HTML Microdata

W3C Working Group Note 29 October 2013

This Version:
http://www.w3.org/TR/2013/NOTE-microdata-20131029/
Latest Published Version:
http://www.w3.org/TR/microdata/
Latest Editor's Draft:
http://www.w3.org/html/wg/drafts/microdata/master/
Previous Versions:
http://www.w3.org/TR/2012/WD-microdata-20121025/
Editor:
Ian Hickson, Google, Inc.

Abstract

This specification defines the HTML microdata mechanism. This mechanism allows machine-readable data to be embedded in HTML documents in an easy-to-write manner, with an unambiguous parsing model. It is compatible with numerous other data formats including RDF and JSON.

Status of This document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document was edited in place on 23 June 2014 to fix a wrong "Previous Version" link.

If you wish to make comments regarding this document in a manner that is tracked by the W3C, please submit them via using our public bug database. If you cannot do this then you can also e-mail feedback to public-html-comments@w3.org (subscribe, archives), and arrangements will be made to transpose the comments to our public bug database. All feedback is welcome.

The bulk of the text of this specification is also available in the WHATWG HTML Living Standard, under a license that permits reuse of the specification text.

Publication as a Working Group Note does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

The latest stable version of the editor's draft of this specification is always available on the W3C HTML git repository.

The W3C HTML Working Group is the W3C working group responsible for this specification's progress. This specification is the 29 October 2013 Working Group Note.

This specification is an extension to the HTML5 language. All normative content in the HTML5 specification, unless specifically overridden by this specification, is intended to be the basis for this specification.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1 Dependencies

This specification depends on the Web IDL and HTML5 specifications. [WEBIDL] [HTML5]

2 Terminology

This specification relies heavily on the HTML5 specification to define underlying terms.

HTML5 defines the concept of DOM **collections** and the `HTMLCollection` interface, as well as the concept of IDL attributes **reflecting** content attributes. It also defines **tree order** and the concept of a node's **home subtree**.

HTML5 defines the terms **URL**, **valid URL**, **absolute URL**, and **resolve a URL**.

HTML5 defines the terms **alphanumeric ASCII characters**, **space characters split a string on spaces**, **converted to ASCII uppercase**, and **prefix match**.

HTML5 defines the meaning of the term **HTML elements**, as well as all the elements referenced in this specification. It also defines the `HTMLElement` and `HTMLDocument` interfaces. It defines the specific concept of **the `title` element** in the context of an `HTMLDocument`. In the context of content models it defines the terms **flow content** and **phrasing content**. It also defines what an element's **ID** or **language** is in HTML.

HTML5 defines the set of **global attributes**, as well as terms used in describing attributes and their processing, such as the concept of a **boolean attribute**, of an **unordered set of unique space-separated tokens**, of a **valid non-negative integer**, of a **date**, a **time**, a **global date and time**, a **valid date string**, and a **valid global date and time string**.

HTML5 defines what **the document's current address** is.

Finally, HTML5 also defines the concepts of **drag-and-drop initialization steps** and of the **list of dragged nodes**, which come up in the context of drag-and-drop interfaces.

3 Conformance requirements

All diagrams, examples, and notes in this specification are non-normative, as are all sections explicitly marked non-normative. Everything else in this specification is normative.

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "MAY", and "OPTIONAL" in the normative parts of this document are to be interpreted as described in RFC2119. The key word "OPTIONALLY" in the normative parts of this document is to be interpreted with the same normative meaning as "MAY" and "OPTIONAL". For readability, these words do not appear in all uppercase letters in this specification. [RFC2119]

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("must", "should", "may", etc) used in introducing the algorithm.

> **Code Example:**
>
> For example, were the spec to say:
>
> ```
> To eat an orange, the user must:
> 1. Peel the orange.
> 2. Separate each slice of the orange.
> 3. Eat the orange slices.
> ```
>
> ...it would be equivalent to the following:
>
> ```
> To eat an orange:
> 1. The user must peel the orange.
> 2. The user must separate each slice of the orange.
> 3. The user must eat the orange slices.
> ```
>
> Here the key word is "must".
>
> The former (imperative) style is generally preferred in this specification for stylistic reasons.

Conformance requirements phrased as algorithms or specific steps may be implemented in any manner, so long as the end result is equivalent. (In particular, the algorithms defined in this specification are intended to be easy to follow, and not intended to be performant.)

4 Introduction

4.1 Overview

This section is non-normative.

Sometimes, it is desirable to annotate content with specific machine-readable labels, e.g. to allow generic scripts to provide services that are customised to the page, or to enable content from a variety of cooperating authors to be processed by a single script in a consistent manner.

For this purpose, authors can use the microdata features described in this section. Microdata allows nested groups of name-value pairs to be added to documents, in parallel with the existing content.

4.2 The basic syntax

This section is non-normative.

At a high level, microdata consists of a group of name-value pairs. The groups are called items, and each name-value pair is a property. Items and properties are represented by regular elements.

To create an item, the `itemscope` attribute is used.

To add a property to an item, the `itemprop` attribute is used on one of the item's descendants.

> **Code Example:**
>
> Here there are two items, each of which has the property "name":
>
> ```
> <div itemscope>
>  My name is Elizabeth.
> 
>
> <div itemscope>
>  My name is Daniel.
> 
> ```

Markup without microdata attributes has no effect on microdata.

> **Code Example:**
>
> These two examples are exactly equivalent, at a microdata level, as the previous two examples respectively:
>
> ```
> <div itemscope>
>  My name is <span
> itemprop="name">Elizabeth.
> 
> ```

```
<section>
 <div itemscope>
  <aside>
   <p>My name is <span itemprop="name"><a href="/?user=daniel">Daniel</a>
</span>.</p>
  </aside>
 </div>
</section>
```

Properties generally have values that are strings.

Code Example:

Here the item has three properties:

```
<div itemscope>
 <p>My name is <span itemprop="name">Neil</span>.</p>
 <p>My band is called <span itemprop="band">Four Parts Water</span>.</p>
 <p>I am <span itemprop="nationality">British</span>.</p>
</div>
```

When a string value is a URL, it is expressed using the `a` element and its `href` attribute, the `img` element and its `src` attribute, or other elements that link to or embed external resources.

Code Example:

In this example, the item has one property, "image", whose value is a URL:

```
<div itemscope>
 <img itemprop="image" src="google-logo.png" alt="Google">
</div>
```

When a string value is in some machine-readable format unsuitable for human consumption, it is expressed using the `value` attribute of the `data` element, with the human-readable version given in the element's contents.

Code Example:

Here, there is an item with a property whose value is a product ID. The ID is not human-friendly, so the product's name is used the human-visible text instead of the ID.

```
<h1 itemscope>
 <data itemprop="product-id" value="9678AOU879">The Instigator
2000</data>
</h1>
```

For numeric data, the `meter` element and its `value` attribute can be used instead.

Code Example:

> Here a rating is given using a `meter` element.
>
> ```
> <div itemscope itemtype="http://schema.org/Product">
>  Panasonic White 60L Refrigerator
>  <img src="panasonic-fridge-60l-white.jpg" alt="">
>   <div itemprop="aggregateRating"
>        itemscope itemtype="http://schema.org/AggregateRating">
>    <meter itemprop="ratingValue" min=0 value=3.5 max=5>Rated
> 3.5/5</meter>
>    (based on 11 customer reviews)
>   
> 
> ```

Similarly, for date- and time-related data, the `time` element and its `datetime` attribute can be used instead.

> **Code Example:**
>
> In this example, the item has one property, "birthday", whose value is a date:
>
> ```
> <div itemscope>
>  I was born on <time itemprop="birthday" datetime="2009-05-10">May 10th
> 2009</time>.
> 
> ```

Properties can also themselves be groups of name-value pairs, by putting the `itemscope` attribute on the element that declares the property.

Items that are not part of others are called top-level microdata items.

> **Code Example:**
>
> In this example, the outer item represents a person, and the inner one represents a band:
>
> ```
> <div itemscope>
>  Name: Amanda
>  Band: <span itemprop="band" itemscope> Jazz
> Band (12 players)
> 
> ```
>
> The outer item here has two properties, "name" and "band". The "name" is "Amanda", and the "band" is an item in its own right, with two properties, "name" and "size". The "name" of the band is "Jazz Band", and the "size" is "12".
>
> The outer item in this example is a top-level microdata item.

Properties that are not descendants of the element with the `itemscope` attribute can be associated with the item using the `itemref` attribute. This attribute takes a list of IDs of elements to crawl in addition to crawling the children of the element with the `itemscope` attribute.

> **Code Example:**

> This example is the same as the previous one, but all the properties are separated from their items:
>
> ```
> <div itemscope id="amanda" itemref="a b">
> Name: Amanda
> <div id="b" itemprop="band" itemscope itemref="c">
> 
>  Band: Jazz Band
>  Size: 12 players
> 
> ```
>
> This gives the same result as the previous example. The first item has two properties, "name", set to "Amanda", and "band", set to another item. That second item has two further properties, "name", set to "Jazz Band", and "size", set to "12".

An item can have multiple properties with the same name and different values.

> **Code Example:**
>
> This example describes an ice cream, with two flavors:
>
> ```
> <div itemscope>
>  Flavors in my favorite ice cream:
>  <ul>
>   <li itemprop="flavor">Lemon sorbet</li>
>   <li itemprop="flavor">Apricot sorbet</li>
>  </ul>
> 
> ```
>
> This thus results in an item with two properties, both "flavor", having the values "Lemon sorbet" and "Apricot sorbet".

An element introducing a property can also introduce multiple properties at once, to avoid duplication when some of the properties have the same value.

> **Code Example:**
>
> Here we see an item with two properties, "favorite-color" and "favorite-fruit", both set to the value "orange":
>
> ```
> <div itemscope>
>  orange
> 
> ```

It's important to note that there is no relationship between the microdata and the content of the document where the microdata is marked up.

> **Code Example:**
>
> There is no semantic difference, for instance, between the following two examples:
>
> ```
> <figure>
>  <img src="castle.jpeg">
>  <figcaption><span itemscope>The Castle
> ```

```
</span> (1986)</figcaption>
</figure>

<span itemscope><meta itemprop="name" content="The Castle"></span>
<figure>
 <img src="castle.jpeg">
 <figcaption>The Castle (1986)</figcaption>
</figure>
```

Both have a figure with a caption, and both, completely unrelated to the figure, have an item with a name-value pair with the name "name" and the value "The Castle". The only difference is that if the user drags the caption out of the document, in the former case, the item will be included in the drag-and-drop data. In neither case is the image in any way associated with the item.

4.3 Typed items

This section is non-normative.

The examples in the previous section show how information could be marked up on a page that doesn't expect its microdata to be re-used. Microdata is most useful, though, when it is used in contexts where other authors and readers are able to cooperate to make new uses of the markup.

For this purpose, it is necessary to give each item a type, such as "http://example.com/person", or "http://example.org/cat", or "http://band.example.net/". Types are identified as URLs.

The type for an item is given as the value of an `itemtype` attribute on the same element as the `itemscope` attribute.

Code Example:

Here, the item's type is "http://example.org/animals#cat":

```
<section itemscope itemtype="http://example.org/animals#cat">
 <h1 itemprop="name">Hedral</h1>
 <p itemprop="desc">Hedral is a male american domestic
 shorthair, with a fluffy black fur with white paws and belly.</p>
 <img itemprop="img" src="hedral.jpeg" alt="" title="Hedral, age 18
months">
</section>
```

In this example the "http://example.org/animals#cat" item has three properties, a "name" ("Hedral"), a "desc" ("Hedral is..."), and an "img" ("hedral.jpeg").

The type gives the context for the properties, thus selecting a vocabulary: a property named "class" given for an item with the type "http://census.example/person" might refer to the economic class of an individual, while a property named "class" given for an item with the type "http://example.com/school/teacher" might refer to the classroom a teacher has been assigned. Several types can share a vocabulary. For example, the types

"`http://example.org/people/teacher`" and "`http://example.org/people/engineer`" could be defined to use the same vocabulary (though maybe some properties would not be especially useful in both cases, e.g. maybe the "`http://example.org/people/engineer`" type might not typically be used with the "`classroom`" property). Multiple types defined to use the same vocabulary can be given for a single item by listing the URLs as a space-separated list in the attribute' value. An item cannot be given two types if they do not use the same vocabulary, however.

4.4 Global identifiers for items

This section is non-normative.

Sometimes, an item gives information about a topic that has a global identifier. For example, books can be identified by their ISBN number.

Vocabularies (as identified by the `itemtype` attribute) can be designed such that items get associated with their global identifier in an unambiguous way by expressing the global identifiers as URLs given in an `itemid` attribute.

The exact meaning of the URLs given in `itemid` attributes depends on the vocabulary used.

> **Code Example:**
>
> Here, an item is talking about a particular book:
>
> ```
> <dl itemscope
>     itemtype="http://vocab.example.net/book"
>     itemid="urn:isbn:0-330-34032-8">
>  <dt>Title
>  <dd itemprop="title">The Reality Dysfunction
>  <dt>Author
>  <dd itemprop="author">Peter F. Hamilton
>  <dt>Publication date
>  <dd><time itemprop="pubdate" datetime="1996-01-26">26 January
> 1996</time>
> </dl>
> ```
>
> The "`http://vocab.example.net/book`" vocabulary in this example would define that the `itemid` attribute takes a `urn:` URL pointing to the ISBN of the book.

4.5 Selecting names when defining vocabularies

This section is non-normative.

Using microdata means using a vocabulary. For some purposes, an ad-hoc vocabulary is adequate. For others, a vocabulary will need to be designed. Where possible, authors are encouraged to re-use existing vocabularies, as this makes content re-use easier.

When designing new vocabularies, identifiers can be created either using URLs, or, for

properties, as plain words (with no dots or colons). For URLs, conflicts with other vocabularies can be avoided by only using identifiers that correspond to pages that the author has control over.

> **Code Example:**
>
> For instance, if Jon and Adam both write content at `example.com`, at `http://example.com/~jon/...` and `http://example.com/~adam/...` respectively, then they could select identifiers of the form "http://example.com/~jon/name" and "http://example.com/~adam/name" respectively.

Properties whose names are just plain words can only be used within the context of the types for which they are intended; properties named using URLs can be reused in items of any type. If an item has no type, and is not part of another item, then if its properties have names that are just plain words, they are not intended to be globally unique, and are instead only intended for limited use. Generally speaking, authors are encouraged to use either properties with globally unique names (URLs) or ensure that their items are typed.

> **Code Example:**
>
> Here, an item is an "http://example.org/animals#cat", and most of the properties have names that are words defined in the context of that type. There are also a few additional properties whose names come from other vocabularies.

```
<section itemscope itemtype="http://example.org/animals#cat">
 <h1 itemprop="name http://example.com/fn">Hedral</h1>
 <p itemprop="desc">Hedral is a male american domestic
 shorthair, with a fluffy <span
 itemprop="http://example.com/color">black</span> fur with <span
 itemprop="http://example.com/color">white</span> paws and belly.</p>
 <img itemprop="img" src="hedral.jpeg" alt="" title="Hedral, age 18
months">
</section>
```

> This example has one item with the type "http://example.org/animals#cat" and the following properties:

Property	Value
name	Hedral
http://example.com/fn	Hedral
desc	Hedral is a male american domestic shorthair, with a fluffy black fur with white paws and belly.
http://example.com/color	black
http://example.com/color	white
img	.../hedral.jpeg

4.6 Using the microdata DOM API

This section is non-normative.

The microdata becomes even more useful when scripts can use it to expose information to the user, for example offering it in a form that can be used by other applications.

The `document.getItems(typeNames)` method provides access to the top-level microdata items. It returns a `NodeList` containing the items with the specified types, or all types if no argument is specified.

Each item is represented in the DOM by the element on which the relevant `itemscope` attribute is found. These elements have their `element.itemScope` IDL attribute set to true.

The type(s) of items can be obtained using the `element.itemType` IDL attribute on the element with the `itemscope` attribute.

> **Code Example:**
>
> This sample shows how the `getItems()` method can be used to obtain a list of all the top-level microdata items of a particular type given in the document:
>
> ```
> var cats = document.getItems("http://example.com/feline");
> ```

Once an element representing an item has been obtained, its properties can be extracted using the `properties` IDL attribute. This attribute returns an `HTMLPropertiesCollection`, which can be enumerated to go through each element that adds one or more properties to the item. It can also be indexed by name, which will return an object with a list of the elements that add properties with that name.

Each element that adds a property also has an `itemValue` IDL attribute that returns its value.

> **Code Example:**
>
> This sample gets the first item of type "http://example.net/user" and then pops up an alert using the "name" property from that item.
>
> ```
> var user = document.getItems('http://example.net/user')[0];
> alert('Hello ' + user.properties['name'][0].itemValue + '!');
> ```

The `HTMLPropertiesCollection` object, when indexed by name in this way, actually returns a `PropertyNodeList` object with all the matching properties. The `PropertyNodeList` object can be used to obtain all the values at once using *its* `getValues` method, which returns an array of all the values.

> **Code Example:**
>
> In an earlier example, a "http://example.org/animals#cat" item had two "http://example.com/color" values. This script looks up the first such item and then lists all its values.

```
var cat = document.getItems('http://example.org/animals#cat')[0];
var colors = cat.properties['http://example.com/color'].getValues();
var result;
if (colors.length == 0) {
  result = 'Color unknown.';
} else if (colors.length == 1) {
  result = 'Color: ' + colors[0];
} else {
  result = 'Colors:';
  for (var i = 0; i < colors.length; i += 1)
    result += ' ' + colors[i];
}
```

It's also possible to get a list of all the property names using the object's `names` IDL attribute.

Code Example:

This example creates a big list with a nested list for each item on the page, each with all of the property names used in that item.

```
var outer = document.createElement('ul');
var items = document.getItems();
for (var item = 0; item < items.length; item += 1) {
  var itemLi = document.createElement('li');
  var inner = document.createElement('ul');
  for (var name = 0; name < items[item].properties.names.length; name +=
1) {
    var propLi = document.createElement('li');

propLi.appendChild(document.createTextNode(items[item].properties.names[name
    inner.appendChild(propLi);
  }
  itemLi.appendChild(inner);
  outer.appendChild(itemLi);
}
document.body.appendChild(outer);
```

If faced with the following from an earlier example:

```
<section itemscope itemtype="http://example.org/animals#cat">
 <h1 itemprop="name http://example.com/fn">Hedral</h1>
 <p itemprop="desc">Hedral is a male american domestic
 shorthair, with a fluffy <span
 itemprop="http://example.com/color">black</span> fur with <span
 itemprop="http://example.com/color">white</span> paws and belly.</p>
 <img itemprop="img" src="hedral.jpeg" alt="" title="Hedral, age 18
months">
</section>
```

...it would result in the following output:

- name
  - http://example.com/fn
  - desc
  - http://example.com/color
  - img

(The duplicate occurrence of "http://example.com/color" is not included in the list.)

5 Encoding microdata

5.1 The microdata model

The microdata model consists of groups of name-value pairs known as items.

Each group is known as an item. Each item can have item types, a global identifier (if the vocabulary specified by the item types support global identifiers for items), and a list of name-value pairs. Each name in the name-value pair is known as a property, and each property has one or more values. Each value is either a string or itself a group of name-value pairs (an item). The names are unordered relative to each other, but if a particular name has multiple values, they do have a relative order.

5.2 Items

Every HTML element may have an **`itemscope`** attribute specified. The `itemscope` attribute is a boolean attribute.

An element with the `itemscope` attribute specified creates a new **item**, a group of name-value pairs.

Elements with an `itemscope` attribute may have an **`itemtype`** attribute specified, to give the item types of the item.

The `itemtype` attribute, if specified, must have a value that is an unordered set of unique space-separated tokens that are case-sensitive, each of which is a valid URL that is an absolute URL, and all of which are defined to use the same vocabulary. The attribute's value must have at least one token.

The **item types** of an item are the tokens obtained by splitting the element's `itemtype` attribute's value on spaces. If the `itemtype` attribute is missing or parsing it in this way finds no tokens, the item is said to have no item types.

The item types must all be types defined in applicable specifications and must all be defined to use the same vocabulary.

Except if otherwise specified by that specification, the URLs given as the item types should not be automatically dereferenced.

> **Note:** A specification could define that its item type can be derefenced to provide the user with help information, for example. In fact, vocabulary authors are encouraged to provide useful information at the given URL.

Item types are opaque identifiers, and user agents must not dereference unknown item types, or otherwise deconstruct them, in order to determine how to process items that use them.

The `itemtype` attribute must not be specified on elements that do not have an `itemscope`

attribute specified.

An item is said to be a **typed item** when either it has an item type, or it is the value of a property of a typed item. The **relevant types** for a typed item is the item's item types, if it has any, or else is the relevant types of the item for which it is a property's value.

Elements with an `itemscope` attribute and an `itemtype` attribute that references a vocabulary that is defined to **support global identifiers for items** may also have an **`itemid`** attribute specified, to give a global identifier for the item, so that it can be related to other items on pages elsewhere on the Web.

The `itemid` attribute, if specified, must have a value that is a valid URL potentially surrounded by spaces.

The **global identifier** of an item is the value of its element's `itemid` attribute, if it has one, resolved relative to the element on which the attribute is specified. If the `itemid` attribute is missing or if resolving it fails, it is said to have no global identifier.

The `itemid` attribute must not be specified on elements that do not have both an `itemscope` attribute and an `itemtype` attribute specified, and must not be specified on elements with an `itemscope` attribute whose `itemtype` attribute specifies a vocabulary that does not support global identifiers for items, as defined by that vocabulary's specification.

The exact meaning of a global identifier is determined by the vocabulary's specification. It is up to such specifications to define whether multiple items with the same global identifier (whether on the same page or on different pages) are allowed to exist, and what the processing rules for that vocabulary are with respect to handling the case of multiple items with the same ID.

Elements with an `itemscope` attribute may have an **`itemref`** attribute specified, to give a list of additional elements to crawl to find the name-value pairs of the item.

The `itemref` attribute, if specified, must have a value that is an unordered set of unique space-separated tokens that are case-sensitive, consisting of IDs of elements in the same home subtree.

The `itemref` attribute must not be specified on elements that do not have an `itemscope` attribute specified.

> **Note:** The `itemref` attribute is not part of the microdata data model. It is merely a syntactic construct to aid authors in adding annotations to pages where the data to be annotated does not follow a convenient tree structure. For example, it allows authors to mark up data in a table so that each column defines a separate item, while keeping the properties in the cells.

> **Code Example:**
>
> This example shows a simple vocabulary used to describe the products of a model railway manufacturer. The vocabulary has just five property names:

product-code
: An integer that names the product in the manufacturer's catalog.

name
: A brief description of the product.

scale
: One of "HO", "1", or "Z" (potentially with leading or trailing whitespace), indicating the scale of the product.

digital
: If present, one of "Digital", "Delta", or "Systems" (potentially with leading or trailing whitespace) indicating that the product has a digital decoder of the given type.

track-type
: For track-specific products, one of "K", "M", "C" (potentially with leading or trailing whitespace) indicating the type of track for which the product is intended.

This vocabulary has four defined item types:

http://md.example.com/loco
: Rolling stock with an engine

http://md.example.com/passengers
: Passenger rolling stock

http://md.example.com/track
: Track pieces

http://md.example.com/lighting
: Equipment with lighting

Each item that uses this vocabulary can be given one or more of these types, depending on what the product is.

Thus, a locomotive might be marked up as:

```
<dl itemscope itemtype="http://md.example.com/loco
                        http://md.example.com/lighting">
 <dt>Name:
 <dd itemprop="name">Tank Locomotive (DB 80)
 <dt>Product code:
 <dd itemprop="product-code">33041
 <dt>Scale:
 <dd itemprop="scale">HO
 <dt>Digital:
 <dd itemprop="digital">Delta
</dl>
```

A turnout lantern retrofit kit might be marked up as:

```
<dl itemscope itemtype="http://md.example.com/track
                       http://md.example.com/lighting">
 <dt>Name:
 <dd itemprop="name">Turnout Lantern Kit
 <dt>Product code:
 <dd itemprop="product-code">74470
 <dt>Purpose:
 <dd>For retrofitting 2 <span itemprop="track-type">C</span> Track
 turnouts. <meta itemprop="scale" content="HO">
</dl>
```

A passenger car with no lighting might be marked up as:

```
<dl itemscope itemtype="http://md.example.com/passengers">
 <dt>Name:
 <dd itemprop="name">Express Train Passenger Car (DB Am 203)
 <dt>Product code:
 <dd itemprop="product-code">8710
 <dt>Scale:
 <dd itemprop="scale">Z
</dl>
```

Great care is necessary when creating new vocabularies. Often, a hierarchical approach to types can be taken that results in a vocabulary where each item only ever has a single type, which is generally much simpler to manage.

5.3 Names: the `itemprop` attribute

Every HTML element may have an `itemprop` attribute specified, if doing so adds one or more properties to one or more items (as defined below).

The `itemprop` attribute, if specified, must have a value that is an unordered set of unique space-separated tokens that are case-sensitive, representing the names of the name-value pairs that it adds. The attribute's value must have at least one token.

Each token must be either:

- If the item is a typed item: a **defined property name** allowed in this situation according to the specification that defines the relevant types for the item, or
- A valid URL that is an absolute URL defined as an item property name allowed in this situation by a vocabulary specification, or
- A valid URL that is an absolute URL, used as a proprietary item property name (i.e. one used by the author for private purposes, not defined in a public specification), or
- If the item is not a typed item: a string that contains no "." (U+002E) characters and no ":" (U+003A) characters, used as a proprietary item property name (i.e. one used by the author for private purposes, not defined in a public specification).

Specifications that introduce defined property names must ensure all such property names contain no "." (U+002E) characters, no ":" (U+003A) characters, and no space characters.

When an element with an `itemprop` attribute adds a property to multiple items, the requirement above regarding the tokens applies for each item individually.

The **property names** of an element are the tokens that the element's `itemprop` attribute is found to contain when its value is split on spaces, with the order preserved but with duplicates removed (leaving only the first occurrence of each name).

Within an item, the properties are unordered with respect to each other, except for properties with the same name, which are ordered in the order they are given by the algorithm that defines the properties of an item.

> **Code Example:**
>
> In the following example, the "a" property has the values "1" and "2", *in that order*, but whether the "a" property comes before the "b" property or not is not important:
>
> ```
> <div itemscope>
>  1
>  2
>  test
> 
> ```
>
> Thus, the following is equivalent:
>
> ```
> <div itemscope>
>  test
>  1
>  2
> 
> ```
>
> As is the following:
>
> ```
> <div itemscope>
>  1
>  test
>  2
> 
> ```
>
> And the following:
>
> ```
> 
>  1
> 
> <div itemscope itemref="x">
>  test
>  2
> 
> ```

5.4 Values

The **property value** of a name-value pair added by an element with an `itemprop` attribute is as given for the first matching case in the following list:

- **If the element also has an `itemscope` attribute**
  The value is the item created by the element.
- **If the element is a `meta` element**
  The value is the value of the element's `content` attribute, if any, or the empty string if there is no such attribute.
- **If the element is an `audio`, `embed`, `iframe`, `img`, `source`, `track`, or `video` element**

The value is the absolute URL that results from resolving the value of the element's `src` attribute relative to the element at the time the attribute is set, or the empty string if there is no such attribute or if resolving it results in an error.

↪ **If the element is an `a`, `area`, or `link` element**
The value is the absolute URL that results from resolving the value of the element's `href` attribute relative to the element at the time the attribute is set, or the empty string if there is no such attribute or if resolving it results in an error.

↪ **If the element is an `object` element**
The value is the absolute URL that results from resolving the value of the element's `data` attribute relative to the element at the time the attribute is set, or the empty string if there is no such attribute or if resolving it results in an error.

↪ **If the element is a `data` element**
The value is the value of the element's `value` attribute, if it has one, or the empty string otherwise.

↪ **If the element is a `meter` element**
The value is the value of the element's `value` attribute, if it has one, or the empty string otherwise.

↪ **If the element is a `time` element**
The value is the element's datetime value.

↪ **Otherwise**
The value is the element's `textContent`.

The **URL property elements** are the `a`, `area`, `audio`, `embed`, `iframe`, `img`, `link`, `object`, `source`, `track`, and `video` elements.

If a property's value, as defined by the property's definition, is an absolute URL, the property must be specified using a URL property element.

> **Note:** These requirements do not apply just because a property value happens to match the syntax for a URL. They only apply if the property is explicitly defined as taking such a value.

> For example, a book about the first moon landing could be called "mission:moon". A "title" property from a vocabulary that defines a title as being a string would not expect the title to be given in an `a` element, even though it looks like a URL. On the other hand, if there was a (rather narrowly scoped!) vocabulary for "books whose titles look like URLs" which had a "title" property defined to take a URL, then the property *would* expect the title to be given in an `a` element (or one of the other URL property elements), because of the requirement above.

5.5 Associating names with items

To find **the properties of an item** defined by the element *root*, the user agent must run

the following steps. These steps are also used to flag microdata errors.

1. Let *results*, *memory*, and *pending* be empty lists of elements.
2. Add the element *root* to *memory*.
3. Add the child elements of *root*, if any, to *pending*.
4. If *root* has an `itemref` attribute, split the value of that `itemref` attribute on spaces. For each resulting token *ID*, if there is an element in the home subtree of *root* with the ID *ID*, then add the first such element to *pending*.
5. *Loop*: If *pending* is empty, jump to the step labeled *end of loop*.
6. Remove an element from *pending* and let *current* be that element.
7. If *current* is already in *memory*, there is a microdata error; return to the step labeled *loop*.
8. Add *current* to *memory*.
9. If *current* does not have an `itemscope` attribute, then: add all the child elements of *current* to *pending*.
10. If *current* has an `itemprop` attribute specified and has one or more property names, then add *current* to *results*.
11. Return to the step labeled *loop*.
12. *End of loop*: Sort *results* in tree order.
13. Return *results*.

A document must not contain any items for which the algorithm to find the properties of an item finds any **microdata errors**.

An item is a **top-level microdata item** if its element does not have an `itemprop` attribute.

All `itemref` attributes in a `Document` must be such that there are no cycles in the graph formed from representing each item in the `Document` as a node in the graph and each property of an item whose value is another item as an edge in the graph connecting those two items.

A document must not contain any elements that have an `itemprop` attribute that would not be found to be a property of any of the items in that document were their properties all to be determined.

> **Code Example:**
>
> In this example, a single license statement is applied to two works, using `itemref` from the items representing the works:
>
> ```
> <!DOCTYPE HTML>
> <html>
> ```

```
 <head>
  <title>Photo gallery</title>
 </head>
 <body>
  <h1>My photos</h1>
  <figure itemscope itemtype="http://n.whatwg.org/work"
itemref="licenses">
   <img itemprop="work" src="images/house.jpeg" alt="A white house,
boarded up, sits in a forest.">
   <figcaption itemprop="title">The house I found.</figcaption>
  </figure>
  <figure itemscope itemtype="http://n.whatwg.org/work"
itemref="licenses">
   <img itemprop="work" src="images/mailbox.jpeg" alt="Outside the house
is a mailbox. It has a leaflet inside.">
   <figcaption itemprop="title">The mailbox.</figcaption>
  </figure>
  <footer>
   <p id="licenses">All images licensed under the <a itemprop="license"
   href="http://www.opensource.org/licenses/mit-license.php">MIT
   license</a>.</p>
  </footer>
 </body>
</html>
```

The above results in two items with the type "`http://n.whatwg.org/work`", one with:

work
: `images/house.jpeg`

title
: The house I found.

license
: `http://www.opensource.org/licenses/mit-license.php`

...and one with:

work
: `images/mailbox.jpeg`

title
: The mailbox.

license
: `http://www.opensource.org/licenses/mit-license.php`

5.6 Microdata and other namespaces

Currently, the `itemscope`, `itemprop`, and other microdata attributes are only defined for HTML elements. This means that attributes with the literal names "`itemscope`", "`itemprop`", etc, do not cause microdata processing to occur on elements in other namespaces, such as SVG.

Code Example:

Thus, in the following example there is only one item, not two.

```
<p itemscope></p> <!-- this is an item (with no properties and no type)
```

```
-->
<svg itemscope></svg> <!-- this is not, it's just an svg element with an
invalid unknown attribute -->
```

6 Microdata DOM API

This definition is non-normative. Implementation requirements are given below this definition.

document **.getItems([*types*])**

Returns a `NodeList` of the elements in the `Document` that create items, that are not part of other items, and that are of the types given in the argument, if any are listed.

The *types* argument is interpreted as a space-separated list of types.

element **.properties**

If the element has an `itemscope` attribute, returns an `HTMLPropertiesCollection` object with all the element's properties. Otherwise, an empty `HTMLPropertiesCollection` object.

element **.itemValue [= *value*]**

Returns the element's value.

Can be set, to change the element's value. Setting the value when the element has no `itemprop` attribute or when the element's value is an item throws an `InvalidAccessError` exception.

The **document.getItems(*typeNames*)** method takes a string that contains an unordered set of unique space-separated tokens that are case-sensitive, representing types. When called, the method must return a live `NodeList` object containing all the elements in the document, in tree order, that are each top-level microdata items whose types include all the types specified in the method's argument, having obtained the types by splitting the string on spaces. If there are no tokens specified in the argument, then the method must return a `NodeList` containing all the top-level microdata items in the document. When the method is invoked on a `Document` object again with the same argument, the user agent may return the same object as the object returned by the earlier call. In other cases, a new `NodeList` object must be returned.

The **itemScope** IDL attribute on HTML elements must reflect the `itemscope` content attribute. The **itemType** IDL attribute on HTML elements must reflect the `itemtype` content attribute. The **itemId** IDL attribute on HTML elements must reflect the `itemid` content attribute. The **itemProp** IDL attribute on HTML elements must reflect the `itemprop` content attribute. The **itemRef** IDL attribute on HTML elements must reflect the `itemref` content attribute.

The **properties** IDL attribute on HTML elements must return an `HTMLPropertiesCollection` rooted at the element's root element (which element this is might change during the collection's lifetime, as the element moves between different subtrees), whose filter matches only elements that are the properties of the item created by the element on which the attribute was invoked, while that element is an item, and matches nothing the rest of the time.

The **itemValue** IDL attribute's behavior depends on the element, as follows:

If the element has no `itemprop` attribute

The attribute must return null on getting and must throw an `InvalidAccessError` exception on setting.

If the element has an `itemscope` attribute
The attribute must return the element itself on getting and must throw an `InvalidAccessError` exception on setting.

If the element is a `meta` element
The attribute must act as it would if it was reflecting the element's `content` content attribute.

If the element is an `audio`, `embed`, `iframe`, `img`, `source`, `track`, or `video` element
The attribute must act as it would if it was reflecting the element's `src` content attribute.

If the element is an `a`, `area`, or `link` element
The attribute must act as it would if it was reflecting the element's `href` content attribute.

If the element is an `object` element
The attribute must act as it would if it was reflecting the element's `data` content attribute.

If the element is a `data` element
The attribute must act as it would if it was reflecting the element's `value` content attribute.

If the element is a `meter` element
The attribute must act as it would if it was reflecting the element's `value` content attribute.

If the element is a `time` element
On getting, if the element has a `datetime` content attribute, the IDL attribute must return that content attribute's value; otherwise, it must return the element's `textContent`. On setting, the IDL attribute must act as it would if it was reflecting the element's `datetime` content attribute.

Otherwise
The attribute must act the same as the element's `textContent` attribute.

When the `itemValue` IDL attribute is reflecting a content attribute or acting like the element's `textContent` attribute, the user agent must, on setting, convert the new value to the IDL `DOMString` value before using it according to the mappings described above.

> **Code Example:**
>
> In this example, a script checks to see if a particular element *element* is declaring a particular property, and if it is, it increments a counter:
>
> ```
> if (element.itemProp.contains('color'))
>   count += 1;
> ```
>
> **Code Example:**

This script iterates over each of the values of an element's `itemref` attribute, calling a function for each referenced element:

```
for (var index = 0; index < element.itemRef.length; index += 1)
  process(document.getElementById(element.itemRef[index]));
```

6.1 HTMLPropertiesCollection

The `HTMLPropertiesCollection` interface is used for collections of elements that add name-value pairs to a particular item in the microdata model.

IDL

```
interface HTMLPropertiesCollection : HTMLCollection {
  // inherits length and item()
  getter PropertyNodeList? namedItem(DOMString name); // shadows inherited
namedItem()
  readonly attribute DOMString[] names;
};

typedef sequence<any> PropertyValueArray;

interface PropertyNodeList : NodeList {
  PropertyValueArray getValues();
};
```

This definition is non-normative. Implementation requirements are given below this definition.

collection **.length**

Returns the number of elements in the collection.

element **=** ***collection*** **.item(** ***index*** **)**
collection **[** ***index*** **]**

Returns the element with index *index* from the collection. The items are sorted in tree order.

propertyNodeList **=** ***collection*** **.namedItem(** ***name*** **)**

Returns a `PropertyNodeList` object containing any elements that add a property named *name*.

collection **[** ***name*** **]**

Returns a `PropertyNodeList` object containing any elements that add a property named *name*. The *name* index has to be one of the values listed in the `names` list.

collection **.names**

Returns an array with the property names of the elements in the collection.

propertyNodeList **.getValues()**

Returns an array of the various values that the relevant elements have.

The object's supported property indices are as defined for `HTMLCollection` objects.

The supported property names consist of the property names of all the elements represented by the collection, in tree order, ignoring later duplicates.

The **`names`** attribute must return a live read only array object giving the property names of all the elements represented by the collection, listed in tree order, but with duplicates removed, leaving only the first occurrence of each name. The same object must be returned each time.

The **`namedItem(`*`name`*`)`** method must return a `PropertyNodeList` object representing a live view of the `HTMLPropertiesCollection` object, further filtered so that the only nodes in the `PropertyNodeList` object are those that have a property name equal to *name*. The nodes in the `PropertyNodeList` object must be sorted in tree order, and the same object must be returned each time a particular *name* is queried.

Members of the `PropertyNodeList` interface inherited from the `NodeList` interface must behave as they would on a `NodeList` object.

The **`getValues`** method the `PropertyNodeList` object must return a newly constructed array whose values are the values obtained from the `itemValue` IDL attribute of each of the elements represented by the object, in tree order.

7 Converting HTML to other formats

7.1 JSON

Given a list of nodes *nodes* in a `Document`, a user agent must run the following algorithm to **extract the microdata from those nodes into a JSON form**:

1. Let *result* be an empty object.
2. Let *items* be an empty array.
3. For each *node* in *nodes*, check if the element is a top-level microdata item, and if it is then get the object for that element and add it to *items*.
4. Add an entry to *result* called "`items`" whose value is the array *items*.
5. Return the result of serializing *result* to JSON in the shortest possible way (meaning no whitespace between tokens, no unnecessary zero digits in numbers, and only using Unicode escapes in strings for characters that do not have a dedicated escape sequence), and with a lowercase "`e`" used, when appropriate, in the representation of any numbers. [JSON]

> **Note:** This algorithm returns an object with a single property that is an array, instead of just returning an array, so that it is possible to extend the algorithm in the future if necessary.

When the user agent is to **get the object** for an item *item*, optionally with a list of elements *memory*, it must run the following substeps:

1. Let *result* be an empty object.
2. If no *memory* was passed to the algorithm, let *memory* be an empty list.
3. Add *item* to *memory*.
4. If the *item* has any item types, add an entry to *result* called "`type`" whose value is an array listing the item types of *item*, in the order they were specified on the `itemtype` attribute.
5. If the *item* has a global identifier, add an entry to *result* called "`id`" whose value is the global identifier of *item*.
6. Let *properties* be an empty object.
7. For each element *element* that has one or more property names and is one of the properties of the item *item*, in the order those elements are given by the algorithm that returns the properties of an item, run the following substeps:
    1. Let *value* be the property value of *element*.
    2. If *value* is an item, then: If *value* is in *memory*, then let *value* be the string

"ERROR". Otherwise, get the object for *value*, passing a copy of *memory*, and then replace *value* with the object returned from those steps.

3. For each name *name* in *element*'s property names, run the following substeps:

    1. If there is no entry named *name* in *properties*, then add an entry named *name* to *properties* whose value is an empty array.

    2. Append *value* to the entry named *name* in *properties*.

8. Add an entry to *result* called "properties" whose value is the object *properties*.

9. Return *result*.

> **Code Example:**
>
> For example, take this markup:

```
<!DOCTYPE HTML>
<title>My Blog</title>
<article itemscope itemtype="http://schema.org/BlogPosting">
 <header>
  <h1 itemprop="headline">Progress report</h1>
  <p><time itemprop="datePublished"
datetime="2013-08-29">today</time></p>
  <link itemprop="url" href="?comments=0">
 </header>
 <p>All in all, he's doing well with his swim lessons. The biggest thing
was he had trouble
 putting his head in, but we got it down.</p>
 <section>
  <h1>Comments</h1>
  <article itemprop="comment" itemscope itemtype="http://schema.org
/UserComments" id="c1">
   <link itemprop="url" href="#c1">
   <footer>
    <p>Posted by: <span itemprop="creator" itemscope
itemtype="http://schema.org/Person">
     <span itemprop="name">Greg</span>
    </span></p>
    <p><time itemprop="commentTime" datetime="2013-08-29">15 minutes
ago</time></p>
   </footer>
   <p>Ha!</p>
  </article>
  <article itemprop="comment" itemscope itemtype="http://schema.org
/UserComments" id="c2">
   <link itemprop="url" href="#c2">
   <footer>
    <p>Posted by: <span itemprop="creator" itemscope
itemtype="http://schema.org/Person">
     <span itemprop="name">Charlotte</span>
    </span></p>
    <p><time itemprop="commentTime" datetime="2013-08-29">5 minutes
ago</time></p>
   </footer>
   <p>When you say "we got it down"...</p>
  </article>
 </section>
</article>
```

It would be turned into the following JSON by the algorithm above (supposing that the page's URL was `http://blog.example.com/progress-report`):

```
{
  "items": [
    {
      "type": [ "http://schema.org/BlogPosting" ],
      "properties": {
        "headline": [ "Progress report" ],
        "datePublished": [ "2013-08-29" ],
        "url": [ "http://blog.example.com/progress-report?comments=0" ],
        "comment": [
          {
            "type": [ "http://schema.org/UserComments" ],
            "properties": {
              "url": [ "http://blog.example.com/progress-report#c1" ],
              "creator": [
                {
                  "type": [ "http://schema.org/Person" ],
                  "properties": {
                    "name": [ "Greg" ]
                  }
                }
              ],
              "commentTime": [ "2013-08-29" ]
            }
          },
          {
            "type": [ "http://schema.org/UserComments" ],
            "properties": {
              "url": [ "http://blog.example.com/progress-report#c2" ],
              "creator": [
                {
                  "type": [ "http://schema.org/Person" ],
                  "properties": {
                    "name": [ "Charlotte" ]
                  }
                }
              ],
              "commentTime": [ "2013-08-29" ]
            }
          }
        ]
      }
    }
  ]
}
```

8 Other changes to HTML5

8.1 Content models

If the `itemprop` attribute is present on `link` or `meta`, they are flow content and phrasing content. The `link` and `meta` elements may be used where phrasing content is expected if the `itemprop` attribute is present.

If a `link` element has an `itemprop` attribute, the `rel` attribute may be omitted.

If a `meta` element has an `itemprop` attribute, the `name`, `http-equiv`, and `charset` attributes must be omitted, and the `content` attribute must be present.

If the `itemprop` is specified on an `a` or `area` element, then the `href` attribute must also be specified.

If the `itemprop` is specified on an `iframe` element, then the `data` attribute must also be specified.

If the `itemprop` is specified on an `embed` element, then the `data` attribute must also be specified.

If the `itemprop` is specified on an `object` element, then the `data` attribute must also be specified.

If the `itemprop` is specified on a media element, then the `src` attribute must also be specified.

8.2 Drag-and-drop

The drag-and-drop initialization steps are:

1. The user agent must take the list of dragged nodes and extract the microdata from those nodes into a JSON form, and then must add the resulting string to the `dataTransfer` member, associated with the `application/microdata+json` format.

9 IANA considerations

application/microdata+json

This registration is for community review and will be submitted to the IESG for review, approval, and registration with IANA.

Type name:
application
Subtype name:
microdata+json
Required parameters:
Same as for `application/json` [JSON]
Optional parameters:
Same as for `application/json` [JSON]
Encoding considerations:
8bit (always UTF-8)
Security considerations:
Same as for `application/json` [JSON]
Interoperability considerations:
Same as for `application/json` [JSON]
Published specification:
Labeling a resource with the `application/microdata+json` type asserts that the resource is a JSON text that consists of an object with a single entry called "`items`" consisting of an array of entries, each of which consists of an object with an entry called "`id`" whose value is a string, an entry called "`type`" whose value is another string, and an entry called "`properties`" whose value is an object whose entries each have a value consisting of an array of either objects or strings, the objects being of the same form as the objects in the aforementioned "`items`" entry. Thus, the relevant specifications are the JSON specification and this specification. [JSON]
Applications that use this media type:
Applications that transfer data intended for use with HTML's microdata feature, especially in the context of drag-and-drop, are the primary application class for this type.

Additional information:
Magic number(s):
Same as for `application/json` [JSON]
File extension(s):
Same as for `application/json` [JSON]
Macintosh file type code(s):
Same as for `application/json` [JSON]
Person & email address to contact for further information:
Ian Hickson <ian@hixie.ch>
Intended usage:
Common
Restrictions on usage:
No restrictions apply.
Author:

Ian Hickson <ian@hixie.ch>

Change controller:

W3C

Fragment identifiers used with `application/microdata+json` resources have the same semantics as when used with `application/json` (namely, at the time of writing, no semantics at all). [JSON]

References

All references are normative unless marked "Non-normative".

[ABNF]
Augmented BNF for Syntax Specifications: ABNF, D. Crocker, P. Overell. IETF.

[ABOUT]
The 'about' URI scheme, S. Moonesamy. IETF.

[AES128CTR]
Advanced Encryption Standard (AES). NIST.

[AGIF]
(Non-normative) *GIF Application Extension: NETSCAPE2.0*. R. Frazier.

[APNG]
(Non-normative) *APNG Specification*. S. Parmenter, V. Vukicevic, A. Smith. Mozilla.

[ARIA]
Accessible Rich Internet Applications (WAI-ARIA), J. Craig, M. Cooper, L. Pappas, R. Schwerdtfeger, L. Seeman. W3C.

[ARIAIMPL]
WAI-ARIA 1.0 User Agent Implementation Guide, A. Snow-Weaver, M. Cooper. W3C.

[ATAG]
(Non-normative) *Authoring Tool Accessibility Guidelines (ATAG) 2.0*, J. Richards, J. Spellman, J. Treviranus. W3C.

[ATOM]
(Non-normative) *The Atom Syndication Format*, M. Nottingham, R. Sayre. IETF.

[BCP47]
Tags for Identifying Languages; Matching of Language Tags, A. Phillips, M. Davis. IETF.

[BECSS]
Behavioral Extensions to CSS, I. Hickson. W3C.

[BEZIER]
Courbes à poles, P. de Casteljau. INPI, 1959.

[BIDI]
UAX #9: Unicode Bidirectional Algorithm, M. Davis. Unicode Consortium.

[BOCU1]
(Non-normative) *UTN #6: BOCU-1: MIME-Compatible Unicode Compression*, M. Scherer, M. Davis. Unicode Consortium.

[CANVAS2D]
(Non-normative) *HTML Canvas 2D Context*, R. Cabanier, E. Graff, J. Munro, T. Wiltzius. W3C.

[CESU8]
(Non-normative) *UTR #26: Compatibility Encoding Scheme For UTF-16: 8-BIT (CESU-8)*, T. Phipps. Unicode Consortium.

[CHARMOD]
(Non-normative) *Character Model for the World Wide Web 1.0: Fundamentals*, M. Dürst, F. Yergeau, R. Ishida, M. Wolf, T. Texin. W3C.

[CLDR]
Unicode Common Locale Data Repository. Unicode.

[COMPOSITE]
Compositing and Blending. R. Cabanier, N. Andronikos. W3C.

[COMPUTABLE]
(Non-normative) *On computable numbers, with an application to the Entscheidungsproblem*, A. Turing. In *Proceedings of the London Mathematical Society*, series 2, volume 42, pages 230-265. London Mathematical Society, 1937.
[COOKIES]
HTTP State Management Mechanism, A. Barth. IETF.
[CORS]
Cross-Origin Resource Sharing, A. van Kesteren. WHATWG.
[CP50220]
(Non-normative) *CP50220*, Y. Naruse. IANA.
[CSP]
(Non-normative) *Content Security Policy*, B. Sterne, A. Barth. W3C.
[CSS]
Cascading Style Sheets Level 2 Revision 1, B. Bos, T. Çelik, I. Hickson, H. Lie. W3C.
[CSSANIMATIONS]
(Non-normative) *CSS Animations*, D. Jackson, D. Hyatt, C. Marrin, S. Galineau, L. Baron. W3C.
[CSSATTR]
CSS Styling Attribute Syntax, T. Çelik, E. Etemad. W3C.
[CSSCOLOR]
CSS Color Module Level 3, T. Çelik, C. Lilley, L. Baron. W3C.
[CSSFONTLOAD]
CSS Font Load Events, J. Daggett. W3C.
[CSSFONTS]
CSS Fonts, J. Daggett. W3C.
[CSSIMAGES]
CSS Image Values and Replaced Content Module, E. Etemad, T. Atkins. W3C.
[CSSOM]
Cascading Style Sheets Object Model (CSSOM), S. Pieters, G. Adams. W3C.
[CSSOMVIEW]
CSSOM View Module, S. Pieters, G. Adams. W3C.
[CSSRUBY]
CSS3 Ruby Module, R. Ishida. W3C.
[CSSTRANSITIONS]
(Non-normative) *CSS Transitions*, D. Jackson, D. Hyatt, C. Marrin, L. Baron. W3C.
[CSSUI]
CSS3 Basic User Interface Module, T. Çelik. W3C.
[CSSSCOPED]
CSS Cascading and Inheritance Level 3, H. Lie, E. Etemad, T. Atkins. W3C
[CSSVALUES]
CSS3 Values and Units, H. Lie, T. Atkins, E. Etemad. W3C.
[DASH]
Dynamic adaptive streaming over HTTP (DASH). ISO.
[DOM]
DOM, A. van Kesteren, A. Gregor, Ms2ger. WHATWG.
[DOMEVENTS]
Document Object Model (DOM) Level 3 Events Specification, T. Leithead, J. Rossi, D. Schepers, B. Höhrmann, P. Le Hégaret, T. Pixley. W3C.
[DOMPARSING]
DOM Parsing and Serialization, T. Leithead. Work in Progress. W3C.

[DOT]
(Non-normative) *The DOT Language*. Graphviz.
[E163]
Recommendation E.163 — Numbering Plan for The International Telephone Service, CCITT Blue Book, Fascicle II.2, pp. 128-134, November 1988.
[ECMA262]
ECMAScript Language Specification. ECMA.
[ECMA357]
(Non-normative) *ECMAScript for XML (E4X) Specification*. ECMA.
[EDITING]
HTML Editing APIs, A. Gregor. W3C Editing APIs CG.
[ENCODING]
Encoding, A. van Kesteren, J. Bell. WHATWG.
[EUCKR]
Hangul Unix Environment. Korea Industrial Standards Association. Ref. No. KS C 5861-1992.
[EUCJP]
Definition and Notes of Japanese EUC. UI-OSF-USLP. In English in the abridged translation of the UI-OSF Application Platform Profile for Japanese Environment, Appendix C.
[EVENTSOURCE]
Server-Sent Events, I. Hickson. W3C.
[FILEAPI]
File API, A. Ranganathan. W3C.
[FILESYSTEMAPI]
File API: Directories and System, E. Uhrhane. W3C.
[FULLSCREEN]
Fullscreen, A. van Kesteren, T. Çelik. WHATWG.
[GBK]
Chinese Internal Code Specification. Chinese IT Standardization Technical Committee.
[GIF]
(Non-normative) *Graphics Interchange Format*. CompuServe.
[GRAPHICS]
(Non-normative) *Computer Graphics: Principles and Practice in C*, Second Edition, J. Foley, A. van Dam, S. Feiner, J. Hughes. Addison-Wesley. ISBN 0-201-84840-6.
[GREGORIAN]
(Non-normative) *Inter Gravissimas*, A. Lilius, C. Clavius. Gregory XIII Papal Bull, February 1582.
[HATOM]
(Non-normative) *hAtom*, D Janes. Microformats.
[HMAC]
The Keyed-Hash Message Authentication Code (HMAC). NIST.
[HPAAIG]
HTML to Platform Accessibility APIs Implementation Guide. W3C.
[HTML4]
(Non-normative) *HTML 4.01 Specification*, D. Raggett, A. Le Hors, I. Jacobs. W3C.
[HTML5]
HTML5, R. Berjon, T. Leithead, E. Doyle Navara, E. O'Connor, S. Pfeiffer. W3C.
[HTML]
HTML, I. Hickson. WHATWG.

[HTMLALTTECHS]
(Non-normative) *HTML5: Techniques for providing useful text alternatives*, S. Faulkner. W3C.
[HTMLDIFF]
(Non-normative) *HTML5 differences from HTML4*, S. Pieters. W3C.
[HTTP]
Hypertext Transfer Protocol — HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. IETF.
[HTTPS]
(Non-normative) *HTTP Over TLS*, E. Rescorla. IETF.
[IANALINKTYPE]
Link Relations. IANA.
[IANAPERMHEADERS]
Permanent Message Header Field Names. IANA.
[ICE]
Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols, J. Rosenberg. IETF.
[IEEE754]
IEEE Standard for Floating-Point Arithmetic (IEEE 754). IEEE. ISBN 978-0-7381-5753-5.
[ISO3166]
ISO 3166: Codes for the representation of names of countries and their subdivisions. ISO.
[ISO8601]
(Non-normative) *ISO8601: Data elements and interchange formats — Information interchange — Representation of dates and times*. ISO.
[JLREQ]
Requirements for Japanese Text Layout. W3C.
[JPEG]
JPEG File Interchange Format, E. Hamilton.
[JSON]
The application/json Media Type for JavaScript Object Notation (JSON), D. Crockford. IETF.
[JSURL]
The 'javascript' resource identifier scheme, B. Höhrmann. IETF. Work in progress.
[MAILTO]
(Non-normative) *The 'mailto' URI scheme*, M. Duerst, L. Masinter, J. Zawinski. IETF.
[MATHML]
Mathematical Markup Language (MathML), D. Carlisle, P. Ion, R. Miner, N. Poppelier. W3C.
[MEDIAFRAG]
Media Fragments URI 1.0, R. Troncy, E. Mannens, S. Pfeiffer, D. Van Deursen. W3C CR.
[MFREL]
Microformats Wiki: existing rel values. Microformats.
[MIMESNIFF]
MIME Sniffing, G. Hemsley. WHATWG.
[MNG]
MNG (Multiple-image Network Graphics) Format. G. Randers-Pehrson.
[MPEG2]
ISO/IEC 13818-1: Information technology — Generic coding of moving pictures and

associated audio information: Systems. ISO/IEC.

[MPEG4]
ISO/IEC 14496-12: ISO base media file format. ISO/IEC.

[MQ]
Media Queries, H. Lie, T. Çelik, D. Glazman, A. van Kesteren. W3C.

[NPAPI]
(Non-normative) *Gecko Plugin API Reference*. Mozilla.

[NPN]
Transport Layer Security (TLS) Next Protocol Negotiation Extension, A. Langley. IETF. Work in progress.

[OGGSKELETONHEADERS]
SkeletonHeaders. Xiph.Org.

[OPENSEARCH]
Autodiscovery in HTML/XHTML. In *OpenSearch 1.1 Draft 4*, Section 4.6.2. OpenSearch.org.

[ORIGIN]
The Web Origin Concept, A. Barth. IETF.

[PAGEVIS]
(Non-normative) *Page Visibility*, J. Mann, A. Jain. W3C.

[PDF]
(Non-normative) *Document management — Portable document format — Part 1: PDF*. ISO.

[PNG]
Portable Network Graphics (PNG) Specification, D. Duce. W3C.

[POINTERLOCK]
Pointer Lock, V. Scheib. W3C.

[POLYGLOT]
(Non-normative) *Polyglot Markup: HTML-Compatible XHTML Documents*, E. Graff. W3C.

[PORTERDUFF]
Compositing Digital Images, T. Porter, T. Duff. In *Computer graphics*, volume 18, number 3, pp. 253-259. ACM Press, July 1984.

[PPUTF8]
(Non-normative) *The Properties and Promises of UTF-8*, M. Dürst. University of Zürich. In *Proceedings of the 11th International Unicode Conference*.

[PSL]
Public Suffix List. Mozilla Foundation.

[RFC1034]
Domain Names - Concepts and Facilities, P. Mockapetris. IETF, November 1987.

[RFC1123]
Requirements for Internet Hosts -- Application and Support, R. Braden. IETF, October 1989.

[RFC1321]
The MD5 Message-Digest Algorithm, R. Rivest. IETF.

[RFC1345]
(Non-normative) *Character Mnemonics and Character Sets*, K. Simonsen. IETF.

[RFC1468]
(Non-normative) *Japanese Character Encoding for Internet Messages*, J. Murai, M. Crispin, E. van der Poel. IETF.

[RFC1494]
(Non-normative) *Equivalences between 1988 X.400 and RFC-822 Message*

Bodies, H. Alvestrand, S. Thompson. IETF.

[RFC1554]
(Non-normative) *ISO-2022-JP-2: Multilingual Extension of ISO-2022-JP*, M. Ohta, K. Handa. IETF.

[RFC1557]
(Non-normative) *Korean Character Encoding for Internet Messages*, U. Choi, K. Chon, H. Park. IETF.

[RFC1842]
(Non-normative) *ASCII Printable Characters-Based Chinese Character Encoding for Internet Messages*, Y. Wei, Y. Zhang, J. Li, J. Ding, Y. Jiang. IETF.

[RFC1922]
(Non-normative) *Chinese Character Encoding for Internet Messages*, HF. Zhu, DY. Hu, ZG. Wang, TC. Kao, WCH. Chang, M. Crispin. IETF.

[RFC2045]
Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, N. Freed, N. Borenstein. IETF.

[RFC2046]
Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, N. Freed, N. Borenstein. IETF.

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels, S. Bradner. IETF.

[RFC2237]
(Non-normative) *Japanese Character Encoding for Internet Messages*, K. Tamaru. IETF.

[RFC2246]
The TLS Protocol Version 1.0, T. Dierks, C. Allen. IETF.

[RFC2313]
PKCS #1: RSA Encryption, B. Kaliski. IETF.

[RFC2318]
The text/css Media Type, H. Lie, B. Bos, C. Lilley. IETF.

[RFC2388]
Returning Values from Forms: multipart/form-data, L. Masinter. IETF.

[RFC2397]
The "data" URL scheme, L. Masinter. IETF.

[RFC2445]
Internet Calendaring and Scheduling Core Object Specification (iCalendar), F. Dawson, D. Stenerson. IETF.

[RFC2483]
URI Resolution Services Necessary for URN Resolution, M. Mealling, R. Daniel. IETF.

[RFC3676]
The Text/Plain Format and DelSp Parameters, R. Gellens. IETF.

[RFC2806]
(Non-normative) *URLs for Telephone Calls*, A. Vaha-Sipila. IETF.

[RFC3023]
XML Media Types, M. Murata, S. St. Laurent, D. Kohn. IETF.

[RFC3279]
Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, W. Polk, R. Housley, L. Bassham. IETF.

[RFC3490]

Internationalizing Domain Names in Applications (IDNA), P. Faltstrom, P. Hoffman, A. Costello. IETF.

[RFC3629]
UTF-8, a transformation format of ISO 10646, F. Yergeau. IETF.

[RFC3864]
Registration Procedures for Message Header Fields, G. Klyne, M. Nottingham, J. Mogul. IETF.

[RFC4281]
The Codecs Parameter for "Bucket" Media Types, R. Gellens, D. Singer, P. Frojdh. IETF.

[RFC4329]
(Non-normative) *Scripting Media Types*, B. Höhrmann. IETF.

[RFC4366]
Transport Layer Security (TLS) Extensions, S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, T. Wright. IETF.

[RFC4395]
Guidelines and Registration Procedures for New URI Schemes, T. Hansen, T. Hardie, L. Masinter. IETF.

[RFC4648]
The Base16, Base32, and Base64 Data Encodings, S. Josefsson. IETF.

[RFC5280]
Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk. IETF.

[RFC5322]
Internet Message Format, P. Resnick. IETF.

[RFC5724]
URI Scheme for Global System for Mobile Communications (GSM) Short Message Service (SMS), E. Wilde, A. Vaha-Sipila. IETF.

[RFC6266]
Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP), J. Reschke. IETF.

[RFC6350]
vCard Format Specification, S. Perreault. IETF.

[SCSU]
(Non-normative) *UTR #6: A Standard Compression Scheme For Unicode*, M. Wolf, K. Whistler, C. Wicksteed, M. Davis, A. Freytag, M. Scherer. Unicode Consortium.

[SDP]
SDP: Session Description Protocol, M. Handley, V. Jacobson, C. Perkins. IETF.

[SDPLABEL]
The Session Description Protocol (SDP) Label Attribute, O. Levin, G. Camarillo. IETF.

[SDPOFFERANSWER]
An Offer/Answer Model with the Session Description Protocol (SDP), J. Rosenberg, H. Schulzrinne. IETF.

[SELECTORS]
Selectors, E. Etemad, T. Çelik, D. Glazman, I. Hickson, P. Linss, J. Williams. W3C.

[SHA1]
Secure Hash Standard. NIST.

[SHIFTJIS]
JIS X0208: 7-bit and 8-bit double byte coded KANJI sets for information

interchange. Japanese Industrial Standards Committee.

[SRGB]
IEC 61966-2-1: Multimedia systems and equipment — Colour measurement and management — Part 2-1: Colour management — Default RGB colour space — sRGB. IEC.

[STUN]
Session Traversal Utilities for NAT (STUN), J. Rosenberg, R. Mahy, P. Matthews, D. Wing. IETF.

[SVG]
Scalable Vector Graphics (SVG) Tiny 1.2 Specification, O. Andersson, R. Berjon, E. Dahlström, A. Emmons, J. Ferraiolo, A. Grasso, V. Hardy, S. Hayman, D. Jackson, C. Lilley, C. McCormack, A. Neumann, C. Northway, A. Quint, N. Ramani, D. Schepers, A. Shellshear. W3C.

[TIS620]
UDC 681.3.04:003.62. Thai Industrial Standards Institute, Ministry of Industry, Royal Thai Government. ISBN 974-606-153-4.

[TURN]
Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN), R. Mahy, P. Matthews, J. Rosenberg. IETF.

[TIMEZONES]
(Non-normative) *Working with Time Zones*, A. Phillips, N. Lindenberg, M. Davis, M.J. Dürst, F. Sasaki, R. Ishida. W3C.

[TYPEDARRAY]
Typed Array Specification, D. Herman, K. Russell. Khronos.

[TZDATABASE]
(Non-normative) *Time Zone Database*. IANA.

[UAAG]
(Non-normative) *User Agent Accessibility Guidelines (UAAG) 2.0*, J. Allan, K. Ford, J. Richards, J. Spellman. W3C.

[UCA]
UTR #10: Unicode Collation Algorithm, M. Davis, K. Whistler. Unicode Consortium.

[UNDO]
UndoManager and DOM Transaction, R. Niwa.

[UNICODE]
The Unicode Standard. Unicode Consortium.

[UNIVCHARDET]
(Non-normative) *A composite approach to language/encoding detection*, S. Li, K. Momoi. Netscape. In *Proceedings of the 19th International Unicode Conference*.

[URL]
URL, A. van Kesteren. WHATWG.

[UTF7]
(Non-normative) *UTF-7: A Mail-Safe Transformation Format of Unicode*, D. Goldsmith, M. Davis. IETF.

[UTF8DET]
(Non-normative) *Multilingual form encoding*, M. Dürst. W3C.

[UTR36]
(Non-normative) *UTR #36: Unicode Security Considerations*, M. Davis, M. Suignard. Unicode Consortium.

[WCAG]
(Non-normative) *Web Content Accessibility Guidelines (WCAG) 2.0*, B. Caldwell, M. Cooper, L. Reid, G. Vanderheiden. W3C.

[WEBGL]
WebGL Specification, D. Jackson. Khronos Group.
[WEBIDL]
Web IDL, C. McCormack. W3C.
[WEBLINK]
Web Linking, M. Nottingham. IETF.
[WEBMSG]
Web Messaging, I. Hickson. W3C.
[WEBMCG]
WebM Container Guidelines. The WebM Project.
[WEBSOCKET]
The WebSocket API, I. Hickson. W3C.
[WEBSTORAGE]
Web Storage, I. Hickson. W3C.
[WEBVTT]
WebVTT, I. Hickson. W3C.
[WEBWORKERS]
Web Workers, I. Hickson. W3C.
[WHATWGBLOG]
The WHATWG Blog. WHATWG.
[WHATWGWIKI]
The WHATWG Wiki. WHATWG.
[WIN1252]
Windows 1252. Microsoft.
[WIN1254]
Windows 1254. Microsoft.
[WIN31J]
Windows Codepage 932. Microsoft.
[WIN874]
Windows 874. Microsoft.
[WIN949]
Windows Codepage 949. Microsoft.
[WSP]
The WebSocket protocol, I. Fette, A. Melnikov. IETF.
[X121]
Recommendation X.121 — International Numbering Plan for Public Data Networks, CCITT Blue Book, Fascicle VIII.3, pp. 317-332.
[X690]
Recommendation X.690 — Information Technology — ASN.1 Encoding Rules — Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER). International Telecommunication Union.
[XFN]
XFN 1.1 profile, T. Çelik, M. Mullenweg, E. Meyer. GMPG.
[XHR]
`XMLHttpRequest`, A. van Kesteren. WHATWG.
[XHTML1]
XHTML(TM) 1.0 The Extensible HyperText Markup Language (Second Edition). W3C.
[XHTMLMOD]
Modularization of XHTML(TM), M. Altheim, F. Boumphrey, S. Dooley, S. McCarron, S. Schnitzenbaumer, T. Wugofski. W3C.

[XML]
Extensible Markup Language, T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau. W3C.
[XMLBASE]
XML Base, J. Marsh, R. Tobin. W3C.
[XMLNS]
Namespaces in XML, T. Bray, D. Hollander, A. Layman, R. Tobin. W3C.
[XPATH10]
XML Path Language (XPath) Version 1.0, J. Clark, S. DeRose. W3C.
[XSLT10]
(Non-normative) *XSL Transformations (XSLT) Version 1.0*, J. Clark. W3C.