HTML Microdata

This version is outdated!
For the latest version, please look at https://www.w3.org/TR/microdata/.

▲ expand

W3C Working Draft 4 March 2010

This Version:
http://www.w3.org/TR/2010/WD-microdata-20100304/
Latest Published Version:
http://www.w3.org/TR/microdata/
Latest Editor's Draft:
http://dev.w3.org/html5/md/
Previous Versions:
http://www.w3.org/TR/2009/WD-html5-20090825/
Editors:
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 most recently formally published revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

If you wish to make comments regarding this document, please send them to public-html-comments@w3.org (subscribe, archives) or whatwg@whatwg.org (subscribe, archives), or submit them using our public bug database. All feedback is welcome.

The working groups maintains a list of all bug reports that the editor has not yet tried to address and a list of issues for which the chairs have not yet declared a decision. The editor also maintains a list of all e-mails that he has not yet tried to address. These bugs, issues, and e-mails apply to multiple HTML-related specifications, not just this one.

Implementors should be aware that this specification is not stable. **Implementors who are not taking part in the discussions are likely to find the specification changing out from under them in incompatible ways.** Vendors interested in implementing this specification before it eventually reaches the Candidate Recommendation stage should join the aforementioned mailing lists and take part in the discussions.

The publication of this document by the W3C as a W3C Working Draft does not imply that all of the participants in the W3C HTML working group endorse the contents of the specification. Indeed, for any section of the specification, one can usually find many members of the working group or of the W3C as a whole who object strongly to the current text, the existence of the section at all, or the idea that the working group should even spend time discussing the concept of that section.

The latest stable version of the editor's draft of this specification is always available on the W3C CVS server and in the WHATWG Subversion repository. The latest editor's working copy (which may contain unfinished text in the process of being prepared) contains the latest draft text of this specification (amongst others). For more details, please see the WHATWG FAQ.

There are various ways to follow the change history for the HTML specifications:

E-mail notifications of changes

HTML-Diffs mailing list (diff-marked HTML versions for each change): http://lists.w3.org/Archives/Public/public-html-diffs/latest
Commit-Watchers mailing list (complete source diffs): http://lists.whatwg.org/listinfo.cgi/commit-watchers-whatwg.org

Real-time notifications of changes:

Generated diff-marked HTML versions for each change: http://twitter.com/HTML5
All (non-editorial) changes to the spec source: http://twitter.com/WHATWG

Browsable version-control record of all changes:

CVSWeb interface with side-by-side diffs: http://dev.w3.org/cvsweb/html5/
Annotated summary with unified diffs: http://html5.org/tools/web-apps-tracker
Raw Subversion interface: `svn checkout http://svn.whatwg.org/webapps/`

The W3C HTML Working Group is the W3C working group responsible for this

specification's progress along the W3C Recommendation track. This specification is the 4 March 2010 First Public Working Draft.

The contents of this specification are also part of a specification published by the WHATWG, which is available under a license that permits reuse of the specification text.

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 Common infrastructure

1.1 Dependencies

Status: *Last call for comments*

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

1.2 Terminology

Status: *Last call for comments*

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

HTML5 defines the concept of DOM **collections**, and 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 element**, as well as all the elements referenced in this specification. It defines the specific concept of **the `title` element** in the context of a `Document`. 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.

1.3 Conformance requirements

Status: *Last call for comments*

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", "RECOMMENDED", "MAY", and "OPTIONAL" in the normative parts of this document are to be interpreted as described in RFC2119. 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.

This specification describes the conformance criteria for user agents (relevant to implementors) and documents (relevant to authors and authoring tool implementors).

> ***Note: There is no implied relationship between document conformance requirements and implementation conformance requirements. User agents are not free to handle non-conformant documents as they please; the processing model described in this specification applies to implementations regardless of the conformity of the input documents.***

Some conformance requirements are phrased as requirements on elements, attributes, methods or objects. Such requirements fall into two categories: those describing content model restrictions, and those describing implementation behavior. Those in the former category are requirements on documents and authoring tools. Those in the second category are requirements on user agents. Similarly, some conformance requirements are phrased as requirements on authors; such requirements are to be interpreted as conformance requirements on the documents that authors produce. (In other words, this specification does not distinguish between conformance criteria on authors and conformance criteria on documents.)

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.)

User agents may impose implementation-specific limits on otherwise unconstrained inputs, e.g. to prevent denial of service attacks, to guard against running out of memory, or to work around platform-specific limitations.

1.4 HTMLPropertiesCollection

The `HTMLPropertiesCollection` interface represents a collection of elements that add name-value pairs to a particular item in the microdata model.

```
interface HTMLPropertiesCollection : HTMLCollection {
  // inherits length and item()
  caller getter PropertyNodeList namedItem(in DOMString name); //
overrides inherited namedItem()
  readonly attribute DOMStringList names;
};
```

```
typedef sequence<any> PropertyValueArray;

interface PropertyNodeList : NodeList {
  readonly attribute PropertyValueArray values;
};
```

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

collection **.** **`length`**

Returns the number of elements in the collection.

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

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

Returns null if *index* is out of range.

propertyNodeList **=** ***collection*** **.** **`namedItem`**(***name***)
collection[***name***]
collection(***name***)

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

collection **.** **`names`**

Returns a `DOMStringList` with the property names of the elements in the collection.

propertyNodeList **.** **`values`**

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

The object's indices of the supported indexed properties are as defined for `HTMLCollection` objects.

The names of the supported named properties consist of the property names of all the elements represented by the collection.

The **`names`** attribute must return a live `DOMStringList` 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 **`values`** IDL attribute on the `PropertyNodeList` object, on getting, must return a newly constructed array whose values are the values obtained from the `itemValue` DOM property of each of the elements represented by the object, in tree order.

2 Introduction

Status: *Last call for comments*

2.1 Overview

Status: *Last call for comments*

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.

2.2 The basic syntax

Status: *Last call for comments*

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.

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

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>
```

Properties can also have values that are URLs. This is achieved 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.

> 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">
> 
> ```

Properties can also have values that are dates, times, or dates and times. This is achieved using the `time` element and its `datetime` 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</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.

> 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.

> 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"></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.

There is no semantic difference, for instance, between the following two examples:

```
<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". 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.

2.3 Typed items

Status: *Last call for comments*

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.

> Here, the item is "http://example.org/animals#cat":
>
> ```
> <section itemscope itemtype="http://example.org/animals#cat">
>  <h1 itemprop="name">Hedral</h1>
>  Hedral is a male american domestic
>  shorthair, with a fluffy black fur with white paws and belly.
>  <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").

An item can only have one type. The type gives the context for the properties, thus defining 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.

2.4 Global identifiers for items

Status: *Last call for comments*

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.

> 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.

2.5 Selecting names when defining vocabularies

Status: *Last call for comments*

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.

> 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 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

2.6 Using the microdata DOM API

Status: *Last call for comments*

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 of items can be obtained using the `element.itemType` IDL attribute on the element with the `itemscope` attribute.

This sample shows how the `getItems()` method can be used to obtain a list of all the top-level microdata items of one 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 a `itemValue` IDL attribute that returns its value.

> 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].content + '!');
> ```

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* `values` attribute, which returns an array of all the values.

> 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'].values;
> 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.

> This example creates a big list with a nested list for each item on the page, each with of all 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[na
>     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.)

3 Encoding microdata

Status: *Last call for comments*

The following attributes are added as global attributes to HTML elements:

- `itemid`
- `itemprop`
- `itemref`
- `itemscope`
- `itemtype`

3.1 The microdata model

Status: *Last call for comments*

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

Each group is known as an item. Each item can have an item type, a global identifier (if the item type supports global identifiers for its 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).

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 type** for a typed item is the item's item type, if it has one, or else is the relevant type of the item for which it is a property's value.

3.2 Items

Status: *Last call for comments*

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 type of the item.

The `itemtype` attribute, if specified, must have a value that is a valid URL that is an absolute URL for which the string "`http://www.w3.org/1999/xhtml/microdata#`" is not a prefix match.

The **item type** of an item is the value of its element's `itemtype` attribute, if it has one and its value is not the empty string. If the `itemtype` attribute is missing or its value is the empty string, the item is said to have no item type.

The item type must be a type defined in an applicable specification.

Except if otherwise specified by that specification, the URL given as the item type 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.

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.

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.

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 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.

3.3 Names: the `itemprop` attribute

Status: *Last call for comments*

Every HTML element may have an `itemprop` attribute specified, if doing so adds a property 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 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:

- A valid URL that is an absolute URL for which the string "`http://www.w3.org/1999/xhtml/microdata#`" is not a prefix match, or
- If the item is a typed item: a **defined property name** allowed in this situation according to the specification that defines the relevant type for the item, or
- If the item is not a typed item: a string that contains no U+002E FULL STOP characters (.) and no U+003A COLON 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.

> 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:
>
> ```
> <div itemscope itemref="x">
>  test
>  2
> 
> 
>  1
> 
> ```

3.4 Values

Status: *Last call for comments*

The **property value** of a name-value pair added by an element with an `itemprop` attribute depends on the element, as follows:

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`, 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 `time` element with a `datetime` attribute
: The value is the value of the element's `datetime` attribute.

Otherwise
: The value is the element's `textContent`.

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

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

If a property's value represents a date, time, or global date and time, the property must be specified using the `datetime` attribute of a `time` element.

3.5 Associating names with items

Status: *Last call for comments*

To find **the properties of an item** defined by the element *root*, the user agent must try to crawl the properties of the element *root*, with an empty list as the value of *memory*: if this fails, then the properties of the item defined by the element *root* is an empty list; otherwise, it is the returned list.

To **crawl the properties** of an element *root* with a list *memory*, the user agent must run the following steps:

1. If *root* is in *memory*, then the algorithm fails; abort these steps.
2. Collect all the elements in the item *root*, and let *results* be the result.
3. If *root* is in *results*, then the algorithm fails; abort these steps.
4. If any elements are listed in *results* more than once, then the algorithm fails; abort these steps.
5. Remove any elements from *results* that do not have an `itemprop` attribute specified.
6. Let *new memory* be a new list consisting of the old list *memory* with the addition of *root*.
7. For each element in *results* that has an `itemscope` attribute specified, crawl the properties of the element, with *new memory* as the memory. If this fails, then this instance of the algorithm fails as well; abort these steps. (If it succeeds, the return value is discarded.)
8. Sort *results* in tree order.
9. Return *results*.

To **collect all the elements in the item** *root*, the user agent must run these steps:

1. Let *results* and *pending* be empty lists of elements.
2. Add all the children elements of *root* to *pending*.
3. 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*.
4. *Loop*: Remove an element from *pending* and let *current* be that element.

5. Add *current* to *results*.

6. If *current* does not have an `itemscope` attribute, then: add all the child elements of *current* to *pending*.

7. *End of loop*: If *pending* is not empty, return to the step labeled *loop*.

8. Return *results*.

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

An item is a **used microdata item** if it is a top-level microdata item, or if it has an `itemprop` attribute and would be found to be the property of an item that is itself a used microdata item.

A document must not contain any items that are not used microdata 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.

A document must not contain any items for which crawling the properties of the element, with an empty list as the value of *memory*, fails.

Note: The algorithms in this section are especially inefficient, in the interests of keeping them easy to understand. Implementors are strongly encouraged to refactor and optimise them in their user agents.

> 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>
>    All images licensed under the <a itemprop="license"
>    href="http://www.opensource.org/licenses/mit-license.php">MIT
>    license</a>.
>   </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`

In the following invalid example, the items are all empty, because the `itemref` attribute on the inner nested item indirectly references the same element twice in the same item:

```
<!-- invalid example - do not copy -->
<div itemscope>
 <span itemprop="example">This is <em>not</em> a property of the
 <code>div</code> element.</span>
 <p itemprop="demo" itemscope itemref="test thing"> <!-- "thing" is a
descendant
                   of "test", which leads to it being included twice,
which is invalid -->
  <span itemprop="sample">This isn't part of anything either.</span>
 </p>
</div>
<p id="test">
 <span id="thing">(this element is referenced twice by the
 <code>p</code> above, causing all the items that involve that
 <code>itemref=""</code> attribute to act as if they had no
 properties.)</span>
</p>
```

4 Microdata DOM API

Status: *Last call for comments*

```
[Supplemental] interface HTMLDocument {
  NodeList getItems(in optional DOMString typeNames); // microdata
};

[Supplemental] interface HTMLElement {
  // microdata
           attribute boolean itemScope;
           attribute DOMString itemType;
           attribute DOMString itemId;
  [PutForwards=value] readonly attribute DOMSettableTokenList itemRef;
  [PutForwards=value] readonly attribute DOMSettableTokenList itemProp;
  readonly attribute HTMLPropertiesCollection properties;
           attribute any itemValue;
};
```

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

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 one 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 `INVALID_ACCESS_ERR` exception.

The **`document.getItems(`*`typeNames`*`)`** method takes an optional string that contains an unordered set of unique space-separated tokens 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 with a type equal to one of the types specified in that argument, having obtained the types by splitting the string on spaces. If there are no tokens specified in the argument, or if the argument is missing, then the method must return a `NodeList` containing all the top-level microdata items in the document. A new `NodeList` object must be returned each time unless the argument is the same as the last time the method was invoked on this `Document` object, in which case the

object must be the same as the object returned by the previous call.

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, as if it was a regular string attribute, not a URL string 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 `Document` node, whose filter matches only elements that have property names and 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 `INVALID_ACCESS_ERR` exception on setting.

If the element has an `itemscope` attribute
: The attribute must return the element itself on getting and must throw an `INVALID_ACCESS_ERR` 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`, 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 `time` element with a `datetime` attribute
: The 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.

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;
```

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]));
```

5 Other changes to HTML5

5.1 Content models

Status: *Last call for comments*

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.

5.2 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.

6 Converting HTML to other formats

Status: *Last call for comments*

6.1 JSON

Status: *Last call for comments*

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.

When the user agent is to **get the object** for an item *item*, it must run the following substeps:

1. Let *result* be an empty object.
2. If the *item* has an item type, add an entry to *result* called "`type`" whose value is the item type of *item*.
3. If the *item* has an global identifier, add an entry to *result* called "`id`" whose value is the global identifier of *item*.
4. Let *properties* be an empty object.
5. 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 get the object for *value*, 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*.

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

7. Return *result*.

6.2 RDF

Status: *Last call for comments*

To **convert a `Document` to RDF**, a user agent must run the following algorithm:

1. If the `title` element is not null, then generate the following triple:

   subject : the document's current address
   predicate : `http://purl.org/dc/terms/title`
   object : the concatenation of the data of all the child text nodes of the `title` element, in tree order, as a plain literal, with the language information set from the language of the `title` element, if it is not unknown.

2. For each `a`, `area`, and `link` element in the `Document`, run these substeps:

   1. If the element does not have a `rel` attribute, then skip this element.

   2. If the element does not have an `href` attribute, then skip this element.

   3. If resolving the element's `href` attribute relative to the element is not successful, then skip this element.

   4. Otherwise, split the value of the element's `rel` attribute on spaces, obtaining *list of tokens*.

   5. Convert each token in *list of tokens* that does not contain a U+003A COLON characters (:) to ASCII lowercase.

   6. If *list of tokens* contains more than one instance of the token `up`, then remove all such tokens.

   7. Coalesce duplicate tokens in *list of tokens*.

   8. If *list of tokens* contains both the tokens `alternate` and `stylesheet`, then remove them both and replace them with the single (uppercase) token `ALTERNATE-STYLESHEET`.

   9. For each token *token* in *list of tokens* that contains no U+003A COLON characters (:), generate the following triple:

      subject : the document's current address
      predicate : the concatenation of the string "`http://www.w3.org/1999/xhtml/vocab#`" and *token*, with any characters in *token* that are not valid in the

<ifragment> production of the IRI syntax being %-escaped [RFC3987]
object : the absolute URL that results from resolving the value of the element's `href` attribute relative to the element

For each token *token* in *list of tokens* that is an absolute URL, generate the following triple:

subject : the document's current address
predicate : *token*
object : the absolute URL that results from resolving the value of the element's `href` attribute relative to the element

3. For each `meta` element in the `Document` that has a `name` attribute and a `content` attribute, if the value of the `name` attribute contains no U+003A COLON characters (:), generate the following triple:

   subject : the document's current address
   predicate : the concatenation of the string "`http://www.w3.org/1999/xhtml/vocab#`" and the value of the element's `name` attribute, converted to ASCII lowercase, with any characters in the value that are not valid in the <ifragment> production of the IRI syntax being %-escaped [RFC3987]
   object : the value of the element's `content` attribute, as a plain literal, with the language information set from the language of the element, if it is not unknown

   For each `meta` element in the `Document` that has a `name` attribute and a `content` attribute, if the value of the `name` attribute is an absolute URL, generate the following triple:

   subject : the document's current address
   predicate : the value of the element's `name` attribute
   object : the value of the element's `content` attribute, as a plain literal, with the language information set from the language of the element, if it is not unknown

4. For each `blockquote` and `q` element in the `Document` that has a `cite` attribute that resolves successfully relative to the element, generate the following triple:

   subject : the document's current address
   predicate : `http://purl.org/dc/terms/source`
   object : the absolute URL that results from resolving the value of the element's `cite` attribute relative to the element

5. Let *memory* be a mapping of items to subjects, initially empty.

6. For each element that is also a top-level microdata item, run the following steps:

   1. Generate the triples for the item. Pass a reference to *memory* as the item/subject list. Let *result* be the subject returned.

   2. Generate the following triple:

      subject : the document's current address
      predicate : `http://www.w3.org/1999/xhtml/microdata#item`
      object : *result*

When the user agent is to **generate the triples for an item** *item*, given a reference to an item/subject list *memory*, and optionally given a fallback type *fallback type* and property name *fallback name*, it must follow the following steps:

1. If there is an entry for *item* in *memory*, then let *subject* be the subject of that entry. Otherwise, if *item* has a global identifier and that global identifier is an absolute URL, let *subject* be that global identifier. Otherwise, let *subject* be a new blank node.

2. Add a mapping from *item* to *subject* in *memory*, if there isn't one already.

3. If *item* has an item type and that item type is an absolute URL, let *type* be that item type. Otherwise, let *type* be the empty string.

4. If *type* is not the empty string, run the following steps:

    1. Generate the following triple:

        subject : *subject*
        predicate : `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`
        object : *type*

    2. If *type* does not contain a U+0023 NUMBER SIGN character (#), then append a U+0023 NUMBER SIGN character (#) to *type*.

    3. If *type* does not have a U+003A COLON character (:) after its U+0023 NUMBER SIGN character (#), append a U+003A COLON character (:) to *type*.

5. If *type* is the empty string, but *fallback type* is not, run the following substeps:

    1. Let *type* have the value of *fallback type*.

    2. If *type* does not contain a U+0023 NUMBER SIGN character (#), then append a U+0023 NUMBER SIGN character (#) to *type*.

    3. If *type* does not have a U+003A COLON character (:) after its U+0023 NUMBER SIGN character (#), append a U+003A COLON character (:) to *type*.

    4. If the last character of *type* is not a U+003A COLON character (:), append a U+0025 PERCENT SIGN character (%), a U+0032 DIGIT TWO character (2), and a U+0030 DIGIT ZERO character (0) to *type*.

    5. Append the value of *fallback name* to *type*, with any characters in *fallback name* that are not valid in the <ifragment> production of the IRI syntax being %-escaped. [RFC3987]

6. 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 substep:

    1. For each name *name* in *element*'s property names, run the following

substeps:

1. If *type* is the empty string and *name* is not an absolute URL, then abort these substeps.

2. Let *value* be the property value of *element*.

3. If *value* is an item, then generate the triples for *value*. Pass a reference to *memory* as the item/subject list, and pass *type* as the fallback type and *name* as the fallback property name. Replace *value* by the subject returned from those steps.

4. Otherwise, if *element* is not one of the URL property elements, let *value* be a plain literal, with the language information set from the language of the element, if it is not unknown.

5. **If *name* is an absolute URL**
   Let *predicate* be *name*.

   If *name* contains no U+003A COLON characters (:)

   1. Let *s* be *type*.

   2. If the last character of *s* is not a U+003A COLON character (:), append a U+0025 PERCENT SIGN character (%), a U+0032 DIGIT TWO character (2), and a U+0030 DIGIT ZERO character (0) to *s*.

   3. Append the value of *name* to *s*, with any characters in *name* that are not valid in the <ifragment> production of the IRI syntax being %-escaped. [RFC3987]

   4. Let *predicate* be the concatenation of the string "`http://www.w3.org/1999/xhtml/microdata#`" and *s*, with any characters in *s* that are not valid in the <ifragment> production of the IRI syntax being %-escaped. [RFC3987]

6. Generate the following triple:

   subject : *subject*
   predicate : *predicate*
   object : *value*

7. Return *subject*.

6.2.1 Examples

Status: *Last call for comments*

This section is non-normative.

Here is an example of some HTML using Microdata to express RDF statements:

```
<dl itemscope
    itemtype="http://purl.org/vocab/frbr/core#Work"
    itemid="http://books.example.com/works/45U8QJGZSQKDH8N">
 <dt>Title</dt>
 <dd><cite itemprop="http://purl.org/dc/terms/title">Just a Geek</cite>
</dd>
 <dt>By</dt>
 <dd><span itemprop="http://purl.org/dc/terms/creator">Wil
Wheaton</span></dd>
 <dt>Format</dt>
 <dd itemprop="http://purl.org/vocab/frbr/core#realization"
     itemscope
     itemtype="http://purl.org/vocab/frbr/core#Expression"
     itemid="http://books.example.com/products/9780596007683.BOOK">
  <link itemprop="http://purl.org/dc/terms/type"
href="http://books.example.com/product-types/BOOK">
  Print
 </dd>
 <dd itemprop="http://purl.org/vocab/frbr/core#realization"
     itemscope
     itemtype="http://purl.org/vocab/frbr/core#Expression"
     itemid="http://books.example.com/products/9780596802189.EBOOK">
  <link itemprop="http://purl.org/dc/terms/type"
href="http://books.example.com/product-types/EBOOK">
  Ebook
 </dd>
</dl>
```

This is equivalent to the following Turtle:

```
@prefix dc: <http://purl.org/dc/terms/> .
 @prefix frbr: <http://purl.org/vocab/frbr/core#> .

<http://books.example.com/works/45U8QJGZSQKDH8N> a frbr:Work ;
     dc:creator "Wil Wheaton"@en ;
     dc:title "Just a Geek"@en ;
     frbr:realization <http://books.example.com/products
/9780596007683.BOOK>,
         <http://books.example.com/products/9780596802189.EBOOK> .

<http://books.example.com/products/9780596007683.BOOK> a
frbr:Expression ;
     dc:type <http://books.example.com/product-types/BOOK> .

<http://books.example.com/products/9780596802189.EBOOK> a
frbr:Expression ;
     dc:type <http://books.example.com/product-types/EBOOK> .
```

The following snippet of HTML has microdata for two people with the same address:

```
<p>
 Both
 <span itemscope itemtype="http://microformats.org/profile/hcard"
itemref="home"><span itemprop="fn">Princeton</span></span>
 and
 <span itemscope itemtype="http://microformats.org/profile/hcard"
itemref="home"><span itemprop="fn">Trekkie</span></span>
 live at
 <span id="home" itemprop="adr" itemscope><span itemprop="street-
address">Avenue Q</span>.</span>
```

```
</p>
```

It generates these triples expressed in Turtle (including a triple that in this case is expressed twice, though that is not meaningful in RDF):

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix hcard: <http://www.w3.org/1999/xhtml/microdata#http%3A%2F
%2Fmicroformats.org%2Fprofile%2Fhcard%23%3A> .

<> <http://www.w3.org/1999/xhtml/microdata#item> _:n0 ;
   <http://www.w3.org/1999/xhtml/microdata#item> _:n1 .
_:n0 rdf:type <http://microformats.org/profile/hcard> ;
     hcard:fn "Princeton" ;
     hcard:adr _:n2 .
_:n1 rdf:type <http://microformats.org/profile/hcard> ;
     hcard:fn "Trekkie" ;
     hcard:adr _:n2 .
_:n2 hcard:adr%20street-address "Avenue Q" ;
     hcard:adr%20street-address "Avenue Q" .
```

7 IANA considerations

7.1 `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:
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 two entries, one called "`type`" whose value is an array of strings, and one 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. As such, the relevant specifications are the JSON specification and this specification. [JSON]
Applications that use this media type:
Same as for `application/json` [JSON]
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 and WHATWG

Fragment identifiers used with `application/microdata+json` resources have the same semantics as when used with `application/json`. [JSON]

References

Status: *Implemented and widely deployed*

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

[HTML5]
HTML5, I. Hickson, D. Hyatt. W3C.

[JSON]
The application/json Media Type for JavaScript Object Notation (JSON), D. Crockford. IETF.

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

[RFC3987]
Internationalized Resource Identifiers (IRIs), M. Dürst, M. Suignard. IETF.

[WEBIDL]
Web IDL, C. McCormack. W3C.

Acknowledgements

Status: *Last call for comments*

For a full list of acknowledgements, please see the HTML5 specification. [HTML5]