5 Microdata §

5.1 Introduction §

5.1.1 Overview §

This section is non-normative.

Sometimes, it is desirable to annotate content with specific machine-readable labels, e.g. to allow generic scripts to provide services that are customized 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.

5.1.2 The basic syntax §

This section is non-normative.

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

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

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

Example

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>
```

Markup without the microdata-related attributes does not have any effect on the microdata model.

Example

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>
```

Properties generally have values that are strings.

Example

Here the item has three properties:

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

```
 <p>I am <span itemprop="nationality">British</span>.</p>
</div>
```

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

Example

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

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

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

Example

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

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

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

Example

Here a rating is given using a meter element.

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

Example

In this example, the item has one property, "birthday", whose value is a date:

```
<div itemscope>
 I was born on <time itemprop="birthday" datetime="2009-05-10">May 10th 2009</time>.
</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.

Example

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</span> (<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.

Example

This example is the same as the previous one, but all the properties are separated from their items:

```
<div itemscope id="amanda" itemref="a b"></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.

Example

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.

Example

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.

Example

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.

5.1.3 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 "https://example.com/person", or "https://example.org/cat", or "https://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.

Example

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

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

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

The type gives the context for the properties, thus selecting a vocabulary: a property named "class" given for an item with the type "https://census.example/person" might refer to the economic class of an individual, while a property named "class" given for an item with the type "https://example.com/school/teacher" might refer to the classroom a teacher has been assigned. Several types can share a vocabulary. For example, the types "`https://example.org/people/teacher`" and "`https://example.org/people/engineer`" could be defined to use the same vocabulary (though maybe some properties would not be especially useful in both cases, e.g. maybe the "`https://example.org/people/engineer`" type might not typically be used with the "`classroom`" property). Multiple types defined to use the same vocabulary can be given for a single item by listing the URLs as a space-separated list in the attribute' value. An item cannot be given two types if they do not use the same vocabulary, however.

5.1.4 Global identifiers for items §

This section is non-normative.

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

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

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

Example

Here, an item is talking about a particular book:

```
<dl itemscope
    itemtype="https://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 "`https://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.

5.1.5 Selecting names when defining vocabularies §

This section is non-normative.

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

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

Example

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

Example

Here, an item is an "https://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="https://example.org/animals#cat">
 <h1 itemprop="name https://example.com/fn">Hedral</h1>
 <p itemprop="desc">Hedral is a male American domestic
 shorthair, with a fluffy <span
 itemprop="https://example.com/color">black</span> fur with <span
 itemprop="https://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 "https://example.org/animals#cat" and the following properties:

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

5.2 Encoding microdata §

5.2.1 The microdata model §

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

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

5.2.2 Items §

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

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

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

The `itemtype` attribute, if specified, must have a value that is an unordered set of unique space-separated tokens, none of which are identical to another token and each of which is a valid URL string that is an absolute URL, and all of which are defined to use the same vocabulary. The attribute's value must have at least one token.

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

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

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

Note

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

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

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

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

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

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

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

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

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

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

The `itemref` attribute, if specified, must have a value that is an unordered set of unique space-separated tokens none of which are identical to another token and consisting of IDs of elements in the same tree.

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

Note

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

Example

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

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

name
: A brief description of the product.

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

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

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

This vocabulary has four defined item types:

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">
 <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">
 <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">
 <dt>Name:
 <dd itemprop="name">Express Train Passenger Car (DB Am 203)
 <dt>Product code:
 <dd itemprop="product-code">8710
 <dt>Scale:
 <dd itemprop="scale">Z
</dl>
```

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

5.2.3 Names: the `itemprop` attribute §

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

The `itemprop` attribute, if specified, must have a value that is an unordered set of unique space-separated tokens none of which are identical to another token, representing the names of the name-value pairs that it adds. The attribute's value must have at least one token.

Each token must be either:

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

Specifications that introduce defined property names must ensure all such property names contain no U+002E FULL STOP characters (.), no U+003A COLON characters (:), and no ASCII whitespace.

Note

The rules above disallow U+003A COLON characters (:) in non-URL values because otherwise they could not be distinguished from URLs. Values with U+002E FULL STOP characters (.) are reserved for future extensions. ASCII whitespace are disallowed because otherwise the values would be parsed as multiple tokens.

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 ASCII whitespace, 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.

Example

In the following example, the "a" property has the values "1" and "2", *in that order*, but whether the "a" property comes before the "b" property or not is not important:

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

Thus, the following is equivalent:

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

As is the following:

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

And the following:

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

5.2.4 Values §

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

↪ **If the element also has an `itemscope` attribute**

The value is the item created by the element.

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

The value is the value of the element's `content` attribute, if any, or the empty string if there is no such attribute.

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

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

, or `link` element

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

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

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

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

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

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

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

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

The value is the element's datetime value.

↪ **Otherwise**

The value is the element's descendant text content.

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.

Example

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

5.2.5 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 ASCII whitespace. For each resulting token *ID*, if there is an element in the tree of *root* with the ID *ID*, then add the first such element to *pending*.
5. While *pending* is not empty:
    1. Remove an element from *pending* and let *current* be that element.
    2. If *current* is already in *memory*, there is a microdata error; continue.
    3. Add *current* to *memory*.
    4. If *current* does not have an `itemscope` attribute, then: add all the child elements of *current* to *pending*.
    5. If *current* has an `itemprop` attribute specified and has one or more property names, then add *current* to *results*.
6. Sort *results* in tree order.
7. Return *results*.

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

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

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

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

Example

In this example, a single license statement is applied to two works, using itemref from the items representing the works:

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

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

work
images/house.jpeg

title
The house I found.

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

...and one with:

work
images/mailbox.jpeg

title
The mailbox.

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

5.2.6 Microdata and other namespaces §

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

Example

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

```
<p itemscope></p> <!-- this is an item (with no properties and no type) -->
<svg itemscope></svg> <!-- this is not, it's just an SVG svg element with an invalid unknown attribute -->
```

5.3 Sample microdata vocabularies §

The vocabularies in this section are primarily intended to demonstrate how a vocabulary is specified, though they are also usable in their own right.

5.3.1 vCard §

An item with the item type http://microformats.org/profile/hcard represents a person's or organization's contact information.

This vocabulary does not support global identifiers for items.

The following are the type's defined property names. They are based on the vocabulary defined in *vCard Format Specification* (*vCard*) and its extensions, where more information on how to interpret the values can be found. [RFC6350]

kind

Describes what kind of contact the item represents.

The value must be text that is identical to one of the kind strings.

A single property with the name kind may be present within each item with the type http://microformats.org/profile/hcard.

fn

Gives the formatted text corresponding to the name of the person or organization.

The value must be text.

Exactly one property with the name fn must be present within each item with the type http://microformats.org/profile/hcard.

n

Gives the structured name of the person or organization.

The value must be an item with zero or more of each of the family-name, given-name, additional-name, honorific-prefix, and honorific-suffix properties.

Exactly one property with the name n must be present within each item with the type http://microformats.org/profile/hcard.

family-name (inside n)

Gives the family name of the person, or the full name of the organization.

The value must be text.

Any number of properties with the name family-name may be present within the item that forms the value of the n property of an item with the type http://microformats.org/profile/hcard.

given-name (inside n)

Gives the given-name of the person.

The value must be text.

Any number of properties with the name given-name may be present within the item that forms the value of the n property of an item with the type http://microformats.org/profile/hcard.

additional-name (inside n)

Gives the any additional names of the person.

The value must be text.

Any number of properties with the name additional-name may be present within the item that forms the value of the n property of an item with the type http://microformats.org/profile/hcard.

honorific-prefix (inside n)

Gives the honorific prefix of the person.

The value must be text.

Any number of properties with the name honorific-prefix may be present within the item that forms the value of the n property of an item with the type http://microformats.org/profile/hcard.

honorific-suffix (inside n)

Gives the honorific suffix of the person.

The value must be text.

Any number of properties with the name honorific-suffix may be present within the item that forms the value of the n property of an item with the type http://microformats.org/profile/hcard.

nickname

Gives the nickname of the person or organization.

Note

The nickname is the descriptive name given instead of or in addition to the one belonging to a person, place, or thing. It can also be used to specify a familiar [illegible] ified by the fn or n properties.

The value must be text.

Any number of properties with the name nickname may be present within each item with the type http://microformats.org/profile/hcard.

photo

Gives a photograph of the person or organization.

The value must be an absolute URL.

Any number of properties with the name photo may be present within each item with the type http://microformats.org/profile/hcard.

bday

Gives the birth date of the person or organization.

The value must be a valid date string.

A single property with the name bday may be present within each item with the type http://microformats.org/profile/hcard.

anniversary

Gives the birth date of the person or organization.

The value must be a valid date string.

A single property with the name anniversary may be present within each item with the type http://microformats.org/profile/hcard.

sex

Gives the biological sex of the person.

The value must be one of F, meaning "female", M, meaning "male", N, meaning "none or not applicable", O, meaning "other", or U, meaning "unknown".

A single property with the name sex may be present within each item with the type http://microformats.org/profile/hcard.

gender-identity

Gives the gender identity of the person.

The value must be text.

A single property with the name gender-identity may be present within each item with the type http://microformats.org/profile/hcard.

adr

Gives the delivery address of the person or organization.

The value must be an item with zero or more type, post-office-box, extended-address, and street-address properties, and optionally a locality property, optionally a region property, optionally a postal-code property, and optionally a country-name property.

If no type properties are present within an item that forms the value of an adr property of an item with the type http://microformats.org/profile/hcard, then the address type string work is implied.

Any number of properties with the name adr may be present within each item with the type http://microformats.org/profile/hcard.

type (inside adr)

Gives the type of delivery address.

The value must be text that is identical to one of the address type strings.

Any number of properties with the name type may be present within the item that forms the value of an adr property of an item with the type http://microformats.org/profile/hcard, but within each such adr property item there must only be one type property per distinct value.

post-office-box (inside adr)

Gives the post office box component of the delivery address of the person or organization.

The value must be text.

Any number of properties with the name post-office-box may be present within the item that forms the value of an adr property of an item with the type http://microformats.org/profile/hcard.

Note

vCard *urges authors not to use this field.*

extended-address (inside adr)

Gives an additional component of the delivery address of the person or organization.

Any number of properties with the name `extended-address` may be present within the item that forms the value of an `adr` property of an item with the type `http://microformats.org/profile/hcard`.

Note

vCard *urges authors not to use this field.*

`street-address` (inside `adr`)

Gives the street address component of the delivery address of the person or organization.

The value must be text.

Any number of properties with the name `street-address` may be present within the item that forms the value of an `adr` property of an item with the type `http://microformats.org/profile/hcard`.

`locality` (inside `adr`)

Gives the locality component (e.g. city) of the delivery address of the person or organization.

The value must be text.

A single property with the name `locality` may be present within the item that forms the value of an `adr` property of an item with the type `http://microformats.org/profile/hcard`.

`region` (inside `adr`)

Gives the region component (e.g. state or province) of the delivery address of the person or organization.

The value must be text.

A single property with the name `region` may be present within the item that forms the value of an `adr` property of an item with the type `http://microformats.org/profile/hcard`.

`postal-code` (inside `adr`)

Gives the postal code component of the delivery address of the person or organization.

The value must be text.

A single property with the name `postal-code` may be present within the item that forms the value of an `adr` property of an item with the type `http://microformats.org/profile/hcard`.

`country-name` (inside `adr`)

Gives the country name component of the delivery address of the person or organization.

The value must be text.

A single property with the name `country-name` may be present within the item that forms the value of an `adr` property of an item with the type `http://microformats.org/profile/hcard`.

`tel`

Gives the telephone number of the person or organization.

The value must be either text that can be interpreted as a telephone number as defined in the CCITT specifications E.163 and X.121, or an item with zero or more `type` properties and exactly one `value` property. [E163] [X121]

If no `type` properties are present within an item that forms the value of a `tel` property of an item with the type `http://microformats.org/profile/hcard`, or if the value of such a `tel` property is text, then the telephone type string `voice` is implied.

Any number of properties with the name `tel` may be present within each item with the type `http://microformats.org/profile/hcard`.

`type` (inside `tel`)

Gives the type of telephone number.

The value must be text that is identical to one of the telephone type strings.

Any number of properties with the name `type` may be present within the item that forms the value of a `tel` property of an item with the type `http://microformats.org/profile/hcard`, but within each such `tel` property item there must only be one `type` property per distinct value.

`value` (inside `tel`)

Gives the actual telephone number of the person or organization.

The value must be text that can be interpreted as a telephone number as defined in the CCITT specifications E.163 and X.121. [E163] [X121]

Exactly one property with the name `value` must be present within the item that forms the value of a `tel` property of an item with the type `http://microformats.org/profile/hcard`.

email

Gives the email address of the person or organization.

The value must be text.

Any number of properties with the name email may be present within each item with the type http://microformats.org/profile/hcard.

impp

Gives a URL for instant messaging and presence protocol communications with the person or organization.

The value must be an absolute URL.

Any number of properties with the name impp may be present within each item with the type http://microformats.org/profile/hcard.

lang

Gives a language understood by the person or organization.

The value must be a valid BCP 47 language tag. [BCP47].

Any number of properties with the name lang may be present within each item with the type http://microformats.org/profile/hcard.

tz

Gives the time zone of the person or organization.

The value must be text and must match the following syntax:

1. Either a U+002B PLUS SIGN character (+) or a U+002D HYPHEN-MINUS character (-).
2. A valid non-negative integer that is exactly two digits long and that represents a number in the range 00..23.
3. A U+003A COLON character (:).
4. A valid non-negative integer that is exactly two digits long and that represents a number in the range 00..59.

Any number of properties with the name tz may be present within each item with the type http://microformats.org/profile/hcard.

geo

Gives the geographical position of the person or organization.

The value must be text and must match the following syntax:

1. Optionally, either a U+002B PLUS SIGN character (+) or a U+002D HYPHEN-MINUS character (-).
2. One or more ASCII digits.
3. Optionally*, a U+002E FULL STOP character (.) followed by one or more ASCII digits.
4. A U+003B SEMICOLON character (;).
5. Optionally, either a U+002B PLUS SIGN character (+) or a U+002D HYPHEN-MINUS character (-).
6. One or more ASCII digits.
7. Optionally*, a U+002E FULL STOP character (.) followed by one or more ASCII digits.

The optional components marked with an asterisk (*) should be included, and should have six digits each.

Note

The value specifies latitude and longitude, in that order (i.e., "LAT LON" ordering), in decimal degrees. The longitude represents the location east and west of the prime meridian as a positive or negative real number, respectively. The latitude represents the location north and south of the equator as a positive or negative real number, respectively.

Any number of properties with the name geo may be present within each item with the type http://microformats.org/profile/hcard.

title

Gives the job title, functional position or function of the person or organization.

The value must be text.

Any number of properties with the name title may be present within each item with the type http://microformats.org/profile/hcard.

role

Gives the role, occupation, or business category of the person or organization.

The value must be text.

Any number of properties with the name role may be present within each item with the type http://microformats.org/profile/hcard.

logo

Gives the logo of the person or organization.

The value must be an absolute URL.

Any number of properties with the name logo may be present within each item with the type http://microformats.org/profile/hcard.

agent

Gives the contact information of another person who will act on behalf of the person or organization.

The value must be either an item with the type http://microformats.org/profile/hcard, or an absolute URL, or text.

Any number of properties with the name agent may be present within each item with the type http://microformats.org/profile/hcard.

org

Gives the name and units of the organization.

The value must be either text or an item with one organization-name property and zero or more organization-unit properties.

Any number of properties with the name org may be present within each item with the type http://microformats.org/profile/hcard.

organization-name (inside org)

Gives the name of the organization.

The value must be text.

Exactly one property with the name organization-name must be present within the item that forms the value of an org property of an item with the type http://microformats.org/profile/hcard.

organization-unit (inside org)

Gives the name of the organization unit.

The value must be text.

Any number of properties with the name organization-unit may be present within the item that forms the value of the org property of an item with the type http://microformats.org/profile/hcard.

member

Gives a URL that represents a member of the group.

The value must be an absolute URL.

Any number of properties with the name member may be present within each item with the type http://microformats.org/profile/hcard if the item also has a property with the name kind whose value is "group".

related

Gives a relationship to another entity.

The value must be an item with one url property and one rel properties.

Any number of properties with the name related may be present within each item with the type http://microformats.org/profile/hcard.

url (inside related)

Gives the URL for the related entity.

The value must be an absolute URL.

Exactly one property with the name url must be present within the item that forms the value of a related property of an item with the type http://microformats.org/profile/hcard.

rel (inside related)

Gives the relationship between the entity and the related entity.

The value must be text that is identical to one of the relationship strings.

Exactly one property with the name rel must be present within the item that forms the value of a related property of an item with the type http://microformats.org/profile/hcard.

categories

Gives the name of a category or tag that the person or organization could be classified as.

Any number of properties with the name `categories` may be present within each item with the type `http://microformats.org/profile/hcard`.

`note`

Gives supplemental information or a comment about the person or organization.

The value must be text.

Any number of properties with the name `note` may be present within each item with the type `http://microformats.org/profile/hcard`.

`rev`

Gives the revision date and time of the contact information.

The value must be text that is a valid global date and time string.

Note

The value distinguishes the current revision of the information for other renditions of the information.

Any number of properties with the name `rev` may be present within each item with the type `http://microformats.org/profile/hcard`.

`sound`

Gives a sound file relating to the person or organization.

The value must be an absolute URL.

Any number of properties with the name `sound` may be present within each item with the type `http://microformats.org/profile/hcard`.

`uid`

Gives a globally unique identifier corresponding to the person or organization.

The value must be text.

A single property with the name `uid` may be present within each item with the type `http://microformats.org/profile/hcard`.

`url`

Gives a URL relating to the person or organization.

The value must be an absolute URL.

Any number of properties with the name `url` may be present within each item with the type `http://microformats.org/profile/hcard`.

The **kind strings** are:

`individual`

Indicates a single entity (e.g. a person).

`group`

Indicates multiple entities (e.g. a mailing list).

`org`

Indicates a single entity that is not a person (e.g. a company).

`location`

Indicates a geographical place (e.g. an office building).

The **address type strings** are:

`home`

Indicates a delivery address for a residence.

`work`

Indicates a delivery address for a place of work.

The **telephone type strings** are:

`home`

Indicates a residential number.

`work`

Indicates a telephone number for a place of work.

`text`

 umber supports text messages (SMS).

`voice`
: Indicates a voice telephone number.

`fax`
: Indicates a facsimile telephone number.

`cell`
: Indicates a cellular telephone number.

`video`
: Indicates a video conferencing telephone number.

`pager`
: Indicates a paging device telephone number.

`textphone`
: Indicates a telecommunication device for people with hearing or speech difficulties.

The **relationship strings** are:

`emergency`
: An emergency contact.

`agent`
: Another entity that acts on behalf of this entity.

contact
acquaintance
friend
met
worker
colleague
resident
neighbor
child
parent
sibling
spouse
kin
muse
crush
date
sweetheart
me
: Has the meaning defined in XFN. [XFN]

5.3.1.1 Conversion to vCard §

Given a list of nodes *nodes* in a `Document`, a user agent must run the following algorithm to **extract any vCard data represented by those nodes** (only the first vCard is returned):

1. If none of the nodes in *nodes* are items with the item type `http://microformats.org/profile/hcard`, then there is no vCard. Abort the algorithm, returning nothing.
2. Let *node* be the first node in *nodes* that is an item with the item type `http://microformats.org/profile/hcard`.
3. Let *output* be an empty string.
4. Add a vCard line with the type "`BEGIN`" and the value "`VCARD`" to *output*.
5. Add a vCard line with the type "`PROFILE`" and the value "`VCARD`" to *output*.
6. Add a vCard line with the type "`VERSION`" and the value "`4.0`" to *output*.
7. Add a vCard line with the type "`SOURCE`" and the result of escaping the vCard text string that is the document's URL as the value to *output*.
8. If the `title` element is not null, add a vCard line with the type "`NAME`" and with the result of escaping the vCard text string obtained from the `title` element's

as the value to *output*.

9. Let *sex* be the empty string.
10. Let *gender-identity* be the empty string.
11. For each element *element* that is a property of the item *node*: for each name *name* in *element*'s property names, run the following substeps:
    1. Let *parameters* be an empty set of name-value pairs.
    2. Run the appropriate set of substeps from the following list. The steps will set a variable *value*, which is used in the next step.

    If the property's value is an item *subitem* and *name* is `n`

    1. Let *value* be the empty string.
    2. Append to *value* the result of collecting the first vCard subproperty named `family-name` in *subitem*.
    3. Append a U+003B SEMICOLON character (;) to *value*.
    4. Append to *value* the result of collecting the first vCard subproperty named `given-name` in *subitem*.
    5. Append a U+003B SEMICOLON character (;) to *value*.
    6. Append to *value* the result of collecting the first vCard subproperty named `additional-name` in *subitem*.
    7. Append a U+003B SEMICOLON character (;) to *value*.
    8. Append to *value* the result of collecting the first vCard subproperty named `honorific-prefix` in *subitem*.
    9. Append a U+003B SEMICOLON character (;) to *value*.
    10. Append to *value* the result of collecting the first vCard subproperty named `honorific-suffix` in *subitem*.

    If the property's value is an item *subitem* and *name* is `adr`

    1. Let *value* be the empty string.
    2. Append to *value* the result of collecting vCard subproperties named `post-office-box` in *subitem*.
    3. Append a U+003B SEMICOLON character (;) to *value*.
    4. Append to *value* the result of collecting vCard subproperties named `extended-address` in *subitem*.
    5. Append a U+003B SEMICOLON character (;) to *value*.
    6. Append to *value* the result of collecting vCard subproperties named `street-address` in *subitem*.
    7. Append a U+003B SEMICOLON character (;) to *value*.
    8. Append to *value* the result of collecting the first vCard subproperty named `locality` in *subitem*.
    9. Append a U+003B SEMICOLON character (;) to *value*.
    10. Append to *value* the result of collecting the first vCard subproperty named `region` in *subitem*.
    11. Append a U+003B SEMICOLON character (;) to *value*.
    12. Append to *value* the result of collecting the first vCard subproperty named `postal-code` in *subitem*.
    13. Append a U+003B SEMICOLON character (;) to *value*.
    14. Append to *value* the result of collecting the first vCard subproperty named `country-name` in *subitem*.
    15. If there is a property named `type` in *subitem*, and the first such property has a value that is not an item and whose value consists only of ASCII alphanumerics, then add a parameter named "`TYPE`" whose value is the value of that property to *parameters*.

    If the property's value is an item *subitem* and *name* is `org`

    1. Let *value* be the empty string.
    2. Append to *value* the result of collecting the first vCard subproperty named `organization-name` in *subitem*.
    3. For each property named `organization-unit` in *subitem*, run the following steps:
        1. If the value of the property is an item, then skip this property.
        2. Append a U+003B SEMICOLON character (;) to *value*.
        3. Append the result of escaping the vCard text string given by the value of the property to *value*.

    If the property's value is an item *subitem* with the item type `http://microformats.org/profile/hcard` and *name* is `related`

    1. Let *value* be the empty string.
    2. If there is a property named `url` in *subitem*, and its element is a URL property element, then append the result of escaping the vCard string given by the value of the first such property to *value*, and add a parameter with the name "`VALUE`" and the value "`URI`" to

parameters.

3. If there is a property named `rel` in *subitem*, and the first such property has a value that is not an item and whose value consists only of ASCII alphanumerics, then add a parameter named "`RELATION`" whose value is the value of that property to *parameters*.

If the property's value is an item and *name* is none of the above

1. Let *value* be the result of collecting the first vCard subproperty named `value` in *subitem*.
2. If there is a property named `type` in *subitem*, and the first such property has a value that is not an item and whose value consists only of ASCII alphanumeric, then add a parameter named "`TYPE`" whose value is the value of that property to *parameters*.

If the property's value is not an item and its *name* is `sex`

If this is the first such property to be found, set *sex* to the property's value.

If the property's value is not an item and its *name* is `gender-identity`

If this is the first such property to be found, set *gender-identity* to the property's value.

Otherwise (the property's value is not an item)

1. Let *value* be the property's value.
2. If *element* is one of the URL property elements, add a parameter with the name "`VALUE`" and the value "`URI`" to *parameters*.
3. Otherwise, if *name* is `bday` or `anniversary` and the *value* is a valid date string, add a parameter with the name "`VALUE`" and the value "`DATE`" to *parameters*.
4. Otherwise, if *name* is `rev` and the *value* is a valid global date and time string, add a parameter with the name "`VALUE`" and the value "`DATE-TIME`" to *parameters*.
5. Prefix every U+005C REVERSE SOLIDUS character (\) in *value* with another U+005C REVERSE SOLIDUS character (\).
6. Prefix every U+002C COMMA character (,) in *value* with a U+005C REVERSE SOLIDUS character (\).
7. Unless *name* is `geo`, prefix every U+003B SEMICOLON character (;) in *value* with a U+005C REVERSE SOLIDUS character (\).
8. Replace every U+000D CARRIAGE RETURN U+000A LINE FEED character pair (CRLF) in *value* with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).
9. Replace every remaining U+000D CARRIAGE RETURN (CR) or U+000A LINE FEED (LF) character in *value* with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).

3. Add a vCard line with the type *name*, the parameters *parameters*, and the value *value* to *output*.

12. If either *sex* or *gender-identity* has a value that is not the empty string, add a vCard line with the type "`GENDER`" and the value consisting of the concatenation of *sex*, a U+003B SEMICOLON character (;), and *gender-identity* to *output*.
13. Add a vCard line with the type "`END`" and the value "`VCARD`" to *output*.

When the above algorithm says that the user agent is to **add a vCard line** consisting of a type *type*, optionally some parameters, and a value *value* to a string *output*, it must run the following steps:

1. Let *line* be an empty string.
2. Append *type*, converted to ASCII uppercase, to *line*.
3. If there are any parameters, then for each parameter, in the order that they were added, run these substeps:
    1. Append a U+003B SEMICOLON character (;) to *line*.
    2. Append the parameter's name to *line*.
    3. Append a U+003D EQUALS SIGN character (=) to *line*.
    4. Append the parameter's value to *line*.
4. Append a U+003A COLON character (:) to *line*.
5. Append *value* to *line*.
6. Let *maximum length* be 75.
7. While *line*'s code point length is greater than *maximum length*:
    1. Append the first *maximum length* code points of *line* to *output*.
    2. Remove the first *maximum length* code points from *line*.
    3. Append a U+000D CARRIAGE RETURN character (CR) to *output*.
    4. Append a U+000A LINE FEED character (LF) to *output*.

5. Append a U+0020 SPACE character to *output*.

6. Let *maximum length* be 74.

8. Append (what remains of) *line* to *output*.

9. Append a U+000D CARRIAGE RETURN character (CR) to *output*.

10. Append a U+000A LINE FEED character (LF) to *output*.

When the steps above require the user agent to obtain the result of **collecting vCard subproperties** named *subname* in *subitem*, the user agent must run the following steps:

1. Let *value* be the empty string.

2. For each property named *subname* in the item *subitem*, run the following substeps:

    1. If the value of the property is itself an item, then skip this property.

    2. If this is not the first property named *subname* in *subitem* (ignoring any that were skipped by the previous step), then append a U+002C COMMA character (,) to *value*.

    3. Append the result of escaping the vCard text string given by the value of the property to *value*.

3. Return *value*.

When the steps above require the user agent to obtain the result of **collecting the first vCard subproperty** named *subname* in *subitem*, the user agent must run the following steps:

1. If there are no properties named *subname* in *subitem*, then return the empty string.

2. If the value of the first property named *subname* in *subitem* is an item, then return the empty string.

3. Return the result of escaping the vCard text string given by the value of the first property named *subname* in *subitem*.

When the above algorithms say the user agent is to **escape the vCard text string** *value*, the user agent must use the following steps:

1. Prefix every U+005C REVERSE SOLIDUS character (\) in *value* with another U+005C REVERSE SOLIDUS character (\).

2. Prefix every U+002C COMMA character (,) in *value* with a U+005C REVERSE SOLIDUS character (\).

3. Prefix every U+003B SEMICOLON character (;) in *value* with a U+005C REVERSE SOLIDUS character (\).

4. Replace every U+000D CARRIAGE RETURN U+000A LINE FEED character pair (CRLF) in *value* with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).

5. Replace every remaining U+000D CARRIAGE RETURN (CR) or U+000A LINE FEED (LF) character in *value* with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).

6. Return the mutated *value*.

Note

This algorithm can generate invalid vCard output, if the input does not conform to the rules described for the `http://microformats.org/profile/hcard` *item type and defined property names.*

5.3.1.2 Examples §

This section is non-normative.

Example

Here is a long example vCard for a fictional character called "Jack Bauer":

```
<section id="jack" itemscope itemtype="http://microformats.org/profile/hcard">
 <h1 itemprop="fn">
  <span itemprop="n" itemscope>
   <span itemprop="given-name">Jack</span>
   <span itemprop="family-name">Bauer</span>
  </span>
 </h1>
 <img itemprop="photo" alt="" src="jack-bauer.jpg">
 <p itemprop="org" itemscope>
  <span itemprop="organization-name">Counter-Terrorist Unit</span>
  (<span itemprop="organization-unit">Los Angeles Division</span>)
 </p>
```

File an issue about the selected text

```
  <span itemprop="adr" itemscope>
   <span itemprop="street-address">10201 W. Pico Blvd.</span><br>
   <span itemprop="locality">Los Angeles</span>,
   <span itemprop="region">CA</span>
   <span itemprop="postal-code">90064</span><br>
   <span itemprop="country-name">United States</span><br>
  </span>
  <span itemprop="geo">34.052339;-118.410623</span>
 </p>
 <h2>Assorted Contact Methods</h2>
 <ul>
  <li itemprop="tel" itemscope>
   <span itemprop="value">+1 (310) 597 3781</span> <span itemprop="type">work</span>
   <meta itemprop="type" content="voice">
  </li>
  <li><a itemprop="url" href="https://en.wikipedia.org/wiki/Jack_Bauer">I'm on Wikipedia</a>
  so you can leave a message on my user talk page.</li>
  <li><a itemprop="url" href="http://www.jackbauerfacts.com/">Jack Bauer Facts</a></li>
  <li itemprop="email"><a href="mailto:j.bauer@la.ctu.gov.invalid">j.bauer@la.ctu.gov.invalid</a></li>
  <li itemprop="tel" itemscope>
   <span itemprop="value">+1 (310) 555 3781</span> <span>
   <meta itemprop="type" content="cell">mobile phone</span>
  </li>
 </ul>
 <ins datetime="2008-07-20 21:00:00+01:00">
  <meta itemprop="rev" content="2008-07-20 21:00:00+01:00">
  <p itemprop="tel" itemscope><strong>Update!</strong>
  My new <span itemprop="type">home</span> phone number is
  <span itemprop="value">01632 960 123</span>.</p>
 </ins>
</section>
```

The odd line wrapping is needed because newlines are meaningful in microdata: newlines would be preserved in a conversion to, for example, the vCard format.

Example

This example shows a site's contact details (using the address element) containing an address with two street components:

```
<address itemscope itemtype="http://microformats.org/profile/hcard">
 <strong itemprop="fn"><span itemprop="n" itemscope><span itemprop="given-name">Alfred</span>
 <span itemprop="family-name">Person</span></span></strong> <br>
 <span itemprop="adr" itemscope>
  <span itemprop="street-address">1600 Amphitheatre Parkway</span> <br>
  <span itemprop="street-address">Building 43, Second Floor</span> <br>
  <span itemprop="locality">Mountain View</span>,
   <span itemprop="region">CA</span> <span itemprop="postal-code">94043</span>
 </span>
</address>
```

Example

The vCard vocabulary can be used to just mark up people's names:

```
<span itemscope itemtype="http://microformats.org/profile/hcard"
><span itemprop=fn><span itemprop="n" itemscope><span itemprop="given-name"
>George</span> <span itemprop="family-name">Washington</span></span
></span></span>
```

This creates a single item with a two name-value pairs, one with the name "fn" and the value "George Washington", and the other with the name "n" and a second item as its value, the second item having the two name-value pairs "given-name" and "family-name" with the values "George" and "Washington" respectively. This is defined to map to the following vCard:

```
BEGIN:VCARD
PROFILE:VCARD
VERSION:4.0
SOURCE:document's address
FN:George Washington
N:Washington;George;;;
END:VCARD
```

5.3.2 vEvent §

An item with the item type `http://microformats.org/profile/hcalendar#vevent` represents an event.

This vocabulary does not support global identifiers for items.

The following are the type's defined property names. They are based on the vocabulary defined in *Internet Calendaring and Scheduling Core Object Specification* (*iCalendar*), where more information on how to interpret the values can be found. [RFC5545]

Note

Only the parts of the iCalendar vocabulary relating to events are used here; this vocabulary cannot express a complete iCalendar instance.

`attach`

Gives the address of an associated document for the event.

The value must be an absolute URL.

Any number of properties with the name `attach` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`categories`

Gives the name of a category or tag that the event could be classified as.

The value must be text.

Any number of properties with the name `categories` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`class`

Gives the access classification of the information regarding the event.

The value must be text with one of the following values:

- `public`
- `private`
- `confidential`

⚠Warning!

This is merely advisory and cannot be considered a confidentiality measure.

A single property with the name `class` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`comment`

Gives a comment regarding the event.

The value must be text.

Any number of properties with the name `comment` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`description`

Gives a detailed description of the event.

The value must be text.

A single property with the name `description` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`geo`

Gives the geographical position of the event.

The value must be text and must match the following syntax:

1. Optionally, either a U+002B PLUS SIGN character (+) or a U+002D HYPHEN-MINUS character (-).
2. One or more ASCII digits.
3. Optionally*, a U+002E FULL STOP character (.) followed by one or more ASCII digits.
4. A U+003B SEMICOLON character (;).
5. Optionally, either a U+002B PLUS SIGN character (+) or a U+002D HYPHEN-MINUS character (-).
6. One or more ASCII digits.
7. Optionally*, a U+002E FULL STOP character (.) followed by one or more ASCII digits.

The optional components marked with an asterisk (*) should be included, and should have six digits each.

Note

The value specifies latitude and longitude, in that order (i.e., "LAT LON" ordering), in decimal degrees. The longitude represents the location east and west of the prime meridian as a positive or negative real number, respectively. The latitude represents the location north and south of the equator as a positive or negative real number, respectively.

A single property with the name `geo` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`location`

Gives the location of the event.

The value must be text.

A single property with the name `location` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`resources`

Gives a resource that will be needed for the event.

The value must be text.

Any number of properties with the name `resources` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`status`

Gives the confirmation status of the event.

The value must be text with one of the following values:

- `tentative`
- `confirmed`
- `cancelled`

A single property with the name `status` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`summary`

Gives a short summary of the event.

The value must be text.

User agents should replace U+000A LINE FEED (LF) characters in the value by U+0020 SPACE characters when using the value.

A single property with the name `summary` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`dtend`

Gives the date and time by which the event ends.

If the property with the name `dtend` is present within an item with the type `http://microformats.org/profile/hcalendar#vevent` that has a property with the name `dtstart` whose value is a valid date string, then the value of the property with the name `dtend` must be text that is a valid date string also. Otherwise, the value of the property must be text that is a valid global date and time string.

In either case, the value be later in time than the value of the `dtstart` property of the same item.

Note

The time given by the `dtend` property is not inclusive. For day-long events, therefore, the `dtend` property's value will be the day after *the end of the event.*

A single property with the name `dtend` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`, so long as that `http://microformats.org/profile/hcalendar#vevent` does not have a property with the name `duration`.

`dtstart`

Gives the date and time at which the event starts.

The value must be text that is either a valid date string or a valid global date and time string.

Exactly one property with the name `dtstart` must be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`duration`

Gives the duration of the event.

The value must be text that is a valid vevent duration string.

The duration represented is the sum of all the durations represented by integers in the value.

A single property with the name `duration` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`, so long as that `http://microformats.org/profile/hcalendar#vevent` does not have a property with the name `dtend`.

Gives whether the event is to be considered as consuming time on a calendar, for the purpose of free-busy time searches.

The value must be text with one of the following values:

- `opaque`
- `transparent`

A single property with the name `transp` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`contact`

Gives the contact information for the event.

The value must be text.

Any number of properties with the name `contact` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`url`

Gives a URL for the event.

The value must be an absolute URL.

A single property with the name `url` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`uid`

Gives a globally unique identifier corresponding to the event.

The value must be text.

A single property with the name `uid` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`exdate`

Gives a date and time at which the event does not occur despite the recurrence rules.

The value must be text that is either a valid date string or a valid global date and time string.

Any number of properties with the name `exdate` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`rdate`

Gives a date and time at which the event recurs.

The value must be text that is one of the following:

- A valid date string.
- A valid global date and time string.
- A valid global date and time string followed by a U+002F SOLIDUS character (/) followed by a second valid global date and time string representing a later time.
- A valid global date and time string followed by a U+002F SOLIDUS character (/) followed by a valid vevent duration string.

Any number of properties with the name `rdate` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`rrule`

Gives a rule for finding dates and times at which the event occurs.

The value must be text that matches the RECUR value type defined in *iCalendar*. [RFC5545]

A single property with the name `rrule` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`created`

Gives the date and time at which the event information was first created in a calendaring system.

The value must be text that is a valid global date and time string.

A single property with the name `created` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`last-modified`

Gives the date and time at which the event information was last modified in a calendaring system.

The value must be text that is a valid global date and time string.

A single property with the name `last-modified` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

`sequence`

ıe event information.

The value must be text that is a valid non-negative integer.

A single property with the name `sequence` may be present within each item with the type `http://microformats.org/profile/hcalendar#vevent`.

A string is a **valid vevent duration string** if it matches the following pattern:

1. A U+0050 LATIN CAPITAL LETTER P character (P).
2. One of the following:
   - A valid non-negative integer followed by a U+0057 LATIN CAPITAL LETTER W character (W). The integer represents a duration of that number of weeks.
   - At least one, and possible both in this order, of the following:
     1. A valid non-negative integer followed by a U+0044 LATIN CAPITAL LETTER D character (D). The integer represents a duration of that number of days.
     2. A U+0054 LATIN CAPITAL LETTER T character (T) followed by any one of the following, or the first and second of the following in that order, or the second and third of the following in that order, or all three of the following in this order:
        1. A valid non-negative integer followed by a U+0048 LATIN CAPITAL LETTER H character (H). The integer represents a duration of that number of hours.
        2. A valid non-negative integer followed by a U+004D LATIN CAPITAL LETTER M character (M). The integer represents a duration of that number of minutes.
        3. A valid non-negative integer followed by a U+0053 LATIN CAPITAL LETTER S character (S). The integer represents a duration of that number of seconds.

5.3.2.1 Conversion to iCalendar §

Given a list of nodes *nodes* in a `Document`, a user agent must run the following algorithm to **extract any vEvent data represented by those nodes**:

1. If none of the nodes in *nodes* are items with the type `http://microformats.org/profile/hcalendar#vevent`, then there is no vEvent data. Abort the algorithm, returning nothing.
2. Let *output* be an empty string.
3. Add an iCalendar line with the type "`BEGIN`" and the value "`VCALENDAR`" to *output*.
4. Add an iCalendar line with the type "`PRODID`" and the value equal to a user-agent-specific string representing the user agent to *output*.
5. Add an iCalendar line with the type "`VERSION`" and the value "`2.0`" to *output*.
6. For each node *node* in *nodes* that is an item with the type `http://microformats.org/profile/hcalendar#vevent`, run the following steps:
   1. Add an iCalendar line with the type "`BEGIN`" and the value "`VEVENT`" to *output*.
   2. Add an iCalendar line with the type "`DTSTAMP`" and a value consisting of an iCalendar DATE-TIME string representing the current date and time, with the annotation "`VALUE=DATE-TIME`", to *output*. [RFC5545]
   3. For each element *element* that is a property of the item *node*: for each name *name* in *element*'s property names, run the appropriate set of substeps from the following list:

      If the property's value is an item

      Skip the property.

      If the property is `dtend`
      If the property is `dtstart`
      If the property is `exdate`
      If the property is `rdate`
      If the property is `created`
      If the property is `last-modified`

      Let *value* be the result of stripping all U+002D HYPHEN-MINUS (-) and U+003A COLON (:) characters from the property's value.

      If the property's value is a valid date string then add an iCalendar line with the type *name* and the value *value* to *output*, with the annotation "`VALUE=DATE`".

      Otherwise, if the property's value is a valid global date and time string then add an iCalendar line with the type *name* and the value *value* to *output*, with the annotation "`VALUE=DATE-TIME`".

      Otherwise skip the property.

      Otherwise

      Add an iCalendar line with the type *name* and the property's value to *output*.

   4. Add an iCalendar line with the type "`END`" and the value "`VEVENT`" to *output*.
7. Add an iCalendar line with the type "`END`" and the value "`VCALENDAR`" to *output*.

When the above algorithm says that the user agent is to **add an iCalendar line** consisting of a type *type*, a value *value*, and optionally an annotation, to a string *output*, it must run the following steps:

1. Let *line* be an empty string.
2. Append *type*, converted to ASCII uppercase, to *line*.
3. If there is an annotation:
    1. Append a U+003B SEMICOLON character (;) to *line*.
    2. Append the annotation to *line*.
4. Append a U+003A COLON character (:) to *line*.
5. Prefix every U+005C REVERSE SOLIDUS character (\) in *value* with another U+005C REVERSE SOLIDUS character (\).
6. Prefix every U+002C COMMA character (,) in *value* with a U+005C REVERSE SOLIDUS character (\).
7. Prefix every U+003B SEMICOLON character (;) in *value* with a U+005C REVERSE SOLIDUS character (\).
8. Replace every U+000D CARRIAGE RETURN U+000A LINE FEED character pair (CRLF) in *value* with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).
9. Replace every remaining U+000D CARRIAGE RETURN (CR) or U+000A LINE FEED (LF) character in *value* with a U+005C REVERSE SOLIDUS character (\) followed by a U+006E LATIN SMALL LETTER N character (n).
10. Append *value* to *line*.
11. Let *maximum length* be 75.
12. While *line*'s code point length is greater than *maximum length*:
    1. Append the first *maximum length* code points of *line* to *output*.
    2. Remove the first *maximum length* code points from *line*.
    3. Append a U+000D CARRIAGE RETURN character (CR) to *output*.
    4. Append a U+000A LINE FEED character (LF) to *output*.
    5. Append a U+0020 SPACE character to *output*.
    6. Let *maximum length* be 74.
13. Append (what remains of) *line* to *output*.
14. Append a U+000D CARRIAGE RETURN character (CR) to *output*.
15. Append a U+000A LINE FEED character (LF) to *output*.

Note

This algorithm can generate invalid iCalendar output, if the input does not conform to the rules described for the `http://microformats.org/profile/hcalendar#vevent` *item type and defined property names.*

5.3.2.2 Examples §

This section is non-normative.

Example

Here is an example of a page that uses the vEvent vocabulary to mark up an event:

```
<body itemscope itemtype="http://microformats.org/profile/hcalendar#vevent">
 ...
 <h1 itemprop="summary">Bluesday Tuesday: Money Road</h1>
 ...
 <time itemprop="dtstart" datetime="2009-05-05T19:00:00Z">May 5th @ 7pm</time>
 (until <time itemprop="dtend" datetime="2009-05-05T21:00:00Z">9pm</time>)
 ...
 <a href="http://livebrum.co.uk/2009/05/05/bluesday-tuesday-money-road"
    rel="bookmark" itemprop="url">Link to this page</a>
 ...
 <p>Location: <span itemprop="location">The RoadHouse</span></p>
 ...
 <p><input type=button value="Add to Calendar"
```



```
 "location = getCalendar(this)"></p>
```

```
 ...
 <meta itemprop="description" content="via livebrum.co.uk">
</body>
```

The `getCalendar()` function is left as an exercise for the reader.

The same page could offer some markup, such as the following, for copy-and-pasting into blogs:

```
<div itemscope itemtype="http://microformats.org/profile/hcalendar#vevent">
 <p>I'm going to
 <strong itemprop="summary">Bluesday Tuesday: Money Road</strong>,
 <time itemprop="dtstart" datetime="2009-05-05T19:00:00Z">May 5th at 7pm</time>
 to <time itemprop="dtend" datetime="2009-05-05T21:00:00Z">9pm</time>,
 at <span itemprop="location">The RoadHouse</span>!</p>
 <p><a href="http://livebrum.co.uk/2009/05/05/bluesday-tuesday-money-road"
       itemprop="url">See this event on livebrum.co.uk</a>.</p>
 <meta itemprop="description" content="via livebrum.co.uk">
</div>
```

5.3.3 Licensing works §

An item with the item type **`http://n.whatwg.org/work`** represents a work (e.g. an article, an image, a video, a song, etc.). This type is primarily intended to allow authors to include licensing information for works.

The following are the type's defined property names.

`work`

Identifies the work being described.

The value must be an absolute URL.

Exactly one property with the name `work` must be present within each item with the type `http://n.whatwg.org/work`.

`title`

Gives the name of the work.

A single property with the name `title` may be present within each item with the type `http://n.whatwg.org/work`.

`author`

Gives the name or contact information of one of the authors or creators of the work.

The value must be either an item with the type `http://microformats.org/profile/hcard`, or text.

Any number of properties with the name `author` may be present within each item with the type `http://n.whatwg.org/work`.

`license`

Identifies one of the licenses under which the work is available.

The value must be an absolute URL.

Any number of properties with the name `license` may be present within each item with the type `http://n.whatwg.org/work`.

5.3.3.1 Examples §

This section is non-normative.

Example

This example shows an embedded image entitled *My Pond*, licensed under the Creative Commons Attribution-Share Alike 4.0 International License and the MIT license simultaneously.

```
<figure itemscope itemtype="http://n.whatwg.org/work">
 <img itemprop="work" src="mypond.jpeg">
 <figcaption>
  <p><cite itemprop="title">My Pond</cite></p>
  <p><small>Licensed under the <a itemprop="license"
  href="https://creativecommons.org/licenses/by-sa/4.0/">Creative
```

File an issue about the selected text

```
on-Share Alike 4.0 International License</a>
```

```
  and the <a itemprop="license"
  href="http://www.opensource.org/licenses/mit-license.php">MIT
  license</a>.</small>
 </figcaption>
</figure>
```

5.4 Converting HTML to other formats §

5.4.1 JSON §

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

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

Note

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

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

1. Let *result* be an empty object.
2. If no *memory* was passed to the algorithm, let *memory* be an empty list.
3. Add *item* to *memory*.
4. If the *item* has any item types, add an entry to *result* called "`type`" whose value is an array listing the item types of *item*, in the order they were specified on the `itemtype` attribute.
5. If the *item* has a global identifier, add an entry to *result* called "`id`" whose value is the global identifier of *item*.
6. Let *properties* be an empty object.
7. For each element *element* that has one or more property names and is one of the properties of the item *item*, in the order those elements are given by the algorithm that returns the properties of an item, run the following substeps:
    1. Let *value* be the property value of *element*.
    2. If *value* is an item, then: If *value* is in *memory*, then let *value* be the string "`ERROR`". Otherwise, get the object for *value*, passing a copy of *memory*, and then replace *value* with the object returned from those steps.
    3. For each name *name* in *element*'s property names, run the following substeps:
        1. If there is no entry named *name* in *properties*, then add an entry named *name* to *properties* whose value is an empty array.
        2. Append *value* to the entry named *name* in *properties*.
8. Add an entry to *result* called "`properties`" whose value is the object *properties*.
9. Return *result*.

Example

For example, take this markup:

```
<!DOCTYPE HTML>
<html lang="en">
<title>My Blog</title>
<article itemscope itemtype="http://schema.org/BlogPosting">
 <header>
  <h1 itemprop="headline">Progress report</h1>
  <p><time itemprop="datePublished" datetime="2013-08-29">today</time></p>
  <link itemprop="url" href="?comments=0">
```

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

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

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