

HTML5 fundamentals, Part 1

Getting your feet wet

Skill Level: Introductory

[Grace Walker \(gwalker@walkerautomated.com\)](mailto:gwalker@walkerautomated.com)

IT Consultant

Walker Automated Services

24 May 2011

HTML5 reflects the monumental changes in the way you now do business on the web and in the cloud. This article is the first in a four-part series designed to spotlight changes in HTML5, beginning with the new tags and page organization and providing high-level information on web page design, the creation of forms, the use and value of the APIs, and the creative possibilities that Canvas provides.

HTML5 is a language designed to organize web content. It is intended to make web design and development easier by creating a standardized and intuitive UI markup language. HTML5 provides the means to dissect and compartmentalize your pages, and it allows you to create discrete components that are not only designed to organize your site logically but are also created to give your site syndication capabilities. HTML5 could be called the "information mapping approach to website design" because it incorporates the essence of information mapping, dividing and labeling information to make it easy to use and understand. This is the foundation of HTML5's dramatic semantic and aesthetic utility. HTML5 gives designers and developers of all levels the ability to publish to the world everything from simple text content to rich, interactive multimedia.

Frequently used acronyms

- **API:** Application programming interface
- **CSS3:** Cascading style sheet version 3
- **GUI:** Graphical user interface
- **HTML:** Hypertext Markup Language

- **HTML5:** HTML version 5
- **SQL:** Structured Query Language
- **UI:** User interface

HTML5 provides effective data management, drawing, video, and audio tools. It facilitates the development of cross-browser applications for the web as well as portable devices. HTML5 is one of the technologies driving the advances in mobile cloud computing services, as it allows for greater flexibility, permitting the development of exciting and interactive websites. It also introduces new tags and enhancements, including an elegant structure, form controls, APIs, multimedia, database support, and significantly faster processing speed.

The new tags in HTML5 are highly evocative, encapsulating their role and use. Past versions of HTML used rather nondescript tags. However, HTML5 has highly descriptive, intuitive labels. It provides rich content labels that immediately identify the content. For example, the overworked `<div>` tag has been supplemented by the `<section>` and `<article>` tags. The addition of the `<video>`, `<audio>`, `<canvas>`, and `<figure>` tags also provides a more precise description of the specific type of content.

HTML5 provides:

- Tags that describe exactly what they are designed to contain
- Enhanced network communications
- Greatly improved general storage
- Web Workers for running background processes
- The WebSocket interface to establish continuous connection between the resident application and the server
- Better retrieval of stored data
- Improved page saving and loading speeds
- Support for CSS3 to manage the GUI, which means that HTML5 can be content oriented
- Improved browser form handling
- An SQL-based database API that allows client-side, local storage.
- Canvas and video, for adding graphics and video without installing third-party plug-ins
- The Geolocation API specification, which uses smart phone location

capabilities to incorporate mobile cloud services and applications

- Enhanced forms that reduce the need to download JavaScript code, allowing more efficient communication between mobile devices and cloud servers

HTML5 creates a more engaging user experience: Pages designed using HTML5 can provide an experience similar to desktop applications. HTML5 also provides enhanced multiple platform development by combining the capability of APIs with the ubiquity of the browser. Using HTML5, developers can provide a modern application experience that smoothly crosses platforms.

When you say HTML5, you are using shorthand for continuous innovation. New tags, new methodologies, and a general development framework resting on the interplay of HTML5 and its two counterparts, CSS3 and JavaScript. This is the core of the client-centered application processing phenomena. In addition to the many desktop deployments of HTML5 technology's techniques and methods, HTML5 can be implemented in feature-rich web mobile phone browsers—a growing market, as witnessed by the popularity and pervasiveness of the Apple iPhone, Google Android, and phones running Palm webOS.

A critical aspect of HTML5's power is the information mapping—or content blocking, if you prefer—that produces a much more comprehensible process. You can see how effective this tool has become for design and development by its increasing domination of the web processing world.

HTML5 heralds the advent of a more effective semantic process at the text level and greater control over the construction and use of forms. All of these qualities and the many other fine points of HTML5 innovation are the basis for the growing domination of this paradigm. Many agency entities, commercial and otherwise—even many of those organizations most remotely involved with information processing and communications as their primary agency activity—are to one degree or another seized by the development of this growing phenomena.

HTML5 is not a magic lamp, and there is no genie. However, its technical and methodological assets have made it the next best thing to rubbing a lamp.

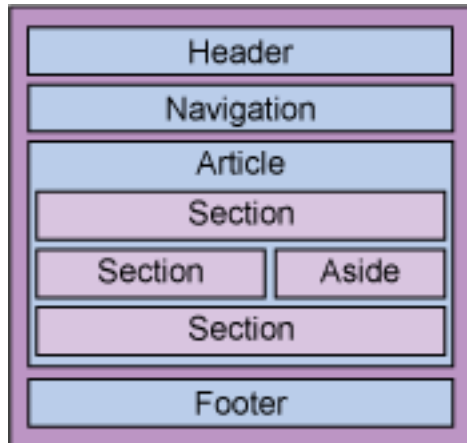
Plan the page

You are going to create a simple web page. During this process, I discuss several of the new tags that have been introduced in HTML5. To create an effective as well as an efficient web page, you must develop a plan that considers all the components you want to craft.

The page you create will have the high-level design shown in Figure 1. The page design contains a Header area, a Navigation area, an Article area that contains

three sections, an Aside area, and, finally, the Footer area. This page is designed to work in the Google Chrome browser, eliminating some of the visual clutter that can accompany browser-compatibility issue rectification as well as hamper understanding of basic structure. The goal is to create a page that clearly delineates the use of the new HTML5 tags, showing how you can use them to create well-formed code and elegant page design.

Figure 1. Acme United web page plan



In the process of creating this page, I touch on CSS3, which is required to properly render HTML5 pages. CSS3 is essential for the styling, navigation, and general feel of the HTML5 page. Its property groups, which you can find at the W3Schools CSS3 reference site (see [Resources](#)), includes some useful elements such as background, font, marquee, and animation.

Before you begin the page construction, however, you need to learn about a few of the new HTML5 tags.

The Header area

The example Header area contains the page title and subtitle. You use the `<header>` tag to create the content for the Header area of the page. The `<header>` tag can contain the opening information about a `<section>` and an `<article>` in addition to the web page itself. The web page created here has a Header area for the page, which is shown in the high-level design, as well as a Header area inside the Article and Section areas. Listing 1 provides a `<header>` tag markup example.

Listing 1. `<header>` tag example

```
<header>
  <h1>Heading Text</h1>
  <p> Text or images can be included here</p>
  <p> Logos are frequently placed here too</p>
```

```
</header>
```

The `<header>` tag can also contain an `<hgroup>` tag, as shown in Listing 2. The `<hgroup>` tag groups headings together, using the `<h1>` to `<h6>` heading levels shown here with a main heading and a sub-heading.

Listing 2. `<hgroup>` tag example

```
<header>
  <hgroup>
    <h1>Main Heading</h1>
    <h2>Sub-heading </h2>
  </hgroup>
  <p> Text or images can be included here</p>
</header>
```

The Navigation area

You create the Navigation area of the page using the `<nav>` tag. The `<nav>` element defines an area specifically intended for navigation. The `<nav>` tag should be used for the main site navigation, not to hold links contained in other areas of the page. The Navigation area can contain code like that shown in Listing 3.

Listing 3. `<nav>` tag example

```
<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About Us</a></li>
    <li><a href="#">Our Products</a></li>
    <li><a href="#">Contact Us</a></li>
  </ul>
</nav>
```

The Article and Section areas

The page you are designing contains one Article section, which holds the actual content of the page. You use the `<article>` tag to create this area, and the tag defines content that can be used independently of the other content found on the page. For example, if you want to create an RSS feed, you can use `<article>` to uniquely identify the content. The `<article>` tag identifies content that can be removed and placed in another context and be completely understandable.

The Article area in the Acme United plan contains three Section areas. You create these areas using the `<section>` tag. A `<section>` contains related component areas of web content. The `<section>` tag—and the `<article>` tag, as well—can

contain headers, footers, or any other components required to complete the section. The `<section>` tag is for grouping content. The content for both the `<section>` tag and the `<article>` tag usually starts with a `<header>` and ends with a `<footer>`, with the content for the tag in between.

The `<section>` tag can also contain `<article>` tags, just as the `<article>` tag can contain the `<section>` tag. The `<section>` tag should be used to group similar information, and the `<article>` tag should be used for information such as an article or a blog that can be removed and placed in a new context without affecting the meaning of the content. The `<article>` tag, as its name implies, provides a complete packet of information. In contrast, the `<section>` tag contains related information, but the information cannot be placed in a different context by itself, because its meaning would be lost.

See Listing 4 for an example of `<article>` and `<section>` tag usage.

Listing 4. `<article>` tag and `<section>` tag example

```
<article>
  <section>
    Content
  </section>
  <section>
    Content
  </section>
</article>
<section>
  <article>
    Content
  </article>
  <article>
    Content
  </article>
</section>
```

Image elements

Both the `<section>` and the `<article>` tags as well as the `<header>` and `<footer>` tags can contain the `<figure>` tag. You use this tag to include images, diagrams, and photos.

The `<figure>` tag can contain the `<figcaption>`, which in turn contains the caption for the figure contained in the `<figure>` tag, allowing you to enter a description that can tie the figure more closely to the content. Listing 5 provides an example of the `<figure>` and `<figcaption>` tag structure.

Listing 5. `<figure>` tag and `<figcaption>` tag example

```
<figure>
  
  <figcaption>Caption for the figure</figcaption>
```

```
</figure>
```

Media elements

The `<section>` and `<article>` tags can also contain various media elements. HTML5 provides tags that quickly convey an understanding of their content. Media elements, such as music and video that were previously only embedded, can now be identified more precisely.

The `<audio>` tag identifies sound content, such as music or any other audio streams. The `<audio>` tag has attributes that control what, when, and how audio will be played. The attributes are `src`, `preload`, `control`, `loop`, and `autoplay`. In the example shown in Listing 6, the audio starts to play as soon as the page loads and will play continuously and provide controls so the user can stop or restart the audio.

Listing 6. `<audio>` tag example

```
<audio src="MyFirstMusic.ogg" controls autoplay loop">
  Your browser does not support the audio tag.
</audio>
```

The `<video>` tag allows you to broadcast video clips or streaming visual media. It has all the attributes of the `<audio>` tag plus three more: `poster`, `width`, and `height`. The `poster` attribute lets you identify an image to be used while the video is loading or in the unfortunate circumstance when the video won't load at all.

Listing 7 provides an example of the `<video>` tag structure.

Listing 7. `<video>` tag example

```
<video src="MyFirstMovie.ogg" controls="controls">
  Your browser does not support the video tag
</video>
```

The `<video>` and `<audio>` tags can contain the `<Source>` tag, which defines multimedia resources for `<video>` and `<audio>` tags. With this element, you specify alternative video and audio files from which the browser can then choose based on its media type or codec support. In Listing 8, there are two choices. If the WMA version of the file cannot be played in the browser being used, then try the MP3. Otherwise, display the message so the user knows why the audio is not available.

Listing 8. `<source>` tag example

```
<audio>
  <source src="/music/good_enough.wma" type="audio/x-ms-wma">
  <source src="/music/good_enough.mp3" type="audio/mpeg">
  <p>Your browser does not support the HTML 'audio' element.</p>
</audio>
```

The `<embed>` tag defines embedded content that can be brought into a page—for example, a plug-in for Adobe Flash SWF files. Listing 9 contains the `type` attribute, identifying the embedded source as a Flash file.

Listing 9. `<embed>` tag example

```
<embed src="MyFirstVideo.swf" type="application/x-shockwave-flash" />
```

In addition to the attributes `src` and `type`, the `<embed>` tag has `height` and `width` attributes.

The Aside area

You create the Aside area in the Acme United plan by using the `<aside>` tag. Think of this tag as holding supplementary content that is not part of the flow of the article it supplements. In magazines, asides are frequently used to highlight a point that was made in the article itself. The `<aside>` tag contains content that can be removed without affecting the information conveyed by the article, section, or page that contains it.

Listing 10 provides an example of the `<aside>` tag's usage.

Listing 10. `<aside>` tag example

```
<p>My family and I visited Euro Disney last year.</p>
<aside>
  <h4>Disney in France</h4>
  <p>Besides Euro Disney, there is a Disneyland in California.</p>
</aside>
```

The Footer area

The `<footer>` element contains information about a page, article, or section, such as the author or date of the article. As a page footer, it may contain copyright or other important legal information, as shown in Listing 11.

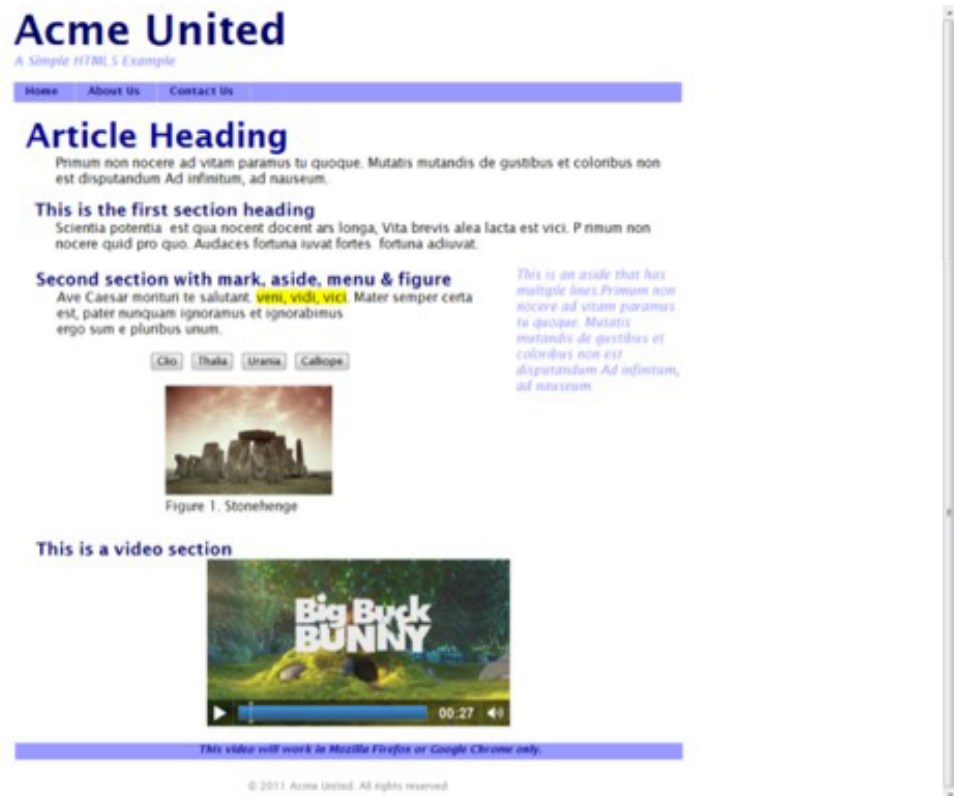
Listing 11. `<footer>` tag example


```
<footer>
  <p>Copyright 2011 Acme United. All rights reserved.</p>
</footer>
```

Constructing the page

Now that you know the basic tags needed to create an HTML5 page, let's begin to construct your page. You're going to construct a web page for Acme United. The completed page is shown in Figure 2 and can be downloaded for your use (see [Download](#)).

Figure 2. The Acme United web page



So, let's set up the page. First, there is the `<!doctype>`. In HTML5, the `<!doctype>` has been simplified: All you need to remember is `html`. This not only simplifies the entry for this tag, it also makes it more future-proof. Note that it is not called `html5`, just `html`. No matter how many versions of HTML may come and go, the `<!doctype>` can always be just `html`.

The `<html>` tag contains all the other HTML elements, excluding the `<!doctype>` tag. Every other element must be nested between the `<html>` and `</html>` tags. See Listing 12.

Listing 12. `<!doctype>` tag example

```
<!doctype html>
<html lang="en">
```

After indicating `html` and the English language, you have the `<head>` element, which can include scripts, browser support information, style sheet links, meta information, and other initialization functions. You can use these tags in the `head` section:

- `<base>`
- `<link>`
- `<meta>`
- `<script>`
- `<style>`
- `<title>`

The `<title>` tag is the holder for the actual title of the document and is a required `<head>` section element. It is the title that you see at the top of the browser when viewing the page. The `<link>` tag in Listing 13 identifies the CSS3 style sheet that will be used to render the HTML5 page. The style sheet called is `main-stylesheet.css`.

Listing 13. `<head>` tag example

```
<head>
  <title>HTML5 Fundamentals Example</title>
  <link rel="stylesheet" href="main-stylesheet.css" />
</head>
```

You use the `<body>` tag next, followed by the `<header>` and `<hgroup>` tags, which were described earlier. The `<h1>` area in this example contains the name of your fictitious company, Acme United, and the `<h2>` area contains the information informing you that it is sub-titled "A Simple HTML5 Example." Listing 14 shows the markup.

Listing 14. `<body>` tag and `<header>` tag example

```
<body >
  <header>
    <hgroup>
      <h1>Acme United</h1>
      <h2>A Simple HTML5 Example</h2>
    </hgroup>
  </header>
```

The CSS3 used to set up the page so far is shown in Listing 15. First, you establish the font for the page, and then the particulars for the body. The dimensions of the body are defined, and then you design the header paragraph structure for first- and second-level heading tags. These are the headers that you will use for the page.

Listing 15. CSS3 example #1

```
* {  
    font-family: Lucida Sans, Arial, Helvetica, sans-serif;  
}  
  
body {  
    width: 800px;  
    margin: 0em auto;  
}  
  
header h1 {  
    font-size: 50px;  
    margin: 0px;  
    color: #006;  
}  
  
header h2 {  
    font-size: 15px;  
    margin: 0px;  
    color: #99f;  
    font-style: italic;  
}
```

Listing 16 shows the `<nav>` tag, which is designed to handle the main site navigation.

Listing 16. `<nav>` example

```
<nav>  
  <ul>  
    <li><a href="#">Home</a></li>  
    <li><a href="#">About Us</a></li>  
    <li><a href="#">Contact Us</a></li>  
  </ul>  
</nav>
```

HTML5 also has a `<menu>` tag—a tag that has caused confusion for some designers and developers. The confusion stems from the fact that navigation is frequently referred to as the "navigation menu." The `<menu>` tag, which was deprecated in HTML version 4.01 and reborn in HTML5, is designed to enhance interactivity. It should not be used for the main navigation. The only tag that should be used for the main navigation is the `<nav>` tag. You will use the `<menu>` tag later in this example.

The formatting of the navigation is handled by CSS3. Each `<nav>` tag definition shown in Listing 17 represents a particular state of the `` and `` elements inside the `<nav>` tag.

Listing 17. CSS3 example #2

```

nav ul {
    list-style: none;
    padding: 0px;
    display: block;
    clear: right;
    background-color: #99f;
    padding-left: 4px;
    height: 24px;
}
nav ul li {
    display: inline;
    padding: 0px 20px 5px 10px;
    height: 24px;
    border-right: 1px solid #ccc;
}

nav ul li a {
    color: #006;
    text-decoration: none;
    font-size: 13px;
    font-weight: bold;
}

nav ul li a:hover {
    color: #fff;
}

```

Next is the Article area. This area, defined by the `<article>` tag, includes its own `<header>` information. The `<section>` contained in the `<article>` also contains a `<header>` tag of its own. See Listing 18.

Listing 18. `<article>` and `<section>` example

```

<article>
  <header>
    <h1>
      <a href="#" title="Link to this post" rel="bookmark">Article Heading</a>
    </h1>
  </header>
  <p> Primum non nocere ad vitam Paramus . . . </p>
  <section>
    <header>
      <h1>This is the first section heading</h1>
    </header>
    <p>Scientia potentia est qua nocent docentp . . . </p>
  </section>

```

Listing 19 shows the CSS3 markup that renders this format. Note that the definition for the paragraph, header, and section areas are all defined for the `<article>` tag in which they are contained. The `<h1>` tag defined here does not have the same format as the `<h1>` tag defined for the page-level `<h1>` tag.

Listing 19. CSS3 example #3

```

article > header h1 {

```

```

        font-size: 40px;
        float: left;
        margin-left: 14px;
    }

    article > header h1 a {
        color: #000090;
        text-decoration: none;
    }

    article > section header h1 {
        font-size: 20px;
        margin-left: 25px;
    }

    article p {
        clear: both;
        margin-top: 0px;
        margin-left: 50px;
    }

```

The second `<section>` tag in the `<article>` contains the same basic information as the first `<section>`, but this time you are going to use an `<aside>`, a `<figure>`, a `<menu>`, and a `<mark>` tag. See Listing 20.

The `<aside>` tag is used here to present information that is not part of the flow that encloses it. The `<figure>` tag includes a graphic of Stonehenge. This `<section>` also contains the `<menu>` tag, which you use to create buttons with the names of the four Muses. When one of the buttons is clicked, it provides information about that particular Muse. The `<mark>` tag is used inside the `<p>` tag to highlight the words *veni, vidi, vici*.

Listing 20. `<article>` and `<section>` example

```

<section>
  <header>
    <h1>Second section with mark, aside, menu & figure</h1>
  </header>
  <p class="next-to-aside"> . . . <mark>veni, vidi, vici</mark>. Mater . . .</p>
  <aside>
    <p>This is an aside that has multiple lines. . . .</p>
  </aside>
  <menu label="File">
    <button type="button" onClick="JavaScript:alert('Clio . . .')">Clio</button>
    <button type="button" onClick="JavaScript:alert('Thalia . . .')">Thalia</button>
    <button type="button" onClick="JavaScript:alert
      ('Urania . . .')">Urania</button>
    <button type="button" onClick="JavaScript:alert
      ('Calliope . . .')">Calliope</button>
  </menu>
  <figure>
    <figcaption>Figure 1. Stonehenge</figcaption>
  </figure>
</section>

```

The CSS3 for this section included a new definition for the `<p>` tag that has a shorter width than the width you set for the page. This change allows the aside to float to the right without overlapping the text. Listing 21 shows the markup.

Listing 21. CSS3 example #4

```
article p.next-to-aside {
    width: 500px;
}

article > section figure {
    margin-left: 180px;
    margin-bottom: 30px;
}

article > section > menu {
    margin-left: 120px;
}

aside p {
    position: relative;
    left: 0px;
    top: -100px;
    z-index: 1;
    width: 200px;
    float: right;
    font-style: italic;
    color: #99f;
}
```

Video section elements

This is the final component of the `<article>`: the `video` section. The example video is an ogg that auto-plays when the page is loaded, continuously loops, and provides controls for pausing and playing. In many contemporary instances, ogg videos use the extension `ogv` (the `v` for video), as shown in Listing 22. The `<audio>` tag works in the same way.

Listing 22. `<article>` and `<section>` example

```
<section>
  <header>
    <h1>This is a video section</h1>
  </header>
  <p><video src="http://people.xiph.org/~maikmerten/demos/BigBuckBunny.ogv"
    controls autoplay loop>
    <div class="no-html5-video"><p>This video will work in
      Mozilla Firefox or Google Chrome only. </p>
    </div>
  </video></p>
</section>
</article>
```

Listing 23 provides the CSS3 definitions for the `video` section.

Listing 23. CSS3 example #5

```
article > section video {
    height: 200px;
    margin-left: 180px;
```

```
}  
  
article > section div.no-html5-video{  
    height: 20px;  
    text-align: center;  
    color: #000090;  
    font-size: 13px;  
    font-style: italic;  
    font-weight: bold ;  
    background-color: #99f;  
}
```

The footer and closing of the page are shown in Listing 24.

Listing 24. <footer> tag example

```
        <footer>  
            <p>Copyright: 2011 Acme United. All rights reserved.</p>  
        </footer>  
    </body>  
</html>
```

The CSS3 for the footer is shown in Listing 25.

Listing 25. CSS3 example #5

```
footer p {  
    text-align: center;  
    font-size: 12px;  
    color: #888;  
    margin-top: 24px;  
}
```

Conclusion

The completion of your web page finishes the first part of this multipart series. The goal of this article was to introduce the new HTML5 regime. HTML5 is more than just an upgrade to HTML4: It is the new way to communicate digitally. With the functionality of CSS3 and JavaScript, HTML5 comes close to giving the developer the whole ball of wax in one pseudo-package. If you are willing to assimilate what you need from the vast amounts of HTML5 information out there for our common use, you will join the growing legion of competent HTML5 multimedia web designers and developers. The next installment in this series will look at how to code and format HTML5 forms.

Downloads

Description	Name	Size	Download method
Sample HTML, CSS3 files	HTML5Fundamentals.zip	10KB	HTTP

[Information about download methods](#)

Resources

Learn

- [Create modern Web sites using HTML5 and CSS3](#) (developerWorks, March 2010) is a multicomponent HTML5 and CSS3 article.
- In [New elements in HTML 5](#) (developerWorks, August 2007), you will find information for several of the new elements in HTML5.
- The [<html>5doctor](#) website provides an excellent view of the current trends in HTML5 today.
- The [W3Schools.com HTML5 Tag Reference](#) provides an extensive list of the HTML5 tags, definitions, and examples.
- The [WHATWG website](#) provides detailed information for the HTML5 specification.
- The [W3Schools.com CSS3 Reference](#) provides extensive information on CSS version 3.
- The [Web development zone](#) specializes in articles covering various Web-based solutions.
- Stay current with developerWorks' [technical events and webcasts](#).
- [developerWorks on-demand demos](#): Watch demos ranging from product installation and setup for beginners to advanced functionality for experienced developers.
- Follow [developerWorks on Twitter](#).

Discuss

- Create your [developerWorks profile](#) today and [set up a watchlist](#) on HTML. Get connected and stay connected with [developerWorks community](#).
- Find other [developerWorks members interested in web development](#).
- Share what you know: [Join one of our developerWorks groups focused on web topics](#).
- Roland Barcia talks about [Web 2.0 and middleware](#) in his blog.

About the author

Grace Walker

Grace Walker, a partner in Walker Automated Services in Chicago,

Illinois, is an IT consultant with a diverse background and broad experience. She has worked in IT as a manager, administrator, programmer, instructor, business analyst, technical analyst, systems analyst, and Web developer in various environments, including telecommunications, education, financial services, and software.

HTML5 fundamentals, Part 2: Organizing inputs

Interacting with your visitor

Skill Level: Intermediate

[Grace Walker](mailto:gwalker@walkerautomated.com) (gwalker@walkerautomated.com)

IT Consultant

Walker Automated Services

31 May 2011

HTML5 reflects the monumental changes in the way you now do business on the Web and in the cloud. This article is the second in a four-part series designed to spotlight changes in HTML5, beginning with the new tags and page organization and providing high-level information on web page design; the creation of forms; the use and value of the APIs; and, finally, the creative possibilities that Canvas provides. This second part introduces the concept of HTML5 form controls and touches on the role of JavaScript and CSS3.

Administration, data analysis, marketing strategy, and the other functions of enterprise-level institutions are all important. However, without a successful digital window that your potential customer can use—or be motivated to use—the necessary initial processes for the development of site visitor conversion will be absent. A positive, user-friendly experience that prompts the interactivity required for your endeavor is a primary institutional goal.

Frequently used acronyms

- **API:** Application programming interface
- **CSS3:** Cascading style sheet version 3
- **HTML5:** Hypertext Markup Language version 5
- **IT:** Information technology
- **UTC:** Coordinated universal time

The heart of interactivity is the site's forms. They facilitate an interactive exchange with the user so that the goals that motivated the website's construction can be advanced by converting site visitors. Forms are the central factor that energizes the interaction between website owners or agents and the website users, and as such, they are of extreme importance to the design and development of a site.

The center of that heart is found in the controls—radio buttons, check boxes, text boxes, number spinners, and the like. These elements are essential to the website users' discourse with the site. In other words, without fully functional controls (both in "mechanical" operation of the control and the control's appropriateness for the given task), there can be no dialogue and, consequently, no potential conversion.

It is imperative that the relationships of the conversion process be carefully considered, including all aspects of the interaction between the site visitor and the system in place. Speed of validation, input, cognition, navigation, loading of pages, and how the pages are set up all have an impact on the conversion process. Validation improvements and enhancements, along with the extended variety of form control options offered and the general multimedia nature of the HTML5 specifications, are all contributors to HTML5's potency in translating a site visitor into an actual site user.

HTML5 is an exceptionally vigorous tool for validation and the general assurance of sound web-based computing—a critical security asset. It is particularly important in the design and development of websites intended to solicit customers. Therefore, its use is essential to maintaining a healthy rate of conversion. If you can't draw prospects, you're in trouble; if you can't convert the prospects you do draw, you're in for a crash landing.

But help is here. HTML5 types, such as `email` and `telephone`, facilitate broad communication options. Combined with the structural clarity imparted by HTML5's semantic foundation, there is no barrier to a clear dialogue between you and the rest of the world.

Given our net-centric world, the hectic pace of the increasingly stochastic global economy, the rapid development of cloud computing, the exponential growth in the use of mobile technology, and cross-platform telecommunications solutions—both commercial and social—it is clear that we stand on the threshold of a brave new world of web-based computing and communication. It is a world that is both a function and a derivative of the evolving marriage between the many elements of communication and IT as well as the needs of a highly competitive global society.

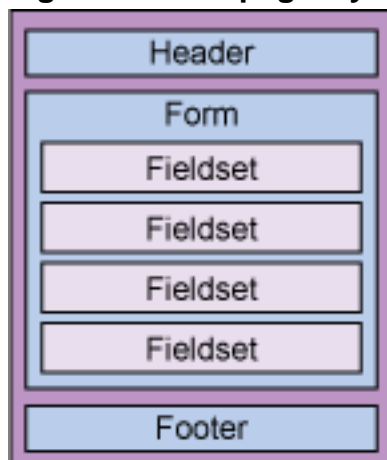
Designing the form

In HTML5, forms have been given a major refurbishing. Several tasks that previously required JavaScript coding can now be easily performed without it. This article's form

example analyzes the use of the HTML5 suite of form innovations. The first step in the process is, of course, planning the form.

The layout of the form example you will develop is shown in Figure 1. You will develop three areas for the form page: the Header area, the Form area, and the Footer area. The Header area includes the page heading and subheading wrapped in the `<header></header>` tags. At the bottom of the page, the Footer area contains the copyright information inside the `<footer></footer>` tags. I discussed the construction of a Header area and a Footer area in the example provided in [part 1](#) of this series: If you are unfamiliar with the `<header>` and `<footer>` tags, refer to that article.

Figure 1. Form page layout



This forms discussion focuses on four tags:

- `<form>`
- `<fieldset>`
- `<label>`
- `<input>`

In HTML5, two new attributes have been added to the `<form>` tag: `autocomplete` and `novalidate`. The `autocomplete` attribute enables the drop-down suggestion lists that appear on sites like Google. The `novalidate` attribute turns validation off for a form, which is valuable during testing.

The `<fieldset>` tag has three new attributes: `disable`, `name`, and `form`. The `disable` attribute deactivates the `<fieldset>`. The `name` attribute sets the name for the `<fieldset>`. The `form` attribute value is the ID of the form or forms to which the `<fieldset>` belongs. In HTML5, a `<fieldset>` can be outside of the form or forms to which it belongs. When a `<fieldset>` is placed outside of the form, you must set the `form` attribute of the `<fieldset>` tag so the `<fieldset>`

can be associated with the correct form or forms.

The `<label>` tag, which defines a categorization for an input element, has one new attribute: `form`. The `form` attribute value is the ID of the form or forms to which the `<label>` belongs. The `<label>` tag can also be placed outside of the form, so the `form` attribute here is also used to associate the `<label>` with the appropriate form.

The `<input>` tag has several new types as well as attributes that enhance the usability of the form. HTML5 has introduced a number of new input types designed to organize and categorize data, replicating the overall semantic approach of HTML5. The old adage that form should follow function is an appropriate one to describe HTML5 forms functions.

In HTML5, the form `<input>` field can be outside of the `<form>` tags. The `form` attribute identifies the form or forms to which the input field belongs. It also identifies the form that it belongs to by referring to the ID of the form. Table 1 shows the new `<input>` types.

Table 1. New `<input>` types

color	date	datetime	datetime-local	month
week	time	email	number	range
search	tel	url		

Table 2 shows the new `<input>` attributes.

Table 2. New `<input>` attributes

autocomplete	autofocus	form	formaction	formenctype
formmethod	formnovalidate	formtarget	height	max
min	multiple	pattern	placeholder	required
step				

During web page creation, you use most of these types and attributes.

Creating the form

The web page shown in Figure 2 is for The Classical Music Place, a site that has the works of several composers for download. It also allows aficionados of classical compositions to upload their recordings of classical pieces. This is the page you're going to create.

Figure 2. The Classical Music Place form

The Classical Music Place

Acme United Subsidiary

Customer Info

Name:

Telephone:

Email address:

Date:

Favorite Composer

Composers

Bach:

Beethoven:

Brahms:

Chopin:

Mendelssohn:

Upload file(s)

Upload one of your orchestra's file(s) for inclusion in our library

No file chosen



© 2011 Acme United. All rights reserved.

The form's structure begins with the `<form>` tag. In this example, you use the new `autocomplete` attribute, shown here:

```
<form id="orderForm" autocomplete="on" action="javascript:alertValues();" method="get">
```

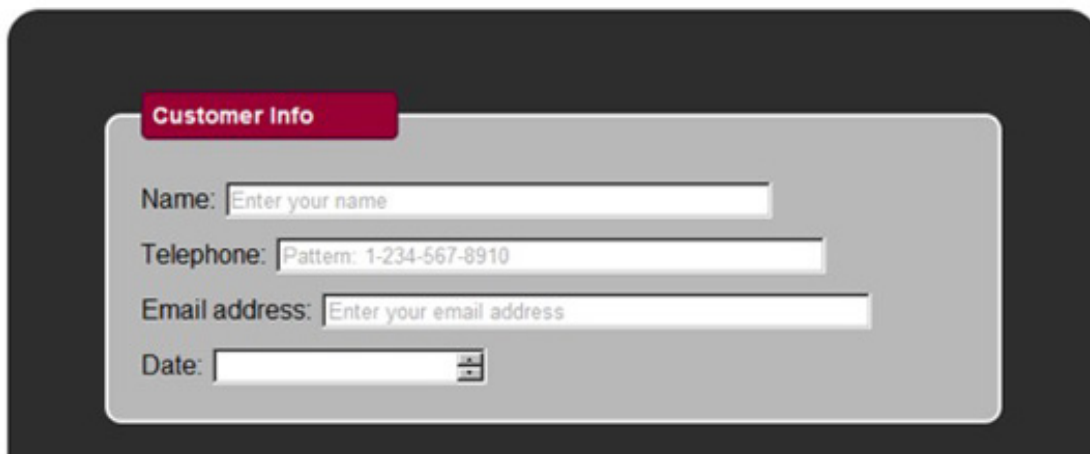
A JavaScript action is contained in the `<form>` tag, as well, which I'll discuss in a later section.

The `<form>` tag contains four `<fieldset>` tags, visually outlined by the gray areas in [Figure 2](#): Name, Telephone, Email address, and date. The `<fieldset>` tag groups like content on the form. Let's look at each `<fieldset>` separately.

First `<fieldset>` tag

The first `<fieldset>` tag contains customer information (see [Figure 3](#)). It has a **Name** field, a **Telephone** field, an **Email address** field, and a **Date** field. The **Name** field has an `autofocus` attribute, which lets you start entering text without having to click in the field.

Figure 3. Customer information fields



The first `<fieldset>` contains a `<legend>`, a `<label>`, and an `<input>` tag, as shown in [Listing 1](#). The Name field, which is type `text`, has three of the new `<input>` attributes: `placeholder`, `autofocus`, and `required`.

Listing 1. The Name field

```
<fieldset>
  <legend> Customer Info </legend>
  <p>
    <label>Name:
      <input id="name" name="name" type="text" placeholder="Enter your name"
              autofocus required size="50">
    </label>
  </p>
```


The `autofocus` ensures that the input focus will be on this field when the page opens. This is a function that causes the page to focus as soon as it loads, allowing the user immediate access to the form.

The `placeholder` attribute puts the text between the quotations marks into the field in a muted gray text. The `placeholder` attribute should tell the user what the field must contain and is displayed when the field is empty. However, because the Name field also has `autofocus` set, when you enter the page, you can't actually see the text. Note that in Figure 3, the Name field does not display the placeholder, and it is framed in a yellow highlight. If you move to another field without entering data, the placeholder text will be displayed. When you use the `autofocus` attribute with the `placeholder` attribute, the placeholder text disappears because of the activated focus in the field.

The `required` attribute facilitates the mandatory population of a field as a precondition to the submission of the form. This is valid for text, search, URL, telephone, email, password, date pickers, number, check box, radio button, and file fields.

The Telephone field, which is type `tel`, has the attributes `required`, `placeholder`, `size`, and `pattern`, as shown in Listing 2. The `tel` type is a text field designed to contain telephone numbers. In the example, the telephone has a pattern restriction that must be strictly observed, because the system will not let you submit until you use the right pattern of characters. The placeholder for the telephone holds the pattern against which your input is matched.

The pattern emulates the functions of a traditional JavaScript regular expression (regex). The input has to match the defined pattern structure of the regex before it can be validated. It works with `text`, `search`, `url`, `telephone`, `email`, and `password` types.

Listing 2. The Telephone field

```
<p>
  <label>Telephone:
    <input id="tel" name="telephone" type="tel" placeholder="Pattern: 1-234-567-8910"
      required size="50" pattern="([0-9]{1}(-[0-9]{3})(-[0-9]{3})(-[0-9]{4}))">
  </label>
</p>
```

The Email address field is type `email`, as shown in Listing 3. The email address is automatically validated without having to use a pattern: The validation is part of HTML5. If the email address is not presented in the appropriate format, the form cannot be submitted.

Listing 3. The Email address field

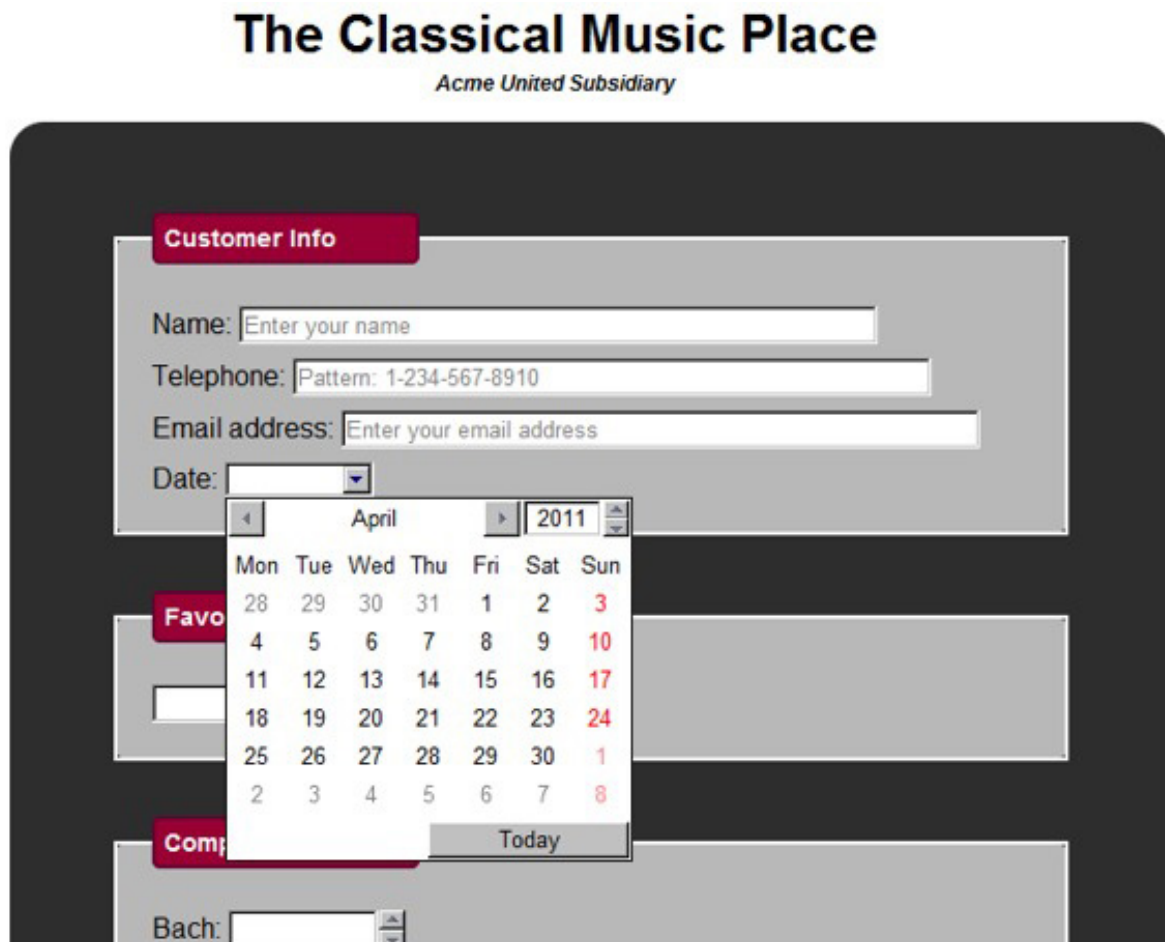
```

<p>
  <label>Email address:
    <input id="email" name="email" type="email" placeholder="Enter your email address"
      required size="50">
  </label>
</p>

```

The last entry in this `<fieldset>` is the Date field. The `date` type provides a calendar picker for date selection in the Opera browser but creates a spinner in Google Chrome, as seen in [Figure 3](#). Figure 4 shows the Opera rendition of the web page with the **Date** field picker displayed. Note that in Opera, the corners of the field set are not rounded, even though the same style sheet was used that rounded the corners in Chrome.

Figure 4. The Date field



Listing 4 shows the code used to create the date picker.

Listing 4. The Date field

```
<p>
  <label>Date: <input type="date">
</label>
</p>
</fieldset>
```

You can generate a report for any date. You can even break it down to the hour. Here are the date types that you can create:

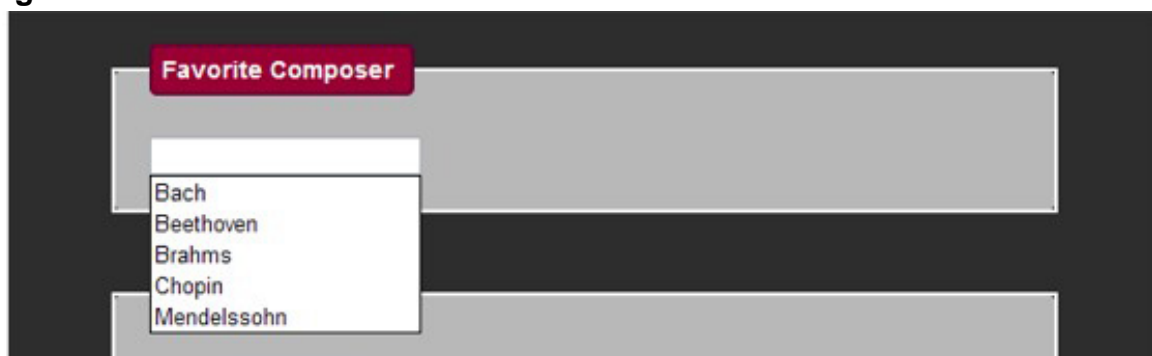
- **date.** Selects date, month, and year
- **month.** Selects month and year
- **week.** Selects week and year
- **time.** Selects time (hour and minute)
- **datetime.** Selects time, date, month, and year (UTC time)
- **datetime-local.** Selects time, date, month, and year (local time)

Second <fieldset> tag

The second <fieldset> tag contains an <input> tag with a `list` attribute and a <datalist> tag. The `list` attribute specifies an input field's <datalist>. The <datalist> tag provides a list of input field options. The `list` attribute works with these <input> types: text, search, url, telephone, email, date pickers, number, range, and color.

The <datalist> is displayed in a drop-down list, as shown in Figure 5. This image was taken in Opera. In Chrome, this list is displayed as a simple text box. The user would not be presented with the list.

Figure 5. The Favorites field



Listing 5 provides the field set that creates the Favorite Composer section.

Listing 5. The Favorite Composer field

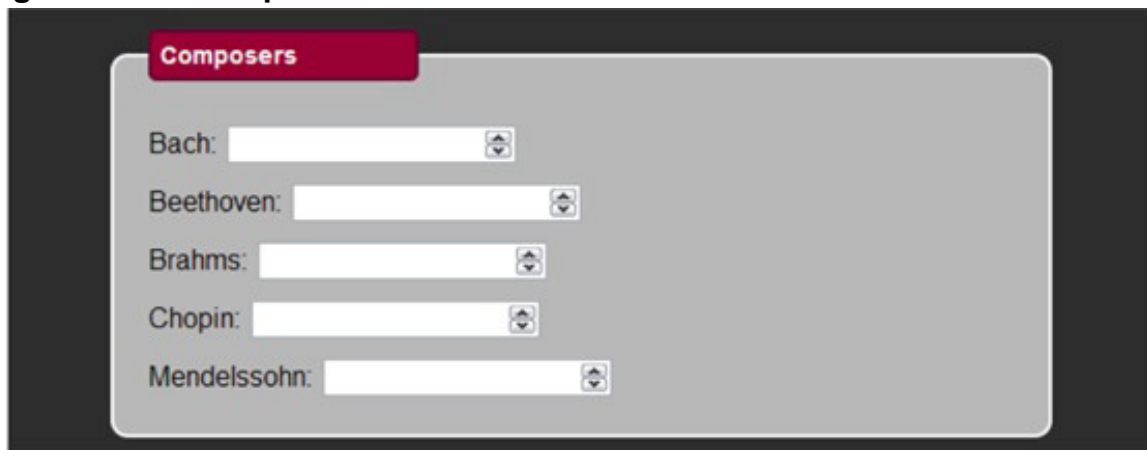
```
<fieldset>
```

```
<legend> Favorite Composer </legend>
<p>
  <label>
    <input type="text" name="favorites" list="composers">
    <datalist id="composers">
      <option value="Bach">
      <option value="Beethoven">
      <option value="Brahms">
      <option value="Chopin">
      <option value="Mendelssohn">
    </datalist>
  </label>
</p>
</fieldset>
```

Third <fieldset> tag

The third <fieldset> tag displays a list of composers, followed by a number field that designates how many works are available for each of the listed composers. Figure 6 shows the section.

Figure 6. The Composers fields



For example, Bach has five works, and Beethoven has 10 works. Listing 6 shows the maximum values for each composer. The number field will not accept more than the maximum value when you submit the form. When submitted, this field responds to incorrect, out-of-scope values by prompting you to correct the input so that it complies with the acceptable numeric limits of the field.

The `number` type creates a spinner field for input. The `min`, `max`, and `step` are used with the `number` type. The default step value is 1. The `min`, `max`, and `step` attributes are used with `number`, `range`, or `date picker` input constraints. The `max` attribute determines the maximum value permitted for the input field. The `min` attribute determines the minimum value permitted for the input field. The `step` attribute determines the valid numeric increment.

Listing 6. The composer fields

```

<fieldset>
  <legend>Composers</legend>
  <p>
    <label>
      Bach: <input name="form_number" id="form_number" type="number" min="1" max="5" >
    </label>
  </p>
  <p>
    <label>
      Beethoven: <input name="form_number" id="form_number" type="number"
        min="1" max="10" >
    </label>
  </p>
  <p>
    <label>
      Brahms: <input name="form_number" id="form_number" type="number" min="1" max="7" >
    </label>
  </p>
  <p>
    <label>
      Chopin: <input name="form_number" id="form_number" type="number" min="1" max="10" >
    </label>
  </p>
  <p>
    <label>
      Mendelssohn: <input name="form_number" id="form_number" type="number"
        min="1" max="4">
    </label>
  </p>
</fieldset>

```

Fourth <fieldset> tag

The fourth <fieldset> tag contains the <input> type file and the attribute multiple, as shown in Figure 7. The multiple attribute specifies that an input field can select multiple values from a data list or file list. In the example, a user can select multiple files for uploading.

Figure 7. The Upload field

The screenshot shows a web form with a dark background. At the top, there is a red button labeled "Upload file(s)". Below it, a light gray box contains the text "Upload one of your orchestra's file(s) for inclusion in our library". Inside this box, there is a "Choose File" button and the text "No file chosen". Below the gray box, there is a large, metallic-looking "Submit" button.

© 2011 Acme United. All rights reserved.

The code for the file type and multiple attribute is shown in Listing 7.

Listing 7. The Upload field

```
<fieldset>
  <legend> Upload file(s) </legend>
  <p>Upload one of your orchestra's file(s) for inclusion in our library</p>
  <p><label>
    <input type="file" multiple="multiple" />
  </label>
</p>
</fieldset>
```

The **Submit** button uses the height and width attributes, as shown in Listing 8. You use these attributes to set the height and width of the image input type. When you set these attributes, the page's spatial area for the image is fixed by the constraints imposed by the preset width and height dimensions, which facilitates the page loading by enhancing the efficacy of the page rendering.

Listing 8. The Submit button

```
<input type="image" src="submitbutton.png" alt="Submit" width="100" height="45" />
</form>
```

JavaScript and CSS3

No HTML5 rendition can be accomplished without CSS3. And, although HTML5 has negated the need for some JavaScript coding, JavaScript is still a valuable tool. Following are the JavaScript code and CSS3 file used to create the example form.

The JavaScript code is a simple alert box that returns the three required fields, as shown in Listing 9. Although the JavaScript code used here is only one line, it has been placed in a separate JavaScript file, following best practices for its use.

Listing 9. Example form JavaScript code

```
function alertValues()
{
  alert("Customer information: " + "\n      " + fullname.value + "\n      "
        + tel.value + "\n      " + email.value);
}
```

Listing 10 shows the CSS3 code that formats the example form. The <header> and <footer> information is not included here.

Listing 10. Example form CSS3

```
form {
```

```
width: 550px;
margin: 0 0 0 0 ;
padding: 50px 60px;
background-color: #2d2d2d;
border-radius: 20px;
}

fieldset {
padding: 0 20px 20px;
margin: 0 0 30px ;
border: 2px solid #ffffff;
background: #B8B8B8 ;
border-radius: 10px;
}

legend {
color: #ffffff;
background: #990033;
font-size: 0.9em;
font-weight: bold;
text-align: left;
padding: 5px;
margin: 0;
width: 10em;
border: 2px solid #660033;
border-radius: 5px;
}

label {
display: block;
float: left;
clear: left;
text-align: left;
width: 100%;
padding: .4em 0 0 0;
margin: .15em 0 0 0;
}
```

Conclusion

The key to tangible individual and agency success is communication. Form controls and general page construction guidelines are critical to this process, and HTML5 has provided a quiver of exceptionally powerful tools appropriate for the task. Those who are prepared for the future—which is here now—will benefit; those who are not will be severely buffeted by the pace and demands of the one-Web-world, netcentric, global society.

Downloads

Description	Name	Size	Download method
Example HTML, CSS3 and JavaScript files	HTML5Forms.zip	10KB	HTTP

[Information about download methods](#)

Resources

Learn

- [Create modern Web sites using HTML5 and CSS3](#) (developerWorks, March 2010) is a multicomponent HTML5 and CSS3 tutorial.
- In [New elements in HTML 5](#) (developerWorks, August 2007), you will find information for several of the new elements in HTML5.
- The [<html>5doctor](#) website provides an excellent view of the current trends in HTML5.
- The [W3Schools.com: HTML5 Tag Reference](#) provides an extensive list of the HTML5 tags, definitions, and examples.
- The [WHATWG website](#) provides detailed information for the HTML5 specification.
- The [Web development zone](#) specializes in articles covering various Web-based solutions.
- Stay current with developerWorks' [technical events and webcasts](#).
- [developerWorks on-demand demos](#): Watch demos ranging from product installation and setup for beginners to advanced functionality for experienced developers.
- Follow [developerWorks on Twitter](#).

Discuss

- Create your [developerWorks profile](#) today and [set up a watchlist](#) on HTML. Get connected and stay connected with [developerWorks community](#).
- Find other [developerWorks members interested in web development](#).
- Share what you know: [Join one of our developerWorks groups focused on web topics](#).
- Roland Barcia talks about [Web 2.0 and middleware](#) in his blog.
- Follow developerWorks' members' [shared bookmarks on web topics](#).
- Get answers quickly: Visit the [Web 2.0 Apps forum](#).
- Get answers quickly: Visit the [Ajax forum](#).

About the author

Grace Walker

Grace Walker, a partner in Walker Automated Services in Chicago, Illinois, is an IT consultant with a diverse background and broad experience. She has worked in IT as a manager, administrator, programmer, instructor, business analyst, technical analyst, systems analyst, and Web developer in various environments, including telecommunications, education, financial services, and software.

HTML5 fundamentals, Part 3: The power of HTML5 APIs

Managing communication

Skill Level: Intermediate

[Grace Walker \(gwalker@walkerautomated.com\)](mailto:gwalker@walkerautomated.com)
IT Consultant
Walker Automated Services

07 Jun 2011

HTML5 reflects the monumental changes in the way you now do business on the web and in the cloud. This article is the third in a four-part series designed to spotlight changes in HTML5, beginning with the new tags and page organization and providing high-level information on web page design, the creation of forms, the use and value of the APIs, and, finally, the creative possibilities that Canvas provides. This installment introduces HTML5 APIs, using an example page to demonstrate functions.

So, what is an API?

An *application programming interface* is a collection of programming instructions and standards for accessing a software application. With an API, you can design products powered by the service the API provides.

HTML5 has several new APIs. For example:

- A 2D drawing API used with the new canvas element for rendering graphs or other visual images
- An API caching mechanism that supports offline web applications
- An API for playing video and audio used with the new video and audio

elements

- A history API that makes the browsing history accessible and allows pages to add to it
- A drag-and-drop API to use with the `draggable` attribute
- An editing API to use with the `contenteditable` attribute
- Client-side storage with JavaScript APIs for key-value pairs and also embedded SQL databases

This article concentrates on two APIs: Geolocation and Web Worker. First, it analyzes the APIs themselves; then, you create a page that contains both the APIs.

Business is everywhere: Geolocation

You use the Geolocation API to determine and share geographical positions. The API returns longitude and latitude coordinates—information that businesses can use to offer services in the area approximate to the coordinates. These services are generally referred to as *location-based services* (LBS).

LBS refers to the geographical data sources used to identify the physical location of the instrument being monitored and, thereby, the human associated with that location. This function gives interested parties the opportunity to interact with that individual based on the market for some geolocation-centric point of interest.

Business is really about the creation of quality, utility, and value for customers, while at the same time creating economic and financial benefits for stakeholders, creditors, stockholders, employees, and vendors. The Geolocation-powered LBS makes it quite easy to track or monitor a package or a person using a non-browser device or a browser. Commercially, geolocation is all about the use of geographical assets to determine where someone or something is located, and then selling that specific set of information to anyone who wants to use it for social, commercial, or other purposes, provided there is legal permission from the owner of the information to do so.

How geolocation works

The Geolocation API is based on a new property of the global `navigator` object: `navigator.geolocation`. The JavaScript `navigator` object provides useful information about the visitor's browser and system. Geolocation can determine latitude and longitude using IP addresses, web-based databases, wireless network connections, and triangulation or GPS technology. It should be noted that the accuracy of Geolocation-provided information varies based on the means of obtaining the information. On occasion, and in some locations, you may not be able to get a clear geolocation reading or any data at all.

Scripts can employ the `navigator.geolocation` object to determine location information related to the user's hosting device. After the location information is retrieved, a position object is created and populated with the data.

The `navigator.geolocation` object has three methods:

- `getCurrentPosition()`
- `watchPosition()`
- `clearWatch()`

The `getCurrentPosition()` method

The `getCurrentPosition()` method retrieves the user's current location, but only once. When it is called by a script, the method asynchronously attempts to obtain the current location of the hosting device. *Asynchronous communication* means that the sender and receiver are not concurrently engaged in communication. Using asynchronous communication lets the browser continue with other activities so that it doesn't have to wait for a response from the receiving entity.

The `getCurrentPosition()` method can have up to three arguments:

- **`geolocationSuccess`**. The callback with the current position (required)
- **`geolocationError`**. The callback if there was an error (optional)
- **`geolocationOptions`**. The geolocation options (optional)

The `navigator.geolocation.getCurrentPosition()` method returns the host device's current position to the `geolocationSuccess` callback with a `Position` object as the parameter. If there is an error, the `geolocationError` callback is invoked with a `PositionError` object. You can set three properties for `geolocationOptions`: `enableHighAccuracy`, `timeout`, and `maximumAge`. These optional properties enable high accuracy if the device supports it, a timeout period by which a position should have been returned, and a maximum amount of time that a cached location can be used, respectively.

The `getCurrentPosition()` method is called as shown here:

```
void navigator.geolocation.getCurrentPosition(  
    geolocationSuccess, geolocationError, geolocationOptions);
```

The `watchPosition()` method

The `watchPosition()` method polls the user location on a regular basis, watching to see whether the user location has changed. It can have up to three arguments.

When `watchPosition` is called, it asynchronously starts a watch process involving the acquisition of a new `Position` object and creation of a `watchID`. If this acquisition is successful, the associated `geolocationSuccess` with a `Position` object as an argument is invoked. Upon a failure involving an invoked method with a non-null `geolocationError` argument, the method generates the `geolocationError` with a `PositionError` object as an argument. When the device position changes, a suitable callback with new `Position` object is invoked.

The `watchPosition()` method is called as shown here:

```
long navigator.geolocation.watchPosition(  
    geolocationSuccess, geolocationError, geolocationOptions);
```

The `clearWatch()` method

The `clearWatch()` method terminates an ongoing `watchPosition()`. This method can have only one argument. When called, it finds the `watchID` argument that was previously started and immediately stops it.

The `clearWatch()` method is called as shown here:

```
void navigator.geolocation.clearWatch(watchID)
```

Geolocation data: The `Position` object

The Geolocation API returns a geographical `Position` object. This object has two properties: `timestamp` and `coords`. The `timestamp` property indicates the time of the geolocation data's creation. The `coords` property has seven attributes:

- **`coords.latitude`**. The estimated latitude
- **`coords.longitude`**. The estimated longitude
- **`coords.altitude`**. The estimated altitude
- **`coords.accuracy`**. The accuracy of the provided latitude and longitude estimates in meters
- **`coords.altitudeAccuracy`**. The accuracy of the provided altitude estimate in meters
- **`coords.heading`**. The current direction of movement for the hosting device in degrees, counting clockwise relative to true north
- **`coords.speed`**. The device's current ground speed in meters per second

Only three of the properties are guaranteed to be there: `coords.latitude`,

`coords.longitude`, and `coords.accuracy`. The rest return `null`, depending on the capabilities of your device and the back-end positioning server that it talks to. The `heading` and `speed` properties are calculated based on the user's previous position, if possible.

Web workers to the rescue

Web workers remedy the problems caused by concurrency. Web workers are the HTML5 family's answer to the JavaScript single-thread problem: They run processes on a separate thread from the main page, preserving the page for the main functions, such as maintaining a stable UI.

A web worker is a JavaScript file that is loaded and executed in the background. These workers allow you to load a JavaScript file dynamically, and then execute a script using a background process that does not affect the UI. Web workers have limited access and are only allowed to pass strings. Because web workers don't use the browser UI thread, they are not permitted access to the DOM. Workers can use both `self` and `this` references for the worker's global scope. Worker and parent page communication is achieved using an event model and the `postMessage()` method.

Because web workers have a multithreaded behavior, they can only access a subset of JavaScript's features. Web workers can:

- Access the navigator object
- Use the read-only location object
- Execute `XMLHttpRequest` to send HTTP or HTTPS requests
- Set a time or interval for an activity using `setTimeout()`/`clearTimeout()` and `setInterval()`/`clearInterval()`
- Access the application cache
- Import external scripts using the `importScripts()` method
- Spawn other web workers (The child (subworker) must have the same origin as the main page and be placed in the same location as the parent worker.)

There are two types of web workers: dedicated workers and shared workers.

Dedicated web worker

A dedicated worker is linked to the script that created it, and it can communicate with

other workers or browser components. However, it cannot communicate with the DOM.

A dedicated worker is created by passing a JavaScript file name to a new worker instance. You create a new worker using the `Worker()` constructor by specifying the worker's executing script URI. To create a dedicated worker, enter the code shown here, which creates a new dedicated `Worker` object:

```
var worker = new Worker('worker.js');
```

Shared web workers

Shared web workers, like dedicated workers, cannot access the DOM and have only limited access to window properties. Shared web workers can only communicate with other shared web workers from the same domain. The workers are created by passing a JavaScript name to a new shared worker instance.

Page scripts can communicate with shared web workers. However, unlike dedicated web workers, you communicate by using a `port` object and attaching a message event handler. In addition, you must call the port's `start()` method before using the first `postMessage()`.

Upon receipt of the first message by the web worker script, the shared web worker attaches an event handler to the active port. Generally, the handler will run its own `postMessage()` method to return a message to the calling code, and then the port's `start()` method generates an enable message process.

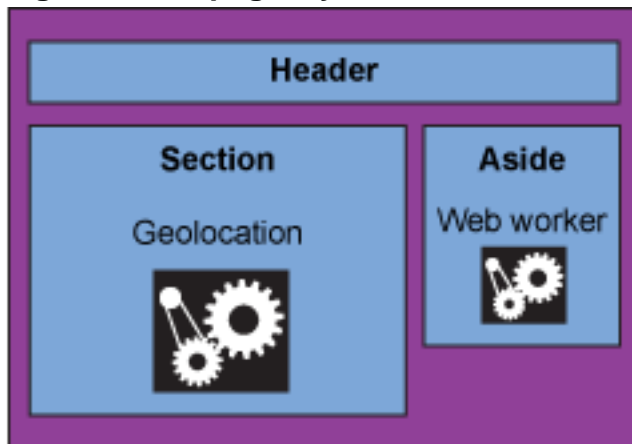
To create a shared web worker, you must create a `SharedWorker` object instead of the `Worker` object. The following code shows how a new `SharedWorker` object is created:

```
var worker = new SharedWorker('worker.js');
```

Constructing a page including the two APIs

You will design a page that contains basic working models of the Geolocation and Web Worker APIs. In addition, you use the Google Map API to render the data gathered as a map.

The page is organized as shown in Figure 1. It contains a Header area created using the `<header></header>` tags, a Section area created using the `<section></section>` tags, and an Aside area created using the `<aside></aside>` tags.

Figure 1. API page layout

The `<section>` and `<aside>` areas contain the APIs. The Section area contains the Geolocation API. The Aside area contains the web worker, which calculates prime numbers.

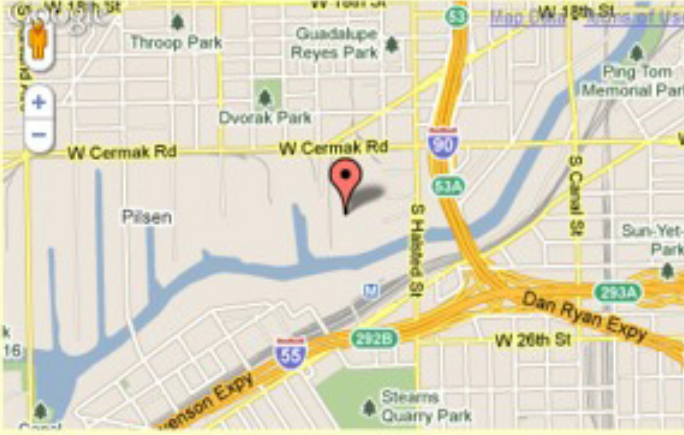
When executed, the web page is displayed as shown in Figure 2. To view the geolocation data, you must first agree to share your information. The web worker starts when the page loads. If you want to see the prime numbers found, click **Display Web Worker**.

Figure 2. The API web page

Geolocation & Web Worker

Making it work

This is the Geolocation example map.



This is the Web Worker.

Prime number calculation result:

Display Web Worker

Stop Web Worker

This is the output from the navigator.geolocation object.

```

accuracy:      24000
altitude:      undefined
altitudeAccuracy: 0
heading:       NaN
latitude:      41.850033
longitude:     -87.650052
speed:         NaN
  
```

The HTML file

The HTML file begins with the standard HTML5 information shown in Listing 1. The `<head>` section contains a call to the Google Maps API, setting the value of `sensor` to `False`. Using the Google Maps API requires that you state whether your application is using a sensor, such as a GPS, to establish location. You must declare a sensor parameter value of `True` or `False` for your Google Maps API application. A *sensor value must be declared*. The `<head>` tag also contains links to the JavaScript and CSS3 files used to handle the functions and format the web page.

Listing 1. HTML file beginning

```

<!doctype html>
<html>
<head>
  <title>Basic GeoLocation Map & Web Worker Prime Number Calculator</title>
  <script src="http://maps.google.com/maps/api/js?sensor=false"
    type="text/javascript"></script>
  <LINK href="GeolocationWebWorker.css" rel="stylesheet" type="text/css">
  <script src="HTML-Part3-GeolocationWebWorker.js" type="text/javascript"></script>

```

```
</head>
```

The `<body>` tag contains an `onLoad` event that calls the initialization function for geolocation, as shown in Listing 2. This function verifies that geolocation can be used in this browser. The initialization function is in the JavaScript file. If the browser can communicate with the Geolocation API, the map will be rendered.

Listing 2. Initialize Geolocation

```
<body onLoad="initGeoApp();">
  <header>
    <hgroup>
      <h1>Geolocation & Web Worker</h1>
      <h2>Making it work</h2>
    </hgroup>
  </header>
```

The `<section>` tag shown in Listing 3 contains the display output information for the `navigator.geolocation` object. A map canvas is created using the longitude and latitude that the API returns. The `Position coords` data is also displayed using the `` tags.

Listing 3. Geolocation map and position

```
<section>
  <p>This is the geolocation example map.</p>
  <div id="map_canvas" ></div>

  <p>This is the output from the navigator.geolocation object.</p>
  <table>
    <tr>
      <td>accuracy:</td>
      <td><span id="accuracyOutput"></span></td>
    </tr>
    <tr>
      <td>altitude:</td>
      <td><span id="altitudeOutput"></span></td>
    </tr>
    <tr>
      <td>altitudeAccuracy:</td>
      <td><span id="altitudeAccuracyOutput"></span></td>
    </tr>
    <tr>
      <td>heading:</td>
      <td><span id="headingOutput"></span></td>
    </tr>
    <tr>
      <td>latitude:</td>
      <td><span id="latitudeOutput"></span></td>
    </tr>
    <tr>
      <td>longitude:</td>
      <td><span id="longitudeOutput"></span></td>
    </tr>
    <tr>
      <td>speed:</td>
      <td><span id="speedOutput"></span></td>
    </tr>
```

```
</table>
</section>
<aside>
  <p>This is the Web Worker. </p>
  <p>Prime number calculation result:
  <output id="result"></output></p>
```

The Web Worker calculates prime numbers. You use the new `<output>` tag to display the calculation that the web worker provides. The ID assigned in the `<output>` tag is the same ID JavaScript uses to identify the calculation it performs. The IDs used in the `` and `<output>` tags makes them accessible to the DOM. Without the reference ID, JavaScript will not know which `` or `<output>` to use. Listing 4 shows the output from the web worker.

Listing 4. Web worker output

```
<aside>
  <p>This is the Web Worker. </p>
  <p>Prime number calculation result:
  <output id="result"></output></p>
```

The `onClick` is used in the `<input>` tag to first display the values being calculated by the Prime Number web worker, and then the second `onClick` is used to stop the web worker. Listing 5 shows the code. The `displayWorker()` function causes the web worker's calculations to be displayed when the button is clicked. The web worker began calculating the prime numbers when the page was loaded.

Listing 5. Inputs for the web worker

```
  <input type="button" value="Display Web Worker" onClick="displayWorker();">
  <input type="button" value="Stop Web Worker"   onClick="stopWorker();">

</aside>
</body>
</html>
```

The JavaScript file

JavaScript is the engine behind the APIs exhibited on the example page. The Geolocation API is initialized with the `initGeoApp()` function. This is the function executed by the `onLoad()` event in the `<body>` tag: It determines whether your browser can use geolocation (see Listing 6). If your browser can use geolocation, then the Geolocation API is called. If successful, a map is drawn using the `Position` attributes. The values of the attributes are then printed below the map.

Listing 6. Geolocation functions

```
function initGeoApp()
{
```

```
if( navigator.geolocation )
{
    navigator.geolocation.getCurrentPosition( success, failure);
}
else
{
    alert("Your browser does not support geolocation services.");
}
```

The values are retrieved using `document.getElementById`, based on the ID that you supplied in the HTML file. `document.getElementById` is a method of the document object and should be accessed by using `document.getElementById`, as shown in Listing 7. The values of the `Position` attributes are stored here so that they can be used to print the attributes below the map to be rendered.

Listing 7. Use `getElementById` to get coords values

```
var map;
function success(position)
{
    document.getElementById("accuracyOutput").innerHTML =
        position.coords.accuracy;
    document.getElementById("altitudeOutput").innerHTML =
        position.coords.altitude;
    document.getElementById("altitudeAccuracyOutput").innerHTML =
        position.coords.altitudeAccuracy;
    document.getElementById("headingOutput").innerHTML =
        position.coords.heading;
    document.getElementById("latitudeOutput").innerHTML =
        position.coords.latitude;
    document.getElementById("longitudeOutput").innerHTML =
        position.coords.longitude;
    document.getElementById("speedOutput").innerHTML =
        position.coords.speed;
```

This section defines the coordinates for the Google Map API's `LatLng` object, as Listing 8 shows. The Google Map API `LatLng` object provides the coordinate information required to create a map. You can set the zoom level and several other options that create the look of the map presented to the user.

Listing 8. Google Map options

```
var coordinates = new google.maps.LatLng(position.coords.latitude,
    position.coords.longitude);

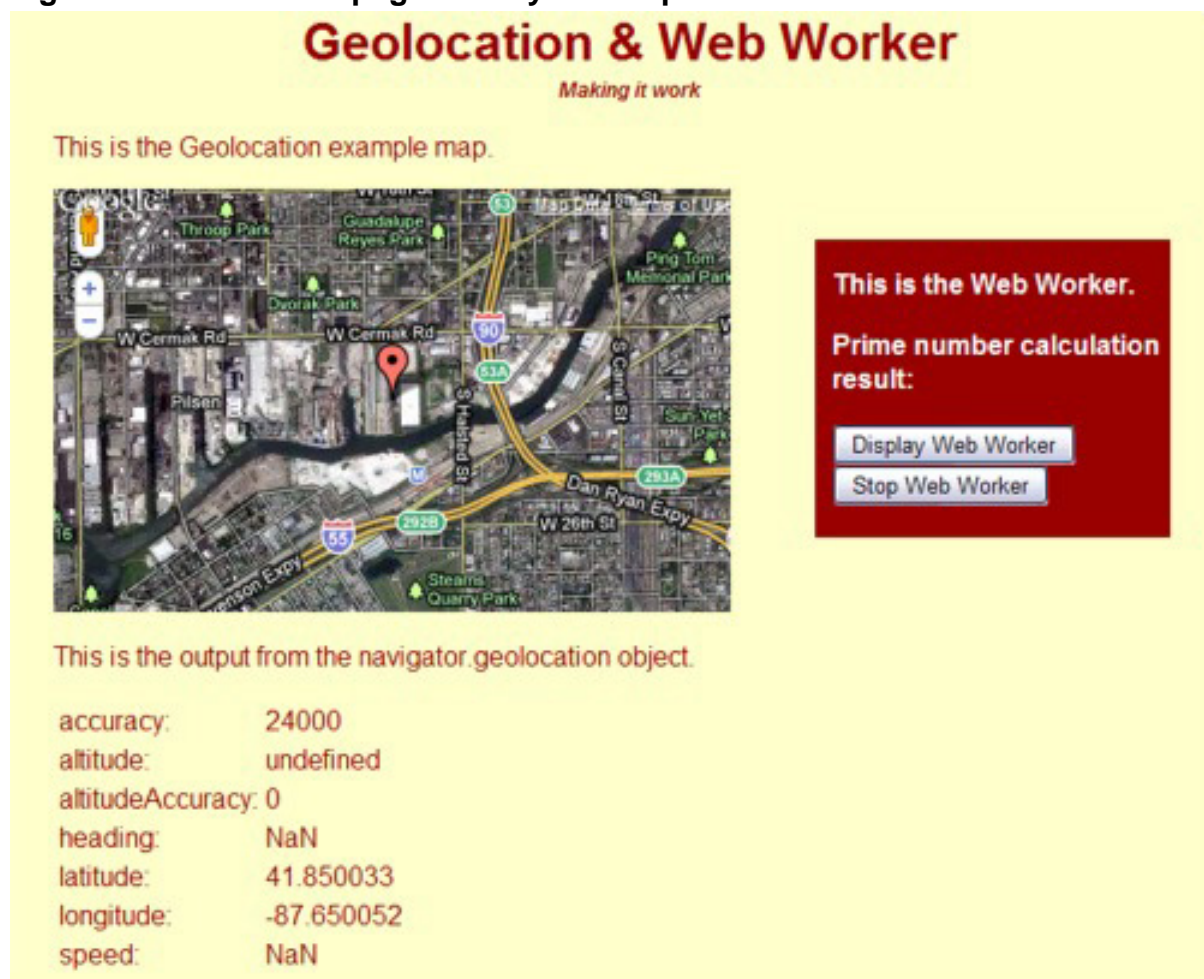
var myOptions =
{
    zoom: 14,
    center: coordinates,
    mapTypeControl: false,
    navigationControlOptions: {style: google.maps.NavigationControlStyle.small},
    mapTypeId: google.maps.MapTypeId.ROADMAP
};
```

Note that in the `mapTypeID` option, the option selected is the `ROADMAP`. This value presents the map so that it appears as shown in Figure 2. There are four possible values:

- `ROADMAP`
- `HYBRID`
- `SATELLITE`
- `TERRAIN`

Figure 3 shows how the page would appear with the **HYBRID** option selected.

Figure 3. The API web page with hybrid map



Create the map using the ID `map_canvas`, which is the ID for the `<div>` in the HTML file:

```
map = new google.maps.Map(document.getElementById("map_canvas"), myOptions);
```

Place an initial position marker on the map. Listing 9 shows the code.

Listing 9. Place an initial map marker

```
var marker = new google.maps.Marker({
  position: coordinates,
  map: map,
  title: "You are here."
});

function failure()
{
  alert("Sorry, could not obtain location");
}
```

The web worker begins executing when the page is initialized. If the user wants to display the output of the calculations being performed, he or she can click **Display Web Worker**, which will call the `displayWorker()` function. Listing 10 shows the code.

Listing 10. The web worker

```
var worker = new Worker('PrimeNumberWebWorker.js');

function displayWorker()
{
  worker.onmessage = function (event)
  {
    document.getElementById('result').innerHTML = event.data;
  };
}
```

If the user wants to stop the web worker, he or she can click **Stop Web Worker**, which will call the `stopWorker()` function shown in Listing 11.

Listing 11. Terminate worker

```
function stopWorker()
{
  worker.terminate();
}
```

The web worker file

This file is the prime number calculator web worker: It calculates every prime number until it is stopped. Listing 12 shows the code.

Listing 12. Calculate prime numbers


```
var n = 1;
search: while (true) {
  n += 1;
  for (var i = 2; i <= Math.sqrt(n); i += 1)
    if (n % i == 0)
      continue search;
  postMessage(n);
}
```

The CSS3 file

The CSS3 file shown in Listing 13 provides the formatting displayed in the HTML5 page.

Listing 13. CSS3 descriptions

```
* {font-family: Arial,Helvetica,sans-serif ;
}

body {
  margin: 0 300px 0 300px;
  color: #990000;
  background-color:#FFFFCC;
}

header > hgroup h1 {
  margin: 0 0 3px 0;
  padding: 0;
  text-align: center;
  font-size: 30px;
}

header > hgroup h2 {
  margin: 0 0 15px 0;
  padding: 0;
  text-align: center;
  font-style: italic;
  font-size: 12px;
}

header p {
  margin: 0 0 20px 0 ;
  padding: 0;
  text-align: center;
  font-size: 12px;
}

aside {
  width: 200px;
  height: 175px;
  margin: -450px 0 0 450px;
  background-color: #990000;
  padding: .5px 0 0 10px ;
  color:#FFFFFF;
  font-weight:bold;
}

div {
  width: 400px;
```



```
    height: 250px;  
}
```

Conclusion

This installment examined the utility of the Geolocation and Web Worker APIs. These two APIs were selected because together they demonstrate both the innovative and the practical use of APIs. Geolocation is an excellent example of the HTML5 specification's use in the creation of new business models. Likewise, the Web Worker's role is the resolution of the problems inherent in JavaScript's concurrency problem.

These two APIs together illustrate a model combination of the use of HTML5 for commercial and social use. Thus, their utility demonstrates the proper facilitation and general management of an HTML5 rich Internet application.

Downloads

Description	Name	Size	Download method
Example HTML, CSS3 and JavaScript files	HTML5APIs.zip	10KB	HTTP

[Information about download methods](#)

Resources

Learn

- The [Web Workers specification](#) published by the WHATWG provides detailed worker information.
- The W3C specification for geolocation, [Geolocation API Level 2 Specification](#), provides a thorough use case and requirements section that is useful in formulating commercial and social applications of the API.
- [Build Web applications with HTML 5](#) (developerWorks, March 2010) discusses several of the new features introduced with HTML5.
- [Creating mobile Web applications with HTML 5, Part 1: Combine HTML 5, geolocation APIs, and Web services to create mobile mashups](#) (developerWorks, May 2010) provides valuable insight into the development of mobile applications.
- [Create modern Websites using HTML5 and CSS3](#) (developerWorks, March 2010) is a multicomponent HTML5 and CSS3 tutorial.
- In [New elements in HTML 5](#) (developerWorks, August 2007), you will find information for several of the new elements in HTML5.
- The [<html>5doctor](#) website provides an excellent view of the current trends in HTML5.
- The [W3Schools.com: HTML5 Tag Reference](#) provides an extensive list of the HTML5 tags, definitions, and examples.
- The [WHATWG website](#) provides detailed information for the HTML5 specification.
- The [Web development zone](#) specializes in articles covering various web-based solutions.
- Stay current with developerWorks' [technical events and webcasts](#).
- [developerWorks on-demand demos](#): Watch demos ranging from product installation and setup for beginners to advanced functionality for experienced developers.
- Follow [developerWorks on Twitter](#).

Get products and technologies

- The [Dojo Toolkit](#), an open source modular JavaScript library, helps you quickly develop cross-platform, JavaScript/Ajax-based applications and websites.

Discuss

- Create your [developerWorks profile](#) today and [set up a watchlist](#) on HTML5. Get connected and stay connected with [developerWorks community](#).
- Find other [developerWorks members interested in web development](#).
- Share what you know: [Join one of our developerWorks groups focused on web topics](#).
- Roland Barcia talks about [Web 2.0 and middleware](#) in his blog.
- Follow developerWorks' members' [shared bookmarks on web topics](#).
- Get answers quickly: Visit the [Web 2.0 Apps forum](#).
- Get answers quickly: Visit the [Ajax forum](#).

About the author

Grace Walker

Grace Walker, a partner in Walker Automated Services in Chicago, Illinois, is an IT consultant with a diverse background and broad experience. She has worked in IT as a manager, administrator, programmer, instructor, business analyst, technical analyst, systems analyst, and Web developer in various environments, including telecommunications, education, financial services, and software.

HTML5 fundamentals, Part 4: The final touch

The canvas

Skill Level: Intermediate

[Grace Walker \(gwalker@walkerautomated.com\)](mailto:gwalker@walkerautomated.com)

IT Consultant

Walker Automated Services

05 Jul 2011

HTML5 reflects the monumental changes in the way we now do business on the web and in the cloud. This article, the fourth in a four-part series designed to spotlight changes in HTML5, introduces the HTML5 Canvas element, using several examples to demonstrate functions.

The role of the HTML5 author, an intrepid combination of designer and developer, is to construct effective rich Internet applications (RIA) and especially rich UIs. By *effective*, I mean the creation of systemic and systematic enhancements that digitally facilitate the dialog among the site owner, agents of the owner, and the site's users.

RIAs are the source and vehicle of a satisfying user experience and, therefore, are an essential part of any successful net-centric venture. Net-centric activities, by nature, are collaborative to one degree or another. A winning approach to digital collaboration is essential for agency success at all levels, including marketing and management. A lot rides on the efficiency with which the site satisfies the quality expectations of its visitors.

HTML5, as you have seen, is tailor-made for the collaborative "one web world" of cross-platform capabilities, converging telecommunications, unified language, ubiquitous computing, and open systems. The first three installments of this series focused on semantics, the proper coding methods, the role that input plays in the vital conversion process, and best practices of site management, all of which was designed to lay the foundation for the creation of RIAs in an organized and logical manner. The common thread in each article has been that the production and management of a rich user experience is critical to achieving the agency objectives

of the owners of the website. The HTML5 Canvas has a critical role to play in the development of effective RIAs.

Frequently used acronyms

- **2D:** Two-dimensional
- **Ajax:** Asynchronous JavaScript + XML
- **API:** Application programming interface
- **HTML:** Hypertext Markup Language
- **HTML5:** HTML version 5
- **UI:** User interface

What is the Canvas?

The HTML5 Canvas is an extremely useful drawing and animation element. Canvas uses JavaScript to draw graphics directly on the page. It is a rectangular area that you define and control and that permits dynamic, scriptable rendering of 2D shapes and bitmap images.

The HTML5 Canvas is perfect for producing great visual material that enhances UIs, diagrams, photo albums, charts, graphs, animations, and embedded drawing applications. The Canvas element has several methods for drawing paths, rectangles, circles, and characters.

Canvas coordinates

A prerequisite to drawing on the canvas is familiarity with the grid or coordinate space. The spatial area measurements for the width and height are given in pixels. The canvas is built around the use of the x and y coordinates. The canvas coordinates at $x=0$, $y=0$ are in the upper-left corner.

The default attributes for the canvas rectangle area is 300 pixels wide and 150 pixels high, but you can determine the exact size of the canvas element by specifying width and height. The diagram in Figure 1 shows how the x and y coordinates are implemented.

Figure 1. Canvas coordinates

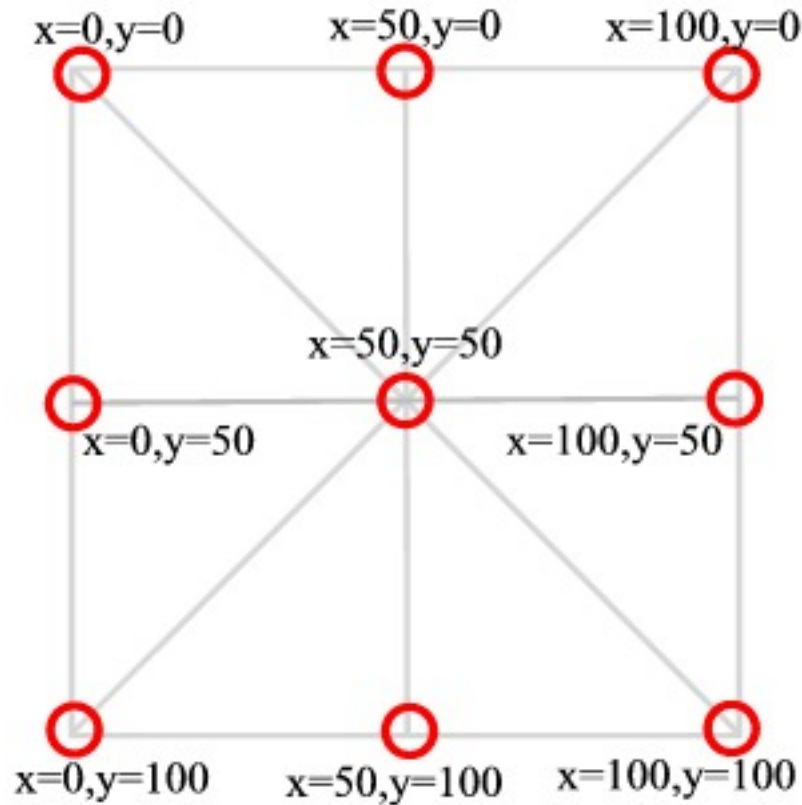


Figure 1 shows a canvas area that is 100 pixels by 100 pixels:

- The top left corner is $x=0, y=0$.
- The x value increases horizontally, and the y value increases vertically.
- The bottom right corner is $x=100, y=100$.
- The center point is $x=50, y=50$.

Getting started

To put anything on the canvas, you must first define the canvas in the HTML file. You must create JavaScript code that accesses the `<canvas>` tag and communicates with the HTML5 Canvas API to draw your image.

The basic structure of the `<canvas>` tag is:

```
<canvas id="myCanvas" width="200" height="200"></canvas>
```

A `canvas` element has two attributes of its own: `width` and `height`. In addition, Canvas possesses all the key HTML5 attributes, such as `class`, `id`, and `name`. The

`id` attribute is used in the code shown above. JavaScript uses the canvas `id` created here to identify the canvas to paint on. JavaScript determines the appropriate canvas using the `document.getElementById()` method, as shown here:

```
var canvas = document.getElementById("myCanvas");
```

Every canvas must have a context definition, as shown below. Currently, the official specification recognizes only a 2D environment:

```
var context = canvas.getContext("2d");
```

After you identify the canvas and specify its context, you're ready to begin drawing.

Drawing tools, effects, and transformations

This discussion of the HTML5 Canvas goes over various drawing tools, effects, and transformations. The drawing tools include:

- Lines
- Rectangles
- Arcs
- Bezier and quadratic curves
- Circles and semicircles

The Canvas effects you will use are:

- Fills and strokes
- Linear and radial gradients

The transformations discussed include:

- Scaling
- Rotation
- Translation

Drawing lines

To draw a line on the canvas, use the `moveTo()`, `lineTo()`, and `stroke()`

methods. In addition, you use the `beginPath()` method to reset the current path:

- `context.beginPath();`
- `Context.moveTo(x,y);`
- `Context.lineTo(x,y);`
- `Context.stroke(x,y);`

The `beginPath()` method starts a new path. Before you draw a new line with different sub-paths, you must use `beginPath()` to indicate that a new starting point for a drawing is to follow. The `beginPath()` method does not have to be called when you draw the first line.

The `moveTo()` method states where the new sub-path is to start. The `lineTo()` method creates sub-paths. You can change the appearance of the line with `lineWidth` and `strokeStyle`. The `lineWidth` element changes the thickness of the line, and `strokeStyle` changes the color.

In Figure 2, three lines are drawn in blue, green, and purple, respectively.

Figure 2. Canvas with lines in three different colors



The lines in Figure 2 were created by the code in [Listing 1](#). The blue line with round ends is created by first establishing that a new path is to begin: `context.beginPath()`. It is followed by:

- `context.moveTo(50, 50)`, which places the starting point for the path at (x=50, y=50)
- `context.lineTo(300, 50)`, which identifies the end point for the line

- `context.lineWidth = 10`, which is the width of the line
- `context.strokeStyle = "#0000FF"`, which is the color of the line
- `context.lineCap = "round"`, which makes the ends round
- `context.stroke()`, which actually paints the line on the canvas

All the lines are 50 pixels in length, though they appear to be different lengths—a visual illusion caused by the line caps. There are three possible line caps:

- `Context.round` (blue)
- `Context.square` (green)
- `Context.butt` (purple—the default)

The butt cap is the default value. When you use a round or square cap style the length of the line increases by an amount equal to the width of the line. For example, a line that is 200 pixels long and 10 pixels wide with a round or square cap style will have a resulting line length of 210 pixels, because each cap will add 5 pixels to each end of the line. A line that is 200 pixels long and 20 pixels wide with a round or square cap style will have a resulting line length of 220 pixels, because each cap will add 10 pixels to each end of the line.

Execute and alter the code in Listing 1 to gain a greater understanding of how lines are drawn.

Listing 1. Create three lines of different colors on the canvas

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Line Example</title>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }

      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>

      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        // blue line with round ends
        context.beginPath();
        context.moveTo(50, 50);
        context.lineTo(300,50);
        context.lineWidth = 10;
        context.strokeStyle = "#0000FF";
```

```
        context.lineCap = "round";
        context.stroke();

        // green line with square ends
        context.beginPath();
        context.moveTo(50, 100);
        context.lineTo(300,100);
        context.lineWidth = 20;
        context.strokeStyle = "#00FF00";
        context.lineCap = "square";
        context.stroke();

        // purple line with butt ends
        context.beginPath();
        context.moveTo(50, 150);
        context.lineTo(300, 150);
        context.lineWidth = 30;
        context.strokeStyle = "#FF00FF";
        context.lineCap = "butt";
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="400" height="200">
    </canvas>

</body>
</html>
```

Drawing rectangles

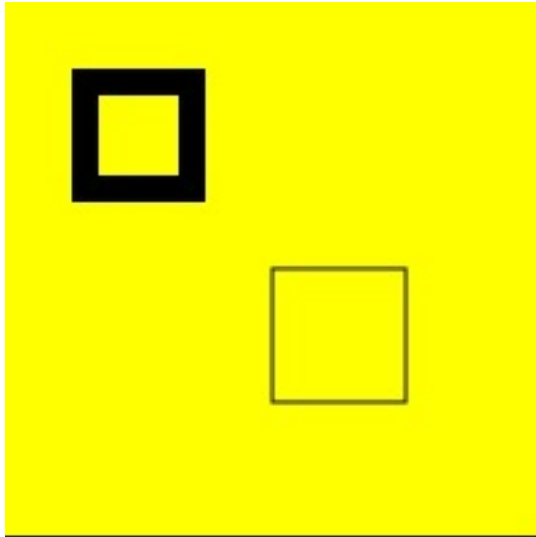
There are three methods for addressing a rectangular area on the canvas:

- `fillRect(x,y,width,height)`, which draws a filled rectangle
- `strokeRect(x,y,width,height)`, which draws a rectangular outline
- `clearRect(x,y,width,height)`, which clears the given area and makes it fully transparent

For each of the three methods, `x` and `y` indicate the position on the canvas relative to the top-left corner of the rectangle (`x=0`, `y=0`), and `width` and `height` are the width and height of the rectangle, respectively.

Figure 3 shows the three rectangular areas created by the code in [Listing 2](#).

Figure 3. Rectangle canvas



The `fillRect()` method creates a rectangle filled in with the default fill color of black. The `clearRect()` method clears a rectangular area in the center of the first rectangle. It is in the center of the rectangle formed by the `fillRect()` method. The `strokeRect` creates a rectangle that has only a visible black border.

Listing 2. Rectangle canvas code

```
<!DOCTYPE HTML>
<html>
<head>
<title>Rectangle Example</title>
<style>
  body {
    margin: 0px;
    padding: 0px;
  }

  #myCanvas {
    border: 1px solid #000000;
    background-color: #ffff00;
  }
</style>
<script type="text/javascript">
function drawShape(){
  var canvas = document.getElementById('myCanvas');

  var context = canvas.getContext('2d');

  context.fillRect(25,25,50,50);
  context.clearRect(35,35,30,30);
  context.strokeRect(100,100,50,50);
}
</script>
</head>
<body onload="drawShape();">
  <canvas id="myCanvas" width="200" height="200"></canvas>
</body>
</html>
```

Drawing arcs, curves, circles, and semicircles

Both the circle and semicircle use the `arc()` method. The `arc()` method takes six arguments:

```
context.arc(centerX, centerY, radius, startingAngle, endingAngle, antiClockwise);
```

The `centerX` and `centerY` arguments are the coordinates of the circle's center. The `radius` is the same as the mathematical equivalent: a straight line from the center to the circumference. The arc created will be part of the defined circle. The `startAngle` and `endAngle` arguments are the starting and ending points of the arc, respectively, in radians. The `anticlockwise` argument is a Boolean value. When the value is `true` the arc is drawn counterclockwise; when it is `false` the arc is drawn in the clockwise direction.

To draw a circle using the `arc()` method, define the starting angle as 0 and the ending angle as $2 * \text{PI}$, as shown here:

```
context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);
```

To draw a semicircle using the `arc()` method, define the ending angle as `startingAngle + PI`, as shown here:

```
context.arc(centerX, centerY, radius, startingAngle, startingAngle + Math.PI, false);
```

Quadratic curve

You create a quadratic curve using the `quadraticCurveTo()` method shown below. Quadratic curves are defined by the context point, a control point, and an ending point. The control point determines the curvature of the line.

```
context.moveTo(x, y);  
context.quadraticCurveTo(controlX, controlY, endX, endY);
```

Bezier curve

Just as with the quadratic curve, a Bezier curve has a start and end point; but unlike the quadratic curve, it has two control points:

```
context.moveTo(x, y);  
context.bezierCurveTo(controlX1, controlY1, controlX2, controlY2, endX, endY);
```

You create a Bezier curve using the `bezierCurveTo()` method. Because the Bezier curve is defined with two control points rather than just one, you can create more complex curvatures.

Figure 4 shows—from left to right—an arc, a quadratic curve, a Bezier curve, a semicircle, and a circle.

Figure 4. Arc, curves, and circles

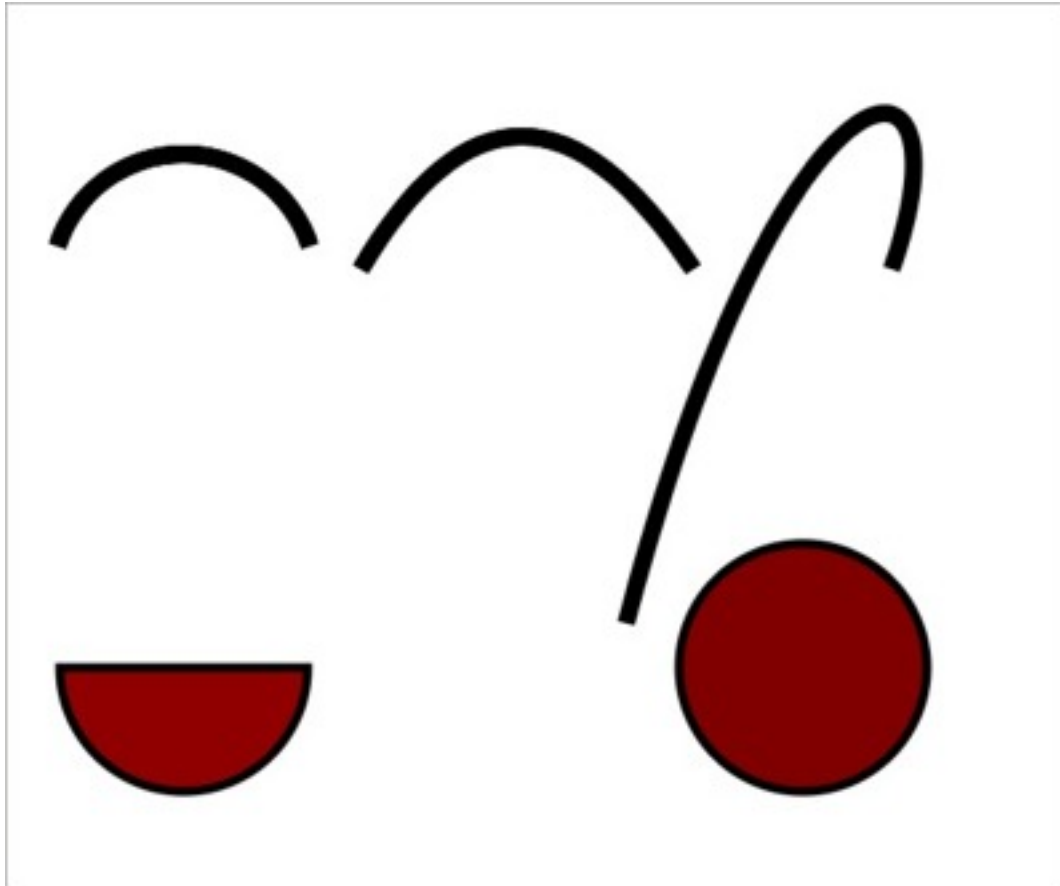


Figure 4 was created using the code in Listing 3.

Listing 3. Arc, curve, and circle code

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Arcs, Curves, Circles, & Semicircles</title>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
  </head>
  <body>
    <img alt="Figure 4: Arc, curves, and circles" data-bbox="117 223 784 642"/>
  </body>
</html>
```

```
    }
    </style>
<script>
function drawArc(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    var centerX = 100;
    var centerY = 160;
    var radius = 75;
    var startingAngle = 1.1 * Math.PI;
    var endingAngle = 1.9 * Math.PI;
    var counterclockwise = false;

    context.arc(centerX, centerY, radius, startingAngle,
        endingAngle, counterclockwise);

    context.lineWidth = 10;
    context.strokeStyle = "black";
    context.stroke();
};

function drawQuadratic(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    context.moveTo(200, 150);

    var controlX = 288;
    var controlY = 0;
    var endX = 388;
    var endY = 150;

    context.quadraticCurveTo(controlX, controlY, endX, endY);
    context.lineWidth = 10;
    context.strokeStyle = "black";
    context.stroke();
};

function drawBezier(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    context.moveTo(350, 350);

    var controlX1 = 440;
    var controlY1 = 10;
    var controlX2 = 550;
    var controlY2 = 10;
    var endX = 500;
    var endY = 150;

    context.bezierCurveTo(controlX1, controlY1, controlX2,
        controlY2, endX, endY);

    context.lineWidth = 10;
    context.strokeStyle = "black";
    context.stroke();
};

function drawCircle(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    var centerX = 450;
    var centerY = 375;
    var radius = 70;

    context.beginPath();
```

```

    context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);

    context.fillStyle = "#800000";
    context.fill();
    context.lineWidth = 5;
    context.strokeStyle = "black";
    context.stroke();
};

function drawSemicircle(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    var centerX = 100;
    var centerY = 375;
    var radius = 70;
    var lineWidth = 5;

    context.beginPath();
    context.arc(centerX, centerY, radius, 0, Math.PI, false);
    context.closePath();

    context.lineWidth = lineWidth;
    context.fillStyle = "#900000";
    context.fill();
    context.strokeStyle = "black";
    context.stroke();
};

window.onload = function (){
drawArc();
drawQuadratic();
drawBezier();
drawCircle();
drawSemicircle()
}
</script>
</head>
<body>
    <canvas id="myCanvas" width="600" height="500">
    </canvas>
</body>
</html>

```

Transformations: translate, scale, and rotate

The `translate()`, `scale()`, and `rotate()` methods all modify the current matrix. The `translate(x, y)` method moves items on the canvas to a different point on the grid. In the `translate(x,y)` method, the `(x,y)` coordinates indicate the number of pixels the image should be moved in the x-direction and the number of pixels the image has to be moved in the y-direction.

If you draw an image at `(15, 25)` with the `drawImage()` method, you can use the `translate()` method with arguments `(20, 30)`, which places the image at position `(15+20, 25+30) = (35, 55)`.

The `scale(x,y)` method changes the size of an image. The `x` argument specifies a horizontal scaling factor, and the `y` argument specifies a vertical scaling factor. For example, `scale(1.5, .75)` would create an image that is 50% larger in the

x-direction and only 75% of the current size in the y-direction. The `rotate(angle)` method turns an object based on the angle specified.

Figure 5 is an example of what can be rendered using the `translate()`, `scale()`, and `rotate()` methods.

Figure 5. Using transformations



Listing 4 provides the code that created the image in Figure 5.

Listing 4. Code to create transformations

```
<!DOCTYPE HTML>
<html>
<head>
<Title>Transformations Example</title>
<script>

window.onload = function() {
  var canvas=document.getElementById("myCanvas");
  var context=canvas.getContext("2d");

  var rectWidth = 250;
  var rectHeight = 75;

  // translate context to center of canvas
  context.translate(canvas.width/2,canvas.height/2);

  // half the y component
  context.scale(1,0.5);

  // rotate 45 degrees clockwise
  context.rotate(-Math.PI/4);

  context.fillStyle="blue";
  context.fillRect(-rectWidth/2,-rectHeight/2,
    rectWidth,rectHeight);

  // flip context horizontally
  context.scale(-1,1);

  context.font="30pt Calibri";
```

```
        context.textAlign="center";
        context.fillStyle="#ffffff";
        context.fillText("Mirror Image",3,10);
    }
</script>
</head>
<body>
    <canvas id="myCanvas" width="400" height="400"></canvas>
</body>
</html>
```

Gradients

A *gradient* is a fill that moves from one color to another, blending the colors where they intersect. There are two types of gradients that you can create in Canvas: linear and radial.

You create a linear gradient using the `createLinearGradient()` method. `createLinearGradient(x0,y0,x1,y1)` produces a gradient along a straight line identified by two points: $(x0,y0)$ and $(x1,y1)$ —the start and end points of the gradient, respectively. The method returns an object.

A color gradient can have many colors. The `addColorStop(offset, color)` method specifies the color stop for the indicated color to the gradient at the given offset. The `addColorStop()` method lets you specify an offset between 0 and 1, where the transition to the next color begins. The value 0 is the offset at one end of the gradient; 1 is the offset at the other end. After the color gradient has been defined, the gradient object can be assigned to `fillStyle()`. You can also draw text with a gradient using the `fillText()` method.

A radial gradient—`createRadialGradient(x0,y0,r0,x1,y1,r1)`—combines two or more colors in a circular or conical pattern using six arguments:

- $(x0,y0)$. The center of the first circle of the cone
- `r0`. The radius of the first circle
- $(x1,y1)$. The center of the second circle of the cone
- `r1`. The radius of the second circle

Figure 6 contains four gradients: a linear gradient, a text gradient, a linear gradient on a diagonal, and a radial gradient.

Figure 6. Gradient example

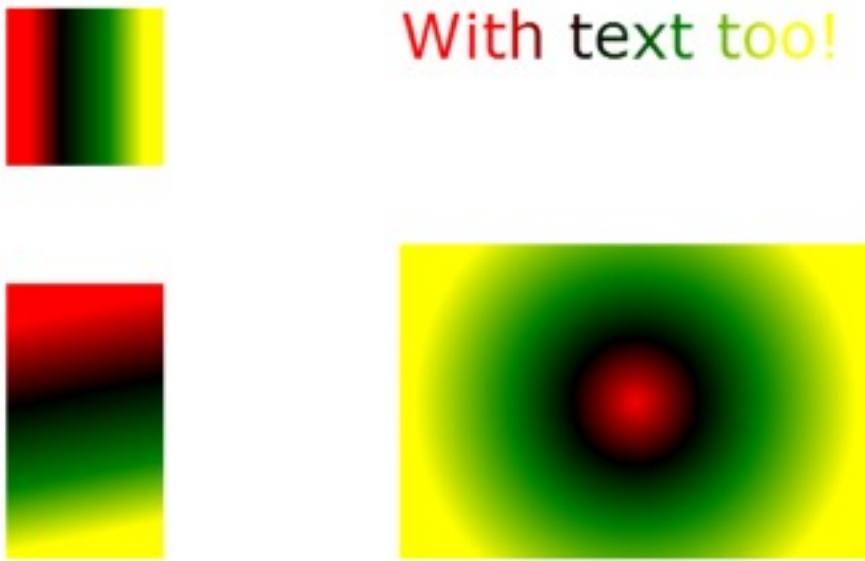


Figure 6 was created using the code in Listing 5.

Listing 5. Gradient example code

```
<!doctype>
<html>
<head>
<title>Gradient Example</title>
<script>
  window.onload = function() {
    var canvas = document.getElementById("myCanvas");

    var context = canvas.getContext("2d");

    //Let's try the gradient on a rectangle

    // Create a linear gradient
    var fillColor = context.createLinearGradient(50,50, 150,50);

    // Set gradient colors
    fillColor.addColorStop(0.15,"red");
    fillColor.addColorStop(0.35,"black");
    fillColor.addColorStop(0.65,"green");
    fillColor.addColorStop(0.87,"yellow");

    // Assign gradient object to fillstyle
    context.fillStyle= fillColor;

    // Draw rectangle
    context.fillRect(50,50,100,100);

    // With text

    var fillColorText = context.createLinearGradient(300,50,600,50);

    fillColorText.addColorStop(0.2,"red");
    fillColorText.addColorStop(0.4,"black");
    fillColorText.addColorStop(0.6,"green");
```

```

        fillColorText.addColorStop(0.8,"yellow");

context.fillStyle= fillColorText;

context.font="40px verdana";
context.textBaseline="top";
context.fillText("With text too!", 300,50)

// Gradient on a diagonal
var fillColordiagonal = context.createLinearGradient(50,200, 100,450);

// Gradient colors
fillColordiagonal.addColorStop(0.2,"red");
fillColordiagonal.addColorStop(0.4,"black");
fillColordiagonal.addColorStop(0.6,"green");
fillColordiagonal.addColorStop(0.75,"yellow");

// Assign gradient object to fillstyle
context.fillStyle= fillColordiagonal;

// Draw rectangle
context.fillRect(50,225, 100,250);

// Draw radial gradient
fillColorRadial = context.createRadialGradient(450,300,0, 450,300,200);
fillColorRadial.addColorStop(0, "red");
fillColorRadial.addColorStop(0.2, "black");
fillColorRadial.addColorStop(0.4, "green");
fillColorRadial.addColorStop(0.7, "yellow");
context.fillStyle = fillColorRadial;
context.rect(300,200,500,400);
context.fill();
}
</script>
</head>
<body>
<div>
<p><canvas id="myCanvas" width="600" height="400"></canvas></p>
</div>
</body>
</html>

```

Image cropping

You can alter images by cropping selected areas out of them. Cropping on the canvas is a function of overloading the `drawImage()` method. The `drawImage()` method has three options. You can use either three, five, or nine arguments.

The three-argument configuration—`drawImage(image, dx, dy)`—draws the image on the canvas at the destination coordinates (`dx, dy`). The coordinates form the upper-left corner of the image.

The-five argument configuration—`drawImage(image, dx, dy, dw, dh)`—provides a width and height for the destination. The image is scaled to fit the destination width and height.

The nine-argument configuration—`drawImage(image, sx, sy, sw, sh, dx,`

dy, dw, dh)—takes an image, clips out a rectangular area that starts at the source (sx, sy) coordinates with a width and height of (sw, sh) , and scales it to fit the destination width and height (dw, dh) , placing it on the canvas at (dx, dy) .

Figure 7 shows the image that you'll crop.

Figure 7. Cropping an image



Using the image shown in Figure 7, a set of images is placed on the canvas. One image is canvas-sized and used as the background. Another image is created that is smaller and inserted at the bottom right of the canvas. A third image is a cut-out of Napoleon's head that you place in the upper-left corner of the canvas. The final cropped image is shown in Figure 8.

Figure 8. The final cropped image



Figure 8 was created using the code in Listing 6. Before executing this code, be sure to [download the Napoleon.png image](#) used in the example.

Listing 6. Code to crop the example image

```
<!doctype>
<html>
<head>
<title>Crop Example</title>
<script type="text/javascript">
  window.onload = function() {
    var canvas=document.getElementById("cropNapolean");
    var context=canvas.getContext("2d");

    var imageObj = new Image();
    imageObj.onload = function() {
      // draw image to cover the entire canvas
      context.drawImage(imageObj,0,0, 600, 400);

      // draw small image in bottom right corner
      var sourceX = 0;
      var sourceY = 0;
      var sourceWidth = 1200;
      var sourceHeight = 801;
      var destX = 300;
      var destY = 200;
      var destWidth = sourceWidth - 900;
      var destHeight = sourceHeight - 600;

      context.drawImage(imageObj, sourceX, sourceY, sourceWidth,
        sourceHeight, destX, destY, destWidth, destHeight);

      //draw Napoleon's head only
      var sourceNapoleanX = 460;
      var sourceNapoleanY = 25;
```

```

        var sourceNapoleanWidth = 250;
        var sourceNapoleanHeight = 175;
        var destNapoleanX = 0;
        var destNapoleanY = 0;
        var destNapoleanWidth = sourceNapoleanWidth - 150 ;
        var destNapoleanHeight = sourceNapoleanHeight - 100;

        context.drawImage(imageObj, sourceNapoleanX, sourceNapoleanY,
            sourceNapoleanWidth, sourceNapoleanHeight,
            destNapoleanX, destNapoleanY,
            destNapoleanWidth, destNapoleanHeight);
    }
    imageObj.src = "Napoleon.png";
}
</script>

</head>
<body>
    <div>
        <p><canvas id="cropNapolean" width="600" height="400"></canvas></p>
    </div>
</body>
</html>

```

Animation and multiple canvases

When working with animation, the question of layers always arises. Layers allow components to be isolated, which makes coding and debugging easier and more efficient. The Canvas API doesn't have layers, but you can create multiple canvases.

Animation must be controlled over time. So, to create an animation, you must address each frame of the animation. The Canvas API has one major limitation when it comes to animation: After a shape is drawn on the canvas, it stays that way. To move the shape, you must redraw it.

To create an animation:

1. Clear the canvas of any shapes that have been drawn previously.
2. Save the canvas state to make sure the original state is used each time a frame is drawn.
3. Perform the steps to render the frames.
4. If you have saved the state, restore it before drawing a new frame.

You can control animation in two ways: by using the `setInterval` or `setTimeout` functions, each of which can be used to call a function over a set time period. The `setInterval` function executes the provided code repeatedly. The `setTimeout` function executes only once after the time provided has elapsed.

Figure 9 shows one frame of the multiple-canvas animation of the swimmer. The

water is on one canvas, and the swimmer is on another.

Figure 9. Animation using images on multiple canvases



You create the swimmer using the code in Listing 7. The swimmer uses a linear gradient to create the water. The water has four shades of blue, which provides a reasonable illusion of water. You create the swimmer's motion using the `positionX` and `positionY` values, which alter the poses of the image. You create the swimmer's head using the `arc()` method. The legs and arms of the swimmer are created by drawing lines and then changing their `lineTo()` positions. You change the torso by altering the `moveTo()` position. Because this is an animation, you will have to execute this code to see how the swimmer moves.

Listing 7. Animation example

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Animation & Multiple Canvas Example</title>
  </head>
  <script>
    // Water canvas
    function drawWater() {
      var canvasWater = document.getElementById("myWaterCanvas");
      var contextWater = canvasWater.getContext("2d");
      contextWater.globalAlpha = .50 ;

      // Create a linear gradient fill
      var linearGrad = contextWater.createLinearGradient(0,0,400,400);
```



```

        linearGrad.addColorStop(0, '#0000ff'); // sets the first color
        linearGrad.addColorStop(.25, '#0099ff'); // sets the second color
        linearGrad.addColorStop(.50, '#00ccff'); // sets the third color
        linearGrad.addColorStop(.75, '#00ffff'); // sets the fourth color
        contextWater.fillStyle = linearGrad;
        contextWater.fillRect(0,0,400,400);
    }

    // Swimmer canvas
    setInterval(drawSwimmer, 30);
    var positionX = 0;
    var positionY = 0;

    function drawSwimmer(){
        var canvasSwimmer = document.getElementById("mySwimmerCanvas");
        var contextSwimmer = canvasSwimmer.getContext("2d");
        contextSwimmer.clearRect(0,0,400,400);

        if (positionX < 30)
        {
            positionX += 1;
            positionY += 1;
        }
        else
        {
            positionX = 0;
            positionY = 0;
        }

        contextSwimmer.save();

        // draw circle for head
        var centerX = 200;
        var centerY = 50;
        var radius = 20;

        contextSwimmer.beginPath();
        contextSwimmer.arc(centerX, centerY+positionY,
                           radius, 0, 2 * Math.PI, false);

        contextSwimmer.fillStyle = "#000000";
        contextSwimmer.fill();
        contextSwimmer.lineWidth = 5;

        // torso
        contextSwimmer.beginPath();
        contextSwimmer.moveTo(200,70+positionY);
        contextSwimmer.lineTo(200,175);
        contextSwimmer.lineWidth = 10;
        contextSwimmer.strokeStyle = "#000000";
        contextSwimmer.lineCap = "round";
        contextSwimmer.stroke();

        // image right arm
        contextSwimmer.beginPath();
        contextSwimmer.moveTo(200, 100);
        contextSwimmer.lineTo(175-positionX,140-positionY);
        contextSwimmer.lineWidth = 10;
        contextSwimmer.strokeStyle = "#000000";
        contextSwimmer.lineCap = "round";
        contextSwimmer.stroke();

        // image left arm
        contextSwimmer.beginPath();
        contextSwimmer.moveTo(200, 100);
        contextSwimmer.lineTo(225+positionX,140-positionY);
        contextSwimmer.lineWidth = 10;

```

```
        contextSwimmer.strokeStyle = "#000000";
        contextSwimmer.lineCap = "round";
        contextSwimmer.stroke();

        // image right leg
        contextSwimmer.beginPath();
        contextSwimmer.moveTo(200, 175);
        contextSwimmer.lineTo(190-positionX,250-positionY);
        contextSwimmer.lineWidth = 10;
        contextSwimmer.strokeStyle = "#000000";
        contextSwimmer.lineCap = "round";
        contextSwimmer.stroke();

        // image left leg
        contextSwimmer.beginPath();
        contextSwimmer.moveTo(200, 175);
        contextSwimmer.lineTo(210+positionX,250-positionY);
        contextSwimmer.lineWidth = 10;
        contextSwimmer.strokeStyle = "#000000";
        contextSwimmer.lineCap = "round";
        contextSwimmer.stroke();

        contextSwimmer.restore();

    };

</script>
</head>
<body onload="drawWater();">
    <canvas id="myWaterCanvas" width="400" height="400" style="z-index: 2;
        position:absolute;left:0px;top:0px;">
    </canvas>
    <canvas id="mySwimmerCanvas" width="400" height="400" style="z-index: 1;
        position:absolute;left:0px;top:0px;">
    </canvas>

</body>
</html>
```

Conclusion

The HTML5 canvas is central to the construction of browser-based RIAs. It provides a utilitarian drawing environment powered by JavaScript and your imagination. It is not really difficult to learn, and there are many support tools on the web for your training and learning needs, including cheat sheets, blogs, online articles, video and non-video tutorials, and sample applications.

The ability to alter text and images visually and to simulate motion makes Canvas an extremely valuable tool. Whether you approach it from the perspective of a designer or developer, use Canvas to build game applications to run on mobile devices, or merely want to enhance the use of the overall screen real estate, Canvas is a critical component of the HTML5 experience.

Downloads

Description	Name	Size	Download method
Napoleon image	Napoleon.zip	2045KB	HTTP

[Information about download methods](#)

Resources

Learn

- "[Create great graphics with the HTML5 canvas](#)" (developerWorks, February 2011) is a guide to the development of graphical techniques and processes.
- The Safari Dev Center has a demo called [Canvas Pixel Manipulation](#) that is an excellent example of managing the canvas to develop effective visual assets.
- WHATWG's [HTML Living Standard](#) provides a window to the evolving development of the HTML5 canvas specs.
- W3Schools.com's [HTML5 Canvas](#) reference has several useful exercises to help you hone your canvas knowledge.
- MDM Docs's [Canvas tutorial](#) is a good, basic tutorial that reflects the development expertise of Mozilla.
- [Let's Call It A Draw\(ing Surface\)](#) is an innovative walk-through for a basic comprehension of the HTML5 canvas.
- The [Web development zone](#) specializes in articles covering various Web-based solutions.
- Stay current with [developerWorks technical events and webcasts](#) focused on a variety of IBM products and IT industry topics.
- Follow [developerWorks on Twitter](#).

Get products and technologies

- The [Dojo Toolkit](#), an open source modular JavaScript library, helps you quickly develop cross-platform, JavaScript/Ajax-based applications and websites.
- [Try out IBM software](#) for free. Download a trial version, log into an online trial, work with a product in a sandbox environment, or access it through the cloud. Choose from over 100 IBM product trials.

Discuss

- Create your [developerWorks profile](#) today and [set up a watchlist](#) on HTML5. Get connected and stay connected with [developerWorks community](#).
- Find other [developerWorks members interested in web development](#).
- Share what you know: [Join one of our developerWorks groups focused on web topics](#).
- Roland Barcia talks about [Web 2.0 and middleware](#) in his blog.
- Follow developerWorks' members' [shared bookmarks on web topics](#).

- Get answers quickly: Visit the [Web 2.0 Apps forum](#).
- Get answers quickly: Visit the [Ajax forum](#).

About the author

Grace Walker

Grace Walker, a partner in Walker Automated Services in Chicago, Illinois, is an IT consultant with a diverse background and broad experience. She has worked in IT as a manager, administrator, programmer, instructor, business analyst, technical analyst, systems analyst, and web developer in various environments, including telecommunications, education, financial services, and software.