

HTML5 Training for Web Developers

Integrated APIs

Lesson 1, Activity 2: Offline Application API

The HTML5 Specification includes an API for creating offline applications. The purpose is two-fold:

1. Allow users to access your web application when they are offline.
2. Make your web applications faster by taking advantage of local caching.

There are four steps involved in turning a regular HTML page into an offline application:

1. Create a cache manifest file.
2. Reference the cache manifest file in your HTML5 document.
3. Write JavaScript to manage caching.

Cache Manifest File

The cache manifest file is a plain text file with a [manifest](#) extension. It is structured as follows:

```
CACHE MANIFEST
#Use pound signs for comments

#Explicitly cached entries.
CACHE:
index.html
style.css
script.js
image.png

#Online-only Resources.
NETWORK:
login.php
/online-only/

#Fallback files (use only when can't access online files)
FALLBACK:
online.js offline.js
```

Note that you must configure your web server to deliver [manifest](#) files using the "text/cache-manifest" mime type.

In Apache, you can use `AddType text/cache-manifest .manifest`

In IIS, you can add Mime types through Computer Management.

The HTML File

Referencing the manifest file in your HTML document is easy:

```
<html manifest="example.manifest">
```

Managing ApplicationCache with JavaScript

You access the application cache through the `window.applicationCache` object. It includes a `status` property indicating the current state of the cache. As the `status` changes, the following events are fired:

1. cached
2. checking
3. downloading
4. error
5. noupdate
6. obsolete
7. progress
8. updateready

You can catch these events using event listeners: `window.applicationCache.addEventListener("error", fnCall, false);`.

A Sample Application

Take a look at our sample application files below:

Code Sample:

[html5-apis/Demos/offline.html](#)

```

<!DOCTYPE HTML>
<html manifest="example.manifest">
<head>
<meta charset="UTF-8">
<title>Offline Application API</title>
<script src="offline/script.js" type="text/javascript"></script>
<link href="offline/style.css" rel="stylesheet">
</head>
<body>
<h1>No Heading</h1>
<ol id="output"></ol>

</body>
</html>

```

Code Sample:[html5-apis/Demos/example.manifest](#)

```

CACHE MANIFEST
#Version 1.2
offline.html
offline/style.css
offline/script.js

NETWORK:
*

FALLBACK:
Images/online.gif Images/offline.gif

#COMMENTS
##We could use the following fallback settings to use
##a different stylesheet and script when offline
##style.css offline-style.css
##script.js offline-script.js

```

Code Sample:[html5-apis/Demos/offline/script.js](#)

```

window.addEventListener("load",function() {
    document.getElementsByTagName("h1")[0].innerHTML="Hello World";
},false);

var appCache = window.applicationCache;

appCache.addEventListener("error", function() {
    alert("Cache failed to update");
    document.getElementById("output").innerHTML+="<li>a cache error has occurred</li>";
}, false);

appCache.addEventListener("updateready", function() {
    var refresh = confirm("An updated version is ready. Press OK refresh your browser.");
    if (refresh) {
        location.reload();
    }
    document.getElementById("output").innerHTML+="<li>cache ready to be updated</li>";
}, false);

appCache.addEventListener("cached", function() {
    document.getElementById("output").innerHTML+="<li>application cached</li>";
}, false);

appCache.addEventListener("checking", function() {
    document.getElementById("output").innerHTML+="<li>checking cache</li>";
}, false);

appCache.addEventListener("downloading", function() {
    document.getElementById("output").innerHTML+="<li>cache downloading</li>";
}, false);

appCache.addEventListener("noupdate", function() {
    document.getElementById("output").innerHTML+="<li>no cache update</li>";
}

```

```

}, false);

appCache.addEventListener("obsolete", function() {
  document.getElementById("output").innerHTML+="- manifest file returned a 404 or 410</li>";
}, false);

appCache.addEventListener("progress", function() {
  document.getElementById("output").innerHTML+="

```

Code Sample:

<html5-apis/Demos/offline/style.css>

```

body {
  background-color:#f00;
}

#output {
  background-color:white;
  float:left;
  width:200px;
  margin-right:25px;
}

```

And here are the two images, the first of which is shown if the browser can connect to the site:



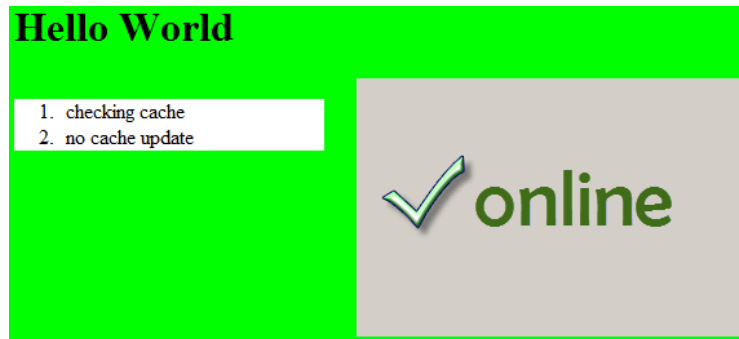
To test this application, you must access it through a web server (e.g., localhost). Here's how it works:

1. The first time you visit the page, all the files will download from the server and get cached:

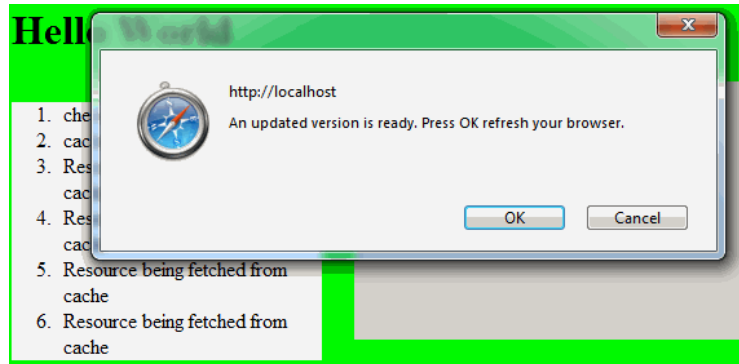
Hello World

1. checking cache
2. cache downloading
3. Resource being fetched from cache
4. Resource being fetched from cache
5. Resource being fetched from cache
6. Resource being fetched from cache
7. application cached

2. Refresh the browser:



3. Modify the CSS document (e.g., change the background color) and refresh. Nothing changes. All the files were fed from cache.
4. Modify the cache manifest file to force it to be redownloaded (e.g., change the version comment from version 1.0 to version 1.1). Refresh the browser:



- Notice that the resources are being fetched again, but the page has not updated.
5. If you press **Cancel** on the confirm dialog, the page will not update. Instead, press **OK** to get the page to update:



6. Our manifest file indicates that offline.gif should be displayed if the browser is offline. To see this, you need to go offline, which is hard to do when you're working locally.

Lesson 1, Activity 3: Drag and Drop API

The HTML5 Drag and Drop API provides a great illustration of the downside of the "paving the cowpaths" approach of HTML5. It standardizes the drag-and-drop API originally created in Internet Explorer 5 (yup, 5). As other browsers implemented the same API, it was decided that this was a cowpath worth paving. We explain basic usage here, which ironically doesn't work very well in Internet Explorer 9, but does work in the other HTML5-compliant browsers.

To build a drag-and-drop application, you need to do the following:

1. Create a draggable element.
2. Create an element to be your drop zone.
3. Capture and respond to drag-and-drop events.
4. Optionally change styles during drag-and-drop process.

It's easiest to understand by looking at a small demo application:

Code Sample:

html5-apis/Demos/drag-and-drop.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Drag and Drop API</title>
<script src="../../html5-common/dateFormat.js" type="text/javascript"></script>
<script src="dragdrop/script.js" type="text/javascript"></script>
<link href="dragdrop/style.css" rel="stylesheet">
</head>
<body>
<h1>Drag and Drop</h1>
<div id="dropzone">Last Dropped: <time>never</time></div>
<div id="drag" draggable="true"></div>
<ol id="output"></ol>
</body>
</html>
```

We have two divs:

1. "dropzone" for capturing the drop events
2. "drag" for dragging - notice the **draggable="true"** attribute

Code Sample:

html5-apis/Demos/dragdrop/style.css

```
#dropzone {
  border:1px solid black;
  width:100px;
  height:100px;
  float:left;
  margin-right:10px;
  text-align:center;
}

#drag {
  background-color:red;
  width:50px;
  height:50px;
  float:left;
  cursor:move;
}

#output {
  background-color:white;
  float:left;
  width:200px;
  margin-right:25px;
}

time {
  text-decoration:underline;
}
```

Nothing too exciting here. Just some basic styling.

Code Sample:

[html5-apis/Demos/dragdrop/script.js](#)

```
var dropzone, drag;
window.addEventListener("load", dragDrop, false);
function dragDrop() {
    dropzone = document.getElementById("dropzone");
    drag = document.getElementById("drag");
    drag.addEventListener("dragstart", handleDragStart, false);
    drag.addEventListener("drag", handleDrag, false);
    drag.addEventListener("dragend", handleDragEnd, false);
    dropzone.addEventListener("dragenter", handleDragEnter, false);
    dropzone.addEventListener("dragover", handleDragOver, false);
    dropzone.addEventListener("dragleave", handleDragLeave, false);
    dropzone.addEventListener("drop", handleDrop, false);
}

function handleDragStart(e) {
    this.style.backgroundColor="green";
    document.getElementById("output").innerHTML+="

```

Here is where the action takes place. We capture these events for the draggable element:

1. dragstart
2. drag
3. dragend

And these events for the drop zone:

1. `dragenter`
2. `dragover`
3. `dragleave`
4. `drop`

Note that the `drag` and `dragover` events fire repeatedly as the draggable element is dragged. That's why we use the booleans `dragReported` and `dragOverReported` to stop the reporting. If you leave those out, you'll likely crash your browser.

The excitement happens in the `handleDrop()` function, which writes out the time of the drop. You could do anything at this point; for example, change the position of the element or add an item to a shopping cart.

You may have noticed the two calls to `cancel()`:

1. In `handleDragOver()`, that allows us to drop the draggable element.
2. In `handleDrop()`, that prevents the browser from trying to navigate to another page when the element is dropped.

The unfortunate reality is that the drag-and-drop API in HTML5 does not make developers' lives easier. If you're working a lot with drag-and-drop, you're almost definitely better off using a framework like jQuery or YUI. However, the concepts covered in this section are the same as you'll see in those libraries.