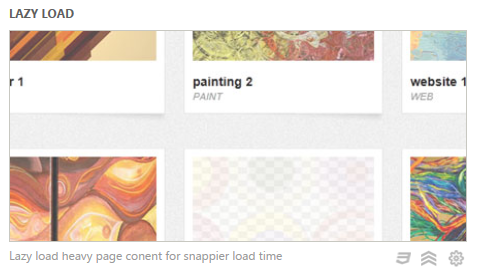
|  |
| --- |
| HTML5 Elements Cookbook |
| H5E Experiment: Lazy Loading |
| Version 1.0  Published 15 July, 2011 |
|  |

|  |
| --- |
|  |
| H5E Scout Team  Windows Web Partners  Microsoft Corporation  Microsoft Confidential |

# Executive Summary

Lazy loading is a technique that is used to improve the user’s perception of load time for a web page. This is particularly useful for sites that use large numbers of images. You can use HTML5 and related technologies to perform lazy loading as well as a variety of simple, but powerful animation techniques that move an HTML-based site experience closer to native application or component-based performance.

**

## Scope

This document describes a basic implementation of lazy loading using jQuery, which was conducted by the H5E scout team using Clarity Consulting. Our objective is to test the limits of HTML5 solving real-world partner questions. This document assumes an existing knowledge of JavaScript and jQuery. This document does not supersede any requirements, guidelines, or instructions provided by the IE team to produce similar behavior.

## Keywords

Lazy loading, lazy load, jQuery, HTML5

## Contact us

To contact us for questions or support, please email Chewy Chong ([ChewyC](mailto:ChewyC?subject=HTML5%20Cookbooks)). Feedback is welcome.

Contents

Executive Summary 2

Scope 2

Keywords 2

Contact us 2

Lazy Loading Overview 4

Walkthrough 4

How do I build this? 6

Files in this solution 6

Using jQuery and the DOM for lazy loading based on the viewport 7

Conclusions and Recommendations 11

Recommendations 11

Resources 11

Relevant Web sites and specifications 11

Microsoft Resources 11

Appendix A: About H5E 12

What is an HTML5 Elements Cookbook? 12

Contact us 12

Copyright and trademark information 12

Document Revision History 13

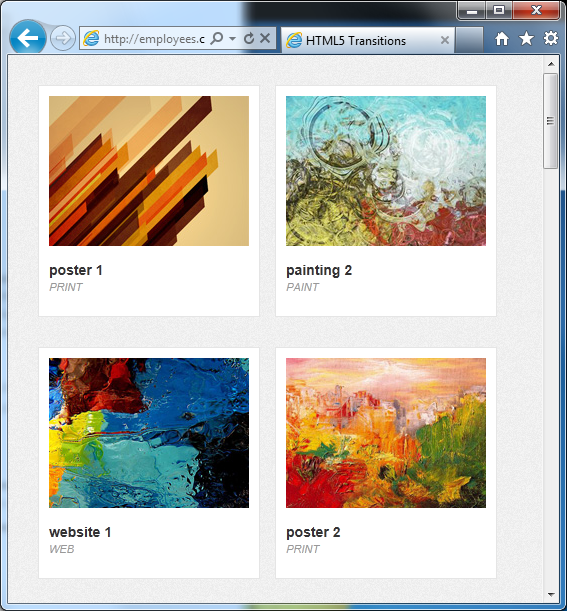
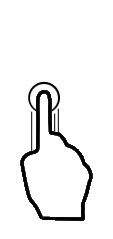
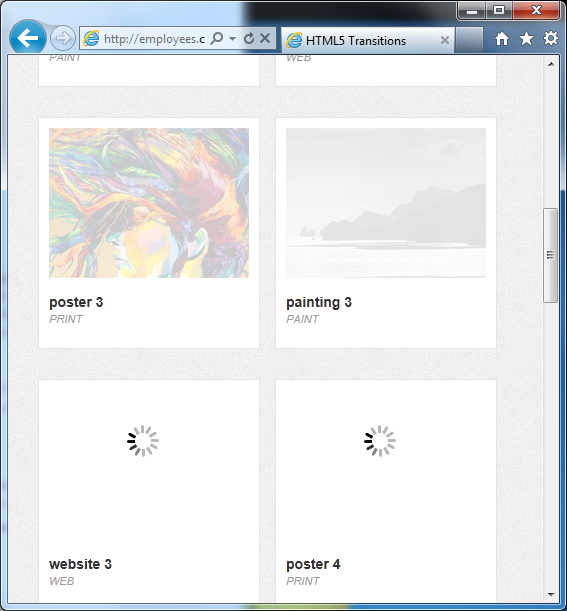
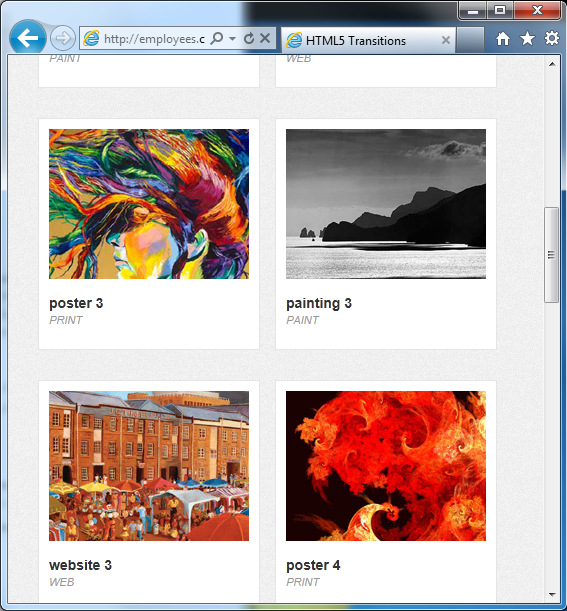
# Lazy Loading Overview

Lazy loading in this context is a technique that is used to improve the performance of a web page by delaying the loading of page elements that are not required for the initial page impression. In this example, if a page includes a set of images that are below the immediate viewing area (“below the fold”), you can delay loading these images until the user scrolls to the lower sections of the page.

The aspirational experience that is replicated here mirrors the behavior of native applications or component-based solutions that provide a sense of polish and richness that web pages often lack. Using some relatively simple logic, this example demonstrates that this effect is well within the grasp of developers that are interested in making their HTML-based experiences more app-like. Rather than these being technical challenges, in some cases it is more a design and experience challenge to make the experience more immersive through subtlety, which we will call out in this document.

## Walkthrough

The Lazy Loading sample does the following:

1. The user navigates to the web page, and the initial content that is visible within their browser window is loaded. The images that are not in this viewable area are not loaded yet.   
   
2. When the user scrolls down, the page dynamically loads the images they come into view. A placeholder animation is inserted while the image loads.  
     
   
3. The image load completes. This cycle repeats each time the user scrolls new content into the viewport.   
     
   

# How do I build this with HTML5?

This HTML5 Experiment focuses on the using JavaScript to detect the viewport, and if the images are within that viewport, jQuery and Ajax are used to display the image. When the user navigates further down the page, the images that come into view are acquired and displayed dynamically. The display detection logic for the example is tied to the scroll event, though in future iterations this could be expanded to include other events like window resizing on the desktop.

***NOTE****: Many of the HTML5 Experiments are still under development. Our initial target is to build prototypes that work on current HTML5-supported browsers and tablet devices, including the iPad. The experiments do not aim for full cross-browser support at this stage, but we will likely build in graceful degradation in future updates.*

To ensure that users have a similar cross-browser experience, the following table describes the compatibility of the solutions in this document:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HTML5 Logo**HTML5 Feature** | IE6.0 | IE7.0 | IE8.0 | IE9.0 | IEPP | Chrome11 | | Chrome12 | Safari4.x | Safari5.x | Firefox3.6 | Firefox4.x | Firefox5.x | Opera11 |
| jQuery |  |  |  |  |  |  | |  |  |  |  |  |  |  |
|  | | | | | | | Full Support | | | | | | |  |
|  | | | | | | | Supported with Shim | | | | | | |  |
|  | | | | | | | No current support | | | | | | |  |

***NOTE****: In general, shims are not incorporated into the HTML5 Experiments at this stage. If a shim or polyfill is required for cross-browser support, see* [*http://browserexperiments.com*](http://browserexperiments.com) *for details on shim implementation.*

## Primary files in this solution

### Source Location

<https://github.com/molant/BrowserExperiments/tree/master/cookbook/2_LazyLoad>

### Sample Location

<http://employees.claritycon.com/eklimczak/html5Cookbook/2_LazyLoad/>

## Using jQuery and the DOM for lazy loading based on the viewport

This sample uses jQuery to handle the event load and simple animation (an opacity change) to complete the effect. When the user scrolls beyond the initial viewport, the images that move into the viewport are displayed. This is accomplished through DOM manipulation to change the class of the element, which then displays the corresponding image.

One minor trick that is used in this sample is the use of an animated .gif file to simulate loading. While this could have been done any number of ways via CSS, a sprite sheet, or similar, the developer chose the simplest (and cleanest) path – a very small animated .gif.

### Initialization and building DOM items

On initialization of the page, the initial data is assembled and DOM manipulation functions are invoked. **GetDataAsync()** uses Ajax to acquire the data elements from data.xml. After the data is acquired, **ParseData()** loops through the XML to build the necessary DOM elements, and calls **BuildDomItem()** to create the DOM elements the images are initially identified as members of the placeholder CSS class, which displays an animated .gif to simulate a load animation.

function Init() {

GetDataAsync();

}

function GetDataAsync() {

$.ajax({

type: "GET",

url: "data/data.xml",

dataType: "xml",

success: function (data) {

ParseData($(data));

}

});

}

function ParseData(xmlData) {

//Create empty object

var tempItem = new Object();

//loop through xml

xmlData.find('item').each(function () {

var temp = $(this);

tempItem.id = temp.attr('id');

tempItem.name = temp.find('name').text();

tempItem.category = temp.find('category').text();

tempItem.url = temp.find('url').text();

tempItem.placeHolder = "images/loader.gif";

tempItem.isLoaded = false;

BuildDomItem(tempItem);

return true;

});

ContentLoader.Init($("#list"));

}

function BuildDomItem(dataItem) {

var domItem = $('<div id="item\_' + dataItem.id + '" class="listItem"></div>').appendTo("#list");

var imgContainer = $('<div class="imgContainer" />').appendTo(domItem);

$('<image class="placeholder" originalUrl="' + dataItem.url + '" />').attr('src', dataItem.placeHolder).appendTo(imgContainer);

$('<h2 />').html(dataItem.name).appendTo(domItem);

$('<h3 />').html(dataItem.category).appendTo(domItem);

}

### Detecting a scroll event

To detect a scroll change, the **scroll** event triggers **InViewPort()** to determine if the elements that are within the viewport have been loaded. If they have not, **LoadImageWithCallback()** is called, and the custom **appear** event is triggered.

$(window).bind("scroll", function (event) {

if (elements != undefined && elements.length > 0) {

elements.each(function () {

var self = this;

if (ContentLoader.InViewPort(self) && !self.loaded) {

ContentLoader.LoadImageWithCallback($(self).attr("originalUrl"), function (img) {

$(self).trigger('appear');

});

}

var temp = $.grep(elements, function (element) {

return !element.loaded;

});

elements = $(temp);

});

### Determining if an element is in the viewport

To determine whether the element is in the viewport , the **InViewPort()** method is invoked. This method uses jQuery to acquire the necessary references and uses the offset properties to determine if the element is within the boundaries of the viewport.

ContentLoader.InViewPort = function (element) {

var uiElement = $(element);

var offset = uiElement.offset();

var win = $(window);

var windowTop = win.scrollTop();

var windowLeft = win.scrollLeft();

//Top Bottom

if (offset.top < windowTop) {

if (offset.top + uiElement.height() >= windowTop) {} else {

return false;

}

}

else {

if (offset.top <= windowTop + win.height()) {} else {

return false;

}

}

//Left Right

if (offset.left < windowLeft) {

if (offset.left + uiElement.width() >= windowLeft) {} else {

return false;

}

}

else {

if (offset.left <= windowLeft + win.width()) {} else {

return false;

}

}

return true;

}

### Loading each image with a callback

To load an image, **LoadImageWithCallback()** is invoked by passing the **originalUrl** property for the DOM div to be displayed as the callback parameter.

if (ContentLoader.InViewPort(self) && !self.loaded) {

ContentLoader.LoadImageWithCallback($(self).attr("originalUrl"), function (img) {

$(self).trigger('appear');

});

**LoadImageWithCallback()** builds a new Image object based on the URL that was passed in as a parameter.

ContentLoader.LoadImageWithCallback = function (url, callback) {

var image = new Image();

image.onload = function () {

callback(image);

}

setTimeout(function () {

image.src = url;

}, 100 + Math.random() \* 300);

return image

}

# Conclusions and Recommendations

This experiment’s goal was to test the level of difficulty in implementing an experience that is usually provided by a component like Flash or Silverlight. In a production scenario, a more common solution would be to use a jQuery plugin that would likely handle much of this, but this solution was also viable.

## Recommendations

While this experiment focuses on the goal of making it work on current devices and then looking at making the solution more robust in future iterations, in retrospect one technique that would definitely have been used is to use the animations that were produced here as a fallback technique. The better solution would likely be to detect the presence of elements like WebKit to take advantage of hardware acceleration, and then fall back to this technique for browsers without hardware acceleration.

# Resources

## Relevant Web sites and specifications

|  |  |
| --- | --- |
| jQuery reference | <http://docs.jquery.com/Main_Page> |
|  |  |

## Microsoft Resources

|  |  |
| --- | --- |
| H5E Primary Contact | Chewy Chong ([ChewyC](mailto:%20chewyc?subject=HTML5%20Experiments%20Cookbooks)) |
| H5E Development Contact | Anton Molleda Quintana ([v-anmoll](mailto:v-anmoll?subject=HTML5%20Experiments%20Cookbooks)) |

# Appendix A: About H5E

## What is an HTML5 Elements Cookbook?

Each HTML5 Elements Cookbook reflects a case study of an aspirational experience that is provided by a native or component-based application. The HTML5 Experiments that are conducted by the H5E team use HTML5 and related technologies to replicate these experiences. Our primary objective is to learn from these experiments to determine if an HTML5 alternative to component-based or native implementations is both possible, and practical. Each Cookbook provides a description of the element and technical details of the HTML5 replication of that feature. We also include recommendations on whether it makes sense to pursue this approach.

## Contact us

If you need assistance with technical solutions or have a best practice to share, please contact us by sending email to Chewy Chong ([ChewyC](mailto:chewyc?subject=H5E%20Cookbooks%20and%20Documentation)).

## Copyright and trademark information

This document is Microsoft Confidential.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

 2011 Microsoft Corporation. All rights reserved.

# Document Revision History

|  |  |  |
| --- | --- | --- |
| **Reviser** | **Date** | **Revisions** |
| **v-jgeige** | 14 July 2011 | Initial draft |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |