HTML Microdata

W3C Working Draft 26 April 2018

This version:
https://www.w3.org/TR/2018/WD-microdata-20180426/

Latest published version:
https://www.w3.org/TR/microdata/

Latest editor's draft:
https://w3c.github.io/microdata/

Previous version:
https://www.w3.org/TR/2017/WD-microdata-20171010/

Editors:
Chaals McCathie Nevile (Yandex / Яндекс)
Dan Brickley (Google, Inc.)

Former editor:
Ian Hickson (Google, Inc.)

Participate:
GitHub w3c/microdata
File a bug
Commit history
Pull requests

Abstract

This [html-extensions] specification defines new HTML attributes to embed simple machine-readable data in HTML documents. It is similar to, but generally less expressive than RDFa, and does not support the same level of internationalization. It is simple to learn and process, but authors who need good internationalization support, or want to include structured content in their data should consider using RDFa. (or another format such as JSON-LD) instead.

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 https://www.w3.org/TR/.

This document is an editor's draft for the Web Platform Working Group, proposed as an update to the 10 October 2017 W3C Working Draft. The editors believe the specification is complete and ready to become a Candidate Recommendation. Review is particularly sought on the algorithm for conversion to RDFa.

If you wish to make comments regarding this document please submit them as github issues. All feedback is welcome, but please note the contribution guidelines require agreement to the terms of the W3C Patent Policy for substantive contributions.

This document was published by the Web Platform Working Group as a Working Draft. This document is intended to become a W3C Recommendation.

Publication as a Working Draft 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.

This document was produced by a group operating under the 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.

This document is governed by the 1 February 2018 W3C Process Document.

Table of Contents

1. Dependencies

This specification is an extension to HTML. All normative content in the HTML specification not specifically overridden by this specification is intended to be normative for this specification. This specification depends on the HTML specification and its extensions for definitions of individual HTML elements and attributes. [HTML52][html-extensions]

2. Terminology

For the purposes of this specification, the terms "URL" and "URI" are equivalent. The URL specification, and RFC 3986 which uses the term URI, define a ***URL***, ***valid URL***, and ***absolute URL***. [RFC3986][URL]

A ***valid absolute URL*** is an absolute URL which is valid.

RFC 3986 defines the term ***resolve a URL*** [RFC3986].

DOM 4.1 defines ***`textContent`*** for attributes, and for elements or nodes, as well as the term ***tree order***. [DOM41]

This specification relies on the HTML specification to define the following terms: [HTML52]

- ***space characters***, ***split a string on spaces***, and an ***unordered set of unique space-separated tokens***.
- ***HTML Element***, ***global attribute***, ***boolean attribute***.
- An element's ***ID*** and ***language***.
- ***flow content*** and ***phrasing content***.

3. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words "must", "must not", "should", "should not", and "may" in the normative parts of this document are to be interpreted as described in RFC2119. [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.

EXAMPLE 1: conformance language

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

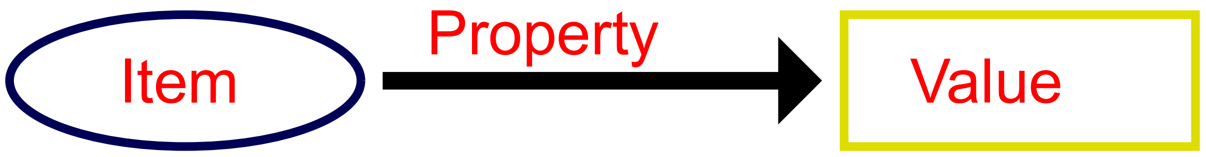
This section is non-normative.

Sometimes, it is desirable to annotate content with specific machine-readable labels. For example, search engines can better identify page content using schema.org annotations, and content management systems can find and use information from documents, if it is marked up in a known way.

Microdata provides a simple mechanism to label content in a document, so it can be processed as a set of items described by name-value pairs.

Each name-value pair identifies a property of the item, and a value of that property.

Figure 1 A common way to represent items, properties and values graphically



The value of a property may be an item.

4.2 The basic syntax

This section is non-normative.

Items and properties are generally represented by regular elements.

The `itemscope` attribute creates an item.

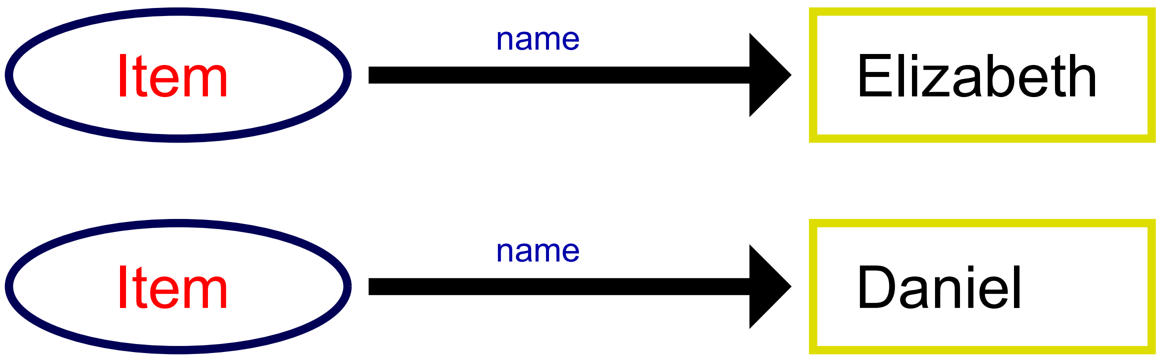
The `itemprop` attribute on a descendent element of an item identifies a property of that item. Typically, the text content of that element is the value of that property.

Here there are two items, each of which has the property "name":

```
<div itemscope>
 <p>My name is
  <span itemprop="name">Elizabeth</span>.</p>
</div>

<div itemscope>
 <p>My name is
  <span itemprop="name">Daniel</span>.</p>
</div>
```

Figure 2 The example represented graphically: two items, each with a value for the property `name`.



Markup other than Microdata attributes has no effect on Microdata.

These two examples are exactly equivalent, at a Microdata level, as the previous two examples respectively:

```
<div itemscope>
 <p>My <em>name</em> is
  <span itemprop="name">E<strong>liz</strong>abeth</span>.</p>
</div>

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

WARNING

Note that this means any information recorded in markup for purposes such as internationalization or accessibility will be lost in the conversion to data. Authors who wish to preserve markup in machine-readable formats should consider using RDFa's ability to use the XMLLiteral datatype.

Properties generally have values that are strings.

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>
```

If the text that would normally be the value of a property, such as the element content, is unsuitable for recording the property value, it can be expressed using the `content` attribute of the element.

Here, the visible content may be added by a script. A Microdata processor can extract the content from the `content` attribute without running scripts. The value of the *product-id* property for this item is `9678AOU879`

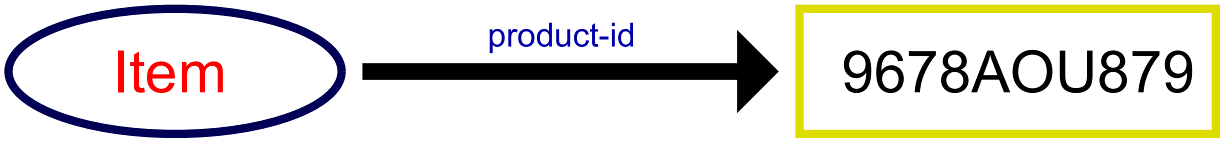
```
<li itemscope>
 <span itemprop="product-id" content="9678AOU879"
     class="reference--id_code-autoinsert"></span>
</li>
```

When a string value is in some machine-readable format unsuitable to present as the content of an element, it can be expressed using the `value` attribute of the `data` element, as long as there is no `content` attribute.

Here, there is an item with a property whose value is a product identifier. The identifier is not human-friendly, so instead it is encoded for Microdata using the `value` attribute of the `data` element, and the product's name is used as the text content of the element that is rendered on the page.

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

Figure 3 The example represented graphically as microdata: an items, whose property `product-id` has the value `9678AOU879`

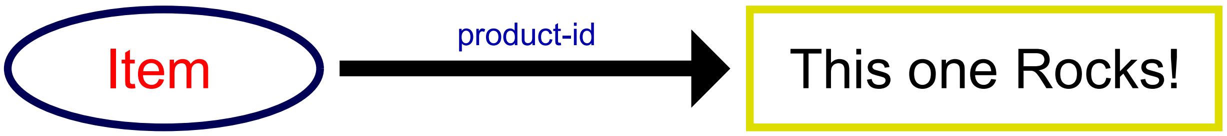


WARNING

This will not work if there is a `content` attribute as well. In the following example, the value of the *product-id* property is taken from the `content` attribute, so it will be `This one rocks!`:

```
<h1 itemscope>
 <data itemprop="product-id" value="9678AOU879"
   content="This one rocks!">The Instigator 2000</data>
</h1>
```

Figure 4 The example represented graphically as microdata: an items, whose property `product-id` has the value `This one Rocks!`



When an `itemprop` is used on an element that can have a `src` or `href` attribute, such as links and media elements, that *does not* have a `content` attribute, the value of the name-value pair is an absolute URL based on the `src` or `href` attribute (or the empty string if they are missing or there is an error).

In this example, the item has one property, *logo*, whose value is a URL based on the location of the page, and ending with `our-logo.png`:

```
<div itemscope itemtype="https://schema.org/LocalBusiness">
 <img itemprop="logo" src="our-logo.png" alt="Our Company">
</div>
```

WARNING

Note that accessibility information, such as the `alt` attribute in the previous example, is ignored. To provide that as a value, repeat it in a `content` attribute. In the following example, the value of the *name* property is `The Company`:

```
<div itemscope itemtype="https://schema.org/LocalBusiness">
 <img itemprop="name" src="our-logo.png"
     content="The Company" alt="Our Company">
</div>
```

For numeric data, the `meter` element and its `value` attribute can be used instead, as long as there is no `content` attribute.

Here a rating of `3.5` is given using a `meter` element.

```
<div itemscope itemtype="https://schema.org/Product">
 <span itemprop="name">Panasonic White 60L Refrigerator</span>
 <img src="panasonic-fridge-60l-white.jpg" alt="">
  <div itemprop="aggregateRating"
       itemscope itemtype="https://schema.org/AggregateRating">
   <meter itemprop="ratingValue" min=0 value=3.5 max=5>Rated 3.5/5</meter>
   (based on <span itemprop="reviewCount">11</span> customer reviews)
  </div>
</div>
```

Similarly, for date- and time-related data, the `time` element and its `datetime` attribute can be used to specify a specifically formatted date or time, as long as there is no `content` attribute.

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</t
</div>
```

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.

In this example, the outer item represents a person, and the inner one represents a band:

```
<div itemscope>
 <p>Name: <span itemprop="name">Amanda</span></p>
 <p>Band: <span itemprop="band" itemscope> <span itemprop="name">Jazz Band</spa
  (<span itemprop="size">12</span> players)</span></p>
</div>
```

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.

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"></div>
<p id="a">Name: <span itemprop="name">Amanda</span></p>
<div id="b" itemprop="band" itemscope itemref="c"></div>
<div id="c">
 <p>Band: <span itemprop="name">Jazz Band</span></p>
 <p>Size: <span itemprop="size">12</span> players</p>
</div>
```

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.

This example describes an ice cream, with two flavors:

```
<div itemscope>
 <p>Flavors in my favorite ice cream:</p>
 <ul>
  <li itemprop="flavor">Lemon sorbet</li>
  <li itemprop="flavor">Apricot sorbet</li>
 </ul>
</div>
```

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.

Here we see an item with two properties, "favorite-color" and "favorite-fruit", both set to the value "orange":

```
<div itemscope>
 <span itemprop="favorite-color favorite-fruit">orange</span>
</div>
```

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.

The following two examples are exactly the same Microdata, because they produce exactly the same information when processed:

```
<figure>
 <img src="castle.jpeg">
 <figcaption><span itemscope><span
  itemprop="name">The Castle</span></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". 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. The value is a URL, which determines the vocabulary identifier for properties

Assuming a page at http://example.net/some/dataexample contains the following code:

```
<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>
```

The item's type is "http://example.org/animals#cat"

In this example the "http://example.org/animals#cat" item has three properties:

`http://example.org/animals#name`
"Hedral"

`http://example.org/animals#desc`
Hedral is a male american domestic shorthair, with a fluffy black fur with white paws and belly.

`http://example.org/animals#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. A vocabulary may define several types. For example, the types "`http://example.org/people/teacher`" and "`http://example.org/people/engineer`" could be defined in the same vocabulary. Some properties might not be especially useful in both cases: the "`classroom`" property might not be meaningful with the "`http://example.org/people/engineer`" type. Multiple types from the same vocabulary can be given for a single item by listing the URLs, separated by spaces, in the attribute's 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, or concepts can be identified by a URL as in [rdf-primer].

The `itemid` attribute associates an item with a global identifier in the form of a URL.

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>
```

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, but authors are encouraged to re-use existing vocabularies to make 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.

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.

Here, an item in the page http://example.net/some/dataexample is an "http://myvocab.example.org/animals/cat", and most of the properties have names defined in the context of that type. There are also a few additional properties whose names come from other vocabularies.

```
<section itemscope itemtype="http://myvocab.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://myvocab.example.org/animals/cat" and the following properties:

http://myvocab.example.org/animals/name
Hedral

http://example.com/fn
Hedral

http://myvocab.example.org/animals/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

http://myvocab.example.org/animals/img
http://example.net/some/hedral.jpeg

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 zero or more item types, global identifier(s), and associated 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). Properties and their values are unordered, and authors must not rely on a particular ordering being preserved.

5.2 Items: `itemscope`, `itemtype`, and `itemid`

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 that describe properties, and their values, of the thing represented by that element.

✧ ✧ ✧

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 absolute URL, and all of which are in 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 determine the ***vocabulary identifier***. This is a URL that is prepended to property names, which identifies them as part of their vocabulary. The value of the vocabulary identifier for an item is determined as follows:

- Let *potential values* be an empty array of URLs.
- Let *tokens be the value of the* `itemtype` *attribute, split on spaces.*
- For each value of *tokens*:

  ↪ **If there is a NUMBER SIGN U+0023 ("#") in the value**

  Append the substring of the value from the beginning to the *first* NUMBER SIGN U+0023 ("#") to *potential values*

  ↪ **Otherwise, if there is a SOLIDUS U+002F ("/") in the value**

Append the substring of the value from the beginning to the *last* SOLIDUS U+002F ("/") to *potential values*

↪ **Otherwise**

Append a SOLIDUS U+002F ("/") to the value, and append the resulting string to *potential values*

- If there is only one unique value in *potential values return that value. Otherwise return the first item in potential values.*

User agents must not automatically dereference *unknown* URLs given as item types and property names. These URLs are *a priori* opaque identifiers.

NOTE

A specification could define that its item types can be derefenced to provide the user with help information. Vocabulary authors are encouraged to provide useful information at the given URL, either in prose or a formal language such as RDF.

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 may also have an ***`itemid`*** attribute specified, to give a global identifier for the item, so that it can be related to other items elsewhere on the Web, or with concepts beyond the Web such as ISBN numbers for published books.

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

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 an `itemscope` attribute.

✧ ✧ ✧

This example shows Microdata used to describe model railway products. manufacturer. The vocabulary identifier is `https://md.example.com/`. The example uses five property names:

product-code
: A number that identifies 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.

It identifies four item types:

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

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

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

https://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="https://md.example.com/loco
                        https://md.example.com/lighting"
                itemid="https://md.example.com/product-catalog/33041">
 <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="https://md.example.com/track
                        https://md.example.com/lighting"
                itemid="https://md.example.com/product-catalog/74470">
  <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="https://md.example.com/passengers"
                itemid="https://md.example.com/product-catalog/8710">
  <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>
```

5.3 Properties: the ***`itemprop`*** and `itemref` attributes

The `itemprop` attribute, when added to any HTML element that is part of an item, identifies a ***property*** of that item. The attribute must be an unordered set of unique space-separated tokens, representing the case-sensitive names of the properties that it adds. The attribute must contain at least one token.

Each token must be either a valid absolute URL or a a string that contains no "." (U+002E) characters and no ":" (U+003A) characters.

Vocabulary specifications must not define property names for Microdata that contain "." (U+002E) characters, ":" (U+003A) characters, nor space characters (defined in [HTML52] as U+0020, U+0009, U+000A, U+000C, and U+000D).

The ***property names*** of an element are determined as follows:

- Let *tokens* be the values of the `itemprop` attribute, Split on spaces.
- Let *properties* be an empty array of strings.
- For each value of *token*, in order:

  ↪ **If the value is a repeated occurrence of an earlier value**
  discard it and process the next value

  ↪ **If the value is an absolute URL**
  append it to *properties*, then process the next value

  ↪ **Otherwise, if the the element is a typed item:**
  Append the value to the vocabulary identifier for the item. If the the resulting value does not match any value in *properties*, then append it to *properties*, and process the next value.

  ↪ **Otherwise**
  append the value to *properties* and process the next value.
- If *properties* is not empty, return *properties*.

Within an item, the properties are unordered with respect to each other and authors must not rely on order in markup being preserved.

In the following example, the orer of the "a" and "b" properties is not important, as it is not guaranteed to be preserved in the resulting microdata:

```
<div itemscope>
 <p itemprop="a">1</p>
 <p itemprop="a">2</p>
 <p itemprop="b">test</p>
</div>
```

Is equivalent to:

```
<div itemscope>
 <p itemprop="b">test</p>
 <p itemprop="a">2</p>
 <p itemprop="a">1</p>
</div>
```

✧ ✧ ✧

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 document.

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

The preceding example:

```
<div itemscope>
 <p itemprop="a">1</p>
 <p itemprop="a">2</p>
 <p itemprop="b">test</p>
</div>
```

Could also be written as follows:

```
<div id="x">
 <p itemprop="a">1</p>
</div>
<div itemscope itemref="x">
 <p itemprop="b">test</p>
 <p itemprop="a">2</p>
</div>
```

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

For the following code:

```
<div itemscope itemtype="http://example.com/a" itemref="x"></div>
<div itemscope itemtype="http://example.com/b" itemref="x"></div>
<meta id="x" itemprop="z" content="">
```

The author should be certain that z is a valid property name for both the http://example.com/a and http://example.com/b vocabularies.

5.4 ***Values***: the `content` attribute, element-specific attributes and element content

The ***algorithm to determine the value*** for a name-value pair is given by applying 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 has a `content` attribute**

The value is the `textContent` of the element's `content` attribute.

> NOTE
>
> HTML only allows the `content` attribute on the `meta` element. This specification changes the content model to allow it on any element, as a global attribute.

↪ **If the element is an `audio`, `embed`, `iframe`, `img`, `source`, `track`, or `video` element**

If the element has a `src` attribute, let *proposed value* be the result of resolving that attribute's `textContent`. If *proposed value* is a valid absolute URL: The value is *proposed value*.
otherwise The value is the empty string.

↪ **If the element is an `a`, `area`, or `link` element**

If the element has an `href` attribute, let *proposed value* be the result of resolving that attribute's `textContent`. If *proposed value* is a valid absolute URL: The value is *proposed value*.
otherwise The value is the empty string.

↪ **If the element is an `object` element**

If the element has a `data` attribute, let *proposed value* be the result of resolving that attribute's `textContent`. If *proposed value* is a valid absolute URL: The value is *proposed value*.
otherwise The value is the empty string.

↪ **If the element is a `data` or `meter` element**

If the element has a `value` attribute, the value is that attribute's `textContent`.

↪ **If the element is a `time` element**

If the element has a `datetime` attribute, the value is that attribute's `textContent`.

↪ **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, on the 20th of July 1967 could be called "mission:moon". A "title" property from a vocabulary that defines a title as a string would not expect the title to be given in an `a` element, even though it is a valid URL. On the other hand, if there was a vocabulary for "books whose titles look like URLs", whose "title" property whose content was defined as a URL, 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 document whose ID is *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 to find the properties of an item finds any ***Microdata Error***.

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 an `itemprop` attribute that would not be a property of any item in that document were the properties all to be determined.

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,
   <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 ma
   <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`

6. Converting Microdata to other formats

6.1 JSON

The section presents an obsolete but conforming feature.

Microdata information can readily be expressed in a JSON form, including the JSON-LD format for RDF. This specification does not seek to limit such conversions, but other than defining a minimal conversion to RDFa which can be used to generate JSON-LD since they are both formally syntaces to express RDF, defines a "reference format". No current implemetation use of this format is known, but it is presented here for historical information.

Given a list of nodes *nodes* in a `Document`, a user agent must run the following ***algorithm to extract the Microdata*** expressed as `application/microdata+json`:

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*, potentially together 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*.

For example, take this markup:

```
<!DOCTYPE HTML>
<title>My Blog</title>
<article itemscope itemtype="https://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
 putting his head in, but we got it down.</p>
 <section>
  <h1>Comments</h1>
  <article itemprop="comment" itemscope itemtype="https://schema.org/Comment" i
   <link itemprop="url" href="#c1">
   <footer>
    <p>Posted by: <span itemprop="creator" itemscope itemtype="https://schema.o
     <span itemprop="name">Greg</span>
    </span></p>
    <p><time itemprop="dateCreated" datetime="2013-08-29">15 minutes ago</time>
   </footer>
   <p>Ha!</p>
  </article>
  <article itemprop="comment" itemscope itemtype="https://schema.org/Comment" i
   <link itemprop="url" href="#c2">
   <footer>
    <p>Posted by: <span itemprop="creator" itemscope itemtype="https://schema.o
     <span itemprop="name">Charlotte</span>
    </span></p>
    <p><time itemprop="dateCreated" datetime="2013-08-29">5 minutes ago</time><
   </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": [ "https://schema.org/BlogPosting" ],
      "properties": {
        "headline": [ "Progress report" ],
        "datePublished": [ "2013-08-29" ],
        "url": [ "http://blog.example.com/progress-report?comments=0" ],
        "comment": [
          {
            "type": [ "https://schema.org/Comment" ],
            "properties": {
              "url": [ "http://blog.example.com/progress-report#c1" ],
              "creator": [
                {
                  "type": [ "https://schema.org/Person" ],
                  "properties": {
                    "name": [ "Greg" ]
                  }
                }
              ],
              "dateCreated": [ "2013-08-29" ]
            }
          },
          {
            "type": [ "https://schema.org/Comment" ],
            "properties": {
              "url": [ "http://blog.example.com/progress-report#c2" ],
              "creator": [
                {
                  "type": [ "https://schema.org/Person" ],
                  "properties": {
                    "name": [ "Charlotte" ]
                  }
                }
              ],
              "dateCreated": [ "2013-08-29" ]
            }
          }
        ]
      }
    }
  ]
}
```

6.2 RDFa

Almost all typed items can be easily converted to RDFa. This is useful for example to support better internationalization, or to include markup in the resulting data, beyond the capabilities of microdata. This specification defines a minimal conversion to the RDFa-Lite dialect of RDFa. which uses a subset of the RDFa functionality. A richer conversion that leverages synergies from using microdata in HTML is described in Microdata to RDF. [microdata-rdf]

To ***convert a Microdata item to RDFa*** the result must include the triples produced by applying the following algorithm:

1. For each element having an `itemscope` attribute:
    1. Replace the `itemscope` attribute with a `vocab` attribute, whose value is the vocabulary identifier for the item.
    2. Replace any `itemtype` attribute with a `typeof` attribute, whose value is taken from the values of `itemtype` where each type is made relative to the vocabulary identifier, ensuring that the element has a `typeof` attribute with at least an empty value.
    3. Replace any `itemid` attribute with a `resource` attribute having the same value.
    4. Otherwise, ensure that element has a `typeof` attribute with at least an empty value.
2. For each `object` element having both an `itemprop` and a `data` attribute, change the `data` attribute to a `resource` attribute with the same value
3. Change each `itemprop` attribute to a `property` attribute with the same value
4. For each element having an `itemref` attribute:
    1. Set *item vocab* to the first value of the `vocab` attribute taken from the element and its ansestors.
    2. For each value *reference* that is a result of splitting the value on spaces follow the following substeps:
        1. Add an element as the immediate parent of the element with `id` *reference*, with a `vocab` attribute whose value is *item vocab*, and a `typeof` attribute with `rel="rdfa:Pattern"` attribute.

NOTE

The choice of element is left to the author, to provide sufficient flexibility to avoid unwanted changes in the rendering of the content.

2. Add a `link` child element to the element that represents the item, with a `rel="rdfa:copy"` attribute and an `href` attribute whose value is a NUMBER SIGN U+0023 followed by *reference*

And then remove the itemref attribute.

NOTE

There is significant scope for optimising this algorithm, which may result in redundant `vocab` declarations in particular.

The example data for a model locomotive given above would be converted to the folling RDFa:

```
<dl vocab="http://md.example.com/" typeof="loco">
 <span rel="rdf:type" resource="http://md.example.com/lighting"></span>
 <dt>Name:
 <dd property="name">Tank Locomotive (DB 80)
 <dt>Product code:
 <dd property="product-code">33041
 <dt>Scale:
 <dd property="http://my.test/scale">HO
 <dt>Digital:
 <dd property="digital">Delta
</dl>
```

An example using `itemref` shows scope for optimisation:

```
  <figure vocab="https://schema.org/" typeof="CreativeWork">
   <link rel="rdfa:copy" href="#licenses">
   <img property="image" src="images/house.jpeg"
    alt="A white house, boarded up, sits in a forest.">
   <figcaption property="name">The house I found.</figcaption>
  </figure>
  <figure vocab="https://schema.org/" typeof="CreativeWork">
   <link rel="rdfa:copy" href="#licenses">
   <img property="image" src="images/mailbox.jpeg"
    alt="Outside the house is a mailbox. It has a leaflet inside.">
   <figcaption property="name">The mailbox.</figcaption>
  </figure>
  <footer>
   <div vocab="https://schema.org/">
    <div vocab="https://schema.org/">
     <p typeof="rdfa:Pattern" resource="#licenses"
      id="licenses">All images licensed under the <a property="license"
      href="http://www.opensource.org/licenses/mit-license.php">MIT license</a>
    </div>
   </div>
  </footer>
```

It could be rewritten as:

```
<div vocab="https://schema.org/">
```

```
  <figure typeof="CreativeWork">
   <link rel="rdfa:copy" href="#licenses">
   <img property="image" src="images/house.jpeg"
    alt="A white house, boarded up, sits in a forest.">
   <figcaption property="name">The house I found.</figcaption>
  </figure>
  <figure typeof="CreativeWork">
   <link rel="rdfa:copy" href="#licenses">
   <img property="image" src="images/mailbox.jpeg"
    alt="Outside the house is a mailbox. It has a leaflet inside.">
   <figcaption property="name">The mailbox.</figcaption>
  </figure>
  <footer>
   <p typeof="rdfa:Pattern" resource="#licenses"
    id="licenses">All images licensed under the <a property="license"
    href="http://www.opensource.org/licenses/mit-license.php">MIT license</a>.<
  </footer>
</div>
```

7. Changes to HTML

7.1 New attributes

This specification adds the following global attributes and associated validity constraints to HTML:

`itemscope`
: This is a boolean attribute. When present on an element, it identifies that element as the container for an item

`itemtype`
: This is a list of absolute URLs that identify an item within a particular vocabulary.

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

  NOTE

  This attribute performs a function similar to the combination of `vocab` and `typeof` attributes in [rdfa-core].

`itemprop`
: When present on an element, it identifies that the element provides the property value of the item in which it appears, and the attribute's value defines the property name.

> NOTE
>
> This attribute is equivalent to the `property` attribute in [rdfa-core].

`itemid`
: This is an absolute URL that provides a global identifier for an item.

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

> NOTE
>
> This is approximately equivalent to declaring that an item is `owl:sameAs` the value of the attribute. [owl-ref]

`itemref`
: This is a space seperated list of IDs of elements which are not descendants of the element on which it appears. It identifies each element whose ID it includes as defining a property of the item on which it is present.

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

7.2 Content models

This section changes the content models defined by HTML in the following ways:

The `content` attribute redefined by this specification as a global attribute that may be present on that element.

This is consistent with [HTML-RDFA], which uses the attribute for the same purpose.

If the `itemprop` attribute is present on a `link` or `meta` element, that element is flow content and phrasing content, and may be used where phrasing content is expected.

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

If the `itemprop` attribute is specified on an `audio`, `embed`, `iframe`, `img`, `source`, `track`, or `video` element, then the `src` attribute must also be specified.

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

8. Microdata and RDF

This section is not normative

Microdata has limited expressivity. There are only two types of data - text strings, and URLs, and microdata does not have a mechanism to describe further datatypes such as numbers or fragments of markup in an interoperable way, or include information such as language, unlike RDF formats including RDFa. To support retaining markup structure, internationalization information, or for more expressivity, authors should consider using [JSON-LD] or [rdfa-core] instead of, or as well as, microdata.

Information expressed as Microdata can be converted to RDFa as described in Section 6. [rdfa-core], A richer process to convert Microdata to RDF, that extends the minimal one required by this specification, is described in Microdata to RDF. [microdata-rdf]

9. Accessibility and Microdata

This section is not normative

Microdata can be used to provide machine-parseable information about content that is processed by tools to improve accessibility.

When editing content that contains Microdata, authors should consider the possibility that this is the case. Authoring and content management tools should implement the Authoring Tool Accessibility Guidelines, and in this context note Guideline B1.1.2 - Ensure Accessibility Information is Preserved, if applicable drawing attention to the fact that changes in content may mean the encoded metadata is not accurate. [ATAG20]

Authors should be aware that a great deal of accessibility information is ignored in extracting Microdata, including attributes such as `alt` and ARIA information. Authors should consider whether to encode accessibility information explicitly, or to use a more expressive approach such as RDFa. [rdfa-core]

10. Internationalisation and localisation

This section is not normative

Microdata does not preserve internationalization-related information in the source document, except if it is specifically encoded as Microdata. In that case it is important to pay attention when editing the source document, as with accessibility, to avoid introducing errors that are only reflected in the microdata. This approach also has the disadvantage that the representation of such information is not based on an established standard, so it may not be understood by downstream processors and users of the information.

Machine-readable data may be presented to users, for example by search engines. Internationalization information is often important for this use case. Authors may prefer to convert their data to RDFa to take advantage of its better support for Internationalization.

Vocabulary design is difficult. Different languages and cultures present view ambiguity differently: two terms with different meanings in one situation may be most naturally translated by a single term that has both meanings, or a single term may have two natural translations. When developing for localisation, it is important to provide sufficient contextual information about terms in a vocabulary to enable accurate translation.

11. Privacy Considerations

This section is not normative

Microdata does not introduce new mechanisms to transmit privacy-sensitive information. However it more clearly identifies information, in a way that facilitates finding data and merging it with data from other sources.

Authors and processors should take care to ensure that their use of Microdata is in line with privacy policies and any applicable regulation.

12. Security Considerations

This section is not normative

Microdata does not generally interact with browsers, being a static document format that lacks any DOM interface.

Microdata makes information machine-readable, but does not automatically include provenance information for the statements it encodes.

Processors of Microdata should consider the trustworthiness of sources they use, including the possibility that data is no longer accurate, and the possibility that data gathered over an insecure connection has been altered by a "man-in-the-middle" attack.

13. 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 Microdata, 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]

For further information:

https://github.com/w3c/microdata/

Intended usage:

Common

Restrictions on usage:

No restrictions apply.

Author:

W3C

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]

14. Changes

An exact history of changes to the text is available in the Github commit log. The following information is provided as an overview of the substantive changes made to the specification between each publication.

Changes made between the current draft and the third Working Draft:

- Specify that properties and their values are unordered.

- Remove the Microdata to JSON-LD conversion.
- Mark Microdata to JSON conversion as an obsolete but conforming feature.

Changes made between the third Working Draft and the second Working Draft:

- Add a section describing how to convert microdata to RDFa.
- Add a section describing how to convert microdata to JSON-LD (now removed).
- `itemid` may be specified on an element with an `itemscope` attribute, not just a typed item

Changes made between the second Working Draft and the First Public Working Draft:

- Allow `itemid` in general, rather than expecting vocabularies to explain what it means.
- Add a mechanism to determine the URL that is a vocabulary identifier, and use it as the start of the property name for any typed item.
- Remove section "Microdata and other namespaces" which implied that Microdata would not work in other namespaces. Because it does.
- Allow the `content` attribute on any element where an `itemprop` attribute is present, to provide a readable value for a property.
- Adjust the algorithm for determining the value of a name-value pair, to match implementation:
  - `data`, `meter`, and `time` elements' `textContent` is used if they do not have an attribute that supplies the value.
  - If there is a `content` attribute present, it provides the value.
- Added non-normative sections for Accessibility and Microdata, Internationalisation and localisation, Privacy, and Security Considerations.
- Remove drag and drop, as it is not implemented in current browsers.

Changes made between the First Public Working Draft and the 23 October 2013 W3C Note:

- Remove DOM API. This API has been removed from browsers that did implement it.
- Remove unused references

15. Acknowledgements

The original specification for Microdata was developed by Ian Hickson. Without him this specification would not exist. Uptake has substantially been driven by its use for the schema.org vocabulary.

The current editors would particularly like to thank Gregg Kellogg and Ivan Herman for invaluable help including providing implementation experience and testing during the development of this specification.

In addition thanks are due to the following people whose direct contributions have helped improve this work:

Addison Phillips, Bruce Lawson, Christine Runnegar, Jeni Tennison, Jens Oliver Meiert, Léonie Watson, Manu Sporny, Marcos Cáceres, Markus Lanthaler, Nick Doty, Nick Levinson, Philippe Le Hégaret, Ralph Swick, Richard Ishida, Rob Sanderson, Robin Berjon, Shane McCarron, Tab Atkins, Tavis Tucker, Tobie Langel, "Unor", Xiaoqian Wu, Yves Lafon.

The editors apologise to people whose names have undeservedly been missed in this list.

A. References

A.1 Normative references

[DOM41]
W3C DOM 4.1. Yongsheng Zhu. W3C. 1 February 2018. W3C Working Draft. URL: https://www.w3.org/TR/dom41/

[HTML52]
HTML 5.2. Steve Faulkner; Arron Eicholz; Travis Leithead; Alex Danilo; Sangwhan Moon. W3C. 14 December 2017. W3C Recommendation. URL: https://www.w3.org/TR/html52/

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC3986]
Uniform Resource Identifier (URI): Generic Syntax. T. Berners-Lee; R. Fielding; L. Masinter. IETF. January 2005. Internet Standard. URL: https://tools.ietf.org/html/rfc3986

A.2 Informative references

[ATAG20]
Authoring Tool Accessibility Guidelines (ATAG) 2.0. Jan Richards; Jeanne F Spellman; Jutta Treviranus. W3C. 24 September 2015. W3C Recommendation. URL: https://www.w3.org/TR/ATAG20/

[html-extensions]
HTML Extension Specifications. Léonie Watson. W3C. 3 May 2017. W3C Editor's Draft. URL: https://w3c.github.io/html-extensions/

[HTML-RDFA]
HTML+RDFa 1.1 - Second Edition. Manu Sporny. W3C. 17 March 2015. W3C Recommendation. URL: https://www.w3.org/TR/html-rdfa/

[JSON]
The application/json Media Type for JavaScript Object Notation (JSON). D. Crockford. IETF. July 2006. Informational. URL: https://tools.ietf.org/html/rfc4627

[JSON-LD]
JSON-LD 1.0. Manu Sporny; Gregg Kellogg; Markus Lanthaler. W3C. 16 January 2014. W3C Recommendation. URL: https://www.w3.org/TR/json-ld/

[microdata-rdf]
Microdata to RDF – Second Edition. Gregg Kellogg. W3C. 16 December 2014. W3C Note. URL: https://www.w3.org/TR/microdata-rdf/

[owl-ref]
OWL Web Ontology Language Reference. Mike Dean; Guus Schreiber. W3C. 10 February 2004. W3C Recommendation. URL: https://www.w3.org/TR/owl-ref/

[rdf-primer]
RDF Primer. Frank Manola; Eric Miller. W3C. 10 February 2004. W3C Recommendation. URL: https://www.w3.org/TR/rdf-primer/

[rdfa-core]
RDFa Core 1.1 - Third Edition. Ben Adida; Mark Birbeck; Shane McCarron; Ivan Herman et al. W3C. 17 March 2015. W3C Recommendation. URL: https://www.w3.org/TR/rdfa-core/

[URL]
URL Standard. Anne van Kesteren. WHATWG. Living Standard. URL: https://url.spec.whatwg.org/

↑