CHAPTER 18

Using Microformats and RDFa As Embeddable Data Formats

The central problem that we will study in this chapter is how to embed information in web pages in a way that is easy to understand by both humans and computer programs. The solution that we will consider in depth is *microformats*, little chunks of structured data that are seamlessly embedded in web pages. (X)HTML is designed primarily to produce a human user interface (via a web browser). However, by carefully following certain conventions (ones that constitute microformats), you can produce (X)HTML out of which data can be unambiguously extracted. The consequence is that it is relatively easy to write computer programs to parse microformats so that the data can be reused in other contexts—giving rise to plenty of mashup possibilities. Moreover, having data embedded right in the context of a user interface is helpful. A user can decide what to do with this data (how to "operate" on a given piece of data) while in the context of normal browsing.

Now, the previous paragraph is a bit abstract. What I will do in this chapter is walk you through some concrete examples describing microformats in general. We will use Operator, a Firefox extension, to help parse and view microformats and to create scripts that enable users to take specific actions in response to microformats. Specifically, we will do the following:

* Study specific examples of microformats
* Look at how to use the Firefox add-on Operator to jump-start your study of microformats
* Look at programmatic approaches to consuming and creating microformats
* Compare microformats to leading alternatives such as RDFa as a way to embed data in human-readable contexts.

Using Operator to Learn About Microformats

Installing the Operator add-on in Firefox and seeing it in action is a good way to learn about microformats. You can download it from here:

`https://addons.mozilla.org/en-US/firefox/addon/4106`

As of writing, the latest stable version is 0.8, the one I will use and describe in this chapter. (Note that there is also a 0.9 beta version.[1])

When using Operator, you should have on hand the closest thing to official documentation for the extension:

`http://www.kaply.com/weblog/operator/`

Now let's see what Operator can do for you once it is installed. Let's look at Operator in action by loading a page from Upcoming.yahoo.com (the event aggregation site you learned about in Chapter 15):

`http://upcoming.yahoo.com/event/144855`

Figure 18-1 shows what happens in the Operator toolbar. I've chosen this page to show you an example of microformats in use "in the wild." (Later in this chapter, I'll show you an example HTML page I created to show examples of microformats.)

Insert 858Xf1801.tif

Figure 18-1. Operator toolbar showing microformats embedded in a page from Upcoming.yahoo.com. Actions available for the location microformat are shown boxed. (Reproduced with permission of Yahoo! Inc. ® 2007 by Yahoo! Inc. YAHOO! and the YAHOO! logo are trademarks of Yahoo! Inc.)

1. See the blog entry announcing 0.9b (http://www.kaply.com/weblog/2007/12/03/operator-09-beta-available/). You can download the latest develop version of Operator from http://www.kaply.com/operator/operator.xpi.

You will notice in the Operator toolbar a list of data formats recognized by Operator, along with the number of instances of each format. By default, these formats (listed by their descriptive and formal names) are as follows:

* Addresses (`adr`)
* Contacts (`hCard`)
* Events (`hCalendar`)
* Location (`geo`)
* Tagspaces (`tag` or `rel-tag`)[2]
* Bookmarks (`xFolk`)
* Resources (`RDF`)

2. The format called tag in Operator is known as rel-tag on http://microformats.org.

By default, the Operator toolbar uses the descriptive names. You can instead display the formal names of the data formats in the Operator toolbar (by doing to the General tab and unchecking Use Descriptive Names under Data Formats). Toggling that option allows you to correlate the formal and descriptive names of the data formats.

I will cover the individual data formats in detail later in the chapter. Continuing with the example from Upcoming.yahoo.com, note that Operator indicates the presence of instances for the following formats: `adr`, `hCard`, `hCalendar`, `geo`, and `tag`. What do these microformats have to do with the event in question (Mashup Camp IV)? The UI for Upcoming.yahoo.com gives you many options to package information about the event:

* You can send it to a number of calendars.
* You can download the event information in the iCalendar format.

* You can use the Upcoming.yahoo.com API (as explained in Chapter 15) to extract the event information from Upcoming.yahoo.com.

For instance, examine the event in iCalendar format, which you can access from http://upcoming.yahoo.com/calendar/v2/event/144855:

```
BEGIN:VCALENDAR
VERSION:2.0
X-WR-CALNAME:Upcoming Event: Mashup Camp IV
PRODID:-//Upcoming.org/Upcoming ICS//EN
CALSCALE:GREGORIAN
METHOD:PUBLISH
BEGIN:VEVENT
DTSTART:20070718T130000
DTEND:20070718T140000
RRULE:FREQ=DAILY;INTERVAL=1;UNTIL=20070720T000000
TRANSP:TRANSPARENT
SUMMARY:Mashup Camp IV
DESCRIPTION: [Full details at http://upcoming.yahoo.com/event/144855/ ] From
Mass
Events Labs\, the organizers of the wildly successfully Mashup Camp
unconferences\,
comes Mashup Camp IV.  Back on the West Coast (the Computer History Museum in
Mountain View\, CA)\, with the same great people\,  great conversations and
discussions.  Same fun\, hacking\, and networking in an Open Space format.
      Have a mashup you'd like to show off?  Enter it in the Best Mashup contest
and
see if you can survive the grueling SpeedGeeking session.  Event submitted by
Eventful.com on behalf of chris_radcliff .
URL;VALUE=URI:http://upcoming.yahoo.com/event/144855/
UID:http://upcoming.yahoo.com/event/144855/
DTSTAMP:20070125T124529
LAST-UPDATED:20070125T124529
CATEGORIES:Other
ORGANIZER;CN=chris_radcliff:X-ADDR:http://upcoming.yahoo.com/user/19139/
LOCATION;VENUE-UID="http://upcoming.yahoo.com/venue/259/":Computer History
Museum @
1401 N Shoreline Blvd.\, Mountain View\, California 94043 US
END:VEVENT
BEGIN:VVENUE
X-VVENUE-INFO:http://evdb.com/docs/ical-venue/draft-norris-ical-venue.html
NAME:Computer History Museum
ADDRESS:1401 N Shoreline Blvd.
CITY:Mountain View
REGION;X-ABBREV=ca:California
COUNTRY;X-ABBREV=us:United States
POSTALCODE:94043
GEO:37.4149;-122.078
URL;X-LABEL=Venue Info:http://www.computerhistory.org/
END:VVENUE
END:VCALENDAR
```

As the Operator toolbar indicates, the event information is also embedded in the (X)HTML source at the following location as a series of microformats:

```
http://upcoming.yahoo.com/event/144855
```

Let’s take a look at each of these example microformats in turn. I’ll give a more formal discussion of each one in the following sections.

Tip You can use Operator to help in this exercise by checking the Debug Mode option (on the General tab) in Operator so that you have access to the Debug action for each microformat instance. The Debug action lists the (X)HTML source fragment containing the microformat instance.

---

adr (Addresses)

From the web page, you can read the address for the event: 1401 N Shoreline Blvd., Mountain View, California, 94043. Operator picks out the address as an instance of the `adr` data format, with the corresponding (X)HTML source fragment:

```
<div class="address adr">
  <span class="street-address">1401 N Shoreline Blvd.</span><br />
  <span class="locality">Mountain View</span>,
  <span class="region">California</span> <span class="postal-code">94043</span>
</div>
```

Note the use of the `<div>` tag to wrap the `address` and `class` attributes to separate and name the parts of the address. This (X)HTML fragment meets two goals simultaneously: it displays an address naturally and appropriately for a human reader of the web page, and it uses (X)HTML elements and attributes to enable programs (such as Operator) to reliably parse an address from the (X)HTML. You will see this design goal of satisfying human and computer readers repeated among all the microformats.

With the `adr` microformat parsed out, you as a user can then apply an *action* to the address. Operator has by default two actions (in addition to Debug) that you can apply to an address: Find with Google Maps and Find with Yahoo! Maps. Selecting the first action, for instance, loads the following into the browser:

```
http://maps.google.com/maps?q=1401%20N%20Shoreline%20Blvd.,%20California,%20Moun
tain~CCC
%20View,%2094043
```

This action, in effect, enables Operator to perform a mashup of Upcoming.yahoo.com and Google Maps—and more generally, any web site that has `adr` microformat data with Google Maps. Note also how Operator enables the user to invoke this action in the context of web browsing. Firefox with Operator joins a web site with an `adr` microformat to Google Maps—and not a third-party web application.

Operator allows you to add other actions. Later in the chapter, I will show you how to add other user scripts to Operator and to write a basic user script to geocode addresses.

hCard (Contacts)

The `hCard` data format is meant to represent a person or organization, specifically contact information for the entity. The (X)HTML source for the embedded `hCard` microformat is as follows:

```
<div class="venue location vcard">
  <span class="fn org">
    <a href="/venue/259/">Computer History Museum</a>
  </span>
  <br />
  <div class="address adr">
    <span class="street-address">1401 N Shoreline Blvd.</span><br />
    <span class="locality">Mountain View</span>,
    <span class="region">California</span>
    <span class="postal-code">94043</span>
  </div>
  <span class="geo" style="display: none">
    <span class="latitude">37.4149</span>,
    <span class="longitude">-122.078</span>
  </span>
</div>
```

You might be wondering why you will see `vcard` (instead of `hcard`) as a class attribute. The reason is that `hCard` is derived from the vCard standard. You can compare the vCard data that Operator creates for this page to the (X)HTML source to see the similarities:

```
BEGIN:VCARD
PRODID:-//kaply.com//Operator 0.8//EN
SOURCE:http://upcoming.yahoo.com/event/144855
NAME:Mashup Camp IV at Computer History Museum (Wednesday, July 18, 2007) -
Upcoming
VERSION:3.0
N:;;;;
ORG;CHARSET=UTF-8:Computer History Museum
FN;CHARSET=UTF-8:Computer History Museum
UID:
ADR;CHARSET=UTF-8:;;1401 N Shoreline Blvd.;Mountain View;California;94043;
GEO:37.4149;-122.078
END:VCARD
```

Among the default actions in Operator for `hCard` is Add to Yahoo! Contacts, which, when invoked for this page, loads the following URL into the browser:

```
http://address.yahoo.com/?fn=Computer%20History%20Museum&co=Computer%20History%2
0Mus~CCC
eum&ha1=1401%20N%20Shoreline%20Blvd.&hc=Mountain%20View&hs=California&hz=94043&A
=C
```

hCalendar (Events)

The `hCalendar` microformat represents events and is roughly speaking the iCalendar format transformed into a microformat. (See Chapter 15 for a discussion of iCalendar.) The (X)HTML source for the `hCalendar` microformat is a large fragment that I will not quote here. To find it, you can use Operator or look at the source and find a `<div>` element that begins with this:

```
<div id="calendarContainer" class="vcalendar"> <!-- Begin vCalendar -->
```

and ends lines later with this:

```
</div> <!-- End vCalendar -->
```

The pieces of (X)HTML in between contain event data, such as this:

```
<abbr class="dtstart" title="20070718T130000">Wednesday, July 18, 2007
</abbr>
```

and the following:

```
<abbr class="dtend" title="20070719T140000">
```

As in the case of `hCard`, you might wonder why the `hCalendar` format would use `class="vcalendar"` and not `class="hcalendar"`. vCalendar was the precursor to iCalendar, a fact that is reflected in the iCalendar standard (which if you look at the iCalendar for the Upcoming.yahoo.com event listed earlier), you have the following structure:

```
BEGIN:VCALENDAR
[...]
DTSTART:20070718T130000
DTEND:20070718T140000
[...]
END:VCALENDAR
```

Among the default actions associated with `hCalendar` are ones to send the event data to Google Calendar, Yahoo! Calendar, and 30boxes.com. Compare how you moved event data with APIs in Chapter 15 with this approach of extracting microformat data and sending that data to other services via an HTTP `GET` request.

geo (Locations)

The `geo` data format represents a geospatial location, specifically a latitude and longitude. The (X)HTML source for the `geo` instance is as follows:

```
<span class="geo" style="display: none">
  <span class="latitude">37.4149</span>,
  <span class="longitude">-122.078</span>
</span>
```

With Operator, you can map this location to Google Maps and Yahoo! Maps, or you can export it as KML.

tag (Tagspaces)

Upcoming.yahoo.com supports the tagging of individual events. For instance, among the tags for the example event is `mashup`. You can find this tag marked up using the `tag` microformat in the (X)HTML source. For example:

```
<a href="/tag/mashup/" rel="tag" class="category">mashup</a>
```

You'll see from the following discussion that the combination of `rel=tag` in an `<a>` element is indicative of a tag microformat and that the last path component of the URL (that is, `mashup`) is the text of the tag. By default, there are actions in Operator to look this tag up in such web sites as del.icio.us, Flickr, Upcoming.yahoo.com, and YouTube.

Definitions and Design Goals of Microformats

This is the current official definition of microformats from `http://microformats.org/`:

> *Designed for humans first and machines second, microformats are a set of simple, open data formats built upon existing and widely adopted standards.*

Many other definitions of microformats have been proposed, some of which I find more illuminating. This is the definition on the Microformats.org site:[3]

> *Microformats can be defined as: simple conventions for embedding semantic markup for a specific problem domain in human-readable (X)HTML/XML documents, Atom/RSS feeds, and "plain" XML that normalize existing content usage patterns using brief, descriptive class names often based on existing interoperable standards to enable decentralized development of resources, tools, and services.*

Out of this definition, I would emphasize the following points:

* Human-readable (X)HTML/XML documents, Atom/RSS feeds, and "plain" XML
* Using brief, descriptive class names

It's important to know about the intentional limitations built into microformats before criticizing them. From the list of things that microformats are *not* at `http://microformats.org/about/`, I will highlight a few. Microformats are not the following:

* Infinitely extensible and open-ended
* A panacea for all taxonomies, ontologies, and other such abstractions

Keep those limitations in mind when we compare microformats to specifications such as RDFa, which are aimed to be highly generalizable.

Because microformats involve both design philosophies and concrete formats (which you can use whether or not you subscribe to the microformats design philosophy), it's easy to just ignore the philosophy. So if you want to just use the work of the microformats community, is this philosophy relevant? Probably—since there are a lot of design decisions that are hard to understand without knowing that philosophy.

Finally, with so many other ways to already get event data, what does having the microformat contribute to the mix? Well, there are at least two advantages:

* For user-centered contexts, programs that have the (X)HTML source already—for instance, a web browser such as Firefox and its associated machinery (browser extensions, Greasemonkey scripts, and so on)—can reliably extract the event data and then present options to the user about what to do with that data. (The Operator Firefox add-on instantiates this idea.)

* Spiders and other screen-scrapers can also extract the information reliably—without the usual heuristical guessing—to build aggregated databases of events across web pages. That's what sites such as Technorati are doing.

In both cases, there is no need to make an extra call to the API to get this desired information about the event.

3. http://microformats.org/wiki/what-are-microformats retrieved as http://microformats.org/wiki?title=what-are-microformats&direction=next&oldid=21422.

Microformats Design Patterns

Microformats are meant to be embeddable in (X)HTML—so anywhere you can put (X)HTML, you should be able to stick in microformats—including RSS and Atom feeds.

The best way to start with microformats is to look at some specific examples while keeping an eye out for some general design patterns that are behind microformats; see `http://microformats.org/wiki/design-patterns`. That is, what are some tried-and-true ways to insert data into HTML?

rel-design-pattern

`rel-design-pattern` uses values in a `rel` attribute to indicate the meaning of a link.[4] You already saw an example with the `tag` or `rel-tag` microformat:

```
<a href="/tag/mashup/" rel="tag" class="category">mashup</a>
```

In addition to `rel-tag`, you'll learn about `rel-license`, a data format that also uses `rel-design-pattern`.

The `rev` attribute is also used in this design pattern. In case you're not up on what `rev` is supposed to mean (don't feel bad if you don't—I didn't before reading about microformats), read the FAQ (`http://microformats.org/wiki/rev#Then_what_does_.22rev.22_mean.3F`):

> *'rev' is the precise opposite (or 'reverse') of the 'rel' attribute. E.g. a `rev="help"` link indicates that the current document is 'help' for the resource indicated by the href.*

MULTIPLE VALUES IN ATTRIBUTES

You can stick multiple values in an attribute by separating the individual values by whitespace. The following is how you associate an HTML element with multiple classes. This is also standard usage for other attributes such as `rel` and `rev`.

```
http://www.w3.org/TR/html401/struct/links.html#adef-rel
http://www.w3.org/TR/html401/struct/links.html#adef-rev
```

class-design-pattern

This design pattern[5] involves selecting an appropriate (X)HTML element and the `class` attribute to hold a whitespace-separated list of semantic class names (`http://microformats.org/wiki/semantic-class-names`). The `<span>` and `<div>` elements can be used in the absence of other appropriate (X)HTML elements to demarcate content.

4. http://microformats.org/wiki/rel-design-pattern
5. http://microformats.org/wiki/class-design-pattern

As you have already seen from the earlier examples, the `adr`, `hCard`, `geo`, and `hCalendar` formats use `class-design-pattern`. For instance:

```
<div class="address adr">
  <span class="street-address">1401 N Shoreline Blvd.</span><br />
  <span class="locality">Mountain View</span>,
  <span class="region">California</span> <span class="postal-code">94043</span>
</div>
```

and for example:

```
<span class="geo" style="display: none">
  <span class="latitude">37.4149</span>,
  <span class="longitude">-122.078</span>
</span>
```

When you come to selecting class names, you should note the ones that are already in use to avoid reinventing the wheel and causing namespace collisions: `http://microformats.org/wiki/existing-classes`.

abbr-design-pattern

This pattern[6] uses the `<abbr>` HTML tag to wrap text with a machine-readable version of that information, which is stored in the `title` attribute of the `<abbr>` element. This pattern is most commonly used to encode the date and time in `datetime-design-pattern` (`http://microformats.org/wiki/datetime-design-pattern`):

```
<abbr class="dtstart" title="20080310T1700-08">
  March 10, 2008 at 5 PM
</abbr>
```

There are some concerns about the accessibility of such constructions (`http://microformats.org/wiki/datetime-design-pattern#Accessibility_issues`). Here is a recommended alternative:

```
<span class="dtstart" title="20080310T1700-08">
  March 10, 2008 at 5 PM, Pacific Standard Time
</span>
```

include-pattern

This pattern[7] is for including data from one microformat into another microformat from the same page. A major reason for this pattern is to avoid redundancy. It's used in the proposed `hResume`, `hReview`, and `hAtom` microformats. For example, I can reuse the content contained in the following `hCard`:

6. http://microformats.org/wiki/abbr-design-pattern
7. http://microformats.org/wiki/include-pattern

```
<span class="vcard">
 <span class="fn n" id="ryee-hcard">
  <span class="given-name">Raymond</span> <span class="family-name">Yee</span>
 </span>
</span>
```

in a second `hCard`:

```
<span class="vcard">
 <a href="#ryee-hcard" class="include" title="Raymond Yee"></a>
 <span class="org">UC Berkeley</span>
 <span class="title">Alumnus</span>
</span>
```

Note the use of `id="ryee-hcard"` in the first `hCard` instance and `href="#ryee-hcard"` to tie the two `hCard` instances, as well as `class="include"` in the `<a>` element in the `hCard` instance that reuses data from the other instance.

Examples of Microformats

In this section, I will show more examples of a number of microformats. (You can find a list of microformats at `http://microformats.org/wiki/Main_Page`.) I have gathered the examples in this section into one page:

`http://examples.mashupguide.net/ch18/sample_microformats.html`

Try Operator on this URL to have it parse the various microformats. Some of the following data formats are not supported natively by Operator, but you might be able to find extra user scripts that add such support. (See "Installing User Scripts to Operator" for a how-to.)

INSTALLING USER SCRIPTS TO OPERATOR

To install user scripts in Operator, download a script to your local drive. Then go to the User Scripts tab in Operator Options. Next, hit the New button, and enter the path of the script on your drive. You can go to the Data formats tab to load any new formats (using the New button) and to the Actions tab to make actions visible on the Operator tab.

Make sure to restart Firefox for the scripts to take effect.

You can find more details at `http://www.kaply.com/weblog/operator/`.

A good place to find Operator user scripts is `http://www.kaply.com/weblog/operator-user-scripts/`.

rel-license

`rel-license` (`http://microformats.org/wiki/rel-license`) is for specifying a license to be associated with the embedding page. For instance, when you go to the Creative Commons site (`http://creativecommons.org/license`) to select a license, you will be given some HTML that uses the `rel-license` microformat to indicate a license. For instance, if you select the defaults, you will get something like this:[8]

```
<a rel="license" href="http://creativecommons.org/licenses/by/3.0/">
```

```
<img alt="Creative Commons License" style="border-width:0"
     src="http://i.creativecommons.org/l/by/3.0/88x31.png" />
</a>
<br />This work is licensed under a
<a rel="license" href="http://creativecommons.org/licenses/by/3.0/">
Creative Commons Attribution 3.0 License</a>.
```

As noted, rel-license uses rel-design-pattern (that is, it uses the rel attribute in an <a> element).

rel-tag

The rel-tag microformat (http://microformats.org/wiki/rel-tag) is used to relate a tag (as in a folksonomic tag; see Chapter 4) to a URL. You use it by implementing rel-design-pattern, specifically, by adding tag to the list of values in the rel attribute. The last path segment of the URL in the href attribute is then considered the value of the tag, and the URL value of href points to a collection of items having the same tag.

Consider the following example generated by WordPress for the Google Maps category for http://blog.mashupguide.net:

```
<a rel="tag" title="View all posts in Google Maps"
   href="http://blog.mashupguide.net/category/google-maps/">Google Maps</a>
```

In this case, google-maps is the tag (the last path segment of the URL in the href attribute), and http://blog.mashupguide.net/category/google-maps/ points to the blog entries that have been tagged with the tag google-maps.

xfn

xfn (http://www.gmpg.org/xfn/), which stands for XHTML Friends Network and also uses rel-design-pattern, is used to denote personal relationships between the author of a web page and the people associated with the linked page. The easiest way is to get started is to fill out the XFN Creator (http://gmpg.org/xfn/creator).

I'll now present two examples. The first is for my wife's weblog:

```
<a href="http://laurashefler.net/blog"
   rel="friend met colleague coresident spouse muse sweetheart">Laura
Shefler</a>
```

The second is for Tim Berners-Lee's web page. In this case, he is a contact since I don't know him personally:

```
<a href="http://www.w3.org/People/Berners-Lee/" rel="contact">Tim Berners-
Lee</a>
```

8. http://creativecommons.org/license/results-one?q_1=2&q_1=1&field_commercial=yes&field_derivatives=yes&field_jurisdiction=&field_format=&lang=en&language=en&n_questions=3

xFolk

xFolk (http://microformats.org/wiki/xfolk) is used to publish a bookmark (like the bookmarks you see in social bookmarking sites, covered in Chapter 14). That is, you can use xFolk to tie a URL to a description and tags. (I imagine that the name xFolk is meant to suggest folksonomies since you can use it to tag URLs.)

xFolk uses class-design-pattern (with class="xfolkentry"). In the following example, the URL of the bookmark is http://www.w3.org/People/Berners-Lee, the description is The inventor of the WWW, and the tag associated with the bookmark is WWW:

```
<div class="xfolkentry">
  <a class="taggedlink" href="http://www.w3.org/People/Berners-Lee/">Tim
Berners-Lee
  </a>: <span class="description">The inventor of the WWW</span>
  <a rel="tag" href="http://technorati.com/tag/WWW"></a>
</div>
```

Note the use of the xfolkentry, taggedlink, and description class names.

With Operator, you can try the "Bookmark with del.icio.us" action, which sends the bookmark to del.icio.us through the following URL:

```
https://secure.del.icio.us/login?url=http%3A%2F%2Fwww.w3.org%2FPeople%2FBerners-
Lee%~CCC
2F&title=Tim%20Berners-Lee&notes=The%20inventer%20of%20the%20WWW&v=4
```

geo

geo (http://microformats.org/wiki/geo) is used to denote the latitude and longitude of the resource tied to the embedding web page. Using http://microformats.org/wiki/geo-cheatsheet, you can figure out how to use class-design-pattern and use the geo, latitude, and longitude class names to write an example such as the following:

```
<div class="geo">Tim Berners-Lee's location is:
  <span class="latitude">42.3633690</span>,
  <span class="longitude">-71.091796</span>.
</div>
```

Given all the attention we paid to mapping geotagged photos in Flickr, I should mention that Flickr uses the geo microformat to denote the location of geotagged photos. For instance, if you load this:

```
http://flickr.com/photos/raymondyee/18389540/
```

into Firefox, you can use Operator to extract the geo instance:

```
<span class="geo" style="display: none">
  <span class="latitude">37.4149</span>,
  <span class="longitude">-122.078</span>
</span>
```

Then you can invoke one of the default actions (for example, Find with Google Maps) to plot the location of the photo at this location:

```
http://maps.google.com/maps?ll=37.8721,-122.257704&q=37.8721,-122.257704
```

hCard and adr

hCard (http://microformats.org/wiki/hcard) is used to represent such entities as people, organizations, companies, and places. An easy way to get started with hCard is to use the hCard Creator at http://microformats.org/code/hcard/creator.

Let’s create an hCard for Tim Berners-Lee, the inventor of the Web, drawing on his web page at http://www.w3.org/People/Berners-Lee/ to come up with the following:

```
<div id="hcard-Tim-Berners-Lee" class="vcard">
  <a class="url fn" href="http://www.w3.org/People/Berners-Lee/">Tim Berners-
Lee</a>
  <div class="org">World Wide Web Consortium</div>
  <a class="email" href="mailto:timbl@w3.org">timbl@w3.org</a>
  <div class="adr">
    <div class="street-address">77 Massachusetts Ave. (MIT Room 32-G524)</div>
    <span class="locality">Cambridge</span>
,
    <span class="region">MA</span>
,
    <span class="postal-code">02139</span>

    <span class="country-name">USA</span>

  </div>
  <div class="tel">+1 (617) 253 5702</div>
  <p style="font-size:smaller;">This <a
href="http://microformats.org/wiki/hcard">
hCard</a> created with the <a href="http://microformats.org/code/hcard/creator">
hCard creator</a>.
  </p>
</div>
```

You’ll notice that inside the hCard microformat is the adr microformat (http://microformats.org/wiki/adr). adr is a mapping of vCard:

> *This specification introduces the adr microformat, which is a 1:1 representation of the aforementioned adr property from the vCard standard, by simply reusing the adr property and sub-properties as-is from the hCard microformat.*

There is support in adr for the following properties, which show up in adr as (X)HTML attributes according to class-design-pattern:

* post-office-box
* extended-address
* street-address
* locality
* region
* postal-code

* country-name

hCalendar

hCalendar (http://microformats.org/wiki/hcalendar) is a microformat-based iCalendar used to represent calendar information. To quickly create an instance, use the hCalendar Creator (http://microformats.org/code/hcalendar/creator), or consult the hCalendar cheat sheet (http://microformats.org/wiki/hcalendar-cheatsheet). Let's create an hCalendar for the WWW 2008 conference (http://www2008.org/):

```
<div class="vevent"
     id="hcalendar-WWW-2008-17th-International-World-Wide-Web-Conference">
  <a class="url" href="http://www2008.org/">
    <abbr class="dtstart" title="20080421">April 21th</abbr> &mdash;
    <abbr class="dtend" title="20080426">25th, 2008</abbr>
    <span class="summary">
      WWW 2008 (17th International World Wide Web Conference)
    </span>&mdash; at
    <span class="location">Beijing International Convention Center, </span>
  </a>
  <div class="description">"The World Wide Web Conference is a global event
bringing
together key researchers, innovators, decision-makers, technologists,
businesses,
and standards bodies working to shape the Web. Since its inception in 1994, the
WWW
conference has become the annual venue for international discussions and debate
on
the future evolution of the Web."</div>
  <p style="font-size: smaller;">This
    <a href="http://microformats.org/wiki/hcalendar">hCalendar event</a>
brought to you by the
    <a href="http://microformats.org/code/hcalendar/creator">hCalendar
Creator</a>.
  </p>
</div>
```

Other Microformats

Here are some other noteworthy microformats:

* xoxo (http://microformats.org/wiki/xoxo) represents hierarchical outlines (that is, nested lists).
* vote-links (http://microformats.org/wiki/vote-links) indicates whether a link represents a vote-for, vote-abstain, or vote-against the link.
* hReview (http://microformats.org/wiki/hreview) represents reviews of URLs.
* hResume (http://microformats.org/wiki/hresume) represents resumes.

Microformats in Practice

You can learn a lot about microformats by studying how they are actually being used on the Web. Some implementations include the following:

* The use of `adr`, `hCard`, `hCalendar`, `tag`, and `geo` by Upcoming.yahoo.com and Eventful.com
* The use of `adr` and `hCard` at Yahoo! Local
* The use of `hCard` and `adr` on Technorati

I suggest using the list of implementations of microformats in the wild (`http://microformats.org/wiki/examples-in-the-wild`), which includes lists for `geo`, `hCalendar`, `hCard`, `hReview`, and `include-pattern`. Go to the listed sites, and use Operator to pick out the microformats.

Programming with Microformats

For simple microformats, including the ones that depend on `rel-design-pattern`, it should be simple enough to write your own code to parse data from and write data to the appropriate `rel` and `rev` attributes. It takes a lot more work to handcraft parsers for complicated microformats such as `hCard` and `hCalendar` because there are many possible properties.

There are no schemas for microformats, only specifications written for direct human interpretation, which makes difficult any autogeneration of high-quality language-specific parsers from the specifications.[9]

A challenge in working with microformats is the lack of validators. Norm Walsh argues that W3C Schema and Relax-NG will not work for the purpose of expressing the syntax of microformats as schemas, though Schematron might be up for the task.[10] You can use XMDP, a schema (of sorts) geared to easy human consumption, to get partway to generating validators, argues Brian Suda, at least for some simple formats.[11]

Hence, you will need to look for some handcrafted language-specific libraries to handle microformats. Start by looking at `http://microformats.org/wiki/implementations`.

Language-Specific Libraries

Here are some language-specific libraries:

* `mofo` (`http://mofo.rubyforge.org/`) is a new Ruby library that has support for a variety of microformats including `hCard`, `hCalendar`, and `xfn`.
* `uformats` (`http://rubyforge.org/projects/uformats/`) is another Ruby library that has support for `hReview`, `hCard`, `hCalendar`, `rel-tag`, `rel-license`, and `include-pattern`.

9. http://smackman.com/2006/06/01/an-old-idea/ and http://lists.w3.org/Archives/Public/public-rdf-in-xhtml-tf/2006Jun/0011.html.

10. See http://norman.walsh.name/2006/04/13/validatingMicroformats for more about validating microformats. Erik van der Vlist adds to this analysis at http://eric.van-der-vlist.com/blog/2277_Validating_microformats.item.

11. http://norman.walsh.name/2005/09/05/microformats#comment0008

* For PHP 5, consider using `hKit` (`http://allinthehead.com/hkit/`), which has support for hCard.
* Probably the best library out there is `Microformats.js`, which is the heart of the Operator add-on.[12]

There are interesting things to do with Operator, both for what it can do today and for how it might be a harbinger of things to come in Firefox 3 (which might have native support for microformats).[13] Operator makes a great sandbox for experimenting with microformats. Here are some things to try:

* Download and install `user-scripts` to add new actions and new microformats (`http://www.kaply.com/weblog/operator-user-scripts/`).
* Try your hand at writing new actions or support for new microformats by studying existing scripts and the documentation.[14]
* Study the code for Operator to pick up on the subtleties that go into working code using microformats.[15]

Writing an Operator Script

In this section, I'll lead you through the process of creating a simple user script for Operator. Start by looking through the best documentation for understanding Operator scripts:

`http://www.kaply.com/weblog/operator-user-scripts/`

There you will find a tutorial for writing a script that lets users find the closest Domino's Pizza to a given instance of an address (`adr`):

`http://www.kaply.com/weblog/operator-user-scripts/creating-a-microformat-action-user-script-basic/`

In this section, I will walk you through the steps to create a script that performs a similar function. Instead of converting an `adr` instance into a URL to the Domino's Pizza web site, our script will geocode the address by creating a URL to `http://geocoder.us`. Since our script is similar to that of the tutorial, we will follow a two-step strategy:

1. Install the tutorial script to understand how it works.
2. Convert the script to one that geocodes the `adr` instance.

12. http://svn.mozilla.org/labs/operator/chrome/operator/content/Microformats/Microformats.js

13. http://www.readwriteweb.com/archives/mozilla_does_microformats_firefox3.php

14. http://www.kaply.com/weblog/2007/04/24/operator-action-architecture/ and http://www.kaply.com/weblog/2007/04/18/microformat-objects-in-javascript/

15. http://svn.mozilla.org/labs/operator/

Studying the Tutorial Script

You will find the tutorial script here:

```
http://www.kaply.com/weblog/wp-content/uploads/2007/07/dominos.js
```

It's possible that after this book is published, there might be a newer version of the referenced user scripts. You can check here: `http://www.kaply.com/weblog/operator-user-scripts/`.

Install it and restart your web browser. If you run the action on this:

```
http://upcoming.yahoo.com/event/144855
```

your browser will conduct a search for the closest Domino's Pizza stores to 1401 N Shoreline Blvd in Mountain View, CA:

```
http://www.dominos.com/apps/storelocator-
EN.jsp?street=1401%20N%20Shoreline%20Blvd.& ~CCC
cityStateZip=California,%20Mountain%20View%2094043
```

Let's now study the script to understand how it works:

```
var dominos = {
  description: "Find the nearest Domino's Pizza",
  shortDescription: "Domino's",
  scope: {
    semantic: {
      "adr" : "adr"
    }
  },
  doAction: function(semanticObject, semanticObjectType) {
    var url;
    if (semanticObjectType == "adr") {
      var adr = semanticObject;
      url = "http://www.dominos.com/apps/storelocator-EN.jsp?";
      if (adr["street-address"]) {
        url += "street=";
        url += adr["street-address"].join(", ");
      }
      if ((adr.region) || (adr.locality) || (adr["postal-code"])) {
        url += "&cityStateZip=";
      }
      if (adr.region) {
        url += adr.region;
        url += ", ";
      }
      if (adr.locality) {
        url += adr.locality;
        url += " ";
      }
      if (adr["postal-code"]) {
        url += adr["postal-code"];
      }
    }
    return url;
  }
};

SemanticActions.add("dominos", dominos);
```

There are several elements to notice about this script as you think about how to adapt it:

* The `dominos` JavaScript object defines an action. An action consists of four properties: `description`, `shortDescription`, `scope`, and `doAction`.
* You should change the name of the JavaScript object, its `description`, and its `shortDescription` to fit the purpose of the new script.
* The `scope` property is used to tie an action to a specific data format. The following:

```
scope: {
    semantic: {
      "adr" : "adr"
    }
```

  means any `adr` instance. You can limit the scope to only `adr` instances with the property `locality` with this:

```
scope: {
    semantic: {
      "adr" : "locality"
    }
```

  or to a certain URL:

```
 scope: {
  url: "http://www.flickr.com"
 }
```

* Associated with the `doAction` property is a function that actually creates the URL for Domino’s Pizza by concatenating the various pieces of the `adr` instance. To adapt this function, you need to understand the URL structure of `http://geocoder.us`, the service we will use to geocode the address.
* Note that the simplest type of action of an Operator script is to return a URL, which the browser then loads. (You can learn how to get Operator actions to perform other operations by reading the advanced tutorials at `http://www.kaply.com/weblog/operator-user-scripts/`.)

Writing a Geocoding Script

As you learned in Chapter 13, there are a variety of sites to use to geocode an address in the United States. One service is Geocoder.us. You can geocode an address here:

```
http://geocoder.us/demo.cgi?address={address}
```

For example:

```
http://geocoder.us/demo.cgi?address=1600+Pennsylvania+Ave%2C+Washington+DC
```

Taking the URL template for Geocoder.us into account, you can adapt the script to come up with something like the following:

```
// based on http://www.kaply.com/weblog/wp-content/uploads/2007/07/dominos.js

var geocoder_us = {
  description: "Geocode with geocoder_us",
  shortDescription: "geocoder_us",
  scope: {
    semantic: {
      "adr" : "adr"
    }
  },
  doAction: function(semanticObject, semanticObjectType) {
    var url;
    if (semanticObjectType == "adr") {
      var adr = semanticObject;
      url = "http://geocoder.us/demo.cgi?address=";
      if (adr["street-address"]) {
        url += adr["street-address"].join(", ");
        url += ", ";
      }
      if (adr.locality) {
        url += adr.locality;
        url += ", ";
      }
      if (adr.region) {
        url += adr.region;
        url += ", ";
      }
      if (adr["postal-code"]) {
        url += adr["postal-code"];
      }
    }
    return url;
  }
};

SemanticActions.add("geocoder_us", geocoder_us);
```

The resulting URL for Mashup Camp IV on Upcoming.yahoo.com is as follows:

```
http://geocoder.us/demo.cgi?address=1401%20N%20Shoreline%20Blvd.,%20Mountain%20V
iew, ~CCC
%20California,%2094043
```

Resources (RDFa): A Promising Complement to Microformats

There's a lot of hype around RDF and the semantic Web, but the core concept of the Resource Description Framework (RDF) is simple:

* An RDF document is just a series of statements about resources in a subject-predicate-object (triplet) form. In other words, they are statements where a resource (R) has a property (P) of a value (V)—a triplet (R,P,V). For example: `("Raymond Yee", "has age of ", 40)`.
* RDF vocabularies define ways to talk about such things as types of resources and terms for properties. For example, a genealogical vocabulary would define properties such as "is mother of" and "is sister of."
* Once we have these types of RPVs around, we can add to the mix various logical propositions. If V > 30 of an RPV with `P="has age of"`, then `(R, "has to trust status", No)`. In other words, a computer program should be able to deduce that Raymond Yee should not be trusted since he is older than 30, since one must not trust anyone older than 30.

Tim Bray's "What is RDF?" (`http://www.xml.com/pub/a/2001/01/24/rdf.html`) was the first essay I read in my attempts to understand RDF. It's still very good. However, I think that the triplets idea was still unclear to me after reading the essay. (And I don't blame Tim Bray for that since the idea is clearly in the essay.) So, you should follow up Bray's essay with reading something like Aaron Schwartz's "RDF Primer Primer" (`http://notabug.com/2002/rdfprimer/`). The two complement each other.

You can express RDF triplets in many ways, including the standard RDF/XML syntax (`http://www.w3.org/TR/rdf-syntax-grammar/`). Since we have been discussing how microformats embed machine-understandable data in (X)HTML, we'll now look at RDFa (`http://rdfa.info/about/`), described in the following way:

> *With RDFa, you can easily include extra "structure" in your HTML to indicate a calendar event, contact information, a document license, etc. . . . RDFa is about total publisher control: you choose which attributes to use, which to reuse from other sites, and how to evolve, over time, the meaning of these attributes.*

Here is a sample RDFa assertion, in which the resource (a book with ISBN of 9781590598580) has a property (namely, the Dublin Core title) whose value is `Pro Web 2.0 Mashups: Remixing Data and Web Services`:[16]

```
<span xmlns:dc="http://purl.org/dc/elements/1.1/" about="isbn:9781590598580"
      property="dc:title">Pro Web 2.0 Mashups: Remixing Data and Web
Services</span>
```

I think that microformats and RDFa will both have a place on the Web. Microformats already have some good uptake and are grounded in today's real-world problems. They are focused on very specific applications. RDFa provides a mechanism for making more general assertions about pieces of data.

16. http://examples.mashupguide.net/ch18/sample_rdf.html

Reference for Further Study

The following are useful resources for more on microformats:

* The microformat book at `http://microformatique.com/book/`

* Micah Dubinko's "What Are Microformats?"[17]
* Uche Ogbuji's "Microformats in Context"[18]

Summary

You can use microformats and RDFa to embed data into the human-readable contexts of (X)HTML. In this chapter, you looked at instances of microformats that you can find "in the wild" (such as on Upcoming.yahoo.com) and ones that you can craft as simple examples, and you learned about how you can use microformats to embed data (such as contact information, addresses, geolocations, bookmarks, tags, and licenses) into (X)HTML. Microformats tend to follow certain common design patterns (that is, use class attributes or use the `rel` attribute) and are adapted from existing standards (such as iCalendar and vCard).

In this chapter, you learned how to use the Operator Firefox extension to work with microformats, including extracting them from web pages and invoking actions on them. You saw how these Operator actions enact simple mashups that move data from any web site with embedded microformats to another web site.

17. http://www.xml.com/pub/a/2005/03/23/deviant.html

18. http://www.xml.com/pub/a/2006/04/26/microformats-grddl-rdfa-nvdl.html