Advanced JavaScript & JQuery

Chapter 15

Objectives

1. JavaScript Pseudo-Classes
2. jQuery Foundations
3. AJAX
4. Asynchronous File Transmission
5. Animation
6. Backbone MVC Frameworks

Section 1 of 6

JAVASCRIPT PSEUDO-CLASSES

Using Object Literals

Without Classes

An object can be instantiated using object literals.

Object literals are a list of key-value pairs with colons between the key and value with commas separating key-value pairs.

Object literals are also known as **Plain Objects** in jQuery.

Object Oriented Design

Without Classes

- JavaScript has no formal class mechanism.

Simple Variable
var car = "Fiat";

Then this
var car = {type:"Fiat", model:500, color:"white"};

Properties
car.name = Fiat
car.model = 500
car.weight = 850kg
car.color = white

Methods:
car.start()
car.drive()
car.brake()
car.stop()

Using Object Literals

Consider a die (single dice)

```
//define a 6 faced dice, with a red color.

var oneDie =     { color : "FF0000",

                   faces : [1,2,3,4,5,6]

                 };
```

These elements can be accessed using dot notation.

For instance

```
oneDie.color="0000FF";
```

Emulate Classes with functions

We told you this would get weird

Use functions to encapsulate variables and methods together.

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];
}
```

LISTING 15.1 Very simple Die pseudo-class definition as a function

Instantiation looks much like in PHP:

var oneDie = new Die("0000FF");

Emulate Classes with functions

Add methods to the object – mimic methods

One technique for adding a method inside of a class definition is by assigning an anonymous function to a variable

```
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];

   // define method randomRoll as an anonymous function
   this.randomRoll = function() {
      var randNum = Math.floor((Math.random() * this.faces.length)+ 1);
      return faces[randNum-1];
   };
}
```

LISTING 15.2 Die pseudo-class with an internally defined method

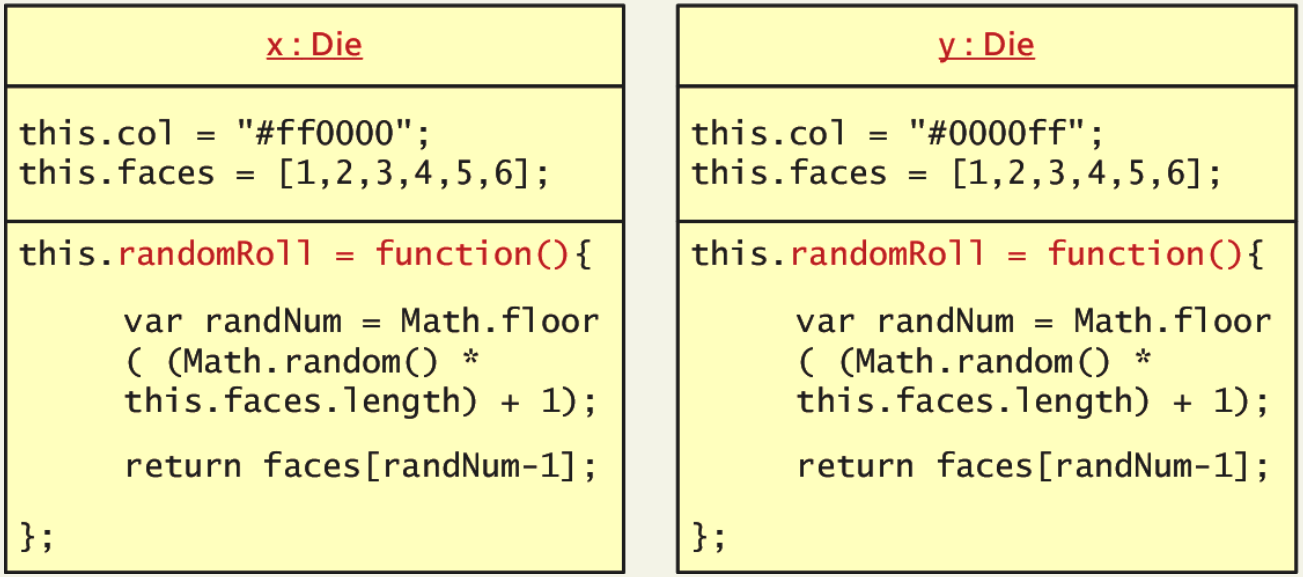
```
var oneDie = new Die("0000FF");
console.log(oneDie.randomRoll() + " was rolled");
```

Emulate Classes with functions

Not very efficient

Defining methods as we have shown is not a memory-efficient

each inline method is redefined for each new object



Using Prototypes

Prototypes

So you can use a *prototype* of the class.

- **Prototypes** are used to make JavaScript behave more like an object-oriented language.
- The prototype properties and methods are defined *once* for all instances of an *object*.
- Every object has a prototype

Using Prototypes

moving the randomRoll() function into the **prototype**.

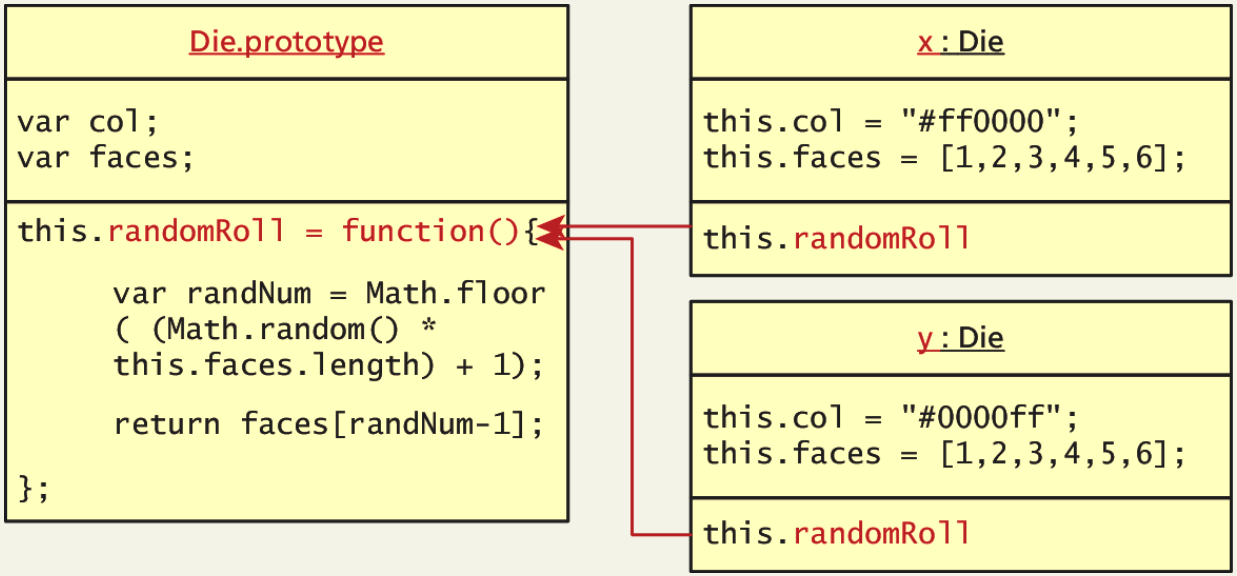
```
// Start Die Class
function Die(col) {
   this.color=col;
   this.faces=[1,2,3,4,5,6];
}

Die.prototype.randomRoll = function() {
   var randNum = Math.floor((Math.random() * this.faces.length) + 1);
   return faces[randNum-1];
};
// End Die Class
```

LISTING 15.3 The Die pseudo-class using the prototype object to define methods

Using Prototypes

No duplication of methods

Since all instances of a Die share the same prototype object, the function declaration only happens one time and is shared with all Die instances.



A prototype is an object from which other objects inherit.

More about Prototypes

Extend any Object

Every object (and method) in JavaScript has a prototype.

In addition to using prototypes for our own pseudo-classes, prototypes enable you to *extend* existing classes by adding to their prototypes

```
String.prototype.countChars = function (c) {
   var count=0;
   for (var i=0;i<this.length;i++) {
      if (this.charAt(i) == c)
         count++;
   }
   return count;
}
```

LISTING 15.4 Adding a method named countChars to the String class

More about Prototypes

Extending String, for example

Now any **new** instances of String will have this method available to them while existing strings will not.

Now we can use the new method in all new Strings.

The following example will output

Hello World has 3 letter l's.

```
var hel = new String("Hello World”);
console.log(hel + "has" + hel.countChars("l") + " letter l's");
```

Section 2 of 6

JQUERY FOUNDATIONS

jQuery

Framework

A **library** or **framework** is software that you can utilize in your own software, which provides some common implementations of standard ideas.

Many developers find that once they start using a framework like jQuery, there's no going back to "pure" JavaScript because the framework offers so many useful shortcuts and succinct ways of doing things

jQuery

A 1 slide history

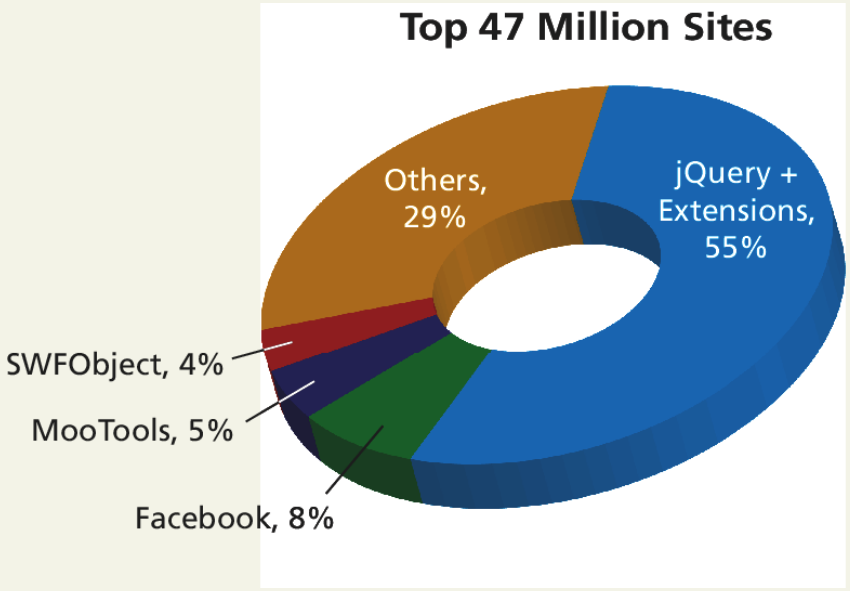
In August 2005 jQuery founder John Resig was looking into how to better combine **CSS selectors with succinct JavaScript notation**.

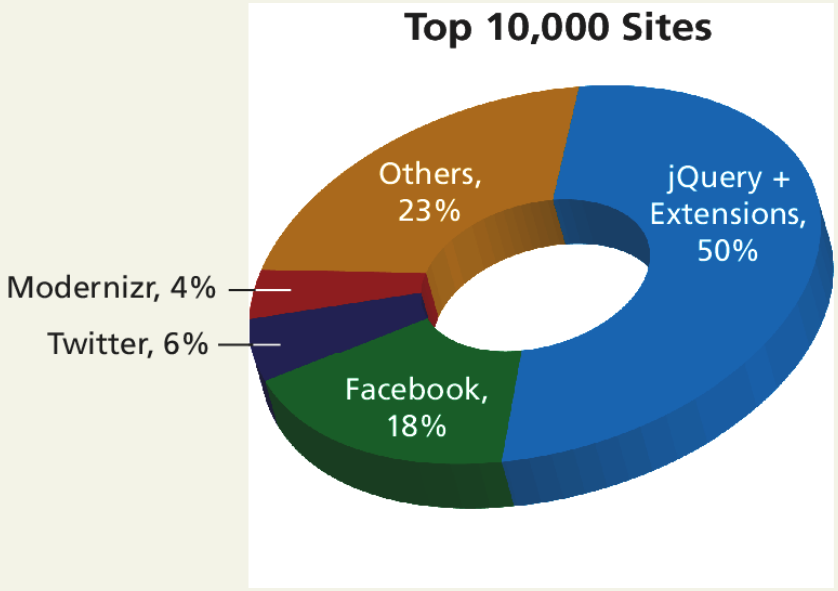
- Within 1 year AJAX and animations were added
- Additional modules
  - jQuery UI extension
  - mobile device support
- Continues to improve.

jQuery

Not the only one, but a popular one

jQuery is now the most popular JavaScript library currently in use as supported by the statistics from BuiltWith.com





Including jQuery

Let's get started

You must either:

- link to a locally hosted version of the library
- Use an approved third-party host, such as Google, Microsoft, or jQuery itself

Including jQuery

Content Delivery Network

Using a third-party **content delivery network (CDN)** is advantageous for several reasons.

- The bandwidth of the file is offloaded to reduce the demand on your servers.
- The user may already have cached the third-party file and thus not have to download it again, thereby reducing the total loading time.

A disadvantage to the third-party CDN is that your jQuery will fail if the third-party host fails (unlikely but possible)

Including jQuery

Content Delivery Network and fallback

```
<script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
<script type="text/javascript">
window.jQuery ||
document.write('<script src="/jquery-1.9.1.min.js"><\/script>');
</script>
```

LISTING 15.5 jQuery loading using a CDN and a local fail-safe if the CDN is offline

jQuery Selectors

Should ring a bell

When discussing basic JavaScript we introduced the **getElementByID()** and **querySelector()** selector functions in JavaScript.

Although the advanced **querySelector()** methods allow selection of DOM elements based on CSS selectors, it is only implemented in newest browsers

jQuery introduces its own way to select an element, which under the hood supports a myriad of older browsers for you!

jQuery Selectors

The easiest way to select an element yet

The relationship between DOM objects and selectors is so important in JavaScript programming that the pseudo-class bearing the name of the framework,

jQuery()

Is reserved for selecting elements from the DOM.

Because it is used so frequently, it has a shortcut notation and can be written as

$()

Basic Selectors

All the way back to CSS

The four basic selectors are:

- $("*") **Universal selector** matches all elements (and is slow).
- $("tag") **Element selector** matches all elements with the given element name.
- $(".class") **Class selector** matches all elements with the given CSS class.
- $("#id") **Id selector** matches all elements with a given HTML id attribute.

Basic Selectors

All the way back to CSS

For example, to select the single element with id="grab" you would write:

```
var singleElement = $("#grab");
```

To get a set of all the <a> elements the selector would be:

```
var allAs = $("a");
```

These selectors replace the use of getElementById() entirely.

More CSS Selectors

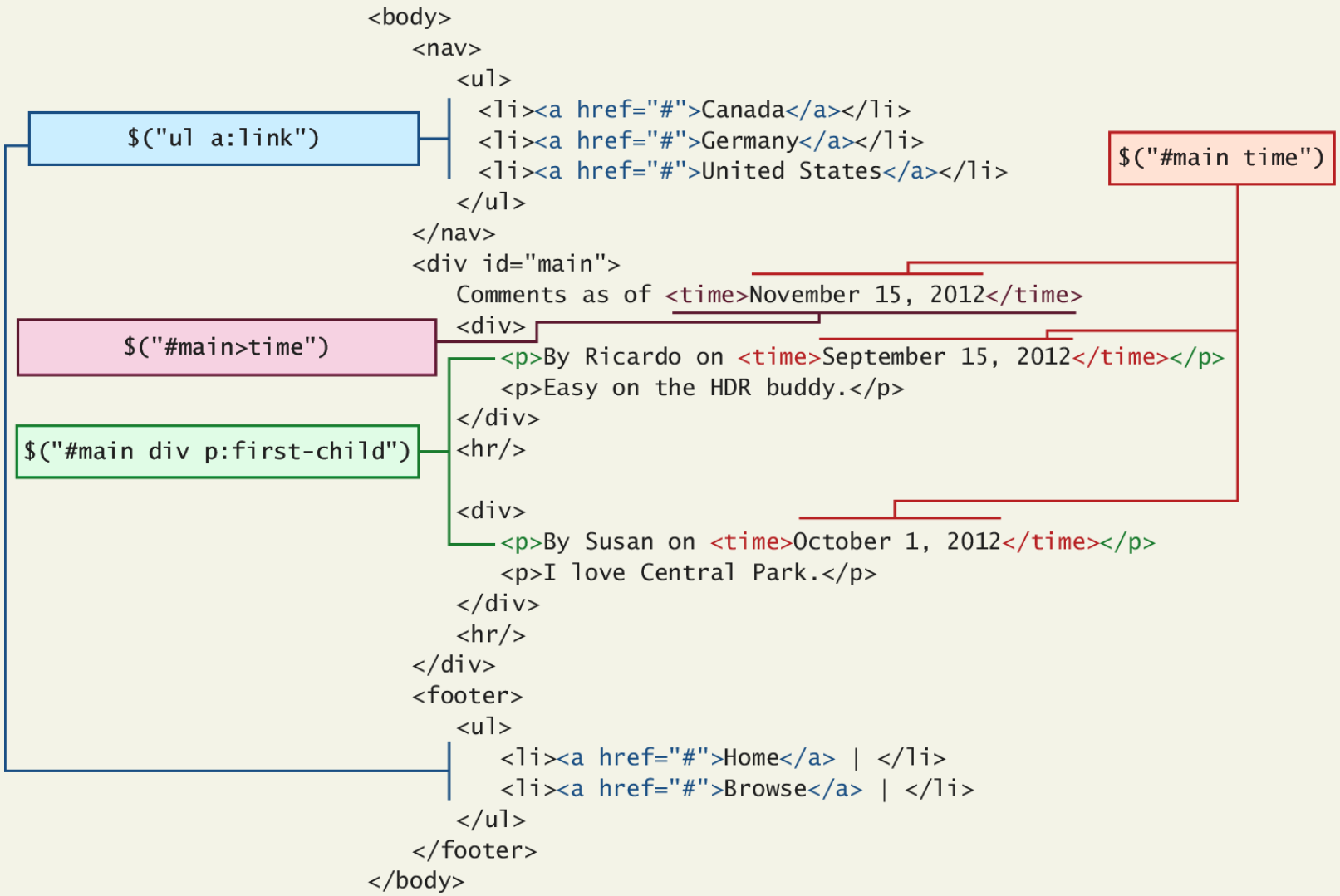
jQuery's selectors are powerful indeed

In addition to these basic selectors, you can use the other CSS selectors that were covered in Chapter 3 on CSS:

- attribute selectors,
- pseudo-element selectors, and
- contextual selectors

More CSS Selectors

jQuery's selectors are powerful indeed



Attribute Selector

Really a review of CSS

Recall from CSS that you can select

- by attribute with square brackets
  - [attribute]
- Specify a value with an equals sign
  - [attribute=value]
- Search for a particular value in the beginning, end, or anywhere inside a string
  - [attribute^=value]
  - [attribute$=value]
  - [attribute*=value]

Attribute Selector

Really a review of CSS

Consider a selector to grab all <img> elements with an src attribute beginning with */artist/* as:

```
var artistImages = $("img[src^='/artist/']");
```

Pseudo-Element Selector

Not to be confused with the pseudo-classes in JavaScript

pseudo-element selectors are also from CSS and allow you to append to any selector using the colon and one of

- :link
- :visited
- :focus
- :hover
- :active
- :checked
- :first-child, :first-line, and :first-letter

Pseudo-Element Selector

Not to be confused with the pseudo-classes in JavaScript

Selecting all links that have been visited, for example, would be specified with:

```
var visitedLinks = $("a:visited");
```

Contextual Selector

Put it into context

Contextual selectors are also from CSS. Recall that these include:

- descendant (space)
- child (>)
- adjacent sibling (+)
- and general sibling (~).

To select all <p> elements inside of elements you would write

```
var para = $("div p");
```

Content Filters

Above and Beyond CSS

The **content filter** is the only jQuery selector that allows you to append filters to all of the selectors you've used thus far and match a particular pattern.

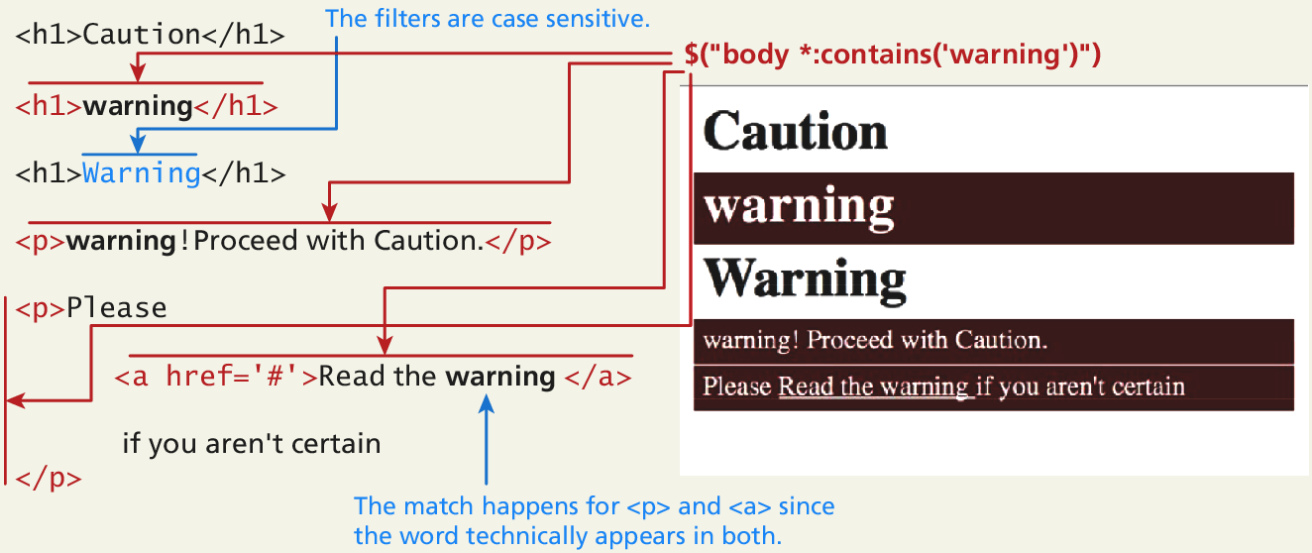
Select elements that have:

- a particular child using :has()
- have no children using :empty
- match a particular piece of text with :contains()

Content Filters

Above and Beyond CSS

$("body *:contains('warning')")



Au: I HAVEN'T TAGGED COMMAS IN THESE LISTS. Please check other lists in this table below.

Form Selectors

Above and Beyond CSS

Selector	CSS Equivalent	Description
$(:button)	$("button, input[type='button']")	Selects all buttons
$(:checkbox)	$('[type=checkbox]‘)	Selects all checkboxes
$(:checked)	No Equivalent	Selects elements that are checked. This includes radio buttons and checkboxes.
$(:disabled)	No Equivalent	Selects form elements that are disabled.
$(:enabled)	No Equivalent	Opposite of :disabled
$(:file)	$('[type=file]')	Selects all elements of type file
$(:focus)	$(document.activeElement)	The element with focus
$(:image)	$('[type=image]')	Selects all elements of type image
$(:input)	No Equivalent	Selects all <input>, <textarea>, <select>, and <button> elements.
$(:password)	$('[type=password]')	Selects all password fields
$(:radio)	$('[type=radio]')	Selects all radio elements
$(:reset)	$('[type=reset]')	Selects all the reset buttons
$(:selected)	No Equivalent	Selects all the elements that are currently selected of type <option>. It does not include checkboxes or radio buttons.
$(:submit)	$('[type=submit]')	Selects all submit input elements
$(:text)	No Equivalent	Selects all input elements of type text. $('[type=text]') is almost the same, except that $(:text) includes <input> fields with no type specified.

jQuery Attributes

Back to HTML now.

In order to understand how to fully manipulate the elements you now have access to, one must understand an element's *attributes* and *properties*.

The core set of attributes related to DOM elements are the ones specified in the HTML tags.

- The *href* attribute of an <a> tag
- The *src* attribute of an <img>
- The *class* attribute of most elements

jQuery Attributes

And some examples

In jQuery we can both set and get an attribute value by using the **attr()** method on any element from a selector.

```
// var link is assigned the href attribute of the first <a> tag
var link = $("a").attr("href");

// change all links in the page to http://funwebdev.com
$("a").attr("href","http://funwebdev.com");

// change the class for all images on the page to fancy
$("img").attr("class","fancy");
```

HTML Properties

Full circle

Many HTML tags include *properties* as well as attributes, the most common being the *checked* property of a radio button or checkbox.

The prop() method is the preferred way to retrieve and set the value of a property although, attr() may return some (less useful) values.

<input **class ="meh"** type="checkbox" checked="checked">

Is accessed by jQuery as follows:

```
var theBox = $(".meh");
theBox.prop("checked"); // evaluates to TRUE
theBox.attr("checked"); // evaluates to "checked"
```

Changing CSS

With jQuery

jQuery provides the extremely intuitive **css()** methods.

To get a css value use the css() method with 1 parameter:

```
$color = $("#colourBox").css("background-color"); // get the color
```

To set a CSS variable use css() with two parameters: the first being the CSS attribute, and the second the value.

```
// set color to red
$("#colourBox").css("background-color", "#FF0000");
```

Shortcut Methods

With jQuery

- The **html()** method is used to get the HTML contents of an element. If passed with a parameter, it updates the HTML of that element.
- The **val()** method returns the value of the element.
- The shortcut methods **addClass**(className) / **removeClass**(className) add or remove a CSS class to the element being worked on. The className used for these functions can contain a space-separated list of classnames to be added or removed.
- The **hasClass**(classname) method returns true if the element has the className currently assigned. False, otherwise.

jQuery Listeners

Set up after page load

In JavaScript, you learned why having your **listeners** set up inside of the window.onload() event was a good practice.

With jQuery we do the same thing but use the $(document).ready() event

```
$(document).ready(function(){
 //set up listeners on the change event for the file items.
 $("input[type=file]").change(function(){
     console.log("The file to upload is "+ this.value);
  });
});
```

LISTING 15.6 jQuery code to listen for file inputs changing, all inside the document's ready event

jQuery Listeners

Listener Management

While pure JavaScript uses the addEventListener() method, jQuery has on() and off() methods as well as shortcut methods to attach events.

```
$(document).ready(function(){
   $(":file").on("change",alertFileName); // add listener
});
// handler function using this
function alertFileName() {
   console.log("The file selected is: "+this.value);
}
```

LISTING 15.7 Using the listener technique in jQuery with on and off methods

Modifying the DOM

Creating Nodes

```
 // pure JavaScript way
var jsLink = document.createElement("a");
jsLink.href = "http://www.funwebdev.com";
jsLink.innerHTML = "Visit Us";
jsLink.title = "JS";

// jQuery way
var jQueryLink = $("<a href='http://funwebdev.com'
title = 'jQuery'>Visit Us</a>");

// jQuery long-form way
var jQueryVerboseLink = $("<a></a>");
jQueryVerboseLink.attr("href",'http://funwebdev.com');
jQueryVerboseLink.attr("title","jQuery verbose");
jQueryVerboseLink.html("Visit Us");
```

LISTING 15.8 A comparison of node creation in JS and jQuery

Modifying the DOM

Appending DOM Elements

The append() method takes as a parameter an HTML string, a DOM object, or a jQuery object. That object is then added as the last child to the element(s) being selected.

HTML Before

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
    </div>
<div>
```

jQuery append

```
$(".linkOut").append(jsLink);
```

HTML After

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
    </div>
<div>
```

Modifying the DOM

Prepending DOM Elements

The prepend() and prependTo() methods operate in a similar manner except that they add the new element as the first child rather than the last.

HTML Before

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
    </div>
<div>
```

jQuery append

```
$(".linkOut").prepend(jsLink);
```

HTML After

```
<div class="external-links">
     <div class="linkOut">
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
        pearson.com
    </div>
<div>
```

Modifying the DOM

Wrapping Existing DOM in New Tags

```
<div class="external-links">
 <div class="gallery">Uffuzi Museum</div>
 <div class="gallery">National Gallery</div>
 <div class="link-out">funwebdev.com</div>
</div>
```

```
$(".gallery").wrap('<div class="galleryLink"/>');
```

```
<div class="external-links">
  <div class="galleryLink">
      <div class="gallery">Uffuzi Museum</div>
  </div>
   <div class="galleryLink">
      <div class="gallery">National Gallery</div>
  </div>
   <div class="link-out">funwebdev.com</div>
</div>
```

LISTING 15.10 HTML from Listing 15.9 modified by executing the wrap statement above

Modifying the DOM

Wrapping Existing DOM in New Tags

A more advanced technique might make use of the content of each div being modified. In that case we use a callback function in place of a simple element.

The wrap() method is a callback function, which is called for each element in a set (often an array).

Each element then becomes this for the duration of one of the wrap() function's executions, allowing the unique title attributes as shown in Listing 15.12.

Modifying the DOM

Wrapping Existing DOM in New Tags

```
$(".contact").wrap(function(){
  return "<div class='galleryLink' title='Visit " + $(this).html()+
         "'></div>";
});
```

LISTING 15.11 Using wrap() with a callback to create a unique div for every matched element

```
<div class="external-links">
  <div class="galleryLink" title="Visit Uffuzi Museum">
     <div class="gallery">Uffuzi Museum</div>
  </div>
  <div class="galleryLink" title="Visit National Gallery">
     <div class="gallery">National Gallery</div>
  </div>
  <div class="link-out">funwebdev.com</div>
</div>
```

LISTING 15.12 The modified HTML from Listing 15.9 after executing using wrap code from Listing 15.11

Section 3 of 6

AJAX

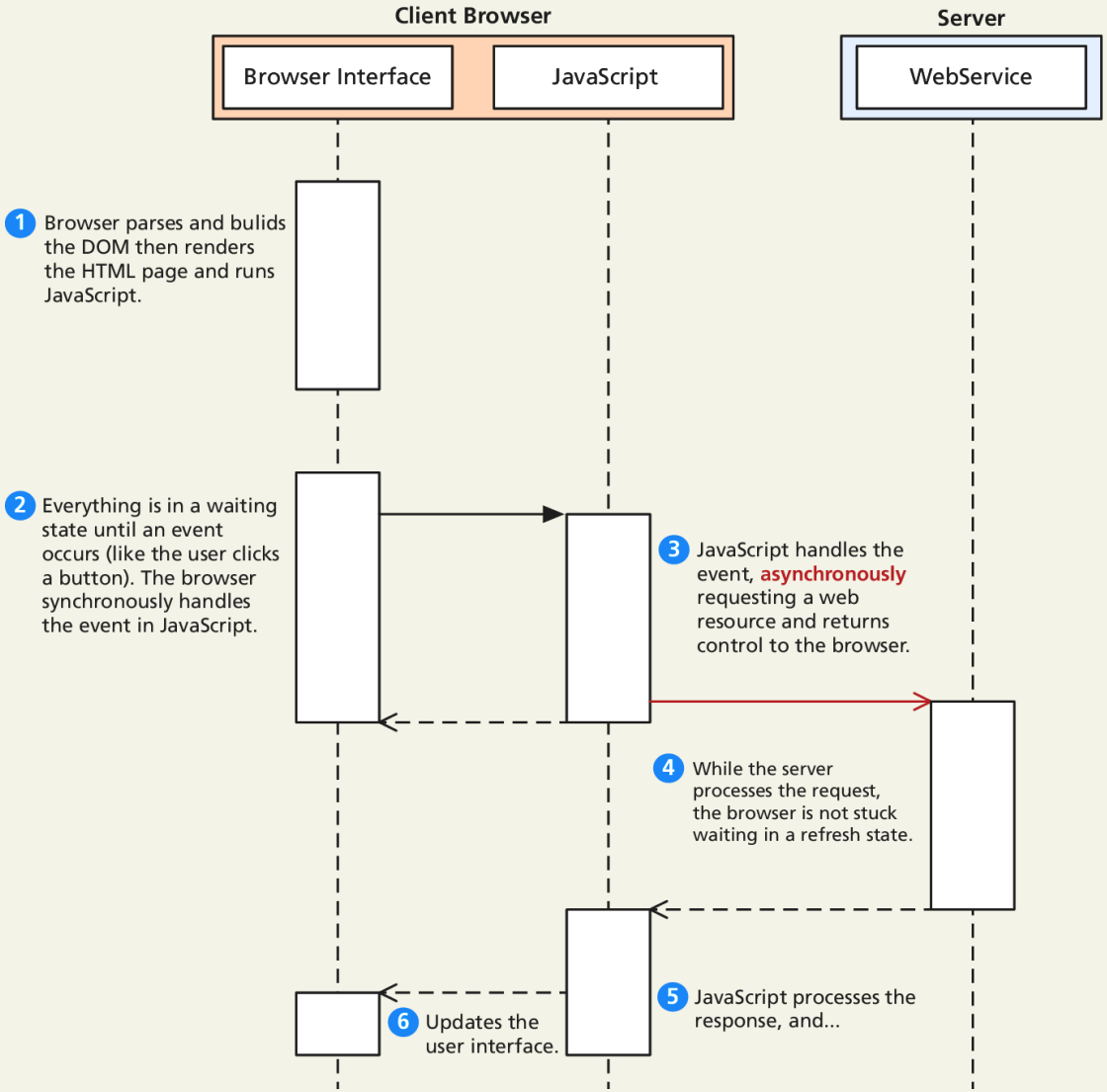
AJAX

Asynchronous JavaScript with XML

Asynchronous JavaScript with XML (AJAX) is a term used to describe a paradigm that allows a web browser to send messages back to the server without interrupting the flow of what's being shown in the browser.

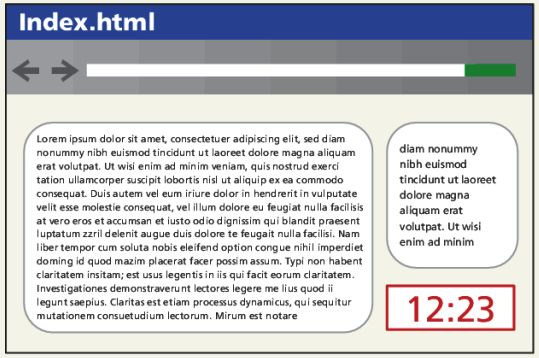
AJAX

UML of an asynchronous request

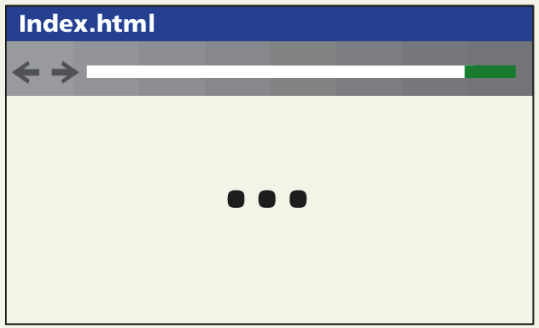


AJAX

Consider a webpage that displays the server's time

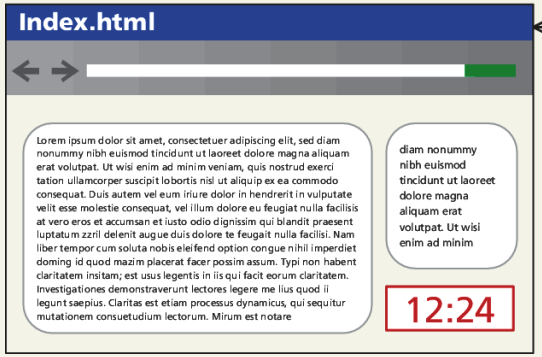


1. The page loads and shows the current server time as a small part of a larger page.



2. A **synchronous** JavaScript call makes an HTTP request for the "freshest" version of the page.

   While waiting for the response, the browser goes into its waiting state.

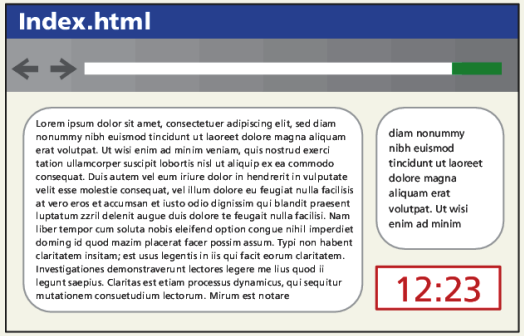


3. The response arrives, so the browser can render the new version of the page, and the functionality in the browser is restored.

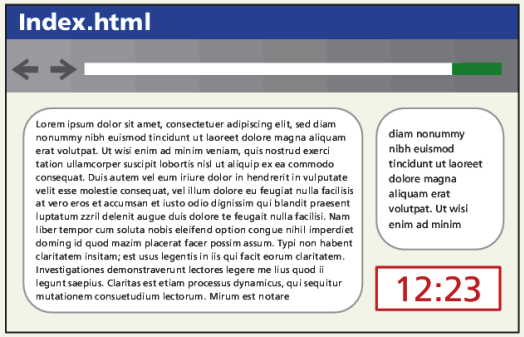
```
<html>
        <head>
        ...
        </head>
        <body>
        ...
        <div id='serverTime'>
                12.24
        </div>
        ...
        </body>
</html>
```

AJAX

Consider a webpage that displays the server's time

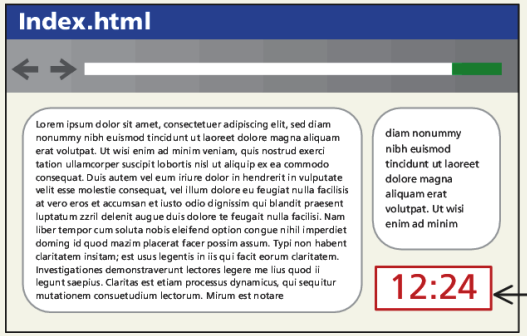


1. The page loads and shows the current server time as a small part of a larger page.



2. An **asynchronous** JavaScript call makes an HTTP request for just the small component of the page that needs updating (the time).

   While waiting for the response, the browser still looks the same and is responsive to user interactions.



3. The response arrives, and through JavaScript, the HTML page is updated.

AJAX

Making Asynchronous requests

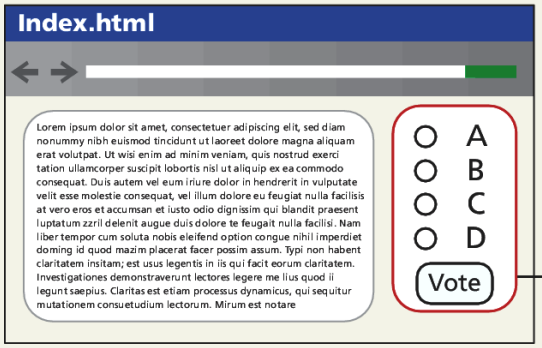
jQuery provides a family of methods to make asynchronous requests. We will start simple and work our way up.

Consider the very simple server time example we just saw. If currentTime.php returns a single string and you want to load that value asynchronously into the element, you could write:

```
$("#timeDiv").load("currentTime.php");
```

AJAX

GET Requests

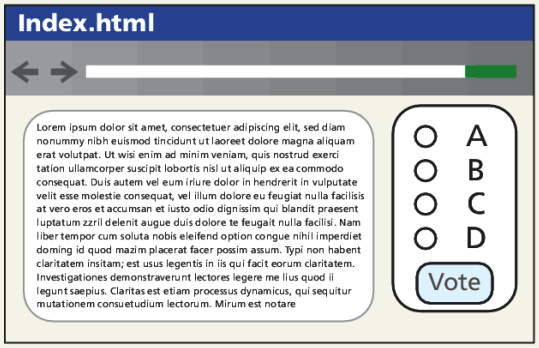


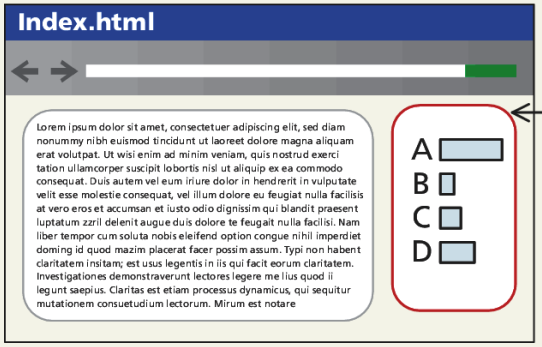
1 The HTML page contains a poll that posts votes asynchronously.

```
$.get("/vote.php?option=C");
```

2 An **asynchronous** vote submits the user's choice.

Meanwhile, the browser remains interactive while the request is processed on the server.





3 The response arrives, and is handled by JavaScript, which uses the response data to update the interface to show poll results.

AJAX

GET Requests – formal definition

jQuery.get (**url [, data] [, success(data, textStatus, jqXHR)] [, dataType]**)

- **url** is a string that holds the location to send the request.
- **data** is an optional parameter that is a query string or a *Plain Object.*
- **success(data,textStatus,jqXHR)** is an optional *callback* function that executes when the response is received.
  - **data** holding the body of the response as a string.
  - **textStatus** holding the status of the request (i.e., “success”).
  - **jqXHR** holding a jqXHR object, described shortly.
- **dataType** is an optional parameter to hold the type of data expected from the server.

AJAX

GET Requests – an example

```
$.get("/vote.php?option=C", function(data,textStatus,jsXHR) {
  if (textStatus=="success") {
     console.log("success! response is:" + data);
  }
  else {
     console.log("There was an error code"+jsXHR.status);
  }
  console.log("all done");
});
```

LISTING 15.13 jQuery to asynchronously get a URL and outputs when the response arrives

AJAX

The jqXHR Object

All of the $.get() requests made by jQuery return a **jqXHR** object to encapsulate the response from the server.

In practice that means the data being referred to in the callback from Listing 15.13 is actually an object with backward compatibility with XMLHttpRequest.

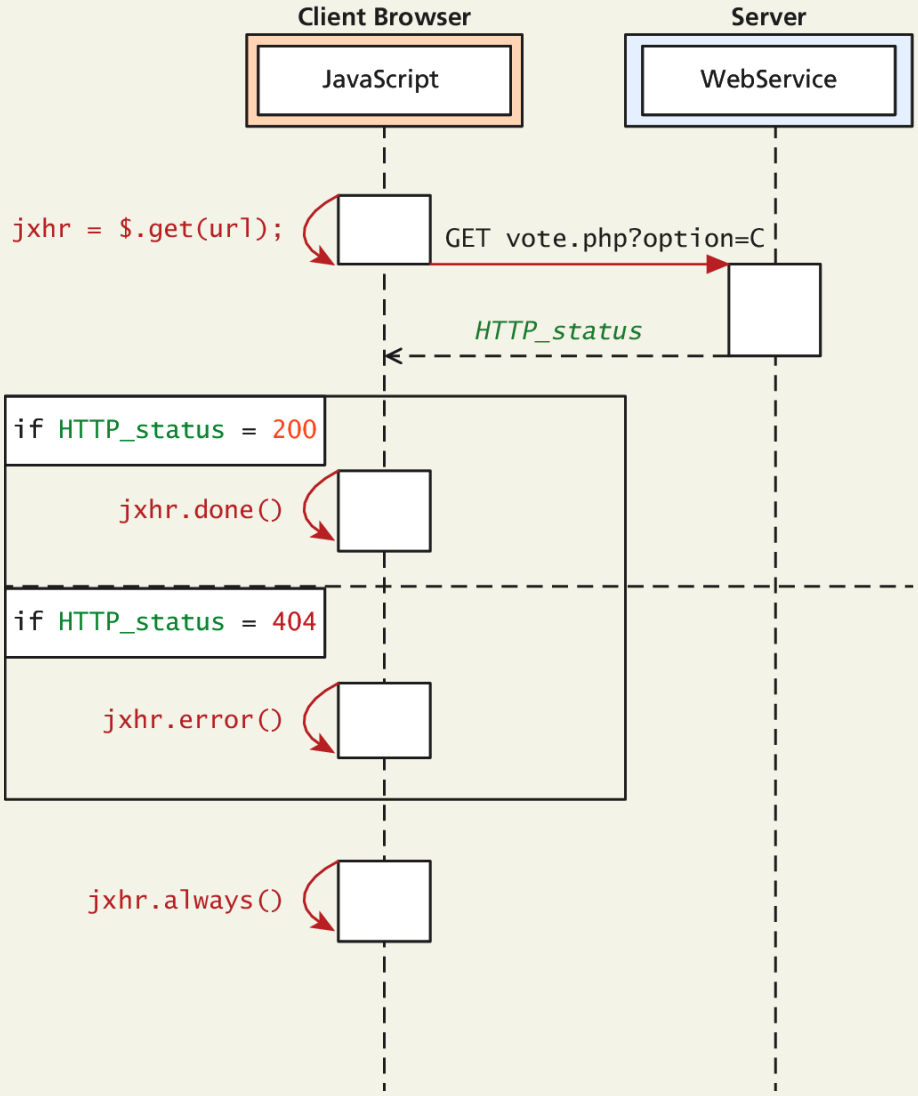
AJAX

jqXHR - XMLHttpRequest compatibility

- **abort()** stops execution and prevents any callback or handlers from receiving the trigger to execute.
- **getResponseHeader()** takes a parameter and gets the current value of that header.
- **readyState** is an integer from 1 to 4 representing the state of the request. The values include 1: sending, 3: response being processed, and 4: completed.
- **responseXML** and/or **responseText** the main response to the request.
- **setRequestHeader(name, value)** when used before actually instantiating the request allows headers to be changed for the request.
- **status** is the HTTP request status codes (200 = ok)
- **statusText** is the associated description of the status code.

jqXHR

Actually quite easy to use



jqXHR objects have methods

- done()
- fail()
- always()

which allow us to structure our code in a more modular way than the inline callback

jqXHR

Very modular code

```
var jqxhr = $.get("/vote.php?option=C");

jqxhr.done(function(data) { console.log(data);});
jqxhr.fail(function(jqXHR) { console.log("Error: "+jqXHR.status); })
jqxhr.always(function() { console.log("all done"); });
```

LISTING 15.14 Modular jQuery code using the jqXHR object

POST requests

Via jQuery AJAX

POST requests are often preferred to GET requests because one can post an unlimited amount of data, and because they do not generate viewable URLs for each action.

GET requests are typically not used when we have forms because of the messy URLs and that limitation on how much data we can transmit.

With POST it is possible to transmit files, something which is not possible with GET.

POST requests

Via jQuery AJAX

The HTTP 1.1 definition describes GET as a **safe method** meaning that they should not change anything, and should only read data.

POSTs on the other hand are not safe, and should be used whenever we are changing the state of our system (like casting a vote). get() method.

POST syntax is almost identical to GET.

jQuery.**post** (url [, data] [, success(data, textStatus, jqXHR)] [, dataType])

POST requests

Via jQuery AJAX

If we were to convert our vote casting code it would simply change the first line from

```
var jqxhr = $.get("/vote.php?option=C");
```

to

```
var jqxhr = $.post("/vote.php", "option=C");
```

POST requests

Serialize() will seriously help

serialize() can be called on any form object to return its current key-value pairing as an & separated string, suitable for use with post().

```
var postData = $("#voteForm").serialize();

$.post("vote.php", postData);
```

Ajax

You have complete control

It turns out both the $.get() and $.post() methods are actually shorthand forms for the jQuery().ajax() method

The ajax() method has two versions. In the first it takes two parameters: a URL and a Plain Object, containing any of over 30 fields.

A second version with only one parameter is more commonly used, where the URL is but one of the key-value pairs in the Plain Object.

Ajax

More verbose

The one line required to post our form using get() becomes the more verbose code

```
$.ajax({ url: "vote.php",
         data: $("#voteForm").serialize(),
         async: true,
         type: post
});
```

LISTING 15.15 A raw AJAX method code to make a post

Ajax

You have complete control

To pass HTTP headers to the ajax() method, you enclose as many as you would like in a Plain Object. To illustrate how you could override User-Agent and Referer headers in the POST

```
$.ajax({ url: "vote.php",
  data: $("#voteForm").serialize(),
  async: true,
  type: post,
  headers: {"User-Agent" : "Homebrew JavaScript Vote Engine agent",
            "Referer": "http://funwebdev.com"
           }
});
```

LISTING 15.16 Adding headers to an AJAX post in jQuery

CORS

Cross Origin Resource Sharing

Since modern browsers prevent cross-origin requests by default (which is good for security), sharing content legitimately between two domains becomes harder.

Cross-origin resource sharing (CORS) uses new headers in the HTML5 standard implemented in most new browsers.

If a site wants to allow any domain to access its content through JavaScript, it would add the following header to all of its responses.

Access-Control-Allow-Origin: *

CORS

Cross Origin Resource Sharing

A better usage is to specify specific domains that are allowed, rather than cast the gates open to each and every domain. To allow our domain to make cross site requests we would add the header:

Access-Control-Allow-Origin: www.funwebdev.com

Rather than the wildcard *.

Section 4 of 6

ASYNCHRONOUS FILE TRANSMISSION

Asynchronous File Transmission

A Primer

Consider a simple form as defined below:

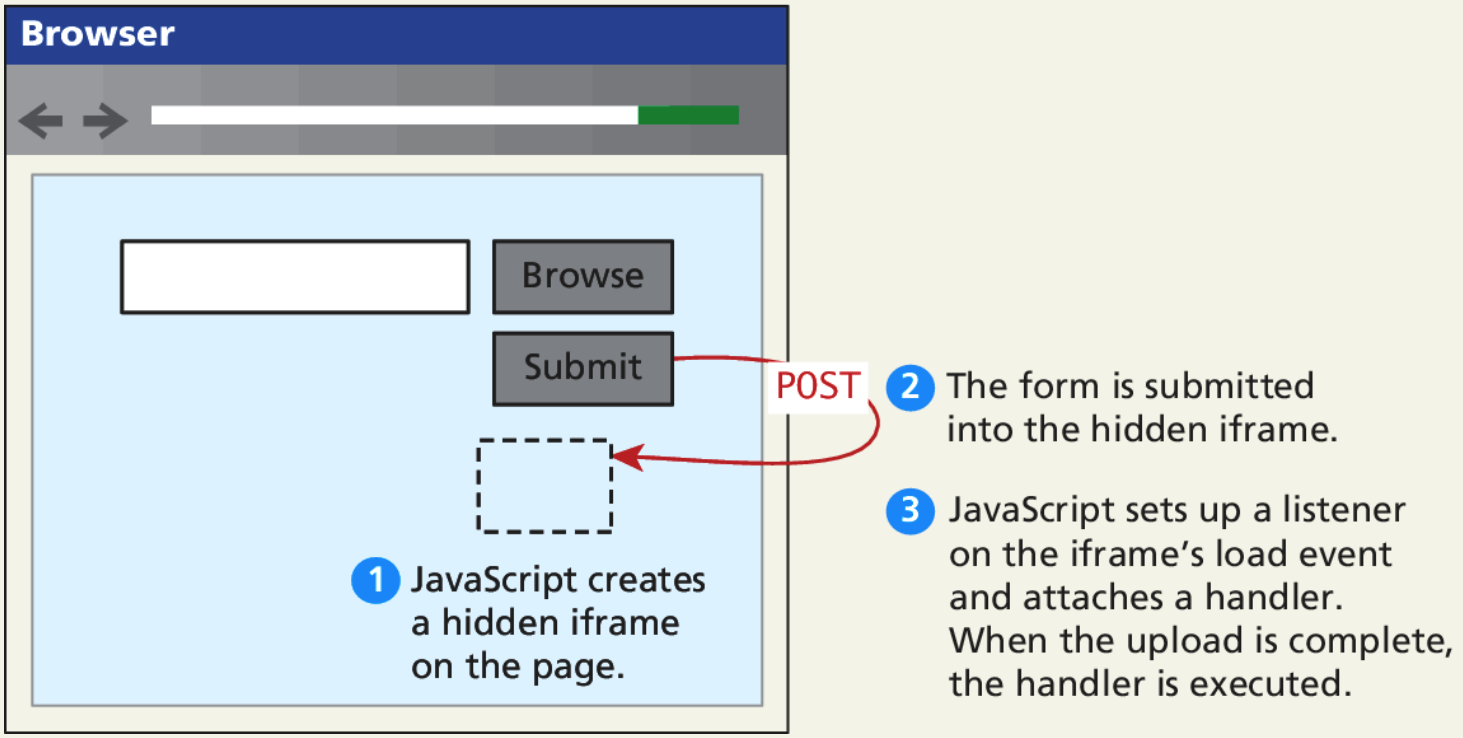
```
<form name="fileUpload" id="fileUpload" enctype="multipart/form-data"
      method="post" action="upload.php">
<input name="images" id="images" type="file" multiple />
<input type="submit" name="submit" value="Upload files!"/>
</form>
```

LISTING 15.17 Simple file upload form

Asynchronous File Transmission

The old iFrame technique

The original workaround to allow the asynchronous posting of files was to use a hidden <iframe> element to receive the posted files.



Asynchronous File Transmission

The old iFrame technique

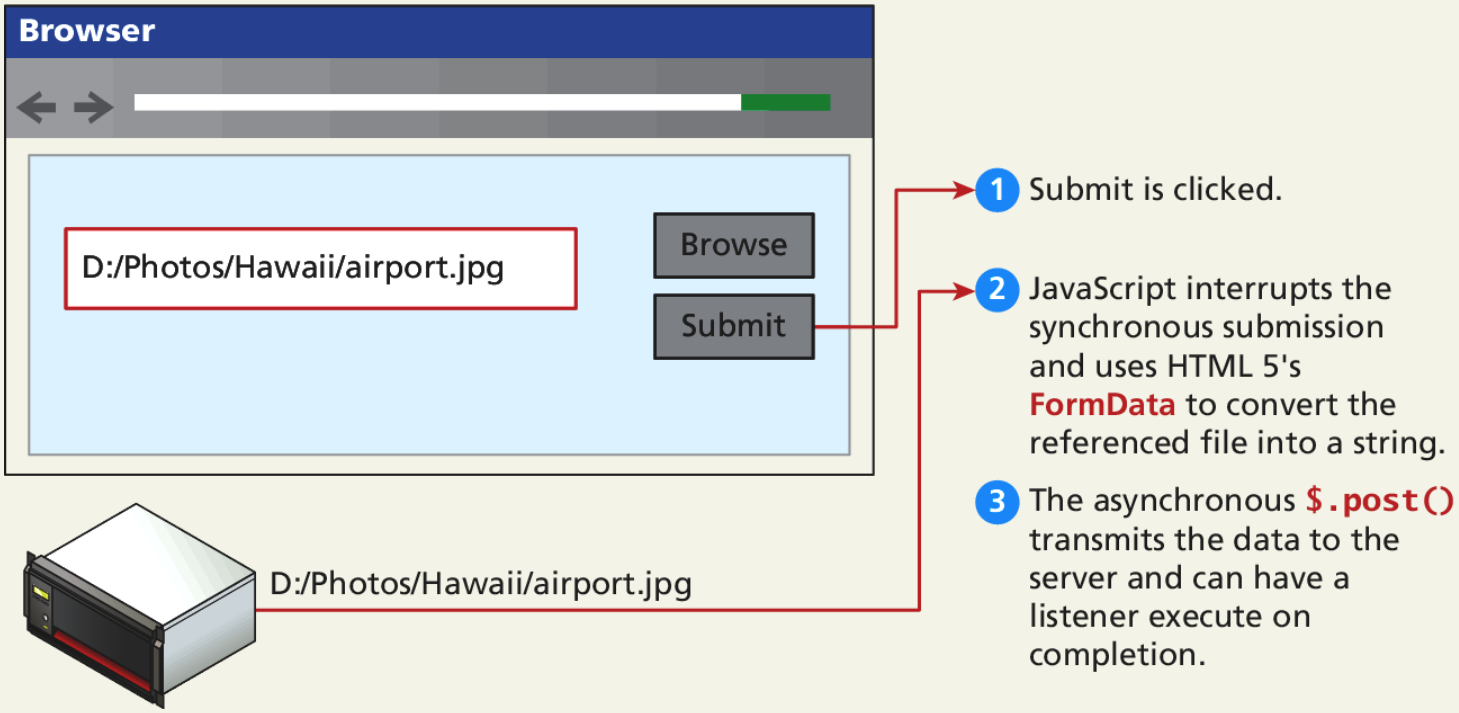
```
$(document).ready(function() {
  // set up listener when the file changes
  $(":file").on("change",uploadFile);
  // hide the submit buttons
  $("input[type=submit]").css("display","none");
});

// function called when the file being chosen changes
function uploadFile () {
  // create a hidden iframe
  var hidName = "hiddenIFrame";
  $("#fileUpload").append("<iframe id='"+hidName+"' name='"+hidName+"'
  style='display:none' src='#' ></iframe>");

  // set form's target to iframe
  $("#fileUpload").prop("target",hidName);
  // submit the form, now that an image is in it.
  $("#fileUpload").submit();
  // Now register the load event of the iframe to give feedback
  $('#'+hidName).load(function() {
  var link = $(this).contents().find('body')[0].innerHTML;
  // add an image dynamically to the page from the file just uploaded
  $("#fileUpload").append("<img src='"+link+"' />");
  });
}
```

LISTING 15.18 Hidden iFrame technique to upload files

Asynchronous File Transmission

New Form Data technique

Using the **FormData** interface and File API, which is part of HTML5, you no longer have to trick the browser into posting your file data asynchronously.



Asynchronous File Transmission

New Form Data technique

```
function uploadFile () {
  // get the file as a string
  var formData = new FormData($("#fileUpload")[0]);

  var xhr = new XMLHttpRequest();
  xhr.addEventListener("load", transferComplete, false);
  xhr.addEventListener("error", transferFailed, false);
  xhr.addEventListener("abort", transferCanceled, false);

  xhr.open('POST', 'upload.php', true);
  xhr.send(formData);                  // actually send the form data

  function transferComplete(evt) {   // stylized upload complete
     $("#progress").css("width","100%");
     $("#progress").html("100%");
  }

  function transferFailed(evt) {
     alert("An error occurred while transferring the file.");
  }

  function transferCanceled(evt) {
     alert("The transfer has been canceled by the user.");
  }
}
```

LISTING 15.19 Using the new FormData interface from the XHR2 Specification to post files asychronously

Asynchronous File Transmission

Advanced modern technique

When we consider uploading multiple files, you may want to upload a single file, rather than the entire form every time. To support that pattern, you can access a single file and post it by appending the raw file to a FormData object.

The advantage of this technique is that you submit each file to the server asynchronously as the user changes it; and it allows multiple files to be transmitted at once.

Asynchronous File Transmission

Advanced modern technique

```
var xhr = new XMLHttpRequest();
// reference to the 1st file input field
var theFile = $(":file")[0].files[0];
var formData = new FormData();
formData.append('images', theFile);
```

LISTING 15.20 Posting a single file from a form

```
var allFiles =  $(":file")[0].files;
for (var i=0;i<allFiles.length;i++) {
  formData.append('images[]', allFiles[i]);
}
```

LISTING 15.21 Looping through multiple files in a file input and appending the data for posting

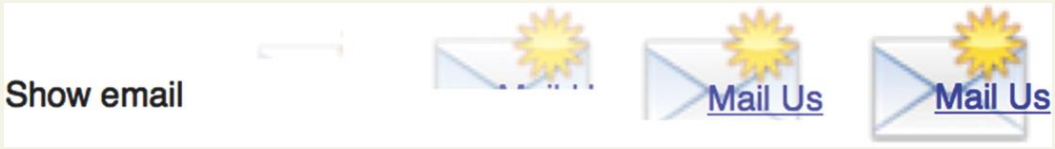
Section 5 of 6

ANIMATION

Animation

Hide() and Show()

The hide() and show() methods can be called with no arguments to perform a default animation. Another version allows two parameters: the duration of the animation (in milliseconds) and a callback method to execute on completion.



Animation

fadeIn() and fadeOut()

The fadeIn() and fadeOut() shortcut methods control the opacity of an element. The parameters passed are the duration and the callback, just like hide() and show(). Unlike hide() and show(), there is no scaling of the element, just strictly control over the transparency.

Animation

SlideUp() and SlideDown()

slideUp() and slideDown() do not touch the opacity of an element, but rather gradually change its height.

email icon from **http://openiconlibrary.sourceforge.net.**

Animation

Toggle()

As you may have seen, the shortcut methods come in pairs, which make them ideal for toggling between a shown and hidden state. Using a toggle method means you don’t have to check the current state and then conditionally call one of the two methods;

- To toggle between the visible and hidden states you can use the **toggle()** methods.
- To toggle between fading in and fading out, use the **fadeToggle()** method
- To toggle between the two sliding states can be achieved using the **slideToggle()** method.

Raw Animation

Full control

The animate() method has several versions, but the one we will look at has the following form:

.animate(properties, options);

The properties parameter contains a Plain Object with all the CSS styles of the final state of the animation.

The options parameter contains another Plain Object with any of the following options set:

Raw Animation

Options parameter

- **always** is the function to be called when the animation completes or stops with a fail condition. This function will always be called (hence the name).
- **done** is a function to be called when the animation completes.
- **duration** is a number controlling the duration of the animation.
- **fail** is the function called if the animation does not complete.
- **progress** is a function to be called after each step of the animation.

Raw Animation

Options parameter

- **queue** is a Boolean value telling the animation whether to wait in the queue of animations or not. If false, the animation begins immediately.
- **step** is a function you can define that will be called periodically while the animation is still going.
- Advanced options called **easing** and **specialEasing** allow for advanced control over the speed of animation.

Raw Animation

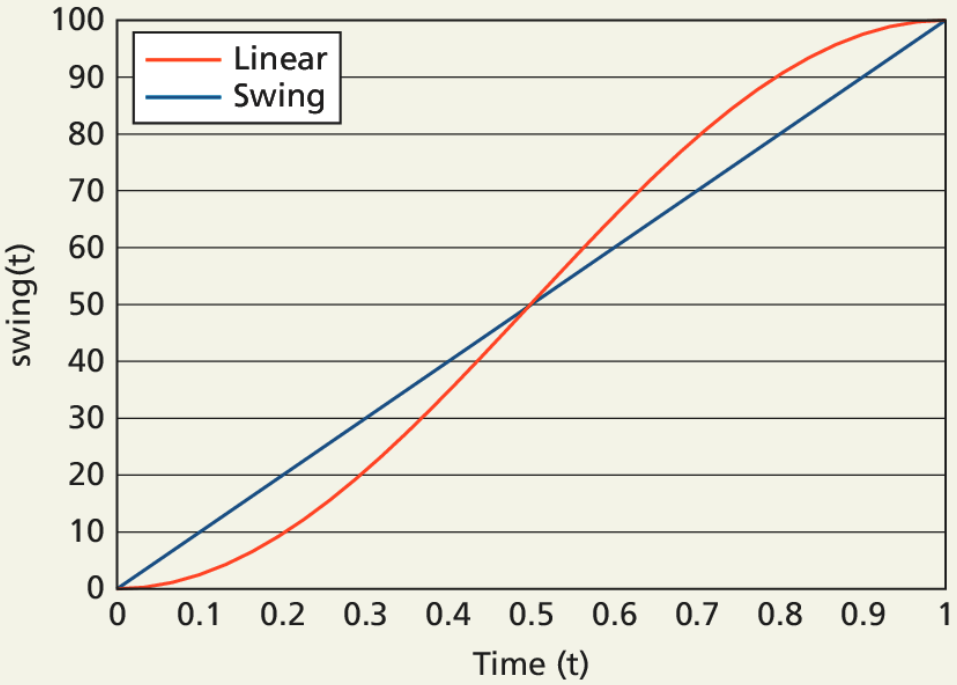
Easing functions – advanced animation

In web development, **easing functions** are used to simulate that natural type of movement. They are mathematical equations that describe how fast or slow the transitions occur at various points during the animation.

Included in jQuery are

- linear
- swing

easing functions.



Raw Animation

Easing functions – advanced animation

For example, the function defining swing for values of time *t* between 0 and 1 is

$$\text{swing}(t) = -\frac{1}{2}\cos(t\pi) + 0.5$$

The jQuery UI extension provides over 30 easing functions, including cubic functions and bouncing effects, so you should not have to define your own.

Advanced example

rotating

```
$(this).animate(
  // parameter one: Plain Object with CSS options.

  {opacity:"show","fontSize":"120%","marginRight":"100px"},
  // parameter 2: Plain Object with other options including a
  // step function
  {step: function(now, fx) {
        // if the method was called for the margin property
        if (fx.prop=="marginRight") {
           var angle=(now/100)*360; //percentage of a full circle
           // Multiple rotation methods to work in multiple browsers
           $(this).css("transform","rotate("+angle+"deg)");
           $(this).css("-webkit-transform","rotate("+angle+"deg)");
           $(this).css("-ms-transform","rotate("+angle+"deg)");
        }
     },
     duration:5000, "easing":"linear"
  }
);
```

LISTING 15.23 Use of animate() with a step function to do CSS3 rotation

Raw Animation

Rotating

t=0

```
//begin animation
$.animate(/* ... */);
```

```
{
opacity:0,
margin-right:0
transform:
angle(0deg);
}
```

t=1500

```
//opacity step
step(0.3, fx);
//margin-right
step(30, fx){


    var angle=(30/100)*360;
    $(this).css("transform",
          "rotate("+angle+"deg)");
}
//no step for transform!
```

```
{
opacity:0.3,
margin-right:30
transform: angle(108deg);
}
```

t=3200

```
//opacity step
step(0.64,fx);
//margin-right
step(64, fx){


    var angle=(64/100)*360;
    $(this).css("transform",
          "rotate("+angle+"deg)");

}
//no step for transform!
```

```
{
opacity:0.64,
margin-right:64
transform: angle(230deg);
}
```

t=5000ms

```
//done animation
done()
```

```
{
opacity:1,
margin-right:100
transform:
angle(0deg);
}
```

Section 6 of 6

BACKBONE MVC FRAMEWORKS

Backbone

Another framework

Backbone is an MVC framework that further abstracts JavaScript with libraries intended to adhere more closely to the MVC model

This library is available from **http://backbonejs.org** and relies on the underscore library, available from http://underscorejs.org/.

Include with:

<script src="underscore-min.js"></script>

<script src="backbone-min.js"></script>

Backbone

Models

In Backbone, you build your client scripts around the concept of **models**.

Backbone.js defines **models** as

the heart of any JavaScript application, containing the interactive data as well as a large part of the logic surrounding it: conversions, validations, computed properties, and access control.

The Models you define using Backbone must *extend* Backbone.Model

Backbone

Collections

In addition to models, Backbone introduces the concept of **Collections**, which are normally used to contain lists of Model objects.

These collections have advanced features and like a database can have indexes to improve search performance.

A collection is defined by extending from Backbone's Collection object.

Backbone

Views

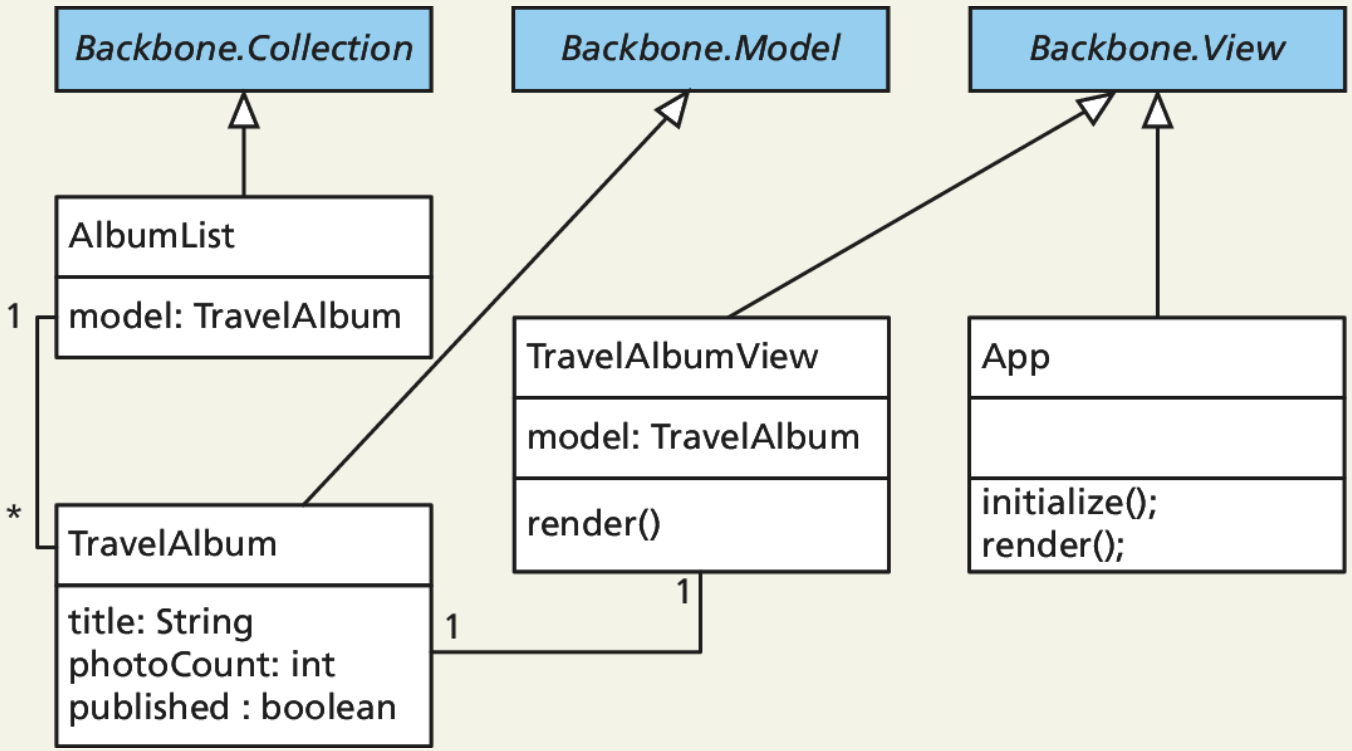
Views allow you to translate your models into the HTML that is seen by the users.

They attach themselves to methods and properties of the Collection and define methods that will be called whenever Backbone determines the view needs refreshing.

You must always override the render() method since it defines the HTML that is output.

Backbone

Example



Backbone

A Model Example

```
// Create a model for the albums
var TravelAlbum = Backbone.Model.extend({
  defaults:{
          title: 'NewAlbum',
          photoCount: 0,
          published: false
      },

      // Function to publish/unpublish
      toggle: function(){
          this.set('checked', !this.get('checked'));
      }
  });
```

LISTING 15.25 A PhotoAlbum Model extending from Backbone.Model

Backbone

A Collection Example

```
// Create a collection of albums
var AlbumList = Backbone.Collection.extend({

  // Set the model type for objects in this Collection
  model: TravelAlbum,

  // Return an array only with the published albums
  GetChecked: function(){
     return this.where({checked:true});
  }
});

// Prefill the collection with some albums.
var albums = new AlbumList([
  new TravelAlbum({ title: 'Banff, Canada', photoCount: 42}),
  new TravelAlbum({ title: 'Santorini, Greece', photoCount: 102}),
]);
```

LISTING 15.26 Demonstration of a Backbone.js Collection defined to hold PhotoAlbums

Backbone

A View Example

```
var TravelAlbumView = Backbone.View.extend({
  TagName: 'li',

  events:{
     'click': 'toggleAlbum'
  },

  initialize: function(){
      // Set up event listeners attached to change
     this.listenTo(this.model, 'change', this.render);
  },

  render: function(){
  // Create the HTML
  this.$el.html('<input type="checkbox" value="1" name="' +
                this.model.get('title') + '" /> ' +
                this.model.get('title') + '<span> ' +
                this.model.get('photoCount') + ' images</span>');
  this.$('input').prop('checked', this.model.get('checked'));

  // Returning the object is a good practice
     return this;
  },

  toggleAlbum: function() {
     this.model.toggle();
  }
});
```

LISTING 15.27 Deriving custom View objects for our model and Collection

Backbone

Bring it all together

```
// The main view of the entire Backbone application
var App = Backbone.View.extend({
  // Base the view on an existing element
  el: $('body'),

  initialize: function() {
     // Define required selectors
     this.total = $('#totalAlbums span');
     this.list = $('#albums');

     // Listen for the change event on the collection.
     this.listenTo(albums, 'change', this. render);

     // Create views for every one of the albums in the collection
     albums.each(function(album) {
        var view = new TravelAlbumView({ model: album });
              this.list.append(view.render().el);
        }, this);   // "this" is the context in the callback
     },

     render: function(){

         // Calculate the count of published albums and photos
            var total = 0; var photos = 0;

           _.each(albums.getChecked(), function(elem) {
              total++;
              photos+= elem.get("photoCount");
           });

        // Update the total price
        this.total.text(total+' Albums ('+photos+' images)');
        return this;
     }
});

new App(); // create the main app
```

LISTING 15.28 Defining the main app's view and making use of the Collections and models defined earlier

What You've Learned

1. JavaScript Pseudo-Classes
2. jQuery Foundations
3. AJAX
4. Asynchronous File Transmission
5. Animation
6. Backbone MVC Frameworks