Lesson 7: Using HTML5 APIs

Objectives

By the end of this lesson, you will be able to:

- 7.2: Define application programming interface (API), and list the common HTML5 APIs.
- 7.3: Use the HTML5 Canvas API for advanced techniques.
- 7.4: Develop offline Web applications using the HTML5 AppCache API.
- 7.5: Use the HTML5 Geolocation API to obtain geographic information about a user.
- 7.6: Create drag-and-drop Web pages using the HTML5 Drag-and-Drop API.
- 7.7: Use the HTML5 File API to interact with files.
- 7.8: Use the HTML5 History API to manipulate a browser’s history.
- 7.9: Retrieve and manipulate HTML5 data using XMLHttpRequest and jQuery.
Pre-Assessment Questions

1. Which HTML5 API allows a Web program to run locally on a client system?
   a. Canvas  
   b. AppCache  
   c. Drag-and-Drop  
   d. Geolocation  

2. Which of the following is a function of the HTML5 File API?
   a. Create preview thumbnail images during image uploads  
   b. Locate the position of the client system  
   c. Move files using a drop zone  
   d. Validate HTML5 code  

3. What is the function of the HTML5 History API?
Introduction to HTML5 APIs

As mentioned in an earlier lesson, experts in Web technology expect that the future of Web development will rely heavily on HTML5 APIs.

HTML5 APIs consist of HTML5, CSS and JavaScript — the Web development trifecta. Used together, these technologies create Web pages that can easily adapt to smartphones, tablets, gaming devices and smart TVs, as well as to traditional desktop computers. In addition to Web pages, they can also be used to create desktop software and apps for mobile devices.

To use HTML5 APIs, you should first understand the Document Object Model (DOM). This lesson will introduce you to the DOM, and then venture more deeply into the functionality and use of some common HTML5 APIs. You will also learn about ways you can request and display data to enrich the user experience.

CIW Online Resources – Movie Clips
Visit CIW Online at http://education.Certification-Partners.com/CIW to watch a movie clip about this topic.

Lesson 7: Using HTML5 APIs

Document Object Model (DOM)

The Document Object Model (DOM) is a standard developed by the W3C. It describes the elements within a document rendered by a Web browser as objects (such as the head object, the body object, the heading objects, etc.). The DOM is a vendor-neutral, cross-platform API that specifies how objects in a Web document can be referred to and manipulated through scripting languages.

The DOM provides a standard interface so you do not need to develop multiple versions of your scripts tailored for each browser. To work with the DOM, you only need to use a scripting language, such as JavaScript. This allows you to access and manipulate objects in the document.

Essentially, the DOM is a standard that enables you to access, change, add or delete HTML elements in a Web document. For example, you could use JavaScript to change all occurrences of the <a> element in the document. You can also use the DOM to access and modify HTML element attributes, HTML events and CSS styles.

All browsers are DOM-compliant. However, browser vendors tend to include their own features in their browsers in hopes of getting them added to the standard. For this reason, you still need to test your scripts in multiple browsers, even when invoking DOM elements.

You can learn more about the W3C DOM by visiting www.w3.org/DOM.
Common HTML5 APIs

Recall that an application programming interface (API) is a set of universal commands, calls and functions that allows developers to communicate with an application or operating system. HTML5 APIs provide an open environment for developing Web applications that does not rely on proprietary browser plug-ins.

HTML5 APIs consist of:

- HTML5 (to create page elements).
- CSS (to format page elements).
- JavaScript (to access the DOM).

Together, these technologies create Web pages that can easily adapt to smartphones, tablets, gaming devices and smart TVs, as well as to traditional PCs. The HTML5 APIs can also be used to create apps for mobile devices. For example, mobile apps for YouTube, Pandora and LinkedIn use HTML5 APIs.

Many expect mass adoption of HTML5 APIs in the next few years as mobile devices continue to proliferate.

Some functions of the HTML5 APIs include:

- Media (audio and video).
- Document editing.
- Cross-document messaging.
- MIME type and protocol handler registration.
- Web storage.

For a complete list of HTML5 APIs from one of the W3C members, visit http://platform.html5.org/. You should bookmark this Web page.

You will learn about some popular APIs in this lesson:

- Canvas
- Offline Application Cache
- Geolocation
- Drag-and-Drop

You have already used the media APIs for audio and video using the <audio> and <video> elements.

The Canvas API

As mentioned in the beginning of this course, Canvas is an HTML5 API that provides a place on a Web page (a “canvas”) where developers can display graphics, animation, video and games “on the fly” without the need for a plug-in.
Canvas is described by the W3C as an HTML5 element that "provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, or other visual images on the fly." By itself, Canvas is a simple pixel-based drawing API that produces a bitmap image.

The Canvas API itself may seem unexciting. It requires an advanced Web application developer to fill it with multimedia content that he or she has created. For this course, you will create a simple canvas area on a Web page and add a rectangle to it. This course does not cover Web application development, so the lab is for introduction purposes only.

The exciting part of the Canvas API is in the opportunities it affords developers:

- You can draw objects on a canvas using JavaScript.
- You can use a canvas to display simple drawings, animations, graphics or graphs.
- You can also use it for more advanced tasks that include interactivity, such as video games, simulations, video editing or image configurations.

The Canvas API uses fewer resources than a plug-in does (such as battery power and CPU usage). This is especially important with regard to mobile devices such as smartphones and tablets that rely upon battery power.

### How the Canvas API works

The canvas element is defined in HTML with the `<canvas></canvas>` tag. The `<canvas>` element uses two new HTML5 attributes:

- `height`
- `width`

You can omit these attributes and use the default height of 300 pixels and width of 150 pixels.

```
The <canvas> element also supports other HTML global attributes.
```

Initially, the canvas is transparent. Adding some basic styles with CSS such as a border, margin, background, etc., will help make it visible in the browser. Styles that are applied will have no effect on the drawing capabilities.

### Coding a canvas for browser support

Older browsers do not fully support the Canvas API. Therefore, you should always provide alternative content. Older browsers will ignore the `<canvas></canvas>` tags and will render the content that is between them. See the following example:

```
<canvas id="canvas" width="250" height="125" style="border:2px solid #000080;">
    The canvas element is not supported by your browser.
</canvas>
```

Looking at the preceding code, an older browser will display the text: "The canvas element is not supported by your browser." All compatible browsers will display the canvas page element styled with a solid 2-pixel navy blue border.

```
The id attribute is not specific to <canvas>. However, you should specify a unique id to identify this element later.
```
Canvas context

The <canvas> element uses a DOM method called `getContext()`. The `getContext()` method returns a context object, which provides properties and methods (i.e., information about lines, colors, fonts, etc.) that can be used to draw and manipulate images on the canvas.

Essentially, the drawing context is where the action happens on the canvas. Without it, the canvas is blank.

You call a <canvas> element’s drawing context by passing the string "2d" to the `getContext()` method:

```javascript
canvas.getContext("2d")
```

You must use the "2d" string because as of this writing, there is not yet a 3D context available.

Canvas shapes

Canvas only supports one primitive shape: the rectangle. All other shapes are created by using a collection of functions and combining one or more paths.

To draw a rectangle, you must specify the x and y coordinates as well as the width and height of the rectangle you want to draw. Table 7-1 describes rectangle properties for the canvas element.

<table>
<thead>
<tr>
<th>Canvas Rectangle Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fillRect(x,y,width,height)</td>
<td>Draws a filled rectangle</td>
</tr>
<tr>
<td>strokeRect(x,y,width,height)</td>
<td>Draws a rectangular outline</td>
</tr>
<tr>
<td>clearRect(x,y,width,height)</td>
<td>Clears the specified area and makes it fully transparent</td>
</tr>
</tbody>
</table>

You can draw paths to create shapes other than rectangles in the canvas element. Table 7-2 describes some methods and properties that will be used in the following lab to draw in the canvas.

<table>
<thead>
<tr>
<th>Canvas Drawing Methods and Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beginPath()</td>
<td>Either begins a drawing path or resets the current path</td>
</tr>
<tr>
<td>closePath()</td>
<td>Ends the current drawing path</td>
</tr>
<tr>
<td>moveTo(x,y)</td>
<td>Moves the drawing pointer to a specified position (x,y)</td>
</tr>
<tr>
<td>arc(x,y,radius,startAngle,endAngle,anticlockwise)</td>
<td>Draws an arcing (curving) line with a center (x,y), a given radius, a starting and ending angle (in radians), and instruction to draw in a clockwise or counterclockwise direction</td>
</tr>
<tr>
<td>lineTo(x,y)</td>
<td>Draws a straight line from the current drawing position to the specified position (x,y)</td>
</tr>
<tr>
<td>stroke()</td>
<td>Adds a stroke (line) to the path drawn</td>
</tr>
</tbody>
</table>
In the following lab, you will explore the Canvas API. Suppose your supervisor has informed you that the company Web site will need to move away from the use of plug-ins to display content. You know that the site needs to be able to display a variety of graphics and animations. You need to start learning about the Canvas API to see what it can do.

### Lab 7-1: Using the HTML5 Canvas API

In this lab, you will demonstrate the functionality of the HTML5 Canvas API. The JavaScript code in this lab is beyond the scope of this course, but is provided to show the potential of the Canvas API.

1. **Windows Explorer:** Copy the Lesson07 folder from your student lab files to your Desktop. Open the Lab_7-1 folder from your student files.

2. **Editor:** Open the file **Canvas_API.html**. Study the code, which appears as follows:

```html
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8"/>
<title>HTML5 Canvas API Example</title>
</head>
<body>
<h1>CIW Advanced HTML5 and CSS3 Specialist</h1>
<h2>Canvas Demonstration</h2>
<canvas id="canvas" width="200" height="200"></canvas>
<script>
// and is licensed under CC-BY-SA 2.5.

function draw() {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext)
    var ctx = canvas.getContext('2d');

  ctx.beginPath();
  ctx.lineWidth = 3;
  ctx.fillStyle = "yellow";
  ctx.arc(75,75,50,0,Math.PI*2,true); // Outer circle
  ctx.fill();
  ctx.moveTo(110,75);
  ctx.arc(75,75,35,0,Math.PI,false);   // Mouth (clockwise)
  ctx.moveTo(65,65);
  ctx.arc(60,65,5,0,Math.PI*2,true);  // Left eye
  ctx.moveTo(95,65);
  ctx.arc(90,65,5,0,Math.PI*2,true);  // Right eye
  ctx.stroke();
}
</script>
</body>
</html>
```

### OBJECTIVE

7.3: Canvas API
```html
window.onload = draw;
</script>
</body>
</html>

3. **Editor:** Continue to study the code and observe the following points. Several things happen in this code:

- First, the `draw()` function is pointed to the HTML `<canvas>` element.
- Next, a check is performed to see whether the user's browser supports drawing on the canvas. The code does this by checking the availability of the `getContext()` function on the canvas element.
- If canvas is supported, the code then calls the `getContext()` method of the `<canvas>` element and stores a reference to the 2D drawing context in the `ctx` variable.
- You can now use a variety of methods to draw different shapes and paths on the drawing surface. This code uses the methods and properties described in the preceding table (Table 7-2).

4. **Browser:** Open the file `Canvas_API.html` in the browser. It should resemble Figure 7-1.

![Figure 7-1: Canvas API demonstration](image)

5. **Editor:** Comment out the lines with the `moveTo()` methods (lines 24, 26 and 28 in the code), then refresh the page. This will show the entire path taken to draw the smiley face. It should resemble Figure 7-2.

![Figure 7-2: Drawing using Canvas API](image)
6. Close your browser and text editor.

In this lab, you demonstrated how the Canvas API can be used to draw a smiley face in a browser.

---

**WebGL (Web Graphics Library)**

WebGL is a JavaScript API that enables rendering of interactive 3D and 2D graphics in an HTML canvas, without the need for plug-ins in HTML5-compatible browsers. Early applications of WebGL include Google Maps.

Figure 7-3 shows an example demo of WebGL graphics from GooTechnologies in which the image of the boy in the boat can be grabbed and rotated by the user (try it at [www.gootechnologies.com/pearl-boy](http://www.gootechnologies.com/pearl-boy)).

![Figure 7-3: WebGL 3D image-rendering demo — Goo Technologies' Pearl Boy](image_url)

WebGL elements can be mixed with other HTML elements and combined with other parts of the page or page background. WebGL programs consist of control code written in JavaScript and shader code, which is used to create special effects in images.

WebGL is used with the HTML5 `<canvas>` element. You create a WebGL context by passing the string "webgl" to the `getContext()` method:

```javascript
canvas.getContext("webgl")
```

Recall that a 3D drawing context is not yet available for canvas. WebGL has the ability to render 3D graphics, but at this time it does not yield consistent results. It is wise to code a fallback for an alternative context in case "webgl" fails to initialize:

```javascript
canvas.getContext("experimental-webgl");
```

WebGL has support in modern browsers, including Firefox 4.0, Chrome 9.0, IE 11.0, Safari 5.1 and Opera 12.0 (and later versions of each), although results vary at this time.
Some ways you can create and publish WebGL applications include the following:

- You can design WebGL scenes without programming knowledge by using content creation tools such as CopperCube, Blender or Autodesk Maya. You can then export the scenes you create to WebGL and place them in your page.
- Some online services will publish interactive 3D WebGL content for you.
- Firefox began offering built-in WebGL tools in version 27.
- You can also use WebGL-based 3D editors such as Clara.io to design or edit 3D content in your browser.

WebGL is developed and maintained by the non-profit Khronos Group. Although further study of WebGL is beyond the scope of this course, you can research it at www.khronos.org/webgl.

To see examples of WebGL, visit "9 Mind-Blowing WebGL Demos" at http://davidwalsh.name/webgl-demos.

### The Offline AppCache API

**Offline Web applications** allow the user to interact with Web sites or Web documents while the user is offline. Thus the user can continue to work with Web pages when a network connection is unavailable. In this case, the Web pages will store data locally using the **application cache**.

This feature is especially helpful when users leave their Internet Service Provider’s coverage range. With mobile devices, users may lose connections when traveling through valleys, tunnels or other areas without coverage. An offline Web application allows them to continue working with the application without interruption during coverage loss.

You can create offline Web applications easily with the HTML5 AppCache API. This API uses a **manifest** to tell the browser which files to store locally, which allows the user to continue working without the connection.

**Additional advantages of offline Web applications**

In addition to running when the computer is offline, an offline Web application will also load more quickly the next time it is used. The application loads faster because it can be run locally, from the user’s computer, instead of having to be downloaded from a server.

This process also reduces the load placed upon the server because the server does not have to provide an application each time it is used. The browser only downloads changes or updates from the server.

**How the Offline AppCache API works**

To set up a site and its applications to work offline, you must first create a mandatory **manifest** file. This manifest lists the files needed for the Web application to work offline. For example, if a Web site is cached, your manifest would list the main (or home) .html file, the .css file, and any images or JavaScript files used.

The manifest file name uses a file name extension of .appcache. The file name itself is usually the application name, such as mortgagecalculator.appcache or date.appcache.

---

**OBJECTIVE**

7.4: Offline AppCache API

**offline Web application**
A program that permits the user to continue working with Web sites and documents even when a network connection is unavailable.

**application cache**
A directory on a computer where applications, such as Internet apps, can be stored and run without access to the original application.

**manifest**
A list of the files that are needed for the Web application to work offline.
Best practice dictates that you use the file name extension .appcache for the cache manifest file. However, the manifest file can use any extension as long as the Web server is set to serve the file with a MIME type of text/cache-manifest. Some developers may advise using a .manifest file name extension with the text/cache-manifest MIME type, but this can lead to conflicts with Microsoft application/manifest files that use the .manifest extension.

The manifest file always includes the text "CACHE MANIFEST" in the first line. The contents of the manifest file must use the following format:

```
CACHE MANIFEST
#Comment: The CACHE MANIFEST line is required.
#Comment: Blank lines and comments are ignored

CACHE:
index.html
styles.css
actions.js
banner.jpg
```

Consider the contents of each line of the preceding manifest file:

- The CACHE MANIFEST line is required to identify the purpose of the code.
- The next two lines in this example are comments, which are helpful for developers to write notes to themselves and others reading the code. Comments (as well as blank lines used for readability) are ignored by the program.
- The CACHE section lists the application files that will be used offline (e.g., cached).

Once you have created the cache manifest file, you must point the HTML files that will be cached to the manifest file. To do this, include a `manifest` attribute in the `<html>` tag for each of those pages that identifies the manifest file to which the pages should link, as follows:

```
<html manifest="date.appcache">
```

In the following lab, you will create an offline Web application using the HTML5 Offline AppCache API. Suppose your supervisor wants to test whether your team can add content to the company intranet that is available to company employees even when they are offline. She hopes to offer some tools to sales reps and executives who travel frequently and may not always have Wi-Fi access. You demonstrate a solution to her that uses an offline Web application, This cached application is a simple JavaScript program that displays the date and time in the browser, even when the browser is offline.

### OBJECTIVE
7.4: Offline AppCache API

### Lab 7-2: Creating an offline Web application with HTML5

In this lab, you will demonstrate how the HTML5 Offline AppCache API functions. The JavaScript code in this lab is beyond the scope of this course, but is provided to show the potential of the HTML5 AppCache API.

1. **Windows Explorer:** Open the Lab 7-2 folder from your student files.

2. **Editor:** Open the file date.html. Study the code, which appears as follows:

```
<!DOCTYPE HTML>
<html manifest="date.appcache">
<head>
<meta charset="utf-8"/>
<title>HTML5 Offline Web Application API Example</title>
```
<script src="date.js"></script>
<link rel="stylesheet" href="date.css">

<head>
<h2>HTML5 Offline Web Application API Sample</h2>
<p>The time is: <output id="date"></output></p>
</head>

<body>
Notice that in the <html> element, the manifest attribute is pointing to the manifest file (date.appcache).

3. **Firefox**: Open the file date.html.

3. **Firefox**: Open the file date.html. The browser may block the script and not respond. Determine if this error is a result of browser incompatibility or if the browser security settings are responsible for the error.

4. After clicking to allow local storage, your screen should now resemble Figure 7-5.

5. This date application will now function regardless of whether the user has an Internet connection. Close the files, and exit the browser and editor.

6. **Chrome or IE9**: Open the file date.html. The browser may block the script and not respond. Determine if this error is a result of browser incompatibility or if the browser security settings are responsible for the error.

7. Close your browser.

In this lab, you created an offline Web application using the HTML5 Offline AppCache API.

It is important to note that browsers will continue to display the cached version of an application, regardless of whether it has been updated or changed on the server. To overcome this issue, you must change the manifest file whenever there are changes to the Web application on the server. This will cause the browser to reload the new version of the application.
Also, consider that some browsers have limitations on the cache size per site, so do not exceed 5 MB for the downloaded application if possible.

### The Geolocation API

The HTML5 Geolocation API is used to obtain geographic information about a user, a service known as geolocation. The W3C defines the Geolocation API as "an API that defines an advanced interface for location information associated only with the device hosting the application, such as latitude and longitude."

Geolocation determines how to display a user's location. A number of technologies can be used, such as:

- The user's IP address.
- The user's wireless network.
- GPS hardware on the user’s device.

Because geolocation can compromise a user's privacy, the user may be required to approve the action, depending on his or her device’s security settings. A prompt may display in the browser asking for permission.

#### How the Geolocation API works

Geolocation can:

- Determine a user’s current position.
- Monitor the user’s location.
- Update any changes to that location.

To do this, the Geolocation API uses three methods to obtain the location information from the user:

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

The Geolocation API starts with the `getCurrentPosition()` method to obtain the user's location. All three methods return values of the user’s latitude and longitude. Table 7-3 describes the Geolocation API methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getCurrentPosition()</code></td>
<td>Retrieves the current geographic location of the user</td>
</tr>
<tr>
<td><code>watchPosition()</code></td>
<td>Retrieves periodic updates about the current geographic location of the user</td>
</tr>
<tr>
<td><code>clearWatch()</code></td>
<td>Cancels an ongoing watchPosition() call</td>
</tr>
</tbody>
</table>

Geolocation can display the user's information in a localized map, such as the one shown in Figure 7-6. You need access to a mapping service that can read latitude and longitude values (such as Google Maps) in order to display your geolocation results.
In the following lab, you will experiment with the Geolocation API. Suppose your supervisor has requested an addition to the company Web site to include the location of your site visitors. The marketing department wants to know each visitor’s location so they can determine the nearest retail store that sells your company’s products. You can demonstrate for your supervisor how the Geolocation API works in a Web page to identify a visitor’s location.

Lab 7-3: Using the Geolocation API to obtain geographical information

In this lab, you will demonstrate the functionality of the HTML5 Geolocation API. The JavaScript code in this lab is beyond the scope of this course, but is provided to show the potential of the Geolocation API.

1. **Windows Explorer**: Open the Lab_7-3 folder from your student files.

2. **Editor**: Open the file Geolocation.html. Study the code, which appears as follows (the Geolocation API code is shown in bold):

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<title>HTML5 Geolocation API Example</title>
</head>
<body>

<p id="MyLocation">Click the button to get your position:</p>
<button onclick="getLocation()">My Location</button>
<div id="map"></div>

<script>
var x=document.getElementById("MyLocation");
function getLocation()
{
    if (navigator.geolocation)
    {
        navigator.geolocation.getCurrentPosition(showPosition,showError);
    }
    else{x.innerHTML="Geolocation is not supported by this browser.";}
}
</script>
```

**OBJECTIVE**
7.5: Geolocation API

**NOTE:** Geolocation is most accurate with GPS-enabled devices such as the iPhone, Lumia or Galaxy S phone.
function showPosition(position)
{
  var latlon=position.coords.latitude","+position.coords.longitude;

  var img_url="http://maps.googleapis.com/maps/api/staticmap?center=
"+latlon+'&zoom=14&size=400x300&sensor=false';
  document.getElementById("map").innerHTML="<img src="'+img_url+'" />";
}

function showError(error)
{
  switch(error.code)
  {
  case error.PERMISSION_DENIED:
    x.innerHTML="User denied the request for Geolocation."
    break;
  case error.POSITION_UNAVAILABLE:
    x.innerHTML="Location information is unavailable."
    break;
  case error.TIMEOUT:
    x.innerHTML="The request to get user location timed out."
    break;
  case error.UNKNOWN_ERROR:
    x.innerHTML="An unknown error occurred."
    break;
  }
}
</script>
</body>
</html>

3. Editor: Continue to study the code and observe the following points. Several things happen in this code:

- The function `getLocation()` checks to see if geolocation is supported by the user's browser.
- If geolocation is not supported, then an error message will be displayed to the user.
- If geolocation is supported, then the `getCurrentPosition()` method is run next.
- If the `getCurrentPosition()` method is successful, then it will return the user's coordinates to the function, and the `showPosition()` function will display the latitude and longitude.
- If any errors are encountered when the `getCurrentPosition()` function was run, then the `showError()` function will display the errors.

4. Firefox: Open the file `Geolocation.html` in Firefox. Grant permission if required. Click the My Location button, then click the Share Location button. Your screen should resemble Figure 7-7, depending on your location.
Tech Note: At the time of this writing, not all browsers supported geolocation. Experiment with different browsers and observe the results.

5. **Chrome**: Open the file `Geolocation.html` in the Chrome browser. Click the **My Location** button. This browser may block the Geolocation API from loading, and an error message may appear, as shown in Figure 7-8. Determine if this error is a result of browser incompatibility or if Chrome’s security settings are responsible for the error.

6. Close your browsers and text editor.

In this lab, you demonstrated how the HTML5 Geolocation API displays in a browser using your current location on a map.

### The Drag-and-Drop API

**Drag-and-drop** functionality on a Web page allows a user to move an item from one place on the page to another by dragging it across the screen and dropping it in a different area. The user can also drag an item off the Web page and drop it into an external application such as a word processor, a photo editor or another browser.
You can create drag-and-drop functionality on a Web page with the HTML5 Drag-and-Drop API. When drag-and-drop functionality is enabled on a page, various elements on that page can become draggable (by default in most browsers, any text selections, img elements and anchor elements with the href attribute). With the use of a **drop zone** — an area where dragged items can be dropped — many elements can be manipulated by the user.

There are two types of drag-and-drop functionality:

- Dragging files from the user’s computer onto a Web page
- Dragging items to different locations within the same page

Drag and drop has been available in Chrome since v4.0, in Firefox v3.5, and at least partial support in IE since v5.5, with full support since IE v10.0.

**How the Drag-and-Drop API works**

Adding drag-and-drop functionality to your Web pages requires several steps:

- First, you must specify the HTML element(s) that you want to become draggable.
- Next, you must add an event listener for the `dragstart` event on any of the HTML draggable elements.
- Lastly, you must add an event listener for the `dragover` and `drop` events on any elements that will accept a dragged item.

To begin, you define an element as able to be dragged by setting its `draggable` attribute to the value "true", as shown in the following code example in bold:

```html
<img id="drag1" src="logo\CIWLogo.png" draggable="true" ondragstart="drag(event)" width="336" height="69" />
```

You must also use the `DataTransfer` object to define the draggable element’s data type and value. Assigning an id to the element to be dragged (in this case, the `<img>` element) will allow that id to be saved in the `DataTransfer` object to be used again after the element is dropped. The `DataTransfer` object is advanced JavaScript code that will be demonstrated in the next lab, but further discussion of it is beyond the scope of this course.

In your draggable element, you also add the `ondragstart` event listener to trigger the `drag` event.

Finally, you must define a drop zone on the page. This is done by creating two events:

- `dragover` — called when you drag a page element
- `drop` — called when a page element is dropped after being dragged

In the following lab, you will add drag-and-drop functionality to a Web page. Suppose your supervisor asks you to develop a training site for the company’s sales team. The company has divisions throughout the world that manufacture various products. You suggest that the site include drag-and-drop functionality so the team members in training can drag pictures of company products to the correct company division that creates each product. You begin by experimenting with the HTML5 Drag-and-Drop API to see what it can do.
Lab 7-4: Adding drag-and-drop elements to a Web page with HTML5

In this lab, you will demonstrate the functionality of the HTML5 Drag-and-Drop API. The JavaScript code in this lab is beyond the scope of this course, but is provided to show the potential of the Drag-and-Drop API.

1. **Windows Explorer:** Open the Lab_7-4 folder from your student files.

2. **Editor:** Open the file Drag_and_Drop.html. Study the code, which appears as follows (the Drag-and-Drop API code is shown in bold):

   ```html
   <!DOCTYPE HTML>
   <html>
   <head>
   <meta charset="utf-8"/>
   <title>HTML5 Drag and Drop API Example</title>
   <style>
   #div1 {width:350px;height:70px;padding:10px;border:1px solid #aaaaaa;}
   </style>
   <script>
   function allowDrop(ev)
   {
   ev.preventDefault();
   }
   function drag(ev)
   {
   ev.dataTransfer.setData("text",ev.target.id);
   }
   function drop(ev)
   {
   var data=ev.dataTransfer.getData("text");
   ev.target.appendChild(document.getElementById(data));
   ev.preventDefault();
   }
   </script>
   </head>
   <body>
   <p>Drag the CIW image into the rectangle:</p>
   <div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
   <img id="drag1" src="logo\CIWLogo.png" alt="CIW logo" draggable="true" ondragstart="drag(event)" width="336" height="69" />
   </body>
   </html>
   ```

3. **Editor:** Continue to study the code and observe the following points. Several things happen in this code:

   - The draggable element — the CIW logo image — is set to `draggable="true"`. The `ondragstart` attribute named `drag(event)` is an event listener specifying that this element (the logo image) can be dragged. A drag event (or user action) triggers (or starts) the dragging functionality.
   - The `ondragover` event specifies where the logo can be dropped. For this reason, it is specified in the `<div>` element, which defines an area of the page. By default,
data elements cannot be dropped in other elements — they will open as a link when dropped. The `event.preventDefault()` method is called to prevent that action.

- When the logo is dropped, the drop event occurs.

4. **Firefox**: Open the file `Drag_and_Drop.html` in the browser. It should resemble Figure 7-9.

![Drag the CIW image into the rectangle.](image1)

**Figure 7-9**: CIW logo before it is dragged to new location

5. **Firefox**: Click the CIW logo, drag it into the box and drop it. Your screen should resemble Figure 7-10.

![Drag the CIW image into the rectangle.](image2)

**Figure 7-10**: CIW logo after it is dropped in new location

6. **Chrome or IE9**: Open the file `Drag_and_Drop.html` in one of these browsers. Does it work in these browsers? Why or why not? Determine whether any errors or lack of response is due to browser incompatibility, or if the browser security settings are responsible for the errors.

7. Close your browsers and text editor.

In this lab, you demonstrated how the Drag-and-Drop API works by dragging the CIW logo into a box on a Web page.
The File API

The HTML5 File API provides a standard way to interact with files on the client machine using a Web browser. It provides access to a local file system on a client for Web applications to use.

The File API can:

- Retrieve and display the properties of a file in the browser.
- Save file references when users are offline.
- Create preview thumbnail images during image uploads.
- Restrict upload sizes.
- Check for errors by comparing MIME types and verifying they are the correct type for the upload. (For example, the File API can check MIME types for an image upload program to ensure that files being uploaded are PNG, JPEG or GIF files.)

How the File API works

The File API allows you to interact with files using three interface types:

- FileList — provides a directory list of file objects. You can list information such as the file name, MIME type and size.
- FileReader — provides read-only access to individual files, such as images.
- Blob — provides the ability to divide a file into byte ranges.

The <input type="file"> element is the simplest way to reference a file. The list of selected file objects are returned as a FileList using JavaScript. Several files can be selected at once using the multiple attribute. Once the file is referenced, you can load its contents into memory using the FileReader object to access it.

In general, the FileList interface is similar to the form <input type="file" multiple> value. However, FileList always returns an array of file handles selected, regardless of the presence of the multiple attribute in the <input> element. The File API’s event model is also similar to the XMLHttpRequest event model.

In the following lab, you will use the HTML5 File API to list and read files. Suppose you want employees at your workplace to upload their timesheets to the company intranet. The File API provides the ability for users to find and select files, such as timesheets or any other type of document supported on your company intranet.

Lab 7-5: Using the HTML5 File API

In this lab, you will use the HTML5 File API to list and read files.

1. Windows Explorer: Open the Lab_7-5 folder in your student files.

2. Editor: Open the file File_API.html. Study the code, which appears as follows:

```html
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8"/>
<title>HTML5 File API Example</title>
</head>
```
3. **Editor:** Continue to study the code and observe the following points. Several things happen in this code:

- On the last line in the script element, we use `getElementById` to reference the `<input>` element with `id` attribute value of "files" and to add an event listener for the `change` event that will call the `handleFileSelect` function when the user selects a file in the file browser.

- When the `handleFileSelect` event handler is called, the code first gets the array of `FileList` objects and stores them in the `files` variable, and then creates an empty string variable called `output` to store the HTML code that will be used to display the listing of files and their properties.

- We use a `for` loop to iterate the `files` array and get the details of each file, such as the file name, type, size and last-modified date. We use this information to create a string representation of an HTML list item containing the file information.

- Once the script has iterated through the list of selected files, it adds the contents of our output string variable to the `innerHTML` property of the output element in our document, displaying the properties list from the list of files selected.

4. **Browser:** Open `File_API.html` in your browser. It should resemble Figure 7-11.
5. **Browser:** Click the Browse button, select some files from the file picker pop-up, and then click the **Open** button. Your browser should resemble Figure 7-12 but with details for the files you have chosen.

6. Close your browser and text editor.

In this lab, you demonstrated how the File API can be used to retrieve and display the properties of a file in the browser.

### The History API

The HTML5 History API allows Web designers to control the browser history. Most users will probably not notice the addition of the History API to your Web site, but this control can help make your site more user-friendly.

For example, suppose you have a Web page with a slideshow of a Top 10 list. Each item on the list is an image. The images change, but the Web page does not. If you used a language such as AJAX or JavaScript to create the slideshow and the user closes his or her Web browser or refreshes the page, then the slideshow starts from the beginning.

The History API allows you to identify each image URL within the browser history so that users start where they left off in the Top 10 list. You can also configure the API so that when users click the Back button, they access a URL that you identified.

### How the History API works

The History API allows you to add, change and delete URLs and titles in the browser history field. The History API uses several methods to control the browser history, as described in Table 7-4. Note that some of these history object methods have been available for some time, whereas a few methods are new in HTML5.
Table 7-4: History API methods

<table>
<thead>
<tr>
<th>History API Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>history.back()</td>
<td>Sends the browser back one page in the history</td>
</tr>
<tr>
<td>history.forward()</td>
<td>Sends the browser forward one page in the history</td>
</tr>
<tr>
<td>history.go(number)</td>
<td>Sends the browser forward (positive number) or backward (negative number) the specified number of pages in the history</td>
</tr>
<tr>
<td></td>
<td>The current page is history.go(0)</td>
</tr>
<tr>
<td>history.pushState(state object, title, url)</td>
<td>Changes the URL in the browser window by adding a URL to the history stack</td>
</tr>
<tr>
<td></td>
<td>New method in HTML5</td>
</tr>
<tr>
<td>history.replaceState(state object, title, url)</td>
<td>Replaces the current URL in the history with a new one, instead of adding another URL to the history stack</td>
</tr>
<tr>
<td></td>
<td>New method in HTML5</td>
</tr>
</tbody>
</table>

There are also three objects you can use to help configure the History API (see Table 7-5).

Table 7-5: History API objects

<table>
<thead>
<tr>
<th>History API Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>history.length</td>
<td>Returns the number of URLs in the history for this browser window</td>
</tr>
<tr>
<td>history.state</td>
<td>Returns the state object at the top of the history stack</td>
</tr>
<tr>
<td>window.onpopstate</td>
<td>An event handler that is called when the history changes for a document in the browser window</td>
</tr>
</tbody>
</table>

In the Top 10 List example, you can identify each image in the browser history using the pushState() method. This method has three values, but the first two are not used and are given null values. The third value is the URL you want to change:

```javascript
history.pushState(null, null, "http://www.ciw.org/");
```

However, the History API only adds the location to the history list. You must also write a script for users to directly access the URL. For example, you can append a parameter to the URL that corresponds to each slide image:

```javascript
http://www.ciw.org/top10list?slide=2
```

Then write a script to read the new parameters and load the corresponding slide. These actions will cause three results:

- If a user closes her browser, reopens it and returns to the site, she will return to the same slide image where she left off.
- If a user refreshes her browser, she will remain on the same slide image.
- If a user enters the URL with the new parameter, she will open the corresponding slide image.

The History API may seem dull compared to Canvas, Geolocation and Drag-and-Drop, but it helps create a site that is easier to navigate. Use it whenever you need a more user-friendly site.

*The JavaScript code required for the History API script is beyond the scope of this course. You can learn more about the History API and see an example called “Dive into "Dogs" by visiting the Dive Into HTML5 site at http://diveintohtml5.info/history.html.*
Retrieving Data with XMLHttpRequest

Remember that focusing your design efforts on usability makes your sites easier to use. Requesting and loading data is another important aspect of site usability. You can use the XMLHttpRequest object to retrieve data in a client browser.

The XMLHttpRequest object is used to exchange data with a server. Scripting languages such as JavaScript use this object to send HTTP or HTTPS requests to a Web server. The server response data is then loaded back into the script.

The XMLHttpRequest object is particularly useful because it allows Web developers to update pages without reloading them. A client device can request and receive data from a server after it has loaded a Web page. This data is exchanged with the server in the background while the user continues to interact with the site.

For example, if you type a character in an input field, an XMLHttpRequest can access a server file to return suggestions. All modern browsers support XMLHttpRequest.

The following syntax is used to create the XMLHttpRequest object:

```javascript
xmlhttp=new XMLHttpRequest();
```

**Also known as AJAX**

This technique for requesting and reusing data using the JavaScript XMLHttpRequest object with HTML pages and CSS is often called AJAX (Asynchronous JavaScript and XML). It is considered asynchronous because it enables the page to retrieve and update data in the background, without disturbing the page's display or behavior.

Keep in mind that AJAX can sometimes negatively affect usability because the user may not realize that updates are coming, since they are processing in the background. The main purpose is to reduce the load on the servers, reduce bandwidth and speed load times for users.

Although the XMLHttpRequest object's name includes XML and HTTP, it can return any type of textual data. Common types of requested data include XML or JSON (JavaScript Object Notation), HTML, and plain text.

Manipulating Data with jQuery

jQuery is a JavaScript library used to simplify JavaScript programming. It requires knowledge of HTML, CSS and JavaScript. It is intended to help developers perform common JavaScript tasks with less code, including HTML/DOM manipulation, CSS manipulation, AJAX calls, HTML event methods, animations, utilities and effects.

Recall that the Document Object Model (DOM) is a W3C-defined interface that allows scripts to dynamically access and update the content, structure and style of HTML and XML documents.
jQuery is designed to work similarly on all major browsers, so compatibility issues should be rare. Cross-browser issues were addressed when the jQuery library was developed.

jQuery can be useful when you are developing pages with HTML5 APIs. jQuery’s DOM-related methods provide a simpler way to access and manipulate HTML elements and attributes. HTML elements are selected and jQuery performs various actions on them using the following code.

\[
\$(\text{selector}).\text{action()}
\]

Table 7-6 shows the various components of this jQuery code.

<table>
<thead>
<tr>
<th>jQuery Code Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>Specifies and accesses jQuery</td>
</tr>
<tr>
<td>(selector)</td>
<td>Pinpoints the HTML element to query Based on CSS selectors</td>
</tr>
<tr>
<td>action()</td>
<td>Determines the action to execute on the HTML element</td>
</tr>
</tbody>
</table>

The jQuery syntax starts with a dollar sign and parentheses, as described in Table 7-7.

<table>
<thead>
<tr>
<th>jQuery Syntax Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&quot;li&quot;.hide()</td>
<td>Hides all &lt;li&gt; elements.</td>
</tr>
<tr>
<td>$&quot;.answers&quot;.hide()</td>
<td>Hides all elements with class=&quot;answers&quot;</td>
</tr>
<tr>
<td>$&quot;#confidential&quot;.hide()</td>
<td>Hides the element with id=&quot;confidential&quot;</td>
</tr>
</tbody>
</table>

jQuery selectors allow you to find and manipulate HTML element(s) based on classes, ids, attributes, values, etc. The jQuery selectors are based on CSS selectors, but they also include custom selectors. Table 7-8 describes common jQuery selectors.

<table>
<thead>
<tr>
<th>jQuery Selector</th>
<th>Description</th>
</tr>
</thead>
</table>
| $\"a\"         | The element selector
Selects instances of the specified element (e.g., <a>) based on the element name |
| $\"#custname\" | The #id selector
Selects instances of elements based on the id attribute (e.g., custname) of the element |
| $\".demo\"     | The .class selector
Selects elements with the specified class (e.g., demo) |
| $\"*\"         | Selects all HTML elements in the document |
| $(this)         | Selects the current HTML element |
| $\"a.demo\"    | Selects all instances of the specified element (<a>) with the specified class (demo) |
| $\"a:first\"   | Selects the first instance of the specified element (<a>) on a page |
Table 7-8: jQuery selectors (cont’d)

<table>
<thead>
<tr>
<th>jQuery Selector</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$('[href]')</code></td>
<td>Selects all elements with the specified attribute (href)</td>
</tr>
<tr>
<td><code>$('a[target='_blank']')</code></td>
<td>Selects all instances of the specified element (&lt;a&gt;) with the specified attribute (target) and specified value (_blank)</td>
</tr>
</tbody>
</table>

**Installing jQuery**

You download the jQuery library as a single JavaScript file to the directory where it will be used. You do this by referencing the file in an HTML `<script>` tag within the `<head>` section:

```html
<head>
  <script src="jquery-1.8.21.min.js"></script>
</head>
```

Instead of downloading the library file, you can also reference a **Content Delivery Network (CDN)** from Google or Microsoft with a `<script>` tag in the `<head>` section:

```html
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.21/jquery.min.js"></script>
```

Or:

```html
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.8.21.min.js"></script>
```

Referencing the CDN address is preferred because site visitors may have already cached the library in their browser from other sites they have visited.

**Using the document.ready event**

jQuery code should not run until the document is finished loading. To ensure this occurs, jQuery methods are placed within a `document.ready` event:

```javascript
$(document).ready(function(){
  // jQuery code
});
```

You can also enter a shorter version of the `document.ready` event:

```javascript
$(function(){
  // jQuery code
});
```

Either way will work, but some prefer the longer version because it is easier to find the jQuery code in a crowded document.

**Retrieving and changing content**

Three jQuery methods for DOM manipulation are used to retrieve (i.e., get) and set (i.e., change) content, as described in Table 7-9.
Table 7-9: jQuery methods to retrieve and change content

<table>
<thead>
<tr>
<th>jQuery Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text()</td>
<td>Sets or returns the text content of selected elements</td>
</tr>
<tr>
<td>html()</td>
<td>Sets or returns the content of selected elements (including HTML markup)</td>
</tr>
<tr>
<td>val()</td>
<td>Sets or returns the values of form fields</td>
</tr>
</tbody>
</table>

**Retrieving and changing attributes**

The jQuery `attr()` method is used to get and set attribute values. The `attr()` method can set multiple attributes simultaneously.

**Adding HTML content**

Four jQuery methods are used to add new content, as described in Table 7-10.

Table 7-10: jQuery methods to add HTML content

<table>
<thead>
<tr>
<th>jQuery Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>append()</td>
<td>Inserts content at the end of the selected elements</td>
</tr>
<tr>
<td>prepend()</td>
<td>Inserts content at the beginning of the selected elements</td>
</tr>
<tr>
<td>after()</td>
<td>Inserts content after the selected elements</td>
</tr>
<tr>
<td>before()</td>
<td>Inserts content before the selected elements</td>
</tr>
</tbody>
</table>

**Removing content**

Table 7-11 describes the two jQuery methods to remove elements and content.

Table 7-11: jQuery methods to remove elements and content

<table>
<thead>
<tr>
<th>jQuery Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove()</td>
<td>Removes the selected element (and its child elements)</td>
</tr>
<tr>
<td>empty()</td>
<td>Removes the child elements from the selected element</td>
</tr>
</tbody>
</table>

**Using jQuery**

Although some of the JavaScript code used in this lesson is beyond the scope of this course, it provides a demonstration of the capabilities of APIs and jQuery. You can put the coding knowledge you have to greater use by employing tools such as jQuery to extend your own capabilities in Web design.

**CIW Online Resources – Online Exercise**

Visit CIW Online at http://education.Certification-Partners.com/CIW to complete an interactive exercise that will reinforce what you have learned about this topic.

*Exercise 7-3: AJAX and jQuery concepts*
Case Study

HTML5 APIs Go to School

Tabitha works in the Human Resources department for a large enterprise organization with more than 50,000 employees. Her company is heavily regulated by the federal government, and must train the company workforce on new regulations and standards issued by various regulatory agencies.

As new banking regulations are passed, Tabitha must teach employees the various terms and processes required. The training will be posted on the company intranet for all employees to access. All content must be created using HTML5.

Tabitha’s supervisor asks her to create a Web page that tests employees on the latest regulatory terms. Tabitha wants to create a page on which users must match the terms in the left column to the proper definitions in the right column.

Consider this scenario and answer the following questions:

- Which HTML5 API could Tabitha use to create the matching exercise she wants?
- How would she set up the matching exercise using that API?
- What other HTML5 APIs might she incorporate into the training site? How could they be used in other training exercises?
Lesson 7: Using HTML5 APIs

Lesson Summary

Application project
This lesson discussed some more advanced Web technologies, including HTML5 APIs, JavaScript and the W3C DOM. Visit the following sites to learn more about these technologies:

- World Wide Web Consortium (www.w3.org)
- HTML5 Rocks – A Google Project (www.html5rocks.com)
- HTML5 Demos and Examples (http://html5demos.com)
- W3 Schools (www.w3schools.com)

Skills review
In this lesson, you studied Web-based HTML5 API technologies that extend the capabilities of your Web pages. You learned about the Document Object Model (DOM) and the importance of DOM Web browser support and APIs. You experimented with some popular HTML5 APIs such as Canvas, Offline AppCache, Geolocation, Drag-and-Drop, File and History. You also learned about some advanced techniques for retrieving and manipulating data using the XMLHttpRequest object (AJAX) and jQuery.

Now that you have completed this lesson, you should be able to:

✓ 7.1: Define the Document Object Model (DOM).
✓ 7.2: Define application programming interface (API), and list the common HTML5 APIs.
✓ 7.3: Use the HTML5 Canvas API for advanced techniques.
✓ 7.4: Develop offline Web applications using the HTML5 AppCache API.
✓ 7.5: Use the HTML5 Geolocation API to obtain geographic information about a user.
✓ 7.6: Create drag-and-drop Web pages using the HTML5 Drag-and-Drop API.
✓ 7.7: Use the HTML5 File API to interact with files.
✓ 7.8: Use the HTML5 History API to manipulate a browser’s history.
✓ 7.9: Retrieve and manipulate HTML5 data using XMLHttpRequest and jQuery.

CIW Practice Exams
Visit CIW Online at http://education.Certification-Partners.com/CIW to take the Practice Exams assessment covering the objectives in this lesson.

Objective 7.01 Review  Objective 7.06 Review
Objective 7.02 Review  Objective 7.07 Review
Objective 7.03 Review  Objective 7.08 Review
Objective 7.04 Review  Objective 7.09 Review
Objective 7.05 Review

Note that some objectives may be only partially covered in this lesson.
Lesson 7 Review

1. How can avoiding browser plug-ins affect mobile devices?

2. What is the term for the API used to standardize the way that JavaScript applications can refer to HTML documents?

3. What is the first step to set up a site and its applications to work offline?

4. What are the two types of drag-and-drop functionality?

5. What can you do with the History API?

6. What is the difference between XMLHttpRequest and jQuery?