APPENDIX

# Associated Technologies

In addition to CSS3, a number of technologies have been associated with HTML5 but actually are defined in their own specifications. Some were once part of HTML5 but have been spun off into their own documents as they have grown, while others are directly used with HTML5 but were never part of the HTML5 specification. This appendix provides a high-level overview of these different technologies that are commonly associated with HTML5. Additionally, you will also find a list of useful website resources for working with HTML5 at the end of the chapter.

## Geolocation

The Geolocation API[1] defines how a web browser can determine a user's geographic location, using their IP or WiFi address or, if they are on a mobile device, the Global Positioning System (GPS) on the device. The location is given as a latitude and longitude coordinate, which may be more or less accurate, depending on the method used in retrieving the location. Determining a user's location with their IP address, for instance, will be far less accurate than using the satellite-based GPS to determine their location.

The implications of retrieving a user's location are quite fascinating, because it enables location-based services to be provided through a web browser. This is perhaps most useful for mobile devices, because information and advertisements can be delivered to a web browser on a device that is on the move.

The JavaScript methods for retrieving a user's location are held in the `Geolocation` object, which is inside `window.navigator.geolocation`. The location can be retrieved one time or can be updated continuously. The API defines three methods:

- `getCurrentPosition()`: Retrieves the current position one time
- `watchPosition()`: Retrieves and updates the current position as it changes
- `clearWatch()`: Stops updating a watched position

When attempting to retrieve the user's location, the web browser will typically provide a prompt as to whether using geolocation is allowed.

[1] See `www.w3.org/TR/geolocation-API/`.

## Retrieving the current position

The `getCurrentPosition()` method is given a parameter for the function to call when the position has been successfully obtained (which could take several minutes if GPS is used). Here's an example:

```
function init() {
          // get the current position and call the "locatedSuccess" function when successful
          window.navigator.geolocation.getCurrentPosition(locatedSuccess);
}
function locatedSuccess(geo) {
          // log the returned Geoposition object
          console.log(geo);
}
window.onload = init;
```

---

■ **Note** Depending on the web browser used, this code will likely work only on a live web server (one running locally is fine). If it is not working, first check that the URL address of the page includes `http://` at the beginning.

---

This script will attempt to obtain the current location and then call the `locatedSuccess()` function when it has done so. The location query is done asynchronously so that other processes can continue to function on the page. The function is handed a parameter that contains a `Geoposition` object that contains information about the location. The `Geoposition` object contains a `timestamp` property and `coords` property, which contains yet another object, a `Coordinates` object. The `Coordinate` object contains the following properties, which may contain `null` values depending on the hardware capabilities of your viewing device (for instance, if your device does not have GPS capabilities, these values will be limited):

- `latitude`: The north-south position on the earth
- `longitude`: The west-east position on the earth
- `altitude`: The height of the position, gathered if the viewing device has the capability to measure altitude
- `accuracy` and `altitudeAccuracy`: The accuracy of the position as measured in meters
- `heading`: The direction of travel as measured in degrees around a circle
- `speed`: The speed of travel in a certain heading in meters per second

---

■ **Note** In addition to the `timestamp` and `coords` properties, Firefox includes an `address` property for retrieving address information such as city, country, and even street information!

---

Update the preceding `locatedSuccess()` function to print the location data on-screen:

```
function locatedSuccess(geo) {
         var lat = geo.coords.latitude;
         var long = geo.coords.longitude;

          document.body.innerHTML = "<ul><li>lat:"+lat+"</li><li>long:"+long+"</li></ul>";
}
```

An additional function name can be given to the `getCurrentPosition()` method to specify a function to run when the request for the user's location has failed. Edit the `init()` code and add a `locatedFailed()` function:

```
function init() {
          // get the current position
          //and call the "locatedSuccess" or "locatedFailed" if
successful or not
          window.navigator.geolocation.getCurrentPosition
(locatedSuccess, locatedFailed);
}
function locatedFailed(e) {
          // log the error code and message
          console.log(e.code , e.message);
}
```

The `locatedFailed()` function will run when the location was unable to be obtained. The parameter handed to it is a `PositionError` object that contains an error code and a message. The following are possible errors:

- *Error code 1, permission denied*: The user didn't authorize using geolocation.
- *Error code 2, position unavailable*: The position can't be determined.
- *Error code 3, position retrieval timed out*: Retrieving the position took too long.

If you want to test this function, the easiest way is to deny the geolocation request from the browser. Depending on the browser and whether you accepted the previous request for geolocation information, the browser will remember your choice and won't ask you again. Google Chrome, for instance, will require you to click the "target" icon in right of the address bar where you will have the option to clear the geolocation settings (the page will need to be reloaded for the settings to take effect). Figure A-1 shows what this dialog looks like. For Firefox, the geolocation permissions are found under Tools ➤ Page Info, which will open a dialog box with information about the page currently being viewed. Selecting the Permissions tab will allow you to set the preferences for sharing location information with a particular page. Safari has location settings set in its Privacy tab in the browser's Preferences pane. If you are unsure of where to clear geolocation information in your preferred browser, check in the application's preferences or the right side of the address bar, because these are the usual locations for the geolocation permission settings.

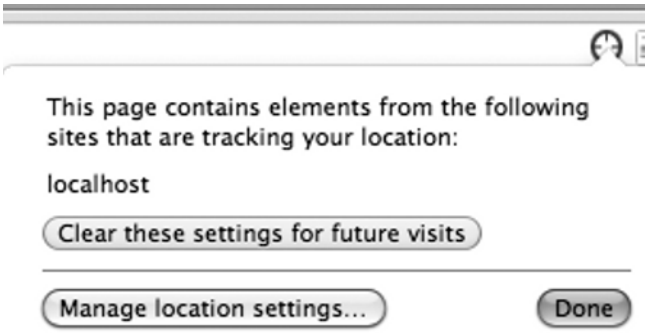


***Figure A-1.*** *Dialog for clearing geolocation settings for a page being viewed in Google Chrome*

Lastly, the `getCurrentPosition()` method can be handed a custom object that can be used to set various options used in retrieving the location. This object can be set up with the following properties and values:

- `enableHighAccuracy`: Set to `true` or `false`. If enabled, the highest accuracy method for determining the location is used, such as GPS. Be aware that this will increase battery usage and the length of time for retrieving the location.
- `timeout`: How long to wait (in milliseconds) when retrieving the location before throwing a position unavailable error.
- `maximumAge`: How long (in milliseconds) a particular position should be considered the current position.

These options may be added to the `getCurrentPosition()` method using shorthand object creation notation, like so:

```
var options = {
          enableHighAccuracy: true,
          timeout: 120000,
          maximumAge: 1000
};
window.navigator.geolocation.getCurrentPosition(locatedSuccess, locatedFailed, options);
```

This will enable high-accuracy positioning (which is dependent on the hardware available), set the timeout to two minutes, and set the maximum age of the location to one second.

## Watching the current position

Getting the location once is fine for a stationary device, such as a desktop computer, but for a mobile device, the location would have to be continually retrieved to be accurate. The `watchPosition()` method is used to continually poll the location (this is where the maximum age option is useful) to update the location information. It takes the same arguments as the `getCurrentPosition()` method, but it should be set to a variable that can later be referenced and handed to the `clearWatch()` method if updating the location continuously is stopped. Here's an example:

```
var geoWatchID = window.navigator.geolocation.watchPosition(locatedSuccess, locatedFailed,
options);
```

Later in the code, `geoWatchID` can be passed as an argument to `clearWatch()` to stop the position from updating:

```
clearWatch(geoWatchID);
```

## SVG and MathML

SVG and MathML have two totally different purposes, but they have one thing in common: they are both XML-based languages that can be embedded in HTML5. Scalable Vector Graphics (SVG) is for describing vector shapes, while Mathematical Markup Language (MathML) is for describing mathematical notation.

SVG is one half, along with `canvas`, of the Web's standard imaging options. While `canvas` deals well with bitmaps, SVG deals well with vector shapes. It also has built-in animation capabilities, which would need to be built from scratch in `canvas`.

The syntax of both is beyond what can be covered here, but being XML-based, they both look very much like HTML, except with a different set of elements. For example, the following code shows an HTML page that includes both MathML and SVG to describe and diagram the trigonometric functions shown in Chapter 7.

```
<!DOCTYPE html>

<html>
          <head>
                    <meta charset="utf-8" />
                    <title>SVG and MathML Demo</title>
          </head>
          <body>
                    <h1>SVG and MathML embedded in an HTML5 page</h1>
                    <p>
                    <math>
                              <mi>x</mi>
                              <mo>=</mo>
                              <mrow>
                                        <msub><mi>x</mi><mn>1</mn></msub>
                                        <mo>&plus;</mo>
                                        <mi>cos</mi>
                                        <mfenced><mi>&#x3B8;</mi></mfenced>
                                        <mo>&InvisibleTimes;</mo>
                                        <mi>c</mi>
                              </mrow>
                    </math>
                    </p><p>
                    <math>
                              <mi>y</mi>
                              <mo>=</mo>
                              <mrow>
                                        <msub><mi>y</mi><mn>1</mn></msub>
                                        <mo>&plus;</mo>
                                        <mi>sin</mi>
                                        <mfenced><mi>&#x3B8;</mi></mfenced>
                                        <mo>&InvisibleTimes;</mo>
                                        <mi>c</mi>
```

```
                        </mrow>
                    </math>
                    </p><p>
                    <svg>
                        <circle r="100" cx="101" cy="101" fill="white" stroke="black"/>
                        <polygon points="101,101 171.710678,30.2893219 171.710678,101"
style="fill:white;stroke:black;" />
                        <rect width="10" height="10" x="161.710678" y="91"
style="fill:white;stroke:black;" />
                        <text x="71" y="101" fill="black" font-family="sans-serif"
font-size="16">x, y</text>
                        <text x="126" y="61" fill="black" font-family="sans-serif"
font-size="16">c</text>
                        <text x="121" y="96" fill="black" font-family="sans-serif"
font-size="16">&#x3B8;</text>
                        <text x="175" y="27" fill="black" font-family="sans-serif"
font-size="16">x<tspan font-size="11" baseline-shift="sub">1</tspan>, y<tspan font-size="11"
baseline-shift ="sub">1</tspan></text>
                    </svg>
                    </p>
</body>
</html>
```

The preceding code creates the notation and diagram in Figure A-2.

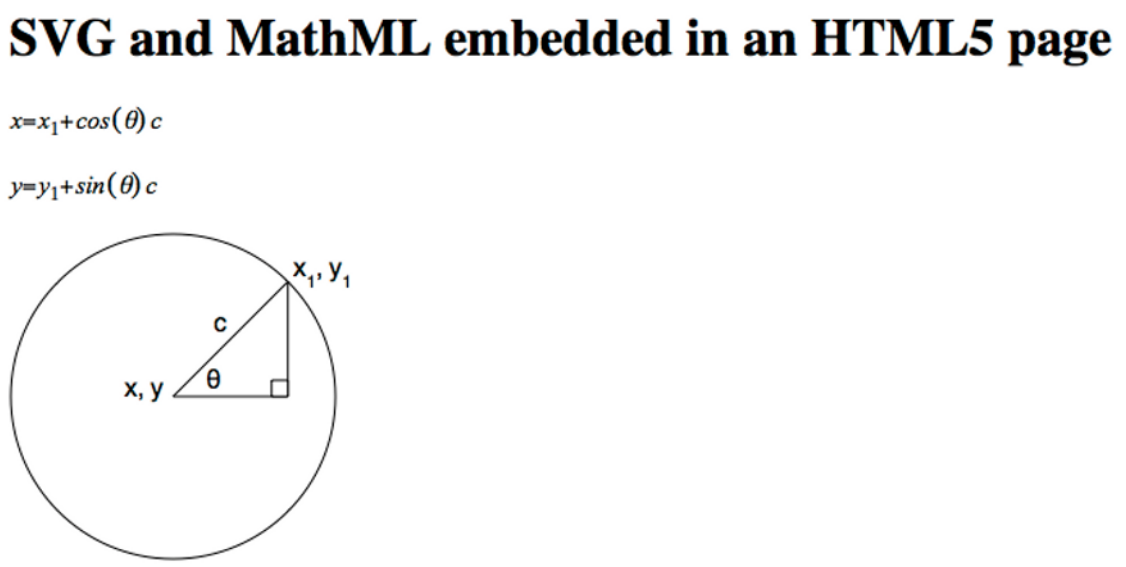


***Figure A-2.*** *A diagram created from markup using HTML, MathML, and SVG*

# Client-side storage

Imagine a web application that could save data the user had worked on in a client-side database and then sync that with a server-based database when the user connected online. Such offline functionality would be tremendously useful for improving the latency of the application, since the user's data would not need to be sent back and forth over the network as frequently, and it would also help in situations where connectivity may be spotty, such as in a mobile environment.

## Web storage

Cookies have long been the method for storing data on the client-side browser. A problem with cookies has been that they are small, allowing only 4 kilobytes of storage each, which is minuscule by the standard of today's data-rich web pages/applications. In response to this, a new generation of client-side storage solutions have emerged. The most stable solution, and the one that can be seen as a replacement for cookies, is the Web Storage API,[2] which allows up to 5 megabytes of storage. Web storage is actually broken into two options, the `localStorage` object and `sessionStorage` object, both of which are properties of the `window` object. The difference between the two is that data stored in `localStorage` is persistent, while data stored in `sessionStorage` is lost when the browser session ends (such as when quitting the browser), but otherwise they are used in the same manner. Each is just a series of key/value pairs, so a key is set with some data, and then the key is used to retrieve the data later.

## Using web storage

Using web storage is really quite straightforward. To add data to the storage, use either of the following syntaxes:

```
window.localStorage.setItem("key","value");
window.localStorage["key"] = "value";
```

In this code, the "key" and "value" can be any string of text. To retrieve the data from the storage, use either of the following:

```
var val = window.localStorage.getItem("key");
var val = window.localStorage["key"];
```

To remove data, either remove a specific key or clear the whole storage:

```
window.localStorage.removeItem("key");
window.localStorage.clear();
```

### WEB STORAGE EXAMPLE

Using the `contenteditable` attribute, you can create a simple text editor that saves changes on the client. For this example, create a new HTML file named `edit.html` and fill it with the following code:

```
<!DOCTYPE html>
```

[2] See `http://dev.w3.org/html5/webstorage/`.

```
<html>
          <head>
                    <meta charset=utf-8 />
                    <title>Contenteditable and localStorage demo</title>
                    <script type="text/javascript" src="js/script.js"></script>
          </head>
          <body>
                    <section id="editable">This text may be edited and the changes will
                    be saved locally.</section>
                    <button id="startEditBtn">Turn editing on</button>
                    <button id="stopEditBtn">Turn editing off and save changes</button>
                    <button id="clearBtn">Clear changes!</button>
          </body>
</html>
```

Now create a new JavaScript file named `script.js` and place it in a directory named `js` that is in the same location as `edit.html`. Fill it with the following script:

```
var editable; // variable for editable area

// initialize the variables and add event handlers
function init()
{
          editable = document.getElementById('editable');
          var startEditBtn = document.getElementById('startEditBtn');
          var stopEditBtn = document.getElementById('stopEditBtn');
          var clearBtn = document.getElementById('clearBtn');

          startEditBtn.onmousedown = startEdit;
          stopEditBtn.onmousedown = stopEdit;
          clearBtn.onmousedown = clear;

          // update text with data in local storage
          if (localStorage.getItem("savedtext")) editable.innerHTML =
          localStorage.getItem("savedtext");
}

function startEdit()
{
    // add the contenteditable attribute
    editable.setAttribute("contenteditable", true);
}

function stopEdit()
{
    // disable the contenteditable attribute
    editable.setAttribute("contenteditable", false);
    // save the text
    localStorage.setItem("savedtext", editable.innerHTML);
}

function clear()
{
```

```
        // clear the local storage
        localStorage.clear();
        // reload the page
        window.location.href = "";
    }
    window.onload = init;
```

Open the HTML page in a web browser, and you will be able to turn on editing (which adds the `contenteditable` attribute), save the edits, and see those edits stick because they will be stored in the local storage (Figure A-3).

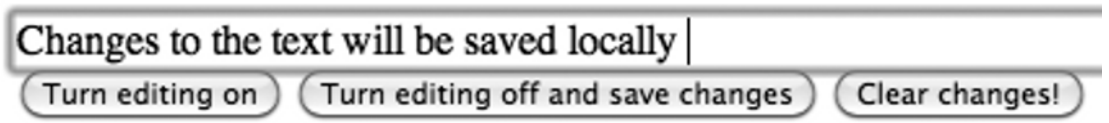


***Figure A-3.*** *A simple application using local storage*

## Other storage options

Local storage is easy to use, but with that ease comes limits on its capabilities. It's really not comparable to a database you would find on the back-end web server, which likely describes the relationship between the stored data and provides methods for ensuring data integrity. Since web technologies are moving toward enabling the creation of web applications, having a fully capable database on the client end is a desirable option. One such option is Web SQL, which essentially embeds an SQLite[3] database into the web browser. This means **Structured Query Language** (SQL) commands can be used directly from JavaScript. Pretty cool! Unfortunately, the future of Web SQL has darkened considerably, because disagreements over standardizing the use of SQLite as the embedded database has led to support being dropped for the initiative by the W3C. Because of this, Mozilla has said it will drop support in Firefox, which means support is spotty and not reliable going forward. Too bad.

Another option, which is currently supported only in Firefox but has planned support from other major web browsers, is the Indexed Database API,[4] also known as IndexedDB. This database solution stores key/value pairs, like web storage, but includes more sophisticated features, such as transactions for ensuring data is successfully committed to the database, which helps guarantee data integrity. IndexedDB is not as sophisticated as Web SQL (it's not a relational database), but it is more capable than web storage and is looking like it will be the option to use in the future for handling client-side data storage that is more complex than what web storage will accommodate.

# Web workers

Web workers are making computationally intensive tasks on the Web a little less painful. JavaScript is a single-threaded language, meaning a script that takes a lot of processing power could completely paralyze any user-interactive scripts that may be running. Using a web worker, a new thread can be spawned that runs a script without interrupting the processing of UI interactions or other events in the main script. Web workers come in two flavors: dedicated workers and shared workers. Shared workers

[3] See `http://sqlite.org`.

[4] See `www.w3.org/TR/IndexedDB/`.

are more powerful than dedicated workers because they can communicate with multiple scripts, while a dedicated worker responds only to the script that spawned it in the first place.

## Web Sockets API

The Web Sockets API[5] is a specification that defines a protocol for providing two-way communication with a remote host. The Web's roots have traditionally been essentially one-way. A server sends a page to a client web browser, and then nothing happens between the two until the user clicks a link and requests another page. What a web socket provides is an open connection over which data can be sent from the client to the server at any time after the page has loaded, and vice versa. This could be used , for example, to create multiplayer online games or applications, because data can be sent to the server from one client and distributed to all other clients connected to the same server.

## Video conferencing and peer-to-peer communication

A project is underway to create a specification for video conferencing between two web browsers. This is a major area of difference between the W3C HTML5 and WHATWG HTML specifications, because it is included in the WHATWG version but omitted from the W3C specification. Instead, the W3C has a separate specification named "WebRTC 1.0: Web Real-time Communication Between Browsers."[6] Since both specifications are in draft status, it is not inconceivable that the version included in the WHATWG HTML draft may well be spun off into a separate specification in the future, as has happened at the W3C.

Anyway, administration issues aside, the actual technology for enabling video conferencing requires that two separate web browsers gather video and audio and stream it over a peer-to-peer connection to each other. Specifically, the following steps need to happen:

1. Gain access to a webcam or other video/audio input device.
2. Record the video/audio locally so that it can be streamed to a remote web browser.
3. Connect and send the video/audio to a remote web browser.
4. Display a video/audio stream in a video or audio element on the local and remote web browsers.

An API called the Stream API, which defines an interface called `MediaStream`, would be used with JavaScript to handle parsing and displaying of the streaming media. In terms of sending the media stream, another API, called the Peer-to-peer Connections API, would be used. This API describes a `PeerConnection` JavaScript interface that defines methods for connecting and sending a media stream to a remote peer.

[5] See `http://dev.w3.org/html5/websockets/`.

[6] See `http://dev.w3.org/2011/webrtc/editor/webrtc.html`.

## WAI-ARIA

WAI-ARIA,[7] the Accessible Rich Internet Applications specification (the WAI stands for Web Accessibility Initiative), aims to provide a syntax for making modern dynamic web applications accessible to people with disabilities. WAI-ARIA uses attributes to mark up user interaction of the page's content and describe how page elements relate to each other. WAI-ARIA defines a `role` attribute with a large set of values for describing how a web feature is presented and how the page is structured. There are also a large number of "`aria-*`" prefixed attributes for describing the state of web page features. These attributes can be used to annotate, for example, whether a menu has a submenu or what actions a dragged object can perform when being dropped on a target. The WAI-ARIA specification is a large specification in its own right; for further information on it, visit the WAI's WAI-ARIA overview page: `www.w3.org/WAI/intro/aria`.

## File API

There are three specifications under development related to reading, browsing, and writing to files on the file system of a client machine. The main one is the File API,[8] which includes interfaces called `File`, `FileList`, `FileReader`, and `FileError` that define methods that, for instance, can be used to read the name and last modification date of files or groups of files. The specification also defines a `Blob` for interfacing with raw binary data, which may be inspected for size and type and sliced into chunks. The File API deals with files that could appear in the web browser through a web form's file input type or even through dragging and dropping a file from the user's system to the web browser window. Extending the File API are the Directories and System[9] and Writer[10] APIs. Directories and Systems is what describes methods of interacting directly with the user's local file system. Obviously, this has security implications, so the file system exposed is sandboxed so web applications don't have unrestrained power to intrude into a user's computer. The Writer API does what you would expect; it defines how files or blobs of raw data can be written to the file system. It also defines a `FileWriterSync` interface for working with writing files in conjunction with the Web Workers API.

## Useful web resources

The following websites are resources you may find useful when developing with HTML5:

- *W3C's Working Draft HTML5 specification*: `http://w3.org/TR/html5/`
- *WHATWG "living" HTML specification*: `www.whatwg.org/specs/web-apps/current-work/`
- *Html5.org*: Includes a HTML5 validator and a tracker for changes to the WHATWG specification: `http://html5.org`
- *Html5rocks.com*: Includes an online code editor playground and slide presentation made entirely with HTML5 technologies: `http://html5rocks.com`

---

[7] See `www.w3.org/TR/wai-aria/`.
[8] See `www.w3.org/TR/FileAPI/`.
[9] See `www.w3.org/TR/file-system-api/`.
[10] See `www.w3.org/TR/file-writer-api/`.

- *Html5doctor.com*: Contains informative articles about HTML5 as well as a comprehensive element reference; `http://html5doctor.com`
- *Caniuse.com*: Compatibility tables for HTML5, CSS3, and related technologies: `http://caniuse.com`
- *Html5test.com*: Browser score for HTML5 and related feature support: `http://html5test.com`
- *CSS3 Selectors Test*: Browser test for support of a wide range of CSS selectors: `http://www.css3.info/selectors-test/`
- *Mobilehtml5.org*: HTML5 feature compatibility tables for mobile and tablet browsers: `http://mobilehtml5.org`
- *HTML5boilerplate.com*: Starting template for HTML5 pages: `http://html5boilerplate.com`
- *Modernizr*: JavaScript library for testing browser support of HTML5, CSS3, and related features: `http://modernizr.com/`
- *Google Chrome Frame*: Method of enabling modern web technology capabilities in older browsers: `http://code.google.com/chrome/chromeframe/`
- *Html5pattern.com*: Regex patterns for client-side validation in web forms: `http://html5pattern.com`
- *Mozilla Developer Network (MDN)*: Great, easy-to-follow resource on HTML5 and other web technologies: `https://developer.mozilla.org/en/HTML/HTML5/`
- *Html5gallery.com*: A showcase of sites using HTML5 technologies: `http://html5gallery.com`
- *Mediaqueri.es*: A showcase of sites using media queries: `http://mediaqueri.es`

# Index

## A, B



## C

## D, E



## F

## G



## H

## I, J, K



## L



## M, N

## O



## P, Q, R



## S

## T, U



## V



## W



## X, Y, Z