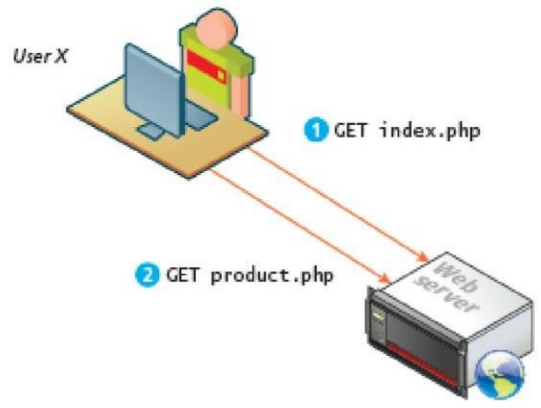
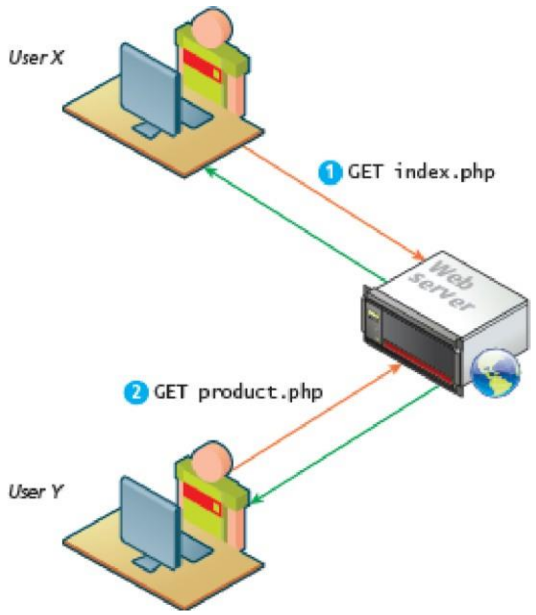
Module V
Managing State, jQuery, XML & WebServices

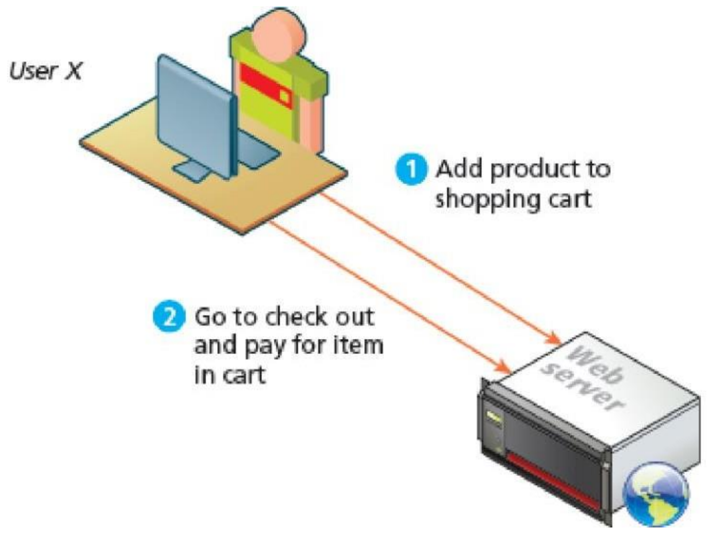
5.1 The Problem of State in Web Applications

- A problem that is unique in web development is the sharing of information among the request. The information obtained from the first request must be saved for the next request.
- In Single-user desktop applications there is no such problem, as all information is stored in memory and can be easily accessed throughout the application. But a web application consists of a series of disconnected HTTP requests to a web server where each request for a server page is a request to run a separate program on server.





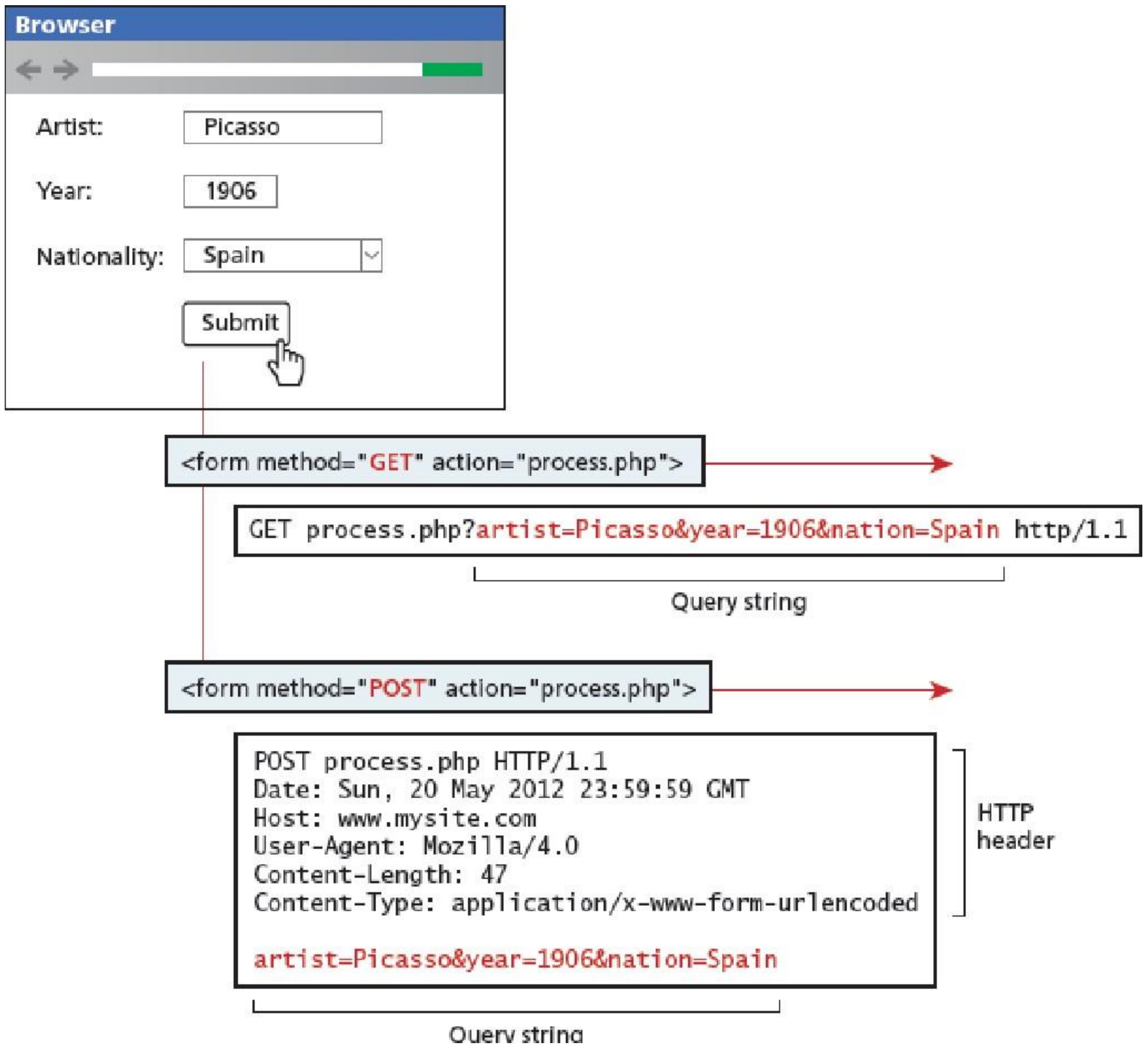
- The web server sees only requests. The HTTP protocol does not, without programming intervention, distinguish two requests by one source from two requests from two different sources.
- While the HTTP protocol disconnects the user’s identity from the requests, there are many occasions when we want the web server to connect requests together.
- Consider the scenario of a web shopping cart, the user (and the website owner) most certainly wants the server to recognize that the request to add an item to the cart and the subsequent request to check out and pay for the item in the cart are connected to the same individual.


User X
1 Add product to shopping cart
2 Go to check out and pay for item in cart
Web server

In HTTP, we can pass information using:
■ Query strings
■ Cookies

5.2 Passing Information via Query Strings

A web page can pass query string information from the browser to the server using one of the two methods: a query string within the URL (GET) and a query string within the HTTP header (POST).



```
<form method="GET" action="process.php">

GET process.php?artist=Picasso&year=1906&nation=Spain http/1.1

<form method="POST" action="process.php">

POST process.php HTTP/1.1
Date: Sun, 20 May 2012 23:59:59 GMT
Host: www.mysite.com
User-Agent: Mozilla/4.0
Content-Length: 47
Content-Type: application/x-www-form-urlencoded

artist=Picasso&year=1906&nation=Spain
```

5.3 Passing Information via the URL Path

- While query strings are a vital way to pass information from one page to another, they become long and complicated. There is some dispute about whether dynamic URLs (i.e., ones with query string parameters) or static URLs are better.
- Users prefer static URL, but dynamic URLs are essential part of web application development to sent informations. The solution is to rewrite the dynamic URL into a static one. This process is commonly called **URL rewriting**.
- The URL is rewritten differently in different servers. Like, instead of using the querystring in URL, the relevant information is sent in the folder path or filename format.

Eg –
www.somedomain.com/DisplayArtist.php?artist=16
may be to rewritten as:
www.somedomain.com/artists/16.php

http://www.1st-art-gallery.com/Raphael?La-Donna-Velata = 1516
is rewritten as,
http://www.1st-art-gallery.com/Raphael/La-Donna-Velata-1516.html

The file name extension is being rewritten to .html, to make the URL friendlier. These are not static HTML file.

URL Rewriting in Apache and Linux
Depending on web development platform, there are different ways to implement URL rewriting. On web servers running Apache, the solution is to use the mod_rewrite module in Apache along with the .htaccess file.

The mod_rewrite module uses a rule-based rewriting engine that uses regular expressions to change the URLs, so that the requested URL can be mapped or redirected to another URL.

5.4 Cookies

- **Cookies** are a client-side approach for preserving state information. They are name=value pairs that are saved within one or more text files that are managed by the browser. These pairs accompany both server requests and responses within the HTTP header.
- Cookies cannot contain viruses. The user-related information is preserved on the user's computer and is managed by the user's browser.
- Cookies are not associated with a specific page. It is associated with the page's domain.
- The browser and server will exchange cookie information, when using the same domain.
- It is mainly used to maintain the continuity of information, after some time in a web application. One typical use of cookies in a website is to "remember" the visitor, so that the server can customize the site for the user.
- Some sites will use cookies as part of their shopping cart implementation so that items added to the cart will remain there even if the user leaves the site and then comes back later.
- Cookies are also frequently used to keep track of whether a user has logged into a site.

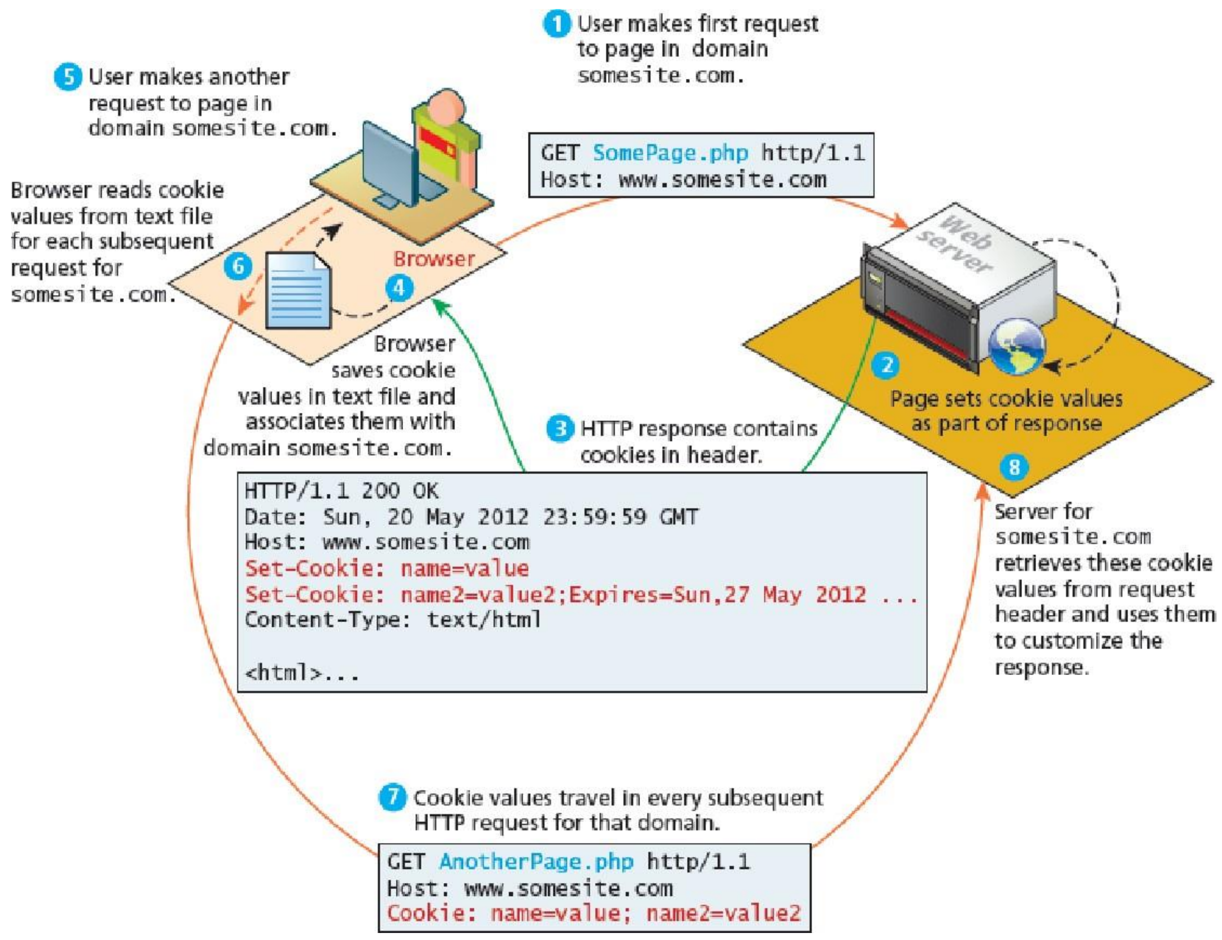
How Do Cookies Work?

- While cookie information is stored and retrieved by the browser, the information in a cookie travels within the HTTP header. There are limitations to the amount of information that can be stored in a cookie (around 4K) and to the number of cookies for a domain (for instance, Internet Explorer 6 limited a domain to 20 cookies).
- Cookies can expire. Expiry time of the cookie is configured during the time of creation. The browser will delete cookies that are beyond their expiry date. If a cookie does not have an expiry date specified, the browser will delete it when the browser closes.

There are two types of cookies: session cookies and persistent cookies.

- A **session cookie** has no expiry stated and thus will be deleted at the end of the user browsing session.
- **Persistent cookies** have an expiry date specified; they will persist in the browser's cookie file until the expiry date occurs, after which they are deleted.

The most important limitation of cookies is that the browser may be configured to refuse the usage of cookies. As a consequence, sites that use cookies should not depend on their availability for critical features. Similarly, the user can also delete cookies or even tamper with the cookies, which may lead to some serious problems if not handled.


1 User makes first request to page in domain somesite.com.
5 User makes another request to page in domain somesite.com.
Browser reads cookie values from text file for each subsequent request for somesite.com.
6
Browser
4
Browser saves cookie values in text file and associates them with domain somesite.com.
Web server
2 Page sets cookie values as part of response
3 HTTP response contains cookies in header.
8
Server for somesite.com retrieves these cookie values from request header and uses them to customize the response.
7 Cookie values travel in every subsequent HTTP request for that domain.

```
GET SomePage.php http/1.1
Host: www.somesite.com
```

```
HTTP/1.1 200 OK
Date: Sun, 20 May 2012 23:59:59 GMT
Host: www.somesite.com
Set-Cookie: name=value
Set-Cookie: name2=value2;Expires=Sun,27 May 2012 ...
Content-Type: text/html

<html>...
```

```
GET AnotherPage.php http/1.1
Host: www.somesite.com
Cookie: name=value; name2=value2
```

Using Cookies

Like any other web development technology, PHP provides mechanisms for writing and reading cookies. Cookies in PHP are *created* using the setcookie() function and are *retrieved* using the $_COOKIES superglobal associative array.

The setcookie() function sets the value to a particular key and also sets the expiry time.

Eg –

Setcookie(key,value, expiry time)

Writing a cookie

```
<?php
// add 1 day to the current time for expiry time
$expiryTime = time()+60*60*24;
// create a persistent cookie
$name = "Username";
$value = "Ricardo";
setcookie($name, $value, $expiryTime);
?>
```

Reading a cookie

```
<?php
if( !isset($_COOKIE['Username']) ) {
        //no valid cookie found
}
else {
        echo "The username retrieved from the cookie is:";
        echo $_COOKIE['Username'];
}
?>
```

Before reading a cookie, we must check to ensure that the cookie exists. In PHP, if the cookie has expired, then the client's browser would not send anything along with the request, and so the $_COOKIE array would be blank

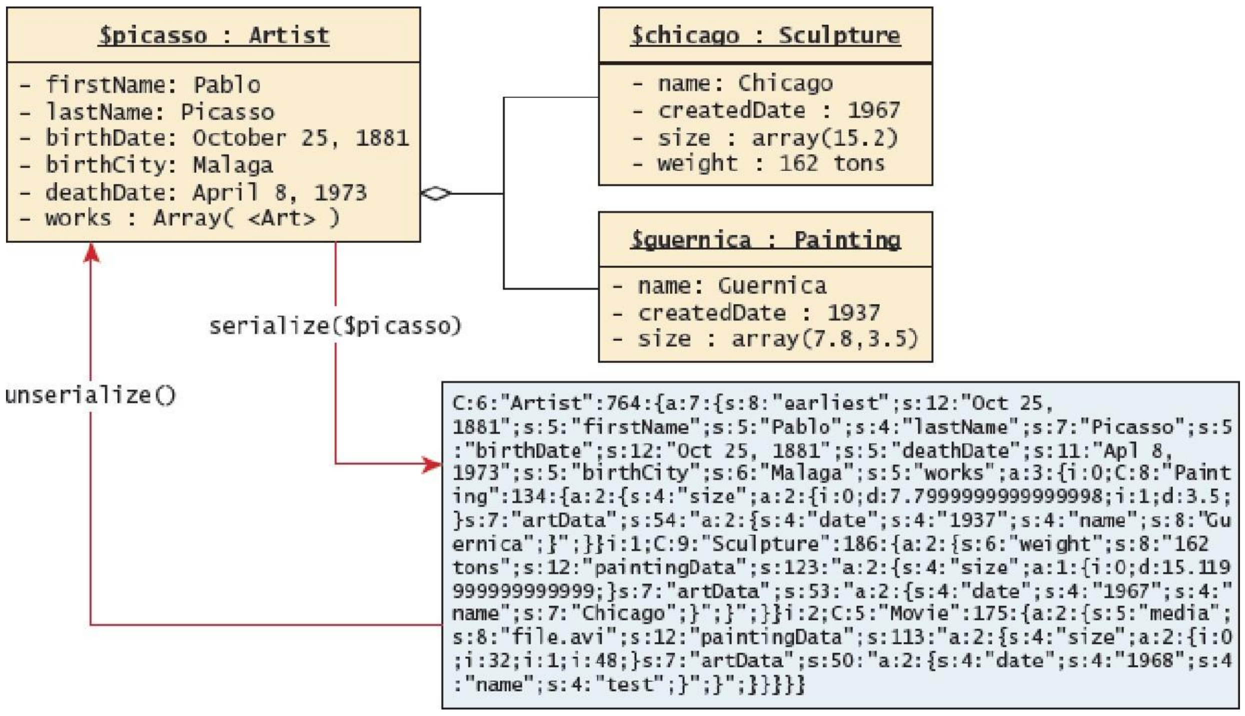
Persistent Cookie Best Practices

- Persistent cookie store the information, which will be retained in the cookie for a long period of time.
- Many sites provide a "Remember Me" checkbox on login forms, which relies on the use of a persistent cookie. This login cookie would contain the user's username but not the password. But cookies should not be used for stored important information required for site's operation.
- Another common use of cookies would be to use them to store user preferences. For instance, some sites allow the user to choose their preferred site color scheme or their country of origin. In these cases, saving the user's preferences in a cookie will make for a more contented user, but if the user's browser does not accept cookies, even then the site will work. But the user will have to reselect his or her preferences again.
- Another use of cookies is to track a user's browsing behavior on a site. This information can be used by the site administrator as an analytic tool to help understand how users navigate through the site.

5.5 Serialization

Serialization is the process of taking a complicated object and converting it to zeros and ones for either storage or transmission. Later that sequence of zeros and ones can be reconstituted into the original object.

In PHP objects can easily be reduced down to a binary string using the serialize() function. The resulting string is a binary representation of the object and therefore may contain unprintable characters. The string can be reconstituted back into an object using the unserialize() method.



```
C:6:"Artist":764:{a:7:{s:8:"earliest";s:12:"Oct 25,
1881";s:5:"firstName";s:5:"Pablo";s:4:"lastName";s:7:"Picasso";s:5
:"birthDate";s:12:"Oct 25, 1881";s:5:"deathDate";s:11:"Apl 8,
1973";s:5:"birthCity";s:6:"Malaga";s:5:"works";a:3:{i:0;C:8:"Paint
ing":134:{a:2:{s:4:"size";a:2:{i:0;d:7.7999999999999998;i:1;d:3.5;
}s:7:"artData";s:54:"a:2:{s:4:"date";s:4:"1937";s:4:"name";s:8:"Gu
ernica";}";}}i:1;C:9:"Sculpture":186:{a:2:{s:6:"weight";s:8:"162
tons";s:12:"paintingData";s:123:"a:2:{s:4:"size";a:1:{i:0;d:15.119
999999999999;}s:7:"artData";s:53:"a:2:{s:4:"date";s:4:"1967";s:4:"
name";s:7:"Chicago";}";}";}}i:2;C:5:"Movie":175:{a:2:{s:5:"media";
s:8:"file.avi";s:12:"paintingData";s:113:"a:2:{s:4:"size";a:2:{i:0
;i:32;i:1;i:48;}s:7:"artData";s:50:"a:2:{s:4:"date";s:4:"1968";s:4
:"name";s:4:"test";}";}";}}}}}
```

Objects must implement the Serializable interface which requires adding implementations for serialize() and unserialize() to any class that implements this interface.

The Serializable interface

```
interface Serializable {
        /* Methods */
        public function serialize();
        public function unserialize($serialized);
}
```

The Student class must be modified to implement the Serializable interface by adding the implements keyword to the class definition and adding implementations for the two methods.

```
class Student implements Serializable {
//...
// Implement the Serializable interface methods
public function serialize() {
// use the built-in PHP serialize function
return serialize(
array("name" => $this->name,
"usn" => $this->usn,
"address" => $this->address,
"average" => $this->avg
);
);
}

public function unserialize($data) {
// use the built-in PHP unserialize function
$data = unserialize($data);
$this->name = $data['name'];
$this->usn = $data['usn'];
$this->address = $data['address'];
$this->avg= $data['average'];
}
//...
}
```

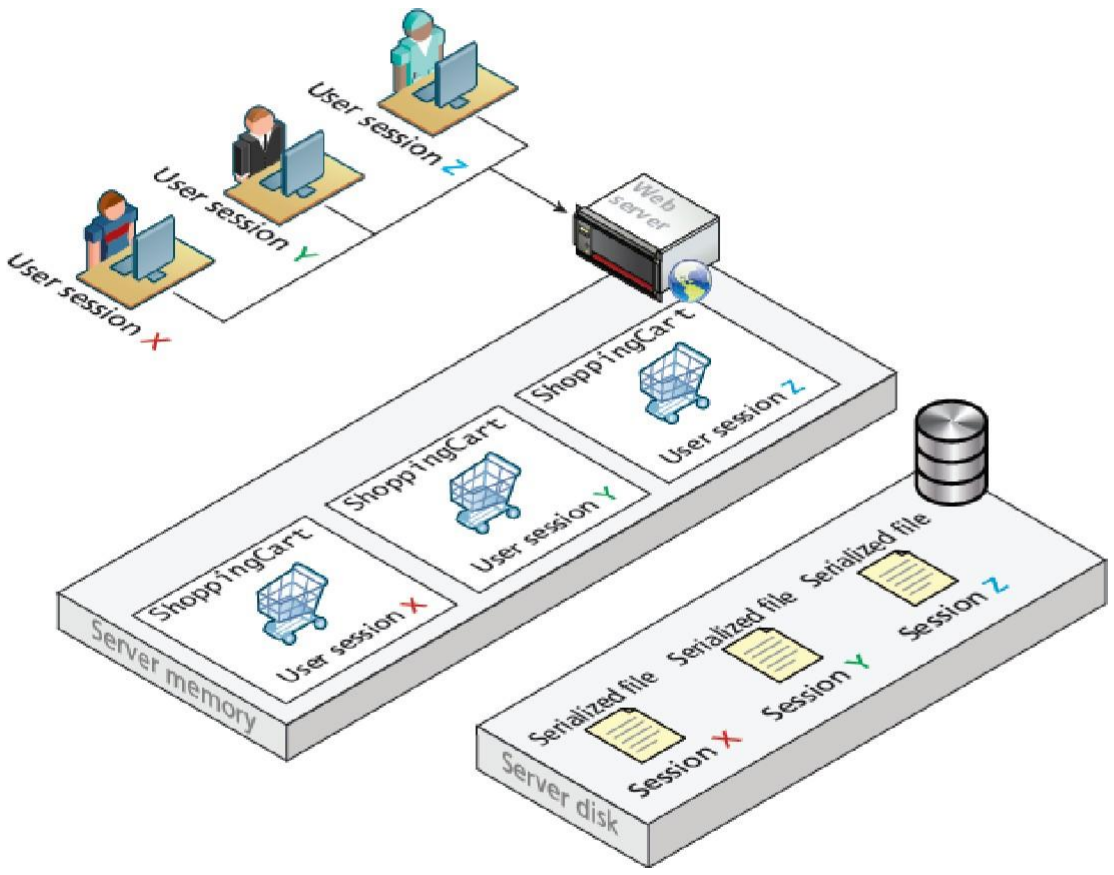
The serialized data can easily be used to reconstruct the object by passing it to unserialize(). In the above example $data is the serialized data. It can be used to recreate the object.
$s1Clone = unserialize($data);

Application of Serialization
Since each request from the user requires objects to be reconstructed, using serialization to store and retrieve objects, it is a rapid way to maintain information between requests. At the end of a request the information is stored in a serialized form, and then the next request would begin by deserializing it to reestablish the previous state.

5.6 Session State

Session state is a server-based state mechanism that allows web applications to store and retrieve objects of any type for each unique user session. That is, each browser session has its own session state stored as a serialized file on the server, which is deserialized and loaded into memory as needed for each request.

As server storage is a finite resource, objects loaded into memory are released when the request completes, making room for other requests and their session objects. This means there can be more active sessions on disk than in memory at any one time. Session state is ideal for storing more complex objects or data structures that are associated with a user session.



In PHP, session state is available to the developer as a superglobal associative array, like the $_GET, $_POST, and $_COOKIE arrays. It can be accessed via the $_SESSION variable.

```
<?php
session_start();
if ( isset($_SESSION['user']) ) {
// User is logged in
}
else {
// No one is logged in (guest)
}
?>
```

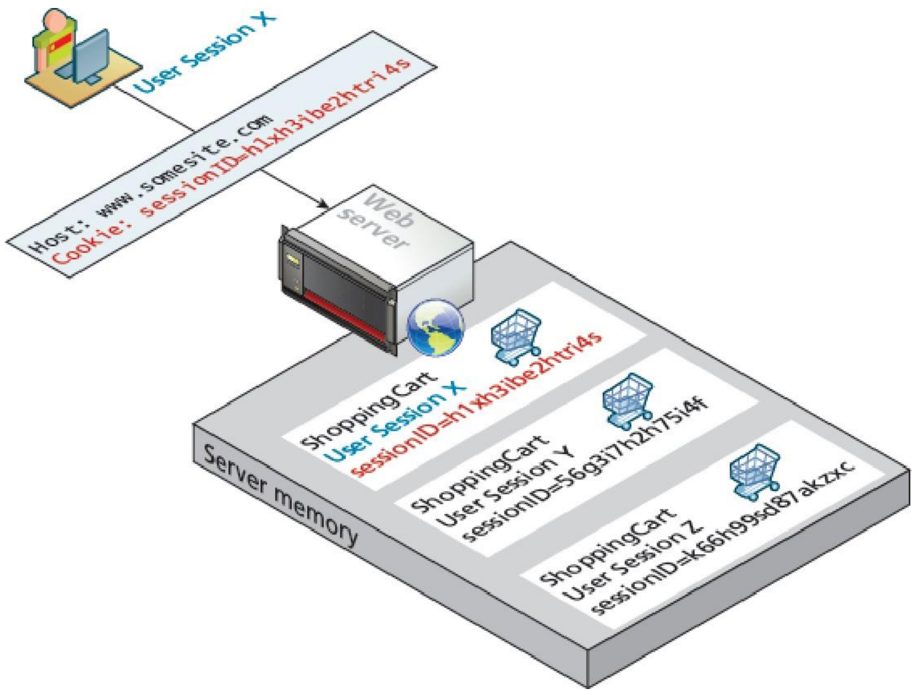
Session information gets deleted after the session. As a result, it should be checked if an item retrieved from session state still exists before using the retrieved object. If the session object does not yet exist (either because it is the first time the user has requested it or because the session has timed out), one might generate an error or redirect to another page.

```
<?php
session_start();
// always check for existence of session object before accessing it
```

```
if ( !isset($_SESSION["Cart"]) ) {
//session variables can be strings, arrays, or objects, but
// smaller is better
$_SESSION["Cart"] = new ShoppingCart();
}
$cart = $_SESSION["Cart"];
?>
```

How Does Session State Work?

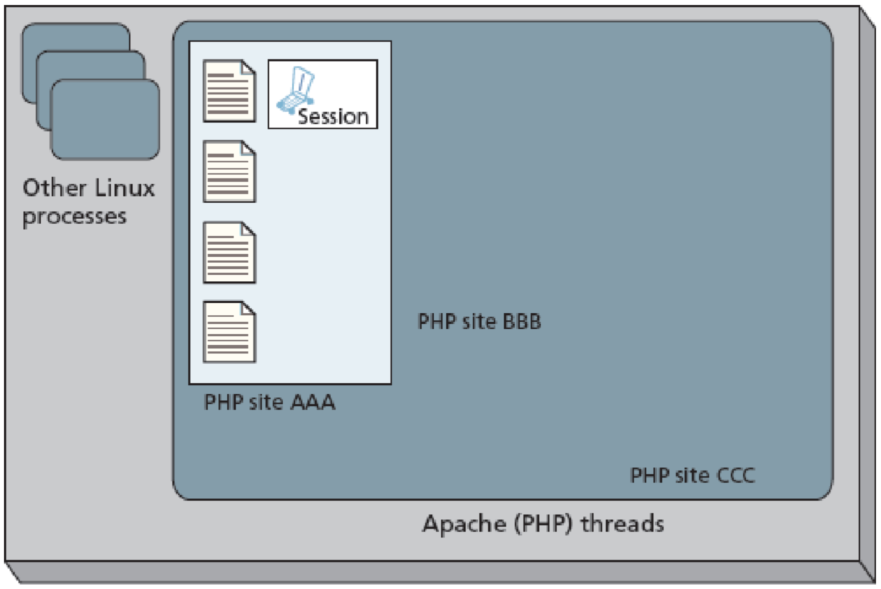
The session state works within the same HTTP context as any web request. Sessions in PHP are identified with a unique session ID. In PHP, this is a unique 32-byte string that is by default transmitted back and forth between the user and the server via a session cookie.



After generating or obtaining of a session ID for a new session, PHP assigns an initially empty dictionary-style collection that can be used to hold any state values for this session. When the request processing is finished, the session state is saved to some type of state storage mechanism, called a session state provider. Finally, when a new request is received for an already existing session, the session's dictionary collection is filled with the previously saved session data from the session state provider.

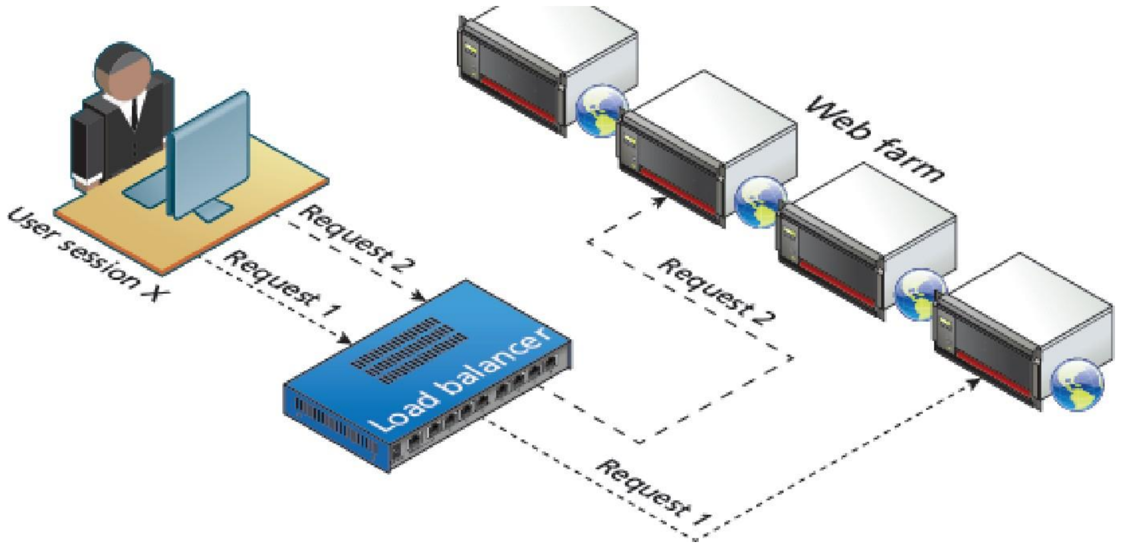
Session Storage and Configuration

The session states can be saved in files. The decision to save sessions to files than in memory resolves the issue of memory usage that can occur on shared hosts. For each application, server memory stores not only session information, but pages being executed, and caching information.



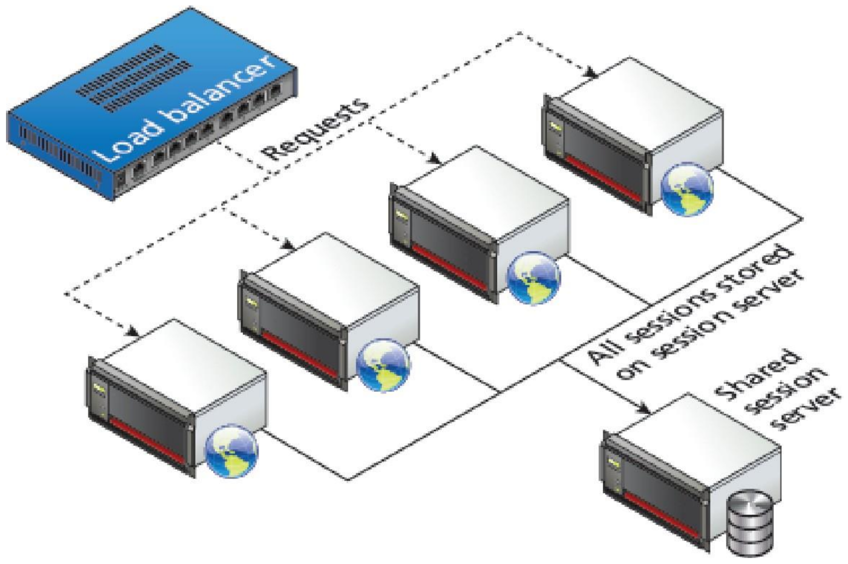
On a busy server hosting multiple sites, it is common for the Apache application process to be restarted on occasion. If the sessions were stored in memory, the sessions would all expire. But as session states are stored into files, they can be instantly recovered as though nothing happened. One disadvantage of storing the sessions in files is degradation in performance compared to memory storage.

Higher-volume web applications often run in an environment in which **multiple web servers** are servicing requests. Each incoming request is forwarded by a load balancer to any one of the available servers in the farm.



In such a situation the in-process session state (session state stored in a server) will not work, since one server may service one request for a particular session, and then a completely different server may service the next request for that session. There are effectively two categories of solution to this problem.

1. Configure the load balancer to be “session aware” and relate all requests using a session to the same server.
2. Use a shared location to store sessions, either in a database, or some other shared session state mechanism.



5.7 HTML5 Web Storage

Web storage is a JavaScript API introduced in HTML5. It is a replacement (or perhaps supplement) to cookies.
Web storage is managed by the browser; and it is not transported to and from the server with every request and response. Web storage is not limited to the 4K size barrier of cookies; browsers are allowed to store more than a limit of 5MB per domain.

There were two types of global web storage objects: localStorage and sessionStorage. The localStorage object is for saving information that will persist between browser sessions. The sessionStorage object is for information that will be lost once the browser session is finished.

Using Web Storage
The below snippet in JavaScript is for writing information to web storage. This mechanism uses JavaScript. There are two ways to store values in web storage: using the setItem() function, or using the property shortcut (e.g., sessionStorage.FavoriteArtist).

```
<form ... >
<h1>Web Storage Writer</h1>
<script language="javascript" type="text/javascript">
if (typeof (localStorage) === "undefined" ||typeof (sessionStorage) === "undefined") {
alert("Web Storage is not supported on this browser...");
}
else {
sessionStorage.setItem("TodaysDate", new Date());
sessionStorage.highestaverage = 92;

localStorage.name = "Ram";
document.write("web storage modified");
}
</script>
</form>
```

The below snippet shows the process of reading from web storage. The difference between sessionStorage and localStorage that if you close the browser after writing and then run the code, only the localStorage item will still contain a value.

```
<form id="form1" runat="server">
<h1>Web Storage Reader</h1>
<script language="javascript" type="text/javascript">
if (typeof (localStorage) === "undefined" || typeof (sessionStorage) === "undefined") {
alert("Web Storage is not supported on this browser...");
}
else {
var today = sessionStorage.getItem("TodaysDate");
var a = sessionStorage. highestaverage;
var user = localStorage.name;
document.write("date saved=" + today);
document.write("<br/>Highest average=" + a);
document.write("<br/>user name = " + user);
}
</script>
</form>
```

Why Would We Use Web Storage?

Cookies have the disadvantage of being limited in size, potentially disabled by the user, vulnerable to security attacks, and for every single request and response the cookie value is sent.

But the advantage of sending cookies with every request and response is also their main advantage, it is easy to implement data sharing between the client browser and the server.

Transporting the information within web storage between the server and browser is a complicated affair involving the construction of a web service on the server and then using asynchronous communication via JavaScript to push the information to the server.

Web storage mechanism is not a cookie replacement but acts as a local cache for relatively static items available to JavaScript. One practical use of web storage is to store static content downloaded asynchronously such as XML or JSON from a web service in web storage, thus reducing server load for subsequent requests by the session.

5.8 Caching

Caching is storing the information. Browser uses caching to speed up the user experience by using locally stored versions of images and other files rather than re-requesting the files from the server.

Caching on server side is necessary, because every time a PHP page is requested, it must be fetched, parsed, and executed by the PHP engine, and the end result is HTML that is sent back to the requestor. For the typical PHP page, this might also involve numerous database queries and processing to build. If this page is being served thousands of times per second, the dynamic generation of that page may become unsustainable.

One way to address this problem is to **cache** the generated markup in server memory so that subsequent requests can be served from memory rather than from the execution of the page. There are two basic strategies to caching web applications –

1. **page output caching**, which saves the rendered output of a page or user control and reuses the output instead of reprocessing the page when a user requests the page again.
2. **application data caching**, which allows the developer to programmatically cache data.

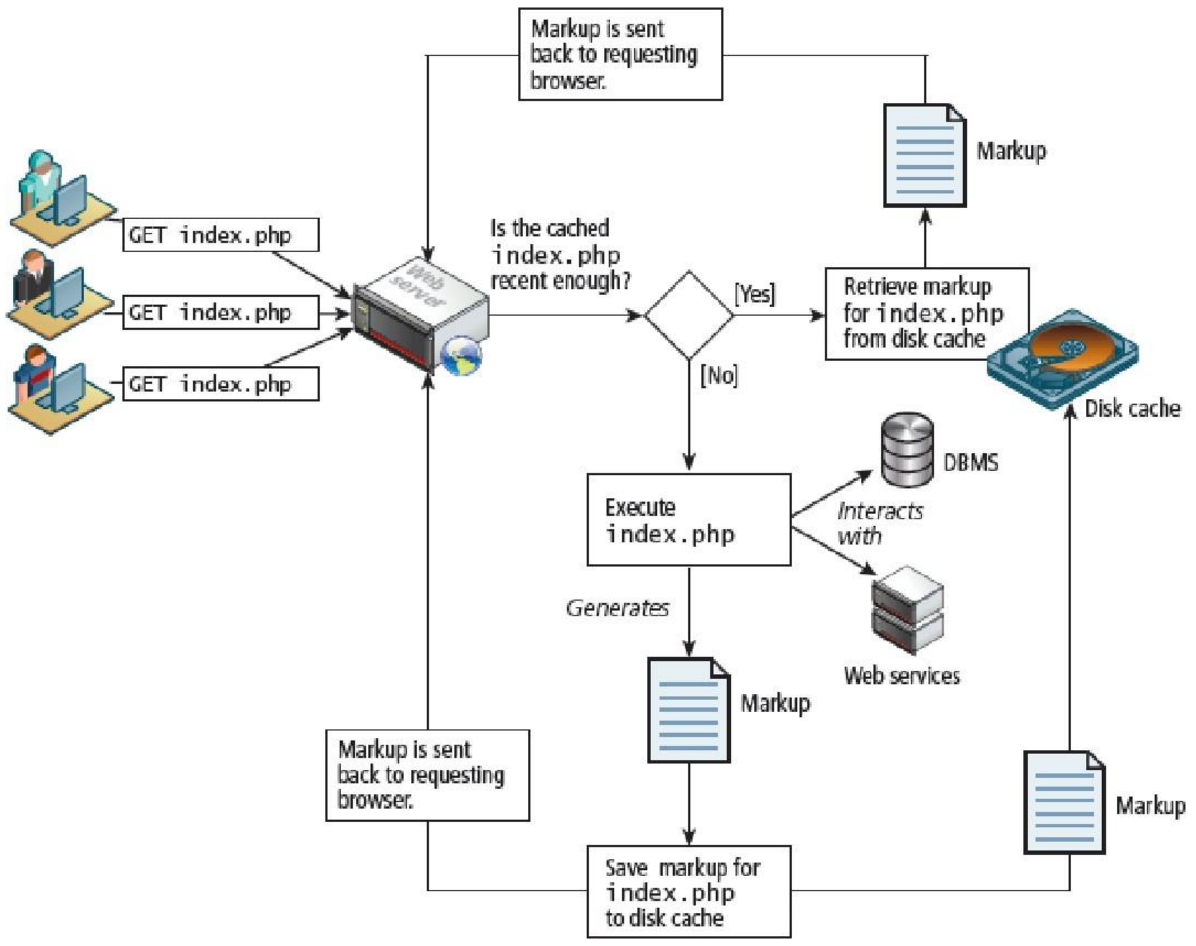
Page Output Caching

In this type of caching, the contents of the rendered PHP page(page ready to display on screen) are written to disk for fast retrieval. This can be particularly helpful because it allows PHP to send a page response to a client without going through the entire page processing life cycle again.

Page output caching is especially useful for pages whose content does not change frequently but which require significant processing to create. There are two models for page caching: full page caching and partial page caching.

In full page caching, the entire contents of a page are cached. In partial page caching, only specific parts of a page are cached while the other parts are dynamically generated in the normal manner.

Page caching is not included in PHP by default, it has to be attached by using add-ons such as Alternative PHP Cache (open source) and Zend (commercial).


Markup is sent back to requesting browser.
Markup
GET index.php
GET index.php
GET index.php
Web server
Is the cached index.php recent enough?
[Yes]
[No]
Retrieve markup for index.php from disk cache
Disk cache
DBMS
Execute index.php
Interacts with
Web services
Generates
Markup
Markup is sent back to requesting browser.
Markup
Save markup for index.php to disk cache

Application Data Caching

One of the biggest drawbacks with page output caching is that performance gains will only be had if the entire cached page is the same for numerous requests.

In application data caching a page will programmatically place commonly used collections of data into cache memory, and then other pages that also need that same data can use the cache version rather than re-retrieve it from its original location. The data that require time-intensive queries from the database or web server are stored in cache memory.

While the default installation of PHP does not come with an application caching ability, a widely available free PECL extension called memcache is widely used to provide this ability. memcache is not used to store large collections. The size of the memory cache is limited. If too many things are placed in it, its performance advantages will be lost as items get paged in and out. Hence it is used to store small collections of data that are frequently accessed on multiple pages.

```
<?php
// create connection to memory cache
$memcache = new Memcache;
```

```
$memcache->connect('localhost', 11211) or die ("Could not connect to memcache server");
$cacheKey = 'topCountries';
/* If cached data exists retrieve it, otherwise generate and cache
it for next time */
if ( ! isset($countries = $memcache->get($cacheKey)) ) {
// since every page displays list of top countries as links
// we will cache the collection
// first get collection from database
$cgate = new CountryTableGateway($dbAdapter);
$countries = cgate->getMostPopular();
// now store data in the cache (data will expire in 240 seconds)
$memcache->set($cacheKey, $countries, false, 240)
or die ("Failed to save cache data at the server");
}
// now use the country collection
displayCountryList($countries);
?>
```

XML Processing & Web Services

5.9 XML Overview

XML (eXtensible Markup Language) is a markup language. It can be used to mark up any type of data. XML is used not only for web development but is also used as a file format in many nonweb applications. One of the key benefits of XML data is that as plain text, it can be read and transferred between applications and different operating systems as well as being human-readable and understandable as well. XML is also used in the web context as a format for moving information between different systems.

XML is not only used on the web server and to communicate asynchronously with the browser, but is also used as a data interchange format for moving information between systems.

Well-Formed XML

Syntax rules for XML -

- Element names are composed of any of the valid characters (most punctuation symbols and spaces are not allowed) in XML.
- Element names can’t start with a number.
- There must be a single-root element. A **root element** is one that contains all the other elements; for instance, in an HTML document, the root element is <html>.
- All elements must have a closing element (or be self-closing).
- Elements must be properly nested.
- Elements can contain attributes.
- Attribute values must always be within quotes.
- Element and attribute names are case sensitive.

Sample XML document –

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
        <painting id="290">
        <title>Balcony</title>
        <artist>
        <name>Manet</name>
        <nationality>France</nationality>
        </artist>
        <year>1868</year>
        <medium>Oil on canvas</medium>
        </painting>

        <painting id="192">
        <title>The Kiss</title>
        <artist>
        <name>Klimt</name>
        <nationality>Austria</nationality>
        </artist>
        <year>1907</year>
        <medium>Oil and gold on canvas</medium>
        </painting>

        <painting id="139">
        <title>The Oath of the Horatii</title>
        <artist>
        <name>David</name>
        <nationality>France</nationality>
        </artist>
        <year>1784</year>
        <medium>Oil on canvas</medium>
        </painting>
</art>
```

In this example, the root element is called <art>. Some type of XML parser is required to verify that an XML document is well formed.

Valid XML

A **valid XML** document is one that is well formed and whose element and content conform to the rules of either its document type definition (DTD) or its schema.

DTDs were the original way for an XML parser to check an XML document for validity. They tell the XML parser which elements and attributes to expect in the document as well as the order and nesting of those elements. A DTD can be defined within an XML document or within an external file.

The DTD for the above XML file is –

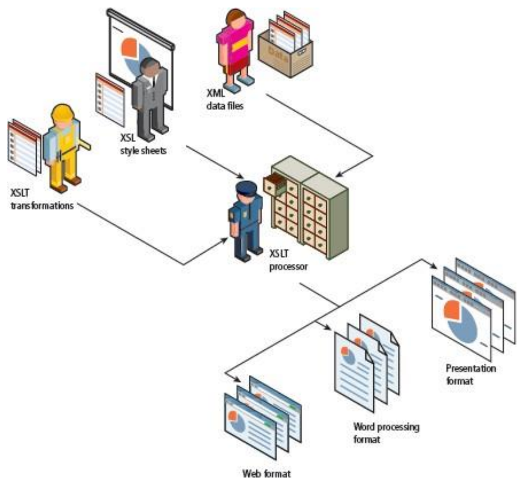
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE art [
<!ELEMENT art (painting*)>
<!ELEMENT painting (title,artist,year,medium)>
<!ATTLIST painting id CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT artist (name,nationality)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT nationality (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT medium (#PCDATA)>
]>
```

The main drawback with DTDs is that they can only validate the existence and ordering of elements (and the existence of attributes). They provide no way to validate the values of attributes or the textual content of elements. For this type of validation, one must instead use XML schemas, which have the added advantage of using XML syntax.

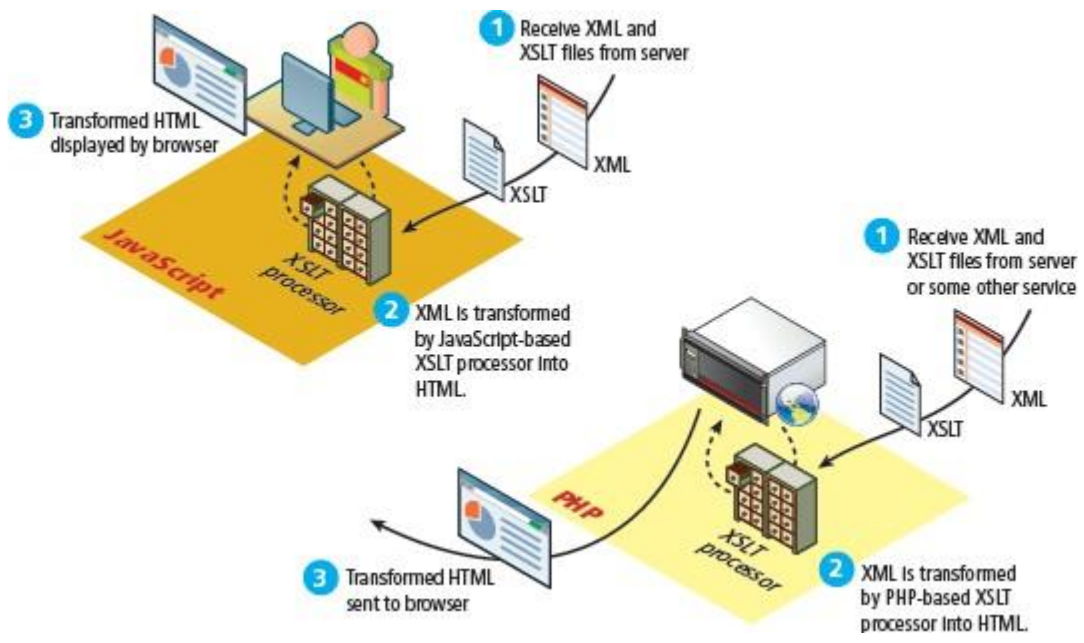
The disadvantage of schema is long-winded and harder for humans to read and comprehend; hence they are usually created with tools.

XSLT

XSLT stands for XML Stylesheet Transformations. XSLT is an XML-based programming language that is used for transforming XML into other document formats. The most common translation is the conversion of XML to HTML.



Usage of XSLT


1 Receive XML and XSLT files from server
XML
XSLT
3 Transformed HTML displayed by browser
JavaScript
XSLT processor
2 XML is transformed by JavaScript-based XSLT processor into HTML.
1 Receive XML and XSLT files from server or some other service
XML
XSLT
PHP
XSLT processor
3 Transformed HTML sent to browser
2 XML is transformed by PHP-based XSLT processor into HTML.

The below example shows an XSLT document that convert the XML into HTML list -

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xsl:version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/1999/xhtml">
<body>
<h1>Catalog</h1>
<ul>
<xsl:for-each select="/art/painting">
<li>
<h2><xsl:value-of select="title"/></h2>
<p>By: <xsl:value-of select="artist/name"/><br/>
Year: <xsl:value-of select="year"/>
[<xsl:value-of select="medium"/>]</p>
</li>
</xsl:for-each>
</ul>
</body>
</html>
```

The strings within the **select attribute**: these are XPath expressions, which are used for selecting specific elements within the XML source document. The <xsl:for-each> element is one of the iteration constructs within XSLT. In this example, it iterates through each of the <painting> elements.

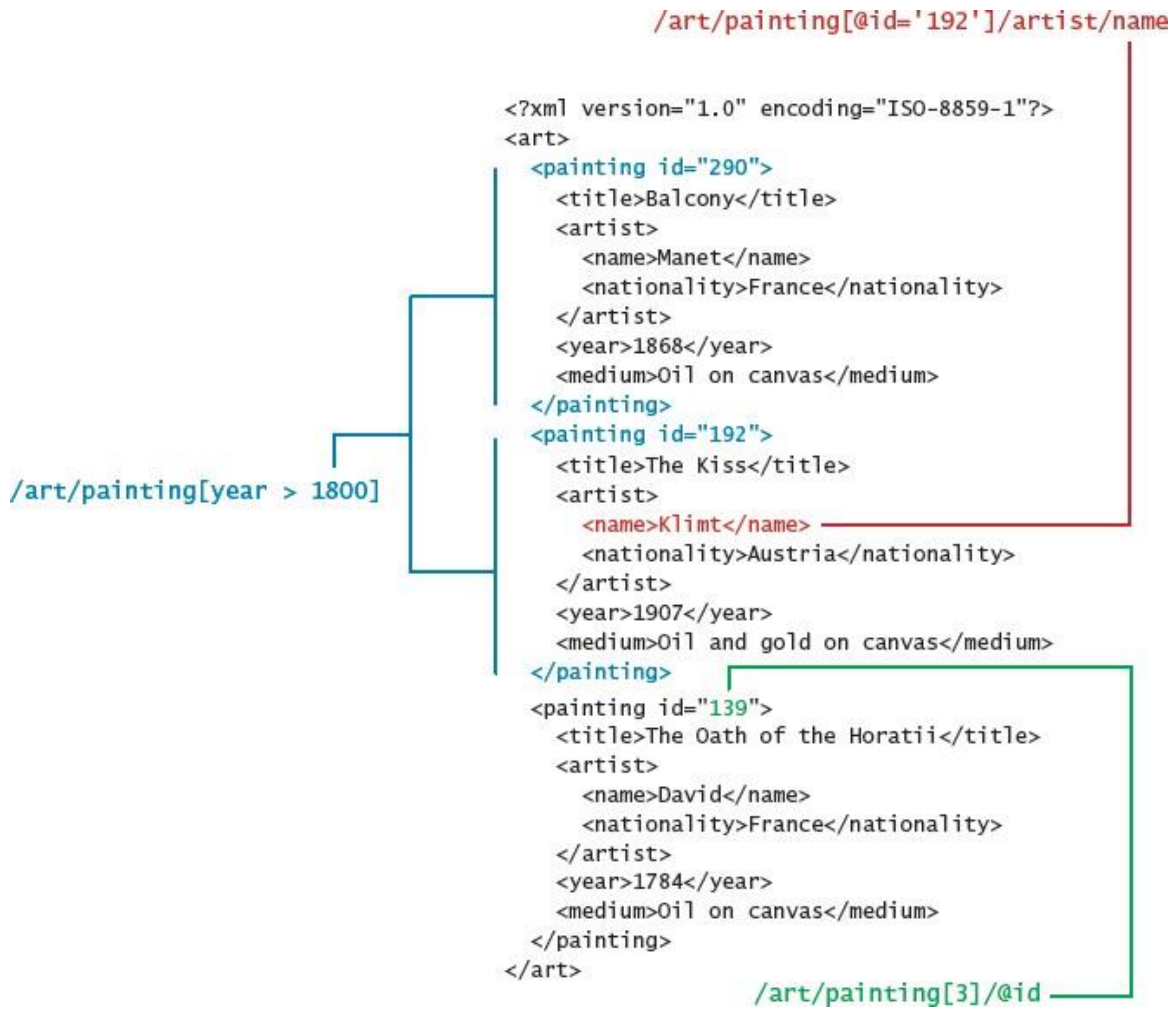
XPath

XPath is a standardized syntax for **searching an XML document** and for navigating to elements within the XML document. XPath is used for programmatically manipulating XML document in PHP and other languages.

XPath uses a syntax that is similar to accessing directories. For instance, to select all the painting elements in the below XML file, XPath expression is - /art/painting. As in file representation, the forward slash is used to separate elements contained within other elements. An XPath expression beginning with a forward slash is an absolute path beginning with the start of the document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
        <painting id="290">
        <title>Balcony</title>
        <artist>
        <name>Manet</name>
        <nationality>France</nationality>
        </artist>
        <year>1868</year>
        <medium>Oil on canvas</medium>
        </painting>

        <painting id="192">
        <title>The Kiss</title>
        <artist>
        <name>Klimt</name>
        <nationality>Austria</nationality>
        </artist>
        <year>1907</year>
        <medium>Oil and gold on canvas</medium>
        </painting>

        <painting id="139">
        <title>The Oath of the Horatii</title>
        <artist>
        <name>David</name>
        <nationality>France</nationality>
        </artist>
        <year>1784</year>
        <medium>Oil on canvas</medium>
        </painting>
</art>
```

In XPath terminology, an XPath expression returns zero, one, or many XML nodes. In XPath, a **node** generally refers to an XML element. From a node, its attributes, textual content, and child nodes can be specifically extracted.


/art/painting[@id='192']/artist/name
/art/painting[year > 1800]
/art/painting[3]/@id

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<art>
  <painting id="290">
    <title>Balcony</title>
    <artist>
      <name>Manet</name>
      <nationality>France</nationality>
    </artist>
    <year>1868</year>
    <medium>Oil on canvas</medium>
  </painting>
  <painting id="192">
    <title>The Kiss</title>
    <artist>
      <name>Klimt</name>
      <nationality>Austria</nationality>
    </artist>
    <year>1907</year>
    <medium>Oil and gold on canvas</medium>
  </painting>
  <painting id="139">
    <title>The Oath of the Horatii</title>
    <artist>
      <name>David</name>
      <nationality>France</nationality>
    </artist>
    <year>1784</year>
    <medium>Oil on canvas</medium>
  </painting>
</art>
```

The XPath expression: '/art/painting[@id='192']/artist/name' - selects the <name> element within the <artist> element for the <painting> element with an id attribute of 192.
Here square brackets are used to specify a criteria expression at the current path node, in the above example is /art/painting (i.e., each painting node is examined to see if its id attribute is equal to the value 192).

The XPath expression: '/art/painting[3]/@id' - selects the id attribute of the element <painting> whose index is 3. XPath expressions begin with one and not zero. Attributes are identified in XPath expressions by being prefaced by the @ character.

The XPath expression: '/art/painting[year >1800]' - selects the whole <painting> element whos <year> element value is > 1800.

5.10 XML Processing

XML processing in PHP, JavaScript, and other modern development environments is divided into two basic styles:

1. The **in-memory approach**, which involves reading the entire XML file into memory into some type of data structure with functions for accessing and manipulating the data.
2. The **event or pull approach**, which lets you pull in just a few elements or lines at a time, thereby avoiding the memory load of large XML files.

XML Processing in JavaScript

All modern browsers have a built-in XML parser and their JavaScript implementations support an **in-memory** XML DOM API, which loads the entire document into memory where it is transformed into a hierarchical tree data structure.

The DOM functions such as getElementById(), getElementsByTagName(), and createElement() are used to access and manipulate the data.
For instance, the below code shows the loading of an XML document into an XML DOM object, and it displays the id attributes of the <painting>elements as well as the content of each painting's <title> element

```
<script>
// load the external XML file
xmlhttp.open("GET","art.xml",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

// now extract a node list of all <painting> elements
paintings = xmlDoc.getElementsByTagName("painting");

if (paintings) {
        // loop through each painting element
        for (var i = 0; i < paintings.length; i++)
        {
                // display its id attribute
                alert("id="+paintings[i].getAttribute("id"));
                // find its <title> element
                title = paintings[i].getElementsByTagName("title");
                if (title) {
                // display the text content of the <title> element
                alert("title="+title[0].textContent);
                }
        }
}
</script>
```

JavaScript supports a variety of node traversal functions as well as properties for accessing information within an XML node.

alert() function – displays the string in a alert dialog box.
GetAttribute(id) - returns the value of the attribute.
getElementsByTagName(tag) – returns an array of tags with the specified title.
textContent() – returns the html content in the specified tag.

XML Processing in PHP
PHP provides several extensions or APIs for working with XML -

- The **DOM extension** loads the entire document into memory where it is transformed into a hierarchical tree data structure. This DOM approach is a standardized approach. Diffeent languages implementing this approach use relatively similarly named functions/methods for accessing and manipulating the data.

- The **SimpleXML** extension loads the data into an object that allows the developer to access the data via array properties and modifying the data via methods.

- The **XML parser** is an event-based XML extension.

- The **XMLReader** is a read-only pull-type extension that uses a cursor-like approach similar to that used with database processing. The XMLWriter provides an analogous approach for creating XML files.

In general, the SimpleXML and the XMLReader extensions are used to read and process XML content. The SimpleXML approach reads the entire XML file into memory and transforms into a complex object. It is not usually used for processing very large XML files because it reads the entire file into server memory; however, since the file is in memory, it offers fast performance.
Eg –

```
<?php
$filename = 'art.xml';
if (file_exists($filename)) {
        $art = simplexml_load_file($filename);

        // access a single element
        $painting = $art->painting[0];
        echo '<h2>' . $painting->title . '</h2>';
        echo '<p>By ' . $painting->artist->name . '</p>';

        // display id attribute
        echo '<p>id=' . $painting["id"] . '</p>';
} else {
exit('Failed to open ' . $filename);
}
?>
```

simplexml_load_file() function - XML file is transformed into an object.
The elements in the XML document is manipulated using regular PHP object techniques using the -> operator.

The SimpleXML extension is used to only small XML files as the read contents are stored directly to memory. For reading the large XML files, the XMLReader is a used. The XMLReader is sometimes referred to as a pull processor, here it reads a single node at a time, the value in that node is processed, and then the next node is read.

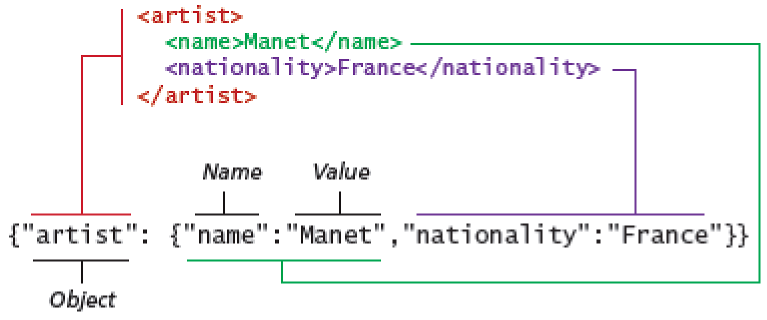
```
$filename = 'art.xml';
if (file_exists($filename)) {
        // create and open the reader
        $reader = new XMLReader();
        $reader->open($filename);

        // loop through the XML file
        while ( $reader->read() ) {
                $nodeName = $reader->name;
                // check node type
                if ($reader->nodeType == XMLREADER::ELEMENT
                && $nodeName == 'painting') {
                        $id = $reader->getAttribute('id');
                        echo '<p>id=' . $id . '</p>';
                }
                if ($reader->nodeType == XMLREADER::ELEMENT
                && $nodeName =='title') {
                // read the next node to get at the text node
                $reader->read();
                echo '<p>' . $reader->value . '</p>';
        }
        }
} else {
exit('Failed to open ' . $filename);
}
```

5.11 JSON

JSON is a data serialization format, like XML. That is, it is used to represent object data in a text format so that it can be transmitted from one computer to another. Many REST web services encode their returned data in the JSON data format instead of XML. While **JSON** stands for **JavaScript Object Notation**, its use is not limited to JavaScript. It was originally designed to provide a lightweight serialization format to represent objects in JavaScript. While it doesn’t have the validation and readability of XML, it has the advantage of generally requiring significantly fewer bytes to represent data than XML.

An example of how an XML data element would be represented in JSON is –

```
{"artist": {"name":"Manet","nationality":"France"}}
```


<artist>
<name>Manet</name>
<nationality>France</nationality>
</artist>
Name
Value
{"artist": {"name":"Manet","nationality":"France"}}
Object

Just like XML, JSON data can be nested to represent objects within objects.

```
{
"paintings": [
{
"id":290,
"title":"Balcony",
"artist":{
"name":"Manet",
"nationality":"France"
},
"year":1868,
"medium":"Oil on canvas"
},
{
"id":192,
"title":"The Kiss",
"artist":{
"name":"Klimt",
"nationality":"Austria"
},
"year":1907,
"medium":"Oil and gold on canvas"
},
{
"id":139,
"title":"The Oath of the Horatii",
"artist":{
"name":"David",
"nationality":"France"
},
"year":1784,
"medium":"Oil on canvas"
}
]
}
```

In general JSON data will have all white space removed to reduce the number of bytes traveling across the network. Square brackets are used to contain the three painting object definitions: this is the JSON syntax for defining an array.

Using JSON in JavaScript

Since the syntax of JSON is the same used for creating objects in JavaScript, it is easy to make use of the JSON format in JavaScript:

```
<script>
var a = {"artist": {"name":"Manet","nationality":"France"}};
alert(a.artist.name + " " + a.artist.nationality);
</script>
```

The JSON information will be contained within a string, and the JSON.parse() function can be used to transform the string containing the JSON data into a JavaScript object:

```
var text = '{"artist": {"name":"Manet","nationality":"France"}}';
var a = JSON.parse(text);
alert(a.artist.nationality);
```

JavaScript also provides a mechanism to translate a JavaScript object into a JSON string:

```
var text = JSON.stringify(artist);
```

Using JSON in PHP

PHP comes with a JSON extension . Converting a JSON string into a PHP object is quite straightforward:

```
<?php
// convert JSON string into PHP object
$text = '{"artist": {"name":"Manet","nationality":"France"}}';
$anObject = json_decode($text);
echo $anObject->artist->nationality;
// convert JSON string into PHP associative array
$anArray = json_decode($text, true);
echo $anArray['artist']['nationality'];
?>
```

The json_decode() function can return either a PHP object or an associative array. Since JSON data is often coming from an external source, we should check for parse errors before using it, which can be done via the json_last_error() function:

```
<?php
```

```
// convert JSON string into PHP object
$text = '{"artist": {"name":"Manet","nationality":"France"}}';
$anObject = json_decode($text);
// check for parse errors
if (json_last_error() == JSON_ERROR_NONE) {
echo $anObject->artist->nationality;
}
?>
```

To convert a PHP object into a JSON string, use the json_encode() function.

```
// convert PHP object into a JSON string
$text = json_encode($anObject);
```

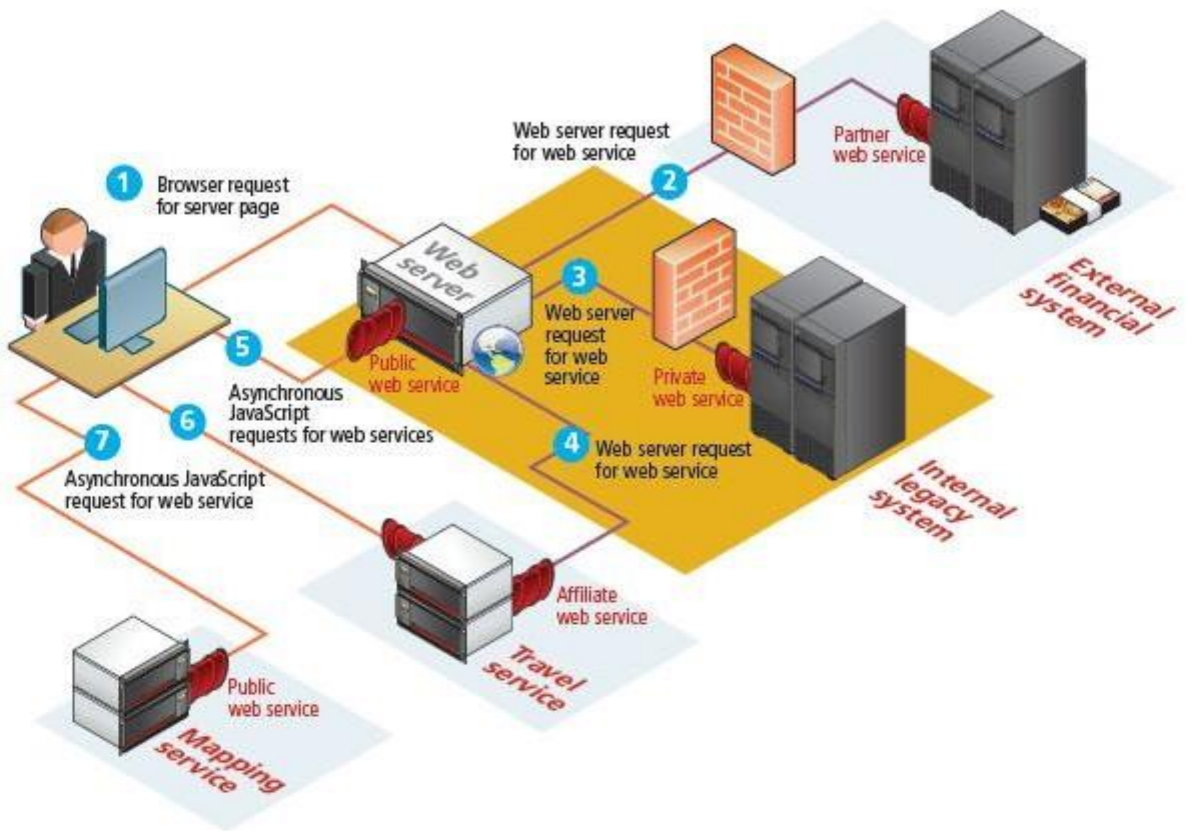
5.12 Overview of Web Services

Web services are the most common example of a computing paradigm commonly referred to as **service-oriented computing** (SOC), which utilizes something called “services” as a key element in the development and operation of software applications.

A **service** is a piece of software with a platform-independent interface that can be dynamically located and invoked. **Web services** are a relatively standardized mechanism by which one software application can connect to and communicate with another software application using web protocols. Web services make use of the **HTTP protocol** so that they can be used by any computer with Internet connectivity. Web services use XML or JSON to encode data within HTTP transmissions so that almost any platform should be able to encode or retrieve the data contained within a web service.

The benefit of web services is that they potentially provide interoperability between different software applications running on different platforms. Because web services use common and universally supported standards (HTTP and XML/ JSON), they are supported on a wide variety of platforms. Another key benefit of web services is that they can be used to implement **service-oriented architecture** (SOA). This type of software architecture aims to achieve very loose coupling among interacting software services.

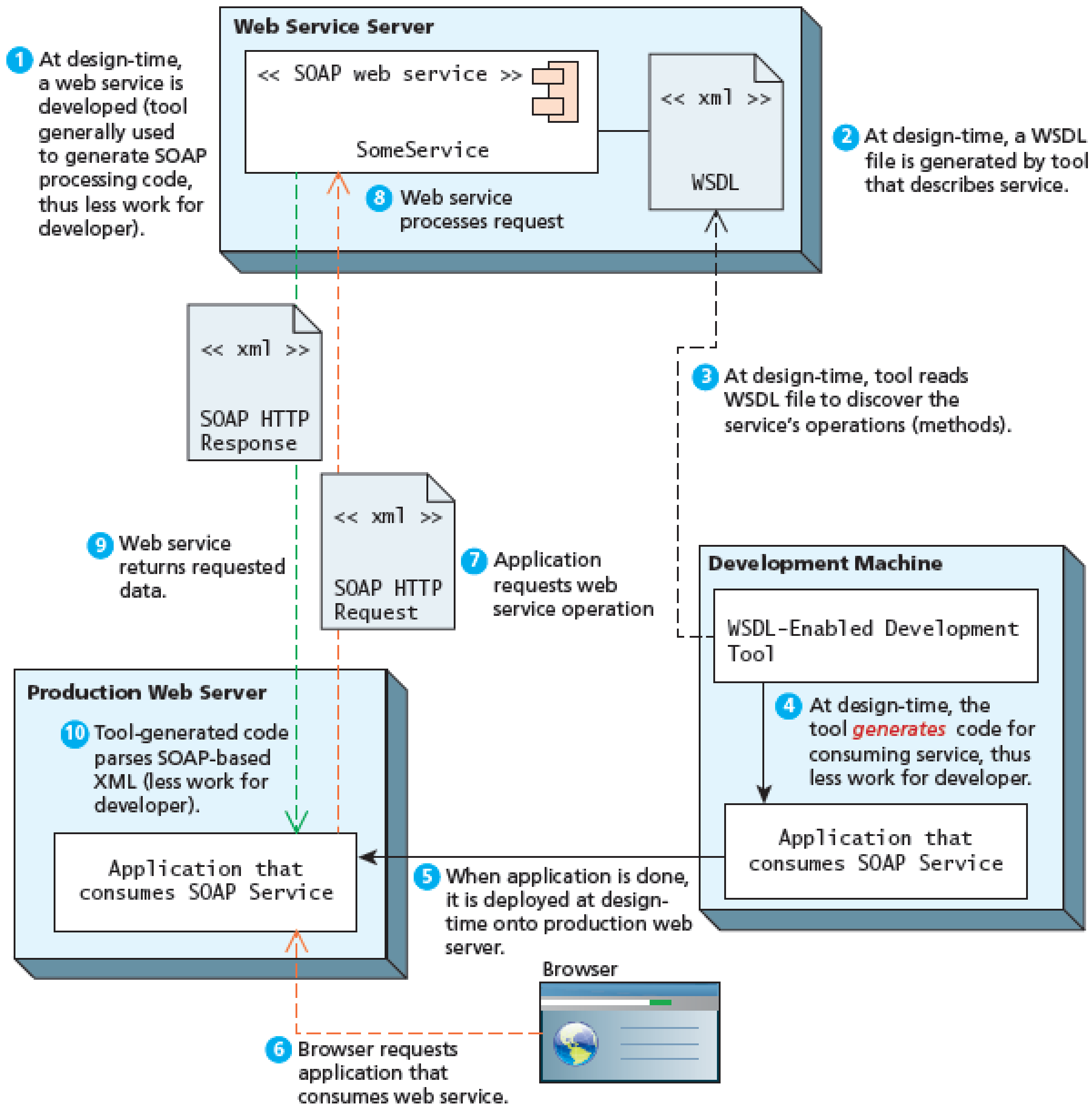
Overview of web services


1 Browser request for server page
2 Web server request for web service
3 Web server request for web service
4 Web server request for web service
5 Asynchronous JavaScript requests for web services
6
7 Asynchronous JavaScript request for web service
Web server
Public web service
Partner web service
Private web service
Affiliate web service
Public web service
External financial system
Internal legacy system
Travel service
Mapping service

SOAP Services
A series of related technologies are used in web services, namely –
SOAP(Simple Object Access Protocol)
WSDL(Web Services Description Language)
REST (Representational State Transfer)

SOAP is the message protocol used to encode the service invocations and their return values via XML within the HTTP header, using POST method.
WSDL is used to describe the operations and data types provided by the service.
REST is the message protocol used to encode the service invocations and their return values via XML/JSON/text format within the HTTP header, GET method.

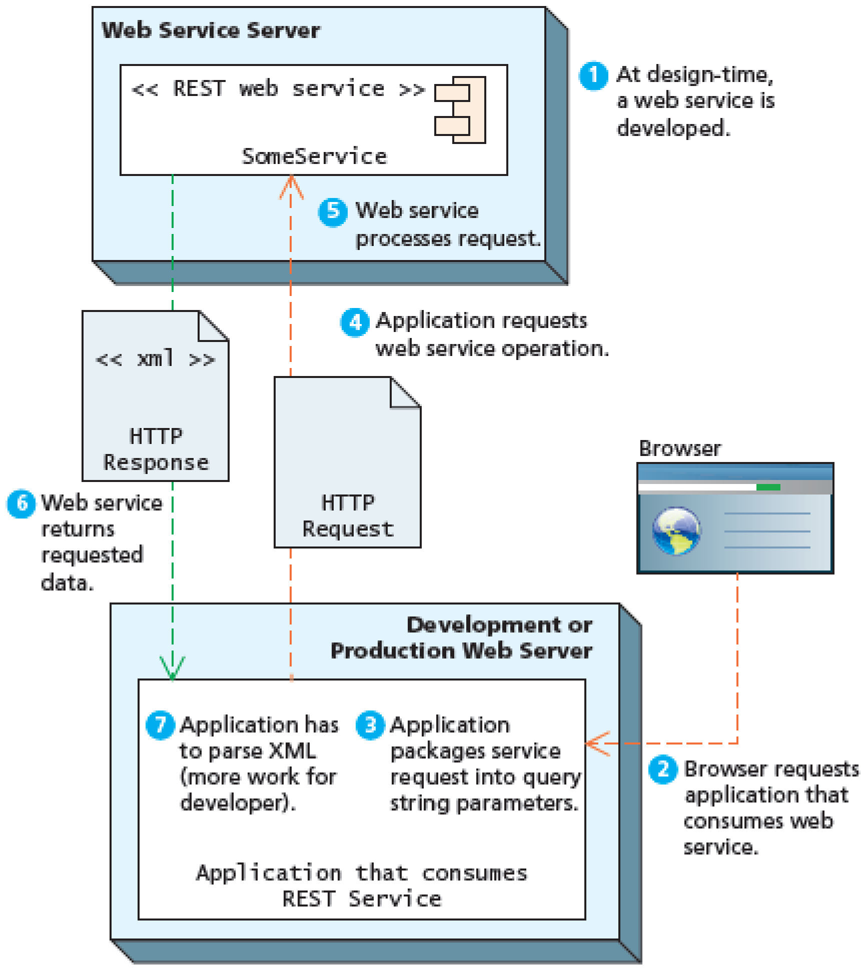
SOAP and WSDL are complex XML schemas, and is well supported in the .NET and Java environments. Detailed knowledge of the SOAP and WSDL specifications is not required to create and consume SOAP-based services. Using SOAP-based services is like compiler: its output may be complicated to understand, but it certainly makes life easier for most programmers. There are standard tools to generate SOAP and WSDL files like compiled files.


Web Service Server
<< SOAP web service >>
SomeService
<< xml >>
WSDL
1 At design-time, a web service is developed (tool generally used to generate SOAP processing code, thus less work for developer).
2 At design-time, a WSDL file is generated by tool that describes service.
3 At design-time, tool reads WSDL file to discover the service's operations (methods).
8 Web service processes request
<< xml >>
SOAP HTTP Response
<< xml >>
SOAP HTTP Request
9 Web service returns requested data.
7 Application requests web service operation
Development Machine
WSDL-Enabled Development Tool
4 At design-time, the tool generates code for consuming service, thus less work for developer.
Application that consumes SOAP Service
Production Web Server
10 Tool-generated code parses SOAP-based XML (less work for developer).
Application that consumes SOAP Service
5 When application is done, it is deployed at design-time onto production web server.
Browser
6 Browser requests application that consumes web service.

REST Services

REST stands for Representational State Transfer. It is implemented as an alternative to SOAP. A RESTful web service does not need a service description layer and also does not need a separate protocol for encoding message requests and responses. Instead it simply uses HTTP URLs for requesting a resource/object. The serialized representation of this object (XML or JSON stream), is then returned to the requestor as a normal HTTP response.

No special steps are needed to deploy a REST-based service, no special tools are generally needed to use a RESTful service, and it is easier to scale for a large number of clients using well-established practices and experience with caching, clustering, and loadbalancing traditional dynamic HTTP websites.


Web Service Server
<< REST web service >>
SomeService
1 At design-time, a web service is developed.
5 Web service processes request.
4 Application requests web service operation.
<< xml >>
HTTP Response
HTTP Request
Browser
6 Web service returns requested data.
Development or Production Web Server
7 Application has to parse XML (more work for developer).
3 Application packages service request into query string parameters.
Application that consumes REST Service
2 Browser requests application that consumes web service.

The lightweight nature of REST made it significantly easier to use than SOAP, hence REST appears to have almost completely displaced SOAP services. If an object is serialized via JSON, it can be turned into a complex JavaScript object in one simple line of JavaScript.

The easy availability of a wide range of RESTful services has given rise to a new style of web development, often referred to as a **mashup**, which generally refers to a website that combines and integrates data from a variety of different sources.

An Example Web Service

The example web service studied here is the Google Geocoding API. The term **geocoding** typically refers to the process of turning a real-world address (such as **British Museum**, **Great Russell Street**, **London**, **WC1B 3DG**) into geographic coordinates, which are usually latitude and longitude values (such as **51.5179231**, **-0.1271022**).
Reverse geocoding is the process of converting geographic coordinates into a human-readable address.
The Google Geocoding API provides a way to perform geocoding operations via an HTTP GET request, and thus is an especially useful example of a RESTful web service.

Like all of the REST web services we will examine a web service beginning with an HTTP request. In this case the request will take the following form:
http://maps.googleapis.com/maps/api/geocode/xml?*parameters*
The parameters in this case are address (for the real-world address to geocode) and sensor (for whether the request comes from a device with a location sensor).

So an example geocode request would look like the following:
http://maps.googleapis.com/maps/api/geocode/xml?address=British%20Museum,+Great+Russell+Street,+London,+WC1B+3DG&sensor=false
Notice that a REST request, like all HTTP requests, must URL encode special characters such as spaces. If the request is well formed and the service is working, it will return an HTTP response similar to that shown below.

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8
Date: Fri, 19 Jul 2013 19:15:54 GMT
Expires: Sat, 20 Jul 2013 19:15:54 GMT
Cache-Control: public, max-age=86400
Vary: Accept-Language
Content-Encoding: gzip
Server: mafe
Content-Length: 512
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
        <status>OK</status>
        <result>
                <type>route</type>
                <formatted_address>
                        Great Russell Street, London Borough of Camden, London, UK
                </formatted_address>
                <address_component>
                        <long_name>Great Russell Street</long_name>
                        <short_name>Great Russell St</short_name>
                        <type>route</type>
                </address_component>
                <address_component>
                        <long_name>London</long_name>
                        <short_name>London</short_name>
                        <type>locality</type>
                        <type>political</type>
                </address_component>
                ...
                <geometry>
                        <location>
```

```
                    <lat>51.5179231</lat>
                    <lng>-0.1271022</lng>
               </location
               <location_type>GEOMETRIC_CENTER</location_type>
               ...
          </geometry>
     </result>
</GeocodeResponse>
```

After receiving this response, XML processing is done in order to extract the latitude and longitude values.

Identifying and Authenticating Service Requests
The Geocoding service illustrated a service that was openly available to any request. Most web services are not open in the same way. Instead, they typically employ one of the following techniques:
■ **Identity**. Each web service request must identify who is making the request.
■ **Authentication**. Each web service request must provide additional evidence that they are who they say they are.
Many web services are not providing information that is especially private or proprietary. For instance, the Flickr web service, which provides URLs to publicly available photos on their site in response to search criteria, is in some ways simply an XML version of the main site's already existing search facility. Since no private user data is being requested, it only expects each web service request to include one or more API keys to identify who is making the request.

This typically is done not only for internal record-keeping, but more importantly to keep service request volume at a manageable level. Most external web service APIs limit the number of web service requests that can be made, generally either per second, per hour, or per day. For instance, Panoramio limits requests to 100,000 per day while Google Maps and Microsoft Bing Maps allow 50,000 geocoding requests per day; Instagram allows 5000 requests per hour but Twitter allows just 100 to 400 requests per hour (it can vary); Amazon and last.fm limit requests to just one per second. Other services such as Flickr, NileGuide, and YouTube have no documented request limits.

While some web services are simply providing information already available on their website, other web services are providing private/proprietary information or are involving financial transactions. In this case, these services not only may require an API key, but they also require some type of user name and password in order to perform an authorization.

Advanced JavaScript & jQuery

5.13 JavaScript Pseudo-Classes

Although JavaScript has no formal class mechanism, it does support objects (such as the DOM). While most object-oriented languages that support objects also support classes formally, JavaScript does not. Instead, you define **pseudo-classes** through a variety of interesting and nonintuitive syntax constructs. Many common features of object-oriented programming, such as inheritance and even simple methods, must be arrived at through these nonintuitive means. Benefits of using object-oriented design in your JavaScript include increased code reuse, better memory management, and easier maintenance.

Almost all modern frameworks (such as jQuery and the Google Maps API) use prototypes to simulate classes, so understanding the mechanism is essential to apply those APIs in applications.
An object can be instantiated using an object represented by the list of key-value pairs with colons between the key and value with commas separating key-value pairs.
A dice object, with a string to hold the color and an array containing the values representing each side (face), could be defined all at once using object literals as follows:

```
var oneDie = { color : "FF0000", faces : [1,2,3,4,5,6] };
```

Once defined, these elements can be accessed using dot notation. For instance, one could change the color to blue by writing:

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

Emulate Classes through Functions
Although a formal *class* mechanism is not available to us in JavaScript, it is possible to get close by using functions to encapsulate variables and methods together, as shown below.

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

The ‘this’ keyword inside of a function refers to the instance, so that every reference to internal properties or methods manages its own variables, as is the case with PHP. One can create an instance of the object as follows, very similar to PHP.

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

Adding Methods to the Object
One of the most common features one expects from a class is the ability to define behaviors with methods. In JavaScript this is relatively easy to do syntactically. To define a method in an object’s function one can either define it internally, or use a reference to a function defined outside the class. External definitions can quickly cause namespace conflict issues, since all method names must remain conflict free with all other methods for other classes. For this reason, one technique for adding a method inside of a class definition is by assigning an anonymous function to a variable, as shown below.

```
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];
};
}
```

With this method so defined, all dice objects can call the randomRoll function, which will return one of the six faces defined in the Die constructor.

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

Although this mechanism for methods is effective, it is not a memory-efficient approach because each inline method is redefined for each new object.

Using Prototypes

Prototypes are an essential syntax mechanism in JavaScript, and 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*. So modification of the definition of the randomRoll() method, is done as shown below by moving the randomRoll() method 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
```

This definition is better because it defines the method only once, no matter how many instances of Die are created. The below figure shows how the prototype object (not class) is updated to contain the method so that subsequent instantiations (x and y) refers that one-method definition. 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.

The above definition sounds almost like a class in an object-oriented language, except that a prototype is itself an *object*, whereas in other oriented-oriented languages a class is an abstraction, not an object. Despite this distinction, you can make use of a function's prototype object, and assign properties or methods to it that are then available to any new objects that are created.

In addition to the obvious application of prototypes to our own pseudo-classes, prototypes enable you to *extend* existing classes by adding to their prototypes.

Imagine a method added to the String object, which allows you to count instances of a character. The below snippet defines a method, named countChars, that takes a character as a parameter.

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

Now any new instances of String will have this method available to them. For instance the following example will output Hello World has 3 letter l's.

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

This technique is also useful to assign properties to a pseudo-class that you want available to all instances. Imagine an array of all the *valid* characters attached to some custom string class.

5.14 jQuery Foundations

A **library** or **framework** is software that can be utilized in your own software, which provides some common implementations of standard ideas. A web framework can be expected to have features related to the web including HTTP headers, AJAX, authentication, DOM manipulation, cross-browser implementations, and more.

jQuery's beginnings date back to August 2005, when jQuery founder John Resig was looking into how to better combine CSS selectors with succinct JavaScript notation. Within a year, AJAX and animations were added, and the project has been improving ever since. 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** is now the most popular JavaScript library currently in use.

Since the entire library exists as a source JavaScript file, importing jQuery for use in application is as easy as including a link to a file in the <head> section of HTML page. Either link to a locally hosted version of the library or use an approved third-party host, such as Google, Microsoft, or jQuery itself. Using a third-party **content delivery network (CDN)** is advantageous for several reasons,

1. the bandwidth of the file is offloaded to reduce the demand on your servers

2. the user may already have cached the third-party file and thus not have to download it again, thereby reducing the total loading time. This probability is increased when using a CDN like Google rather than a developer-focused CDN like jQuery.

A disadvantage to the third-party CDN is that your jQuery will fail if the thirdparty host fails, although that is unlikely given the mission-critical demands of large companies like Google and Microsoft.

jQuery Selectors

Selectors offer the developer a way of accessing and modifying a DOM object from an HTML page in a simple way. Although the advanced querySelector() methods allow selection of DOM elements based on CSS selectors, it is only implemented in newest browsers. To address this issue jQuery introduces its own way to select an element. jQuery builds on the CSS selectors and adds its own to let you access elements as you would in CSS or using new shortcut methods.

The relationship between DOM objects and selectors is so important in JavaScript programming that the pseudo-class bearing the name of the framework, jQuery(), lets programmers easily access DOM objects using selectors passed as parameters. Because it is used so frequently, it has a shortcut notation and can be written as $(). This $() syntax can be confusing to PHP developers at first, since in PHP the $ symbol indicates a variable. Nonetheless jQuery uses this shorthand frequently.
CSS selectors are combined with the $() notation to select DOM objects that match CSS attributes. Pass in the string of a CSS selector to $() and the result will be the set of DOM objects matching the selector. Basic selector syntax is used from CSS, as well as some additional ones defined within jQuery.

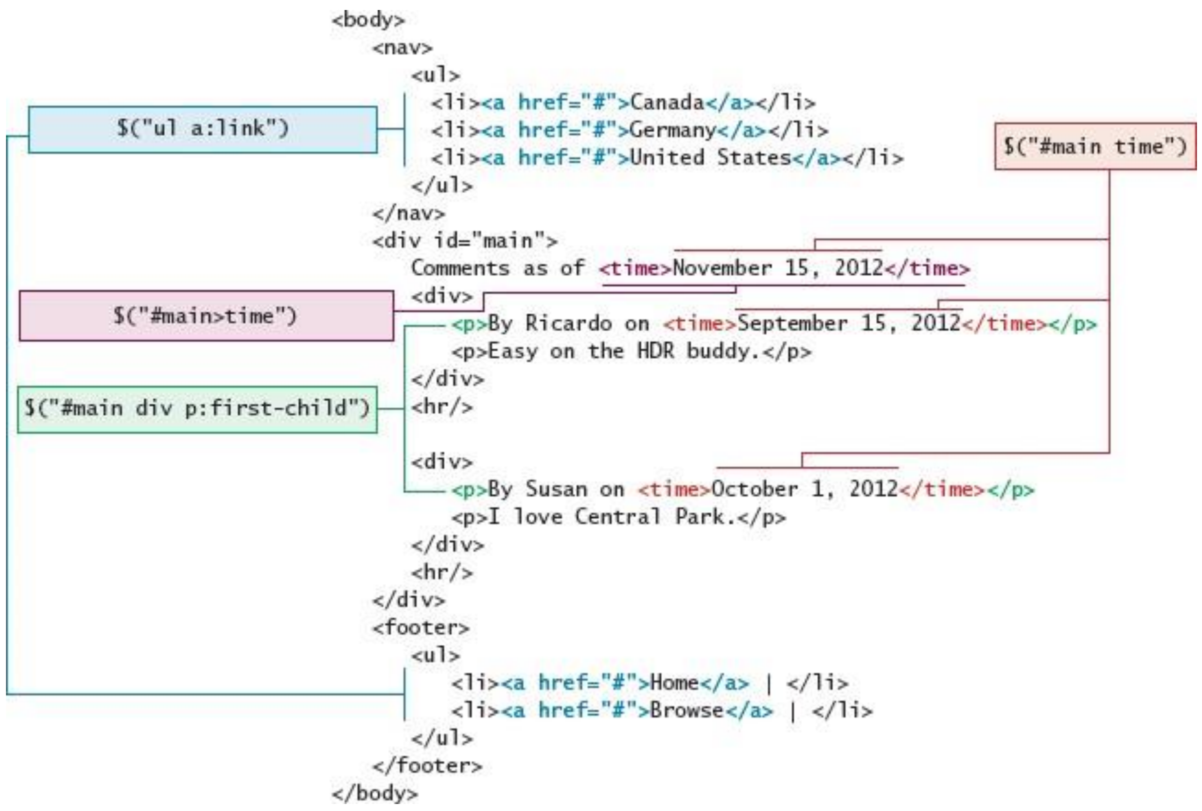
Basic Selectors

The four basic selectors include the universal selector, class selectors, id selectors, and elements selectors:

- $("*") **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.

To select the single <div> element with id="grab", write as
var singleElement = $("#grab");
To get a set of all the <a> elements the selector would be:
var allAs = $("a");

These selectors are powerful enough that they can replace the use of getElementById() entirely. Some of the selections can be written as,



```
<body>
   <nav>
      <ul>
        <li><a href="#">Canada</a></li>
        <li><a href="#">Germany</a></li>
        <li><a href="#">United States</a></li>
      </ul>
   </nav>
   <div id="main">
      Comments as of <time>November 15, 2012</time>
      <div>
         <p>By Ricardo on <time>September 15, 2012</time></p>
         <p>Easy on the HDR buddy.</p>
      </div>
      <hr/>

      <div>
         <p>By Susan on <time>October 1, 2012</time></p>
         <p>I love Central Park.</p>
      </div>
      <hr/>
   </div>
   <footer>
      <ul>
         <li><a href="#">Home</a> | </li>
         <li><a href="#">Browse</a> | </li>
      </ul>
   </footer>
</body>
```

Attribute Selector

An **attribute selector** provides a way to select elements by either the presence of an element attribute or by the value of an attribute.

A selector to grab all <img> elements with an src attribute beginning with /artist/ as:
var artistImages = $("img[src^='/artist/']");

You can select by attribute with square brackets ([attribute]), specify a value with an equals sign ([attribute=value]) and search for a particular value in the beginning, end, or anywhere inside a string with ^, $, and * symbols respectively ([attribute^=value], [attribute$=value], [attribute*=value]).

Pseudo-Element Selector

Pseudo-elements are special elements, which are special cases of regular ones. The yallow 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.
These selectors can be used in combination with any selectors or alone. Selecting all links that have been visited, for example, would be specified with:
var visitedLinks = $("a:visited");
Contextual Selector

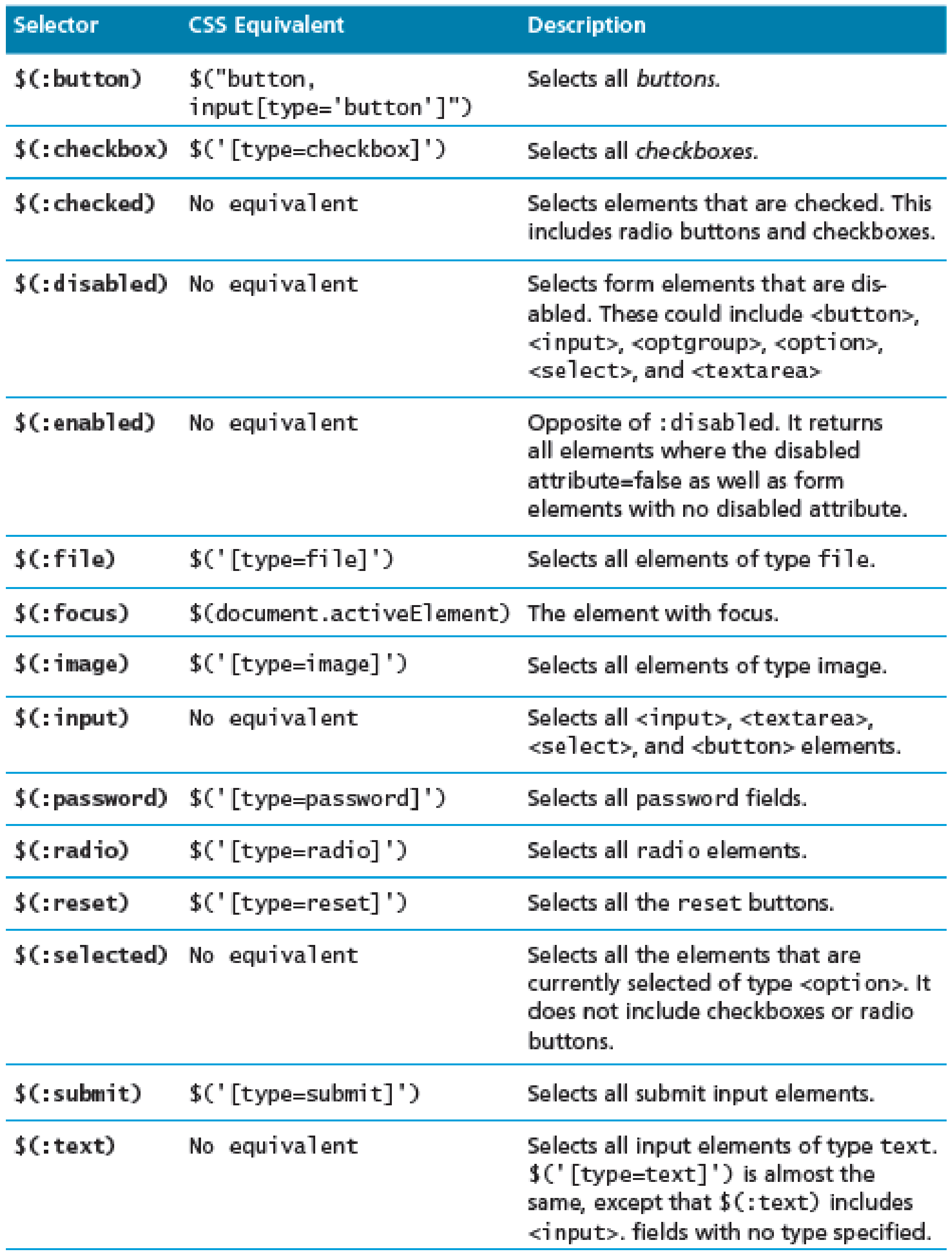
These selectors allowed you to specify elements with certain relationships to one another in your CSS. These relationships included descendant (space), child (>), adjacent sibling (+), and general sibling (~).

To select all <p> elements inside of <div> elements you would write
var para = $("div p");

Content Filters
The **content filter** is the only jQuery selector that allows to append filters to all of
the selectors and match a particular pattern. You can select elements that have a particular child using :has(), have no children using :empty, or match a particular piece of text with :contains().

Consider the following example:
var allWarningText = $("body *:contains('warning')");
It will return a list of all the DOM elements with the word *warning* inside of them.

Form Selectors
There are jQuery selectors written especially for form HTML elements. These selectors are well known and frequently used to collect and transmit data. Examples are listed in below table



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 dis-abled. These could include `<button>`, `<input>`, `<optgroup>`, `<option>`, `<select>`, and `<textarea>`
`$(:enabled)`	`No equivalent`	Opposite of `:disabled`. It returns all elements where the disabled attribute=false as well as form elements with no disabled attribute.
`$(: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
Any set of elements from a web page can be selected. In order to fully manipulate the elements, one must understand an element's *attributes* and *properties*.

HTML Attributes
The core set of attributes related to DOM elements are the ones specified in the HTML tags.
In jQuery we can both set and get an attribute value by using the attr() method on any element from a selector. This function takes a parameter to specify which attribute, and the optional

second parameter is the value to set it to. If no second parameter is passed, then the return value of the call is the current value of the attribute. Some example usages are:

```
// 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

Many HTML tags include properties as well as attributes, the most common being the *checked* property of a radio button or checkbox. In early versions of jQuery, HTML properties could be set using the attr() method. However, since properties are not technically attributes, this resulted in odd behavior. The prop() method is now the preferred way to retrieve and set the value of a property although, attr() may return some (less useful) values.
To illustrate this subtle difference, consider a DOM element defined by

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

The value of the attr() and prop() functions on that element differ as shown below.

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

Changing CSS

Changing a CSS style is syntactically very similar to changing attributes. jQuery provides the extremely intuitive css() methods. There are two versions of this method (with two different method signatures), one to get the value and another to set it. The first version takes a single parameter containing the CSS attribute whose value you want and returns the current value.

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

To modify a CSS attribute you use the second version of css(), which takes two parameters: the first being the CSS attribute, and the second the value.

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

If you want to use classes instead of overriding particular CSS attributes individually, have a look at the additional shortcut methods described in the jQuery documentation.

Shortcut Methods

jQuery allows the programmer to rely on foundational HTML attributes and properties exclusively as described above. However, as with selectors, there are additional functions that provide easier access to common operations such as changing an object’s class or the text within an HTML tag.
The html() method is used to get the HTML contents of an element (the part between the <> and </> tags associated with the innerHTML property in JavaScript).

If passed with a parameter, it updates the HTML of that element. The html() method should be used with caution since the inner HTML of a DOM element can itself contain nested HTML elements! When replacing DOM with text, you may inadvertently introduce DOM errors since no validation is done on the new content (the browser wouldn't want to presume).
You can enforce the DOM by manipulating textNode objects and adding them as children to an element in the DOM tree rather than use html(). While this enforces the DOM structure, it does complicate code. To illustrate, consider that you could replace the content of every <p> element with "jQuery is fun," with the one line of code:

```
$("p").html("jQuery is fun");
```

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. The toggleClass(className) method will add or remove the class className, depending on whether it is currently present in the list of classes. The val() method returns the value of the element. This is typically used to retrieve values from input and select fields.

jQuery Listeners
jQuery supports creation and management of listeners/handlers for JavaScript events. The usage of these events is conceptually the same as with JavaScript with some minor syntactic differences.

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. Namely, it ensured the entire page and all DOM elements are loaded before trying to attach listeners to them. With jQuery we do the same thing but use the $(document).ready() event as shown below.

```
$(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);
});
});
```

Listener Management
Setting up listeners for particular events is done in much the same way as JavaScript. While pure JavaScript uses the addEventListener() method, jQuery has on() and off() methods as well as shortcut methods to attach events. Modifying the code to use listeners rather than one handler yields the more modular code as shown below.

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

Modifying the DOM

jQuery comes with several useful methods to manipulate the DOM elements themselves.

Creating DOM and textNodes

If you decide to think about your page as a DOM object, then you will want to manipulate the tree structure rather than merely manipulate strings. jQuery is able to convert strings containing valid DOM syntax into DOM objects automatically.

The jQuery methods to manipulate the DOM take an HTML string, jQuery objects, or DOM objects as parameters, you might prefer to define your element as

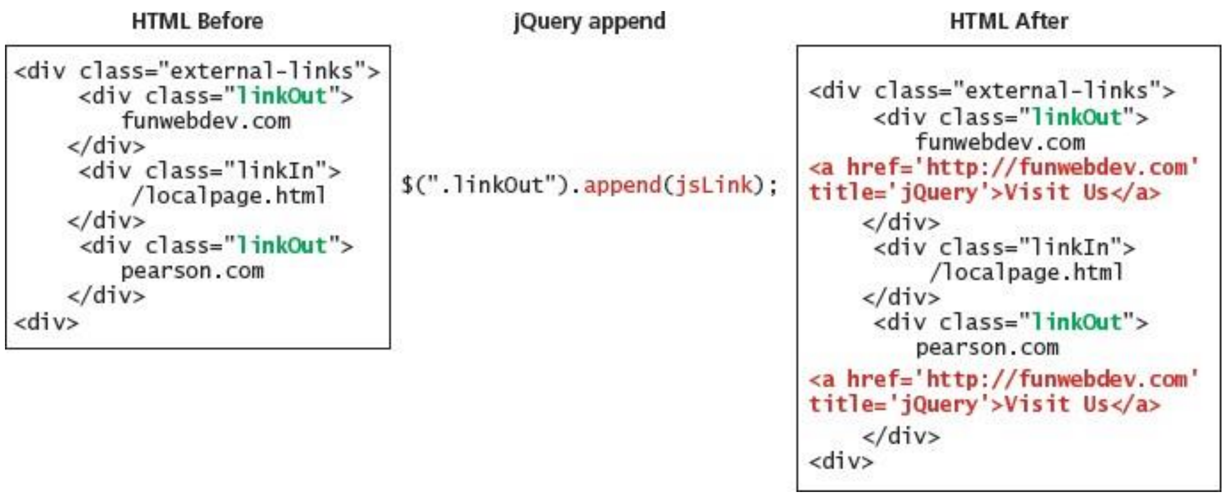
var element = $("<div></div>"); *//create new DOM node based on html*

This way you can apply all the jQuery functions to the object, rather than rely on pure JavaScript, which has fewer shortcuts. If we consider creation of a simple <a> element with multiple attributes, you can see the comparison of the JavaScript and jQuery techniques.

Prepending and Appending DOM Elements

When an element is defined, it must be inserted into the existing DOM tree. You can also insert the element into several places at once if you desire, since selectors can return an array of 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. In below figure we can see the effect of an append() method call. Each element with a class of linkOut has the jsLink element appended to it.



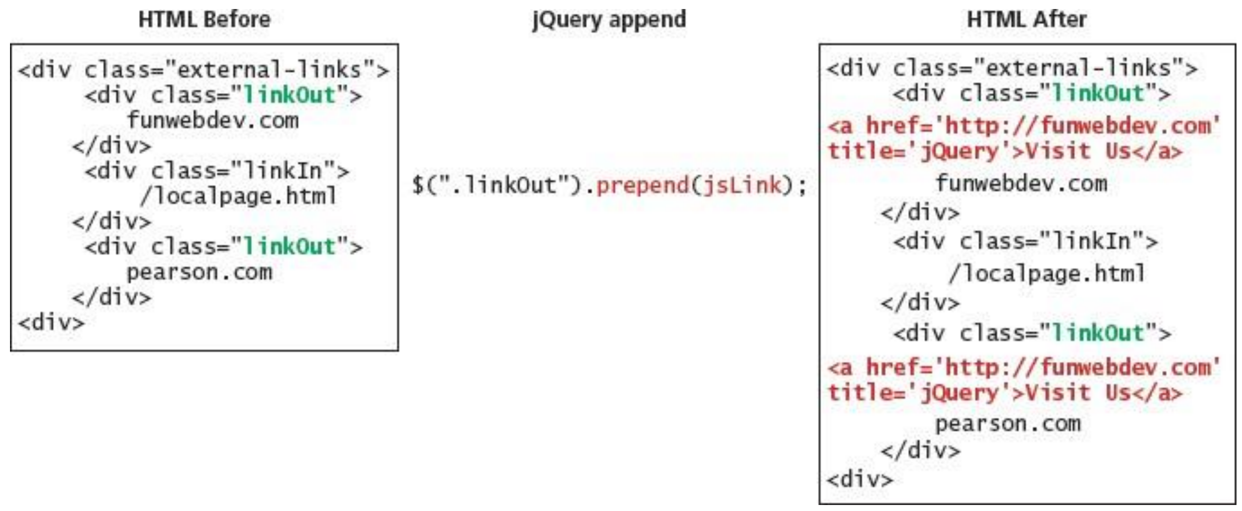
```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
    </div>
<div>
```

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

```
<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>
```

The appendTo() method is similar to append() but is used in the syntactically converse way. If we were to use appendTo(), we would have to switch the object making the call and the parameter to have the same effect as the previous code:
jsLink.appendTo($(".linkOut"));

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.



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

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

```
<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>
```

Wrapping Existing DOM in New Tags

One of the most common ways to enhance a website that supports JavaScript is to add new HTML tags as needed to support some jQuery functions. Imagine for illustration purposes our art galleries being listed alongside some external links as described by the HTML below.

```
<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>
```

If we wanted to wrap all the gallery items in the whole page inside, another
<div> with class galleryLink we could write:
$(".gallery").wrap('<div class="galleryLink"/>');
which modifies the HTML to that shown below. Note how each and every link is wrapped in the correct opening and closing and uses the galleryLink class.

```
<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>
```

consider the situation where you wanted to add a title element to each <div> element that reflected the unique contents inside. To achieve this more sophisticated manipulation, you must pass a function as a parameter rather than a tag to the wrap() method, and that function will return a dynamically created <div> element as shown below.

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

The wrap() method is a callback function, which is called for each element in a set. Each element then becomes this for the duration of one of the wrap() function's executions, allowing the unique title attributes as shown.

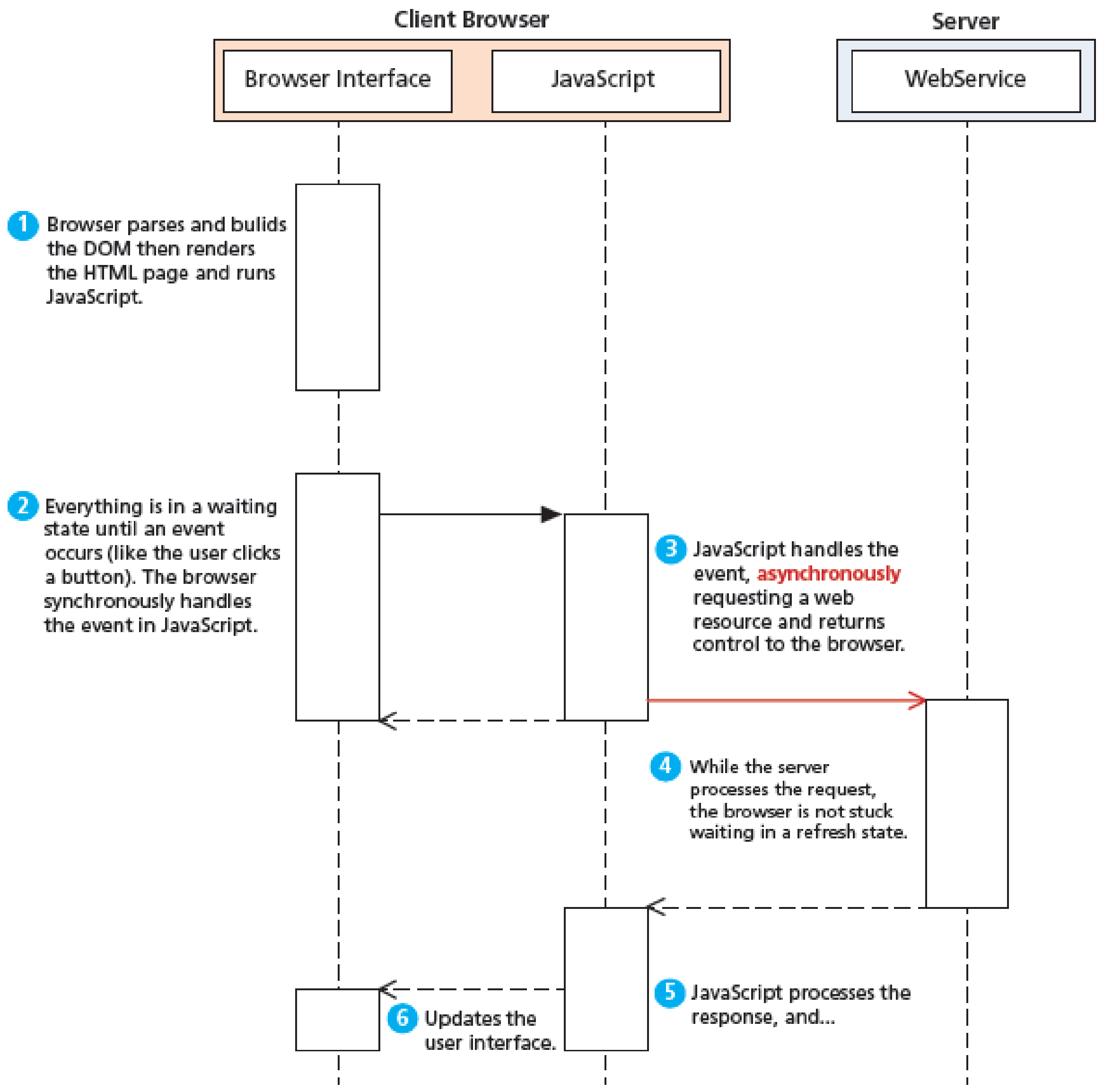
```
<div class="external-links">
<div class="galleryLink" title="Visit Uffuzi Museum">
<div class="gallery">Uffuzi Museum</div>
</div>
<div class="galleryLink" title="Visit National Gallery">
<div class="gallery">National Gallery</div>
</div>
<div class="link-out">funwebdev.com</div>
</div>
```

unwrap() is a method that does not take any parameters and whereas wrap() *added* a parent to the selected element(s), unwrap() *removes* the selected item's parent.

5.15 AJA X

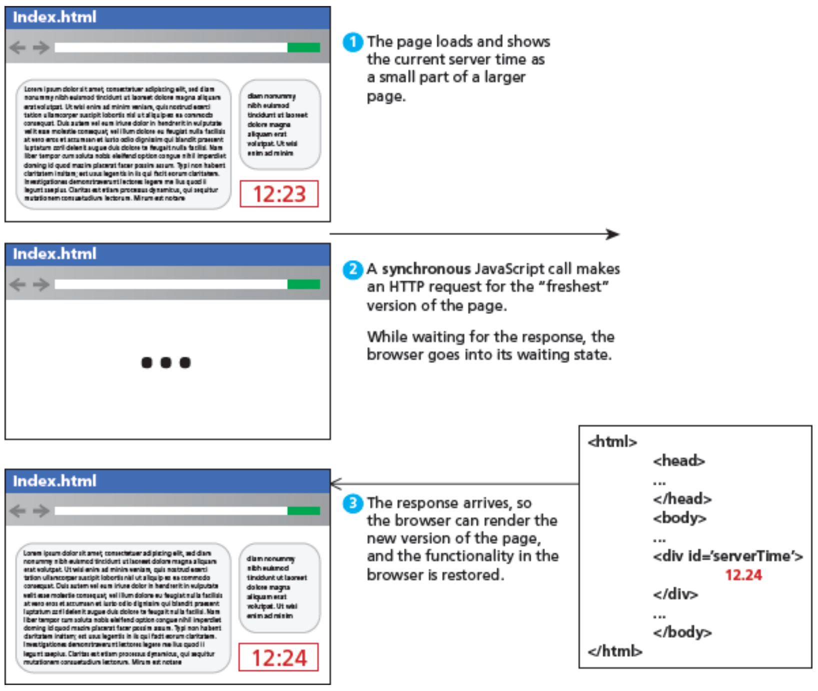
Asynchronous JavaScript with XML (AJAX) is a term used to describe a paradigm that allows a web browser to send messages back to the server without interrupting the flow of what's being shown in the browser. This makes use of a browser's multi-threaded design and lets one thread handle the browser and interactions while other threads wait for responses to asynchronous requests.

The below figure annotates a UML sequence diagram where the white activity bars illustrate where computation is taking place. Between the request being sent and the response being received, the system can continue to process other requests from the client, so it does not appear to be waiting in a loading state.


Client Browser
Server
Browser Interface
JavaScript
WebService
1 Browser parses and bulids the DOM then renders the HTML page and runs JavaScript.
2 Everything is in a waiting state until an event occurs (like the user clicks a button). The browser synchronously handles the event in JavaScript.
3 JavaScript handles the event, asynchronously requesting a web resource and returns control to the browser.
4 While the server processes the request, the browser is not stuck waiting in a refresh state.
5 JavaScript processes the response, and...
6 Updates the user interface.

Responses to asynchronous requests are caught in JavaScript as events. The events can subsequently trigger changes in the user interface or make additional requests. This differs from the typical synchronous requests we have seen thus far, which require the entire web page to refresh in response to a request.
Another way to contrast AJAX and synchronous JavaScript is to consider a web page that displays the current server time. If implemented synchronously, the entire page has to be refreshed from the server just to update the displayed time. During that refresh, the browser enters a waiting state, so the user experience is interrupted.

In contrast, consider the very simple asynchronous implementation of the server time, where an AJAX request updates the server time in the background as illustrated in figure b. In pure JavaScript it is possible to make asynchronous requests, but it’s tricky and it differs greatly between browsers.


Index.html
12:23
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.
12:24

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

Figure (a)

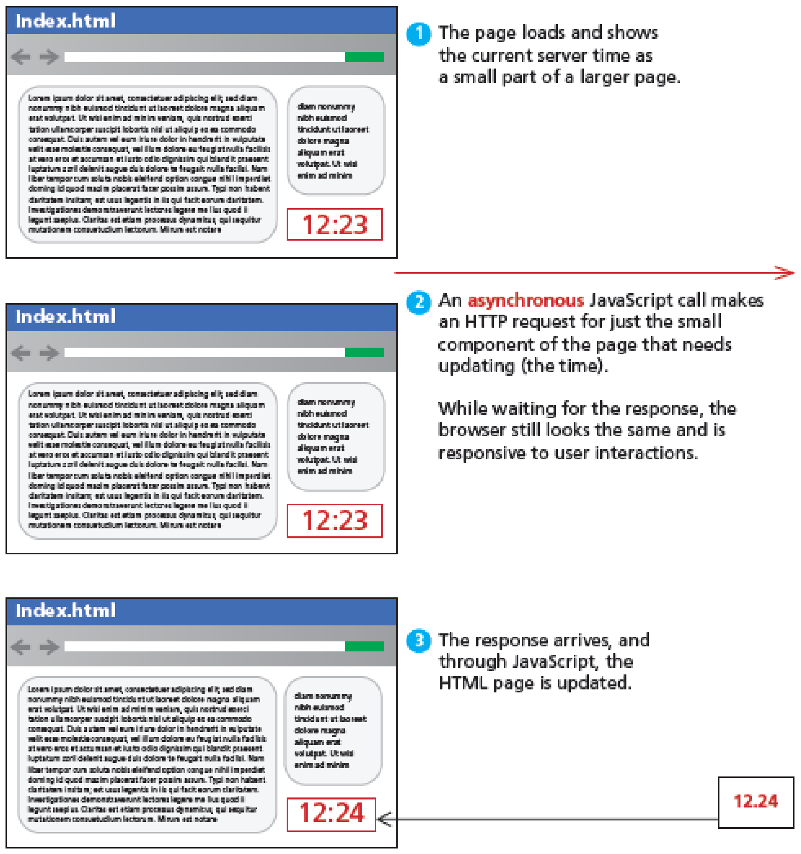

Index.html
1 The page loads and shows the current server time as a small part of a larger page.
12:23
2 An asynchronous JavaScript call makes an HTTP request for just the small component of the page that needs updating (the time).
While waiting for the response, the browser still looks the same and is responsive to user interactions.
3 The response arrives, and through JavaScript, the HTML page is updated.
12:24
12.24

Figure (b)

Get Request via jQuery AJAX

AJAX get request can be done as followes:
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.

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

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

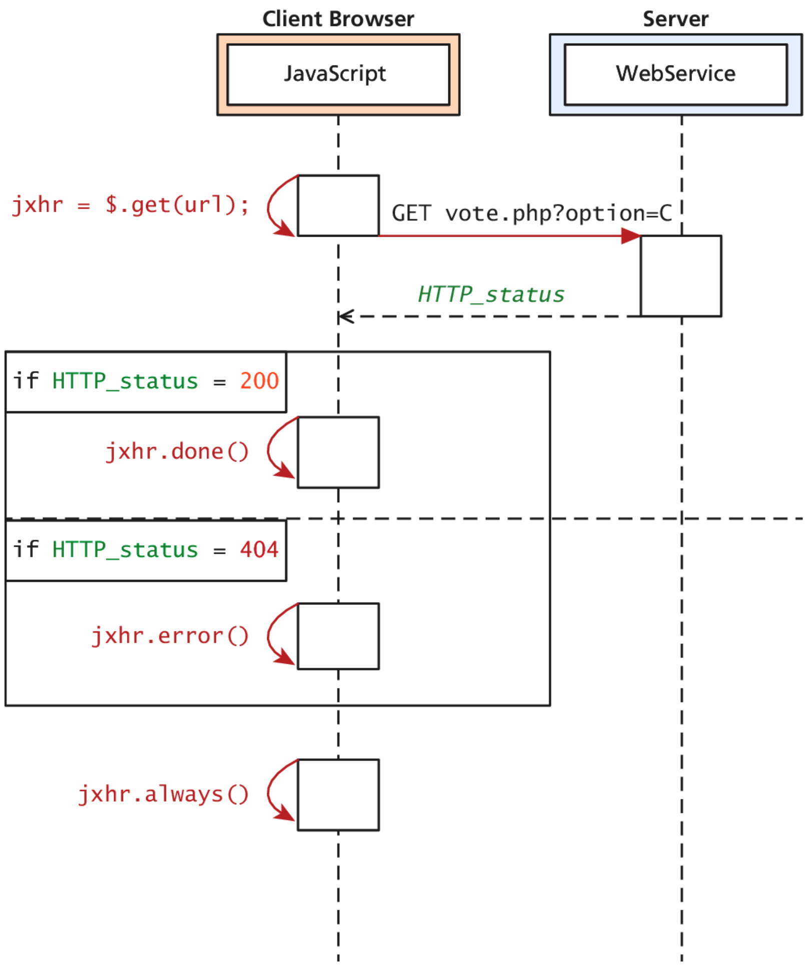
jqXHR - XMLHttpRequest compatibility:

- **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.

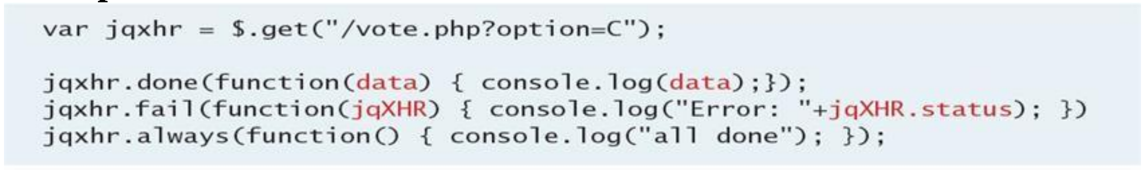
jqXHR objects have methods

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

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



Example:



```
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 Request 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.

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])

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");

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);

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.

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
});
```

LISTING 15.15 A raw AJAX method code to make a post

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

Cross-origin resource sharing (CORS)

- 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: ***
- 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 *.

5.16 Asynchronous File Transmission

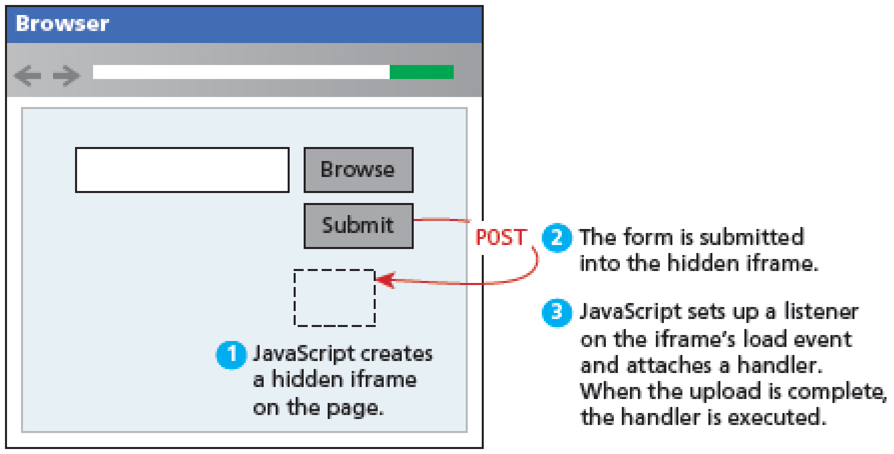
Asynchronous file transmission is one of the most powerful tools for modern web applications. In the days of old, transmitting a large file could require your user to wait idly by while the file uploaded, unable to do anything within the web interface. Since file upload speeds are almost always slower than download speeds, these transmissions can take minutes or even hours, destroying the feeling of a “real” application. Unfortunately jQuery alone does not permit asynchronous file uploads! However, using clever tricks and HTML5 additions, you too can use asynchronous file uploads.

For the following examples consider a simple file-uploading HTML form 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>
```

Old iframe Workarounds

The original workaround to allow the asynchronous posting of files was to use a hidden <iframe> element to receive the posted files. Given that jQuery still does not natively support the asynchronous uploading of files, this technique persists to this day and may be found in older code you have to maintain. As illustrated in the figure and the snippet below a hidden <iframe> allows one to post synchronously to another URL in another window. If JavaScript is enabled, you can also hide the upload button and use the change event instead to trigger a file upload. You then use the <iframe> element’s onload event to trigger an action when it is done loading. When the window is done loading, the file has been received and we use the return message to update our interface much like we do with AJAX normally.


Browser
Browse
Submit
POST
1 JavaScript creates a hidden iframe on the page.
2 The form is submitted into the hidden iframe.
3 JavaScript sets up a listener on the iframe's load event and attaches a handler. When the upload is complete, the handler is executed.

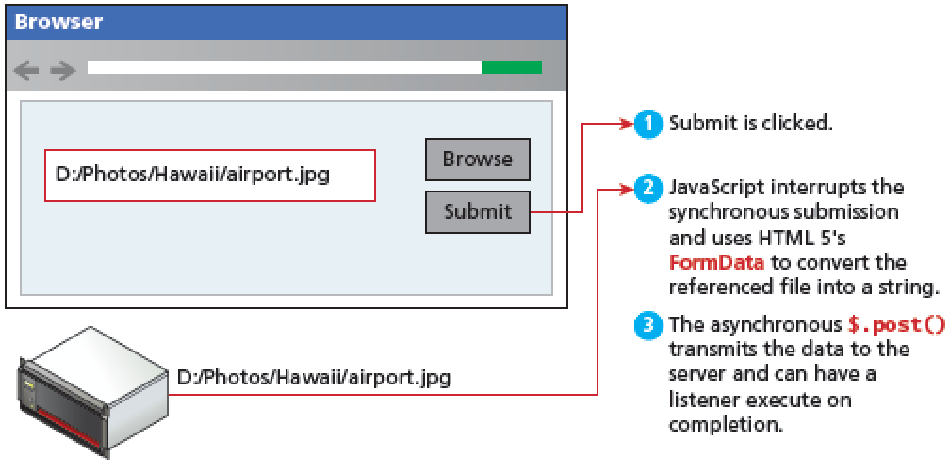
```
$(document).ready(function() {
// set up listener when the file changes
$(":file").on("change",uploadFile);
// hide the submit buttons
$("input[type=submit]").css("display","none");
});

// 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+"' />");
});
}
```

This technique exploits the fact that browsers treat each <iframe> element as a separate window with its own thread. By forcing the post to be handled in another window, we don't lose control of our user interface while the file is uploading.

Although it works, it's a workaround using the fact that every browser can post a file synchronously. A more modular and "pure" technique would be to somehow serialize the data in the file being uploaded with JavaScript and then post it in the body of a post request asynchronously.

The FormData Interface

The FormData interface provides a mechanism for JavaScript to read a file from the user's computer and encode it for upload. You can use this mechanism to upload a file asynchronously. Intuitively the browser is already able to do this, since it can access file data for transmission in synchronous posts. The FormData interface simply exposes this functionality to the developer, so you can turn a file into a string when you need to.


Browser
D:/Photos/Hawaii/airport.jpg
Browse
Submit
D:/Photos/Hawaii/airport.jpg
1 Submit is clicked.
2 JavaScript interrupts the synchronous submission and uses HTML 5's FormData to convert the referenced file into a string.
3 The asynchronous $.post() transmits the data to the server and can have a listener execute on completion.

The <iframe> method uploadFile() can be replaced with the more elegant and straightforward code. In this pure AJAX technique the form object is passed to a FormData constructor, which is then used in the call to send() the XHR2 object. This code attaches listeners for various events that may occur.

```
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.");
}
}
```

Appending Files to a POST

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.

```
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);
```

To support uploading multiple files in our JavaScript code, we must loop through all the files rather than only hard-code the first one. The below snippet handles multiple files being selected and uploaded at once.

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

The main challenge of asynchronous file upload is that your implementation must consider the range of browsers being used by your users.

5.17 Animation

When animation features are used appropriately, they make web applications appear more professional and engaging.

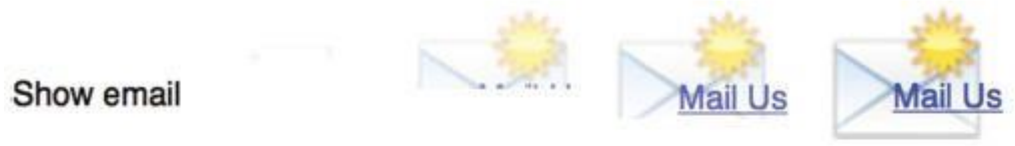
Animation Shortcuts

Animation is done using a raw animate() method and many more easy-to-use shortcuts like fadeIn()/fadeOut(), slideUp()/slideDown().

One of the common things done in a dynamic web page is to show and hide an element. Modifying the visibility of an element can be done using css(), but that causes an element to change instantaneously, which can be visually jarring. To provide a more natural transition from hiding to showing, the hide() and show() methods allow developers to easily hide elements gradually, rather than through an immediate change.

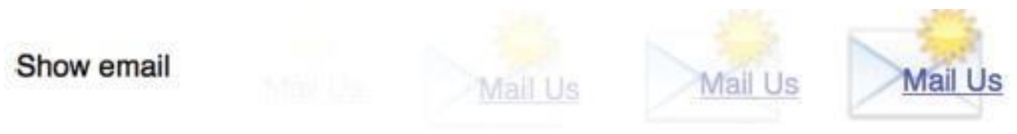
The hide() and show() methods can be called with no arguments to perform a default animation. Another version allows two parameters: the duration of the animation (in milliseconds) and a callback method to execute on completion. Using the callback is a great way to chain animations together, or just ensure elements are fully visible before changing their contents.

A visualization of the show() method is illustrated in Figure below. Note that both the size and opacity are changing during the animation. Although using the very straightforward hide() and show() methods works, you should be aware of some more advanced shortcuts that give you more control.



fadeIn()/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. The below figure shows a span during its animation using fadeIn(). It should be noted that there is another method, fadeTo(), that takes two parameters: a duration in milliseconds and the opacity to fade to (between 0 and 1).



slideDown()/slideUp()

The final shortcut methods we will talk about are slideUp() and slideDown(). These methods do not touch the opacity of an element, but rather gradually change its height. The below figure shows a slideDown() animation using an email icon.

Mail Us
Mail Us

Raw Animation
Just like $.get() and $.post() methods are shortcuts for the complete $.ajax() method, the animations shown this far are all specific versions of the generic animate() method. When you want to do animation that differs from the prepackaged animations, you will need to make use of animate.

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

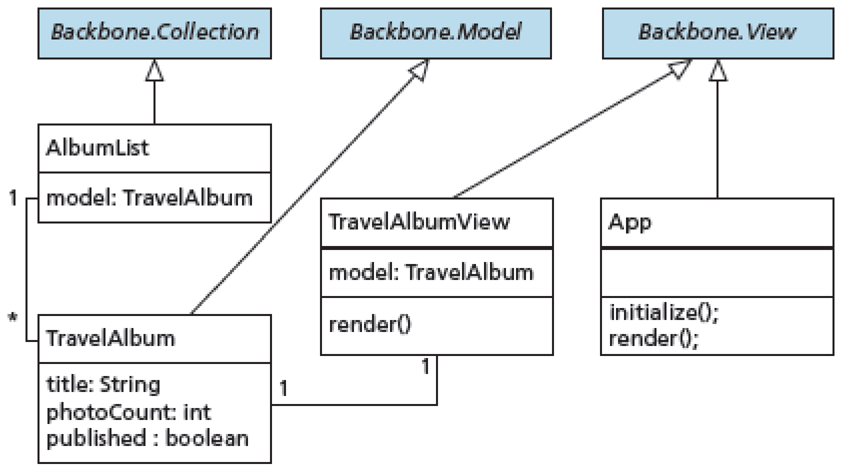
The properties parameter contains a Plain Object with all the CSS styles of the final state of the animation. The options parameter contains another Plain Object with any of the options below set.

■ always is the function to be called when the animation completes or stops with a fail condition.
■ 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.
■ 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. It takes two parameters: a now element, with the current numerical value of a CSS property, and an fx object, which is a temporary object with useful properties like the CSS attribute it represents (called *tween* in jQuery).
■ Advanced options called easing and specialEasing allow for advanced control over the speed of animation.

5.18 Backbone MVC Frameworks
MVC frameworks are overkill for small applications, where a small amount of jQuery is used. Backbone is an MVC framework that further abstracts JavaScript with libraries intended to adhere more closely to the MVC model. In Backbone, you build your client scripts around the concept of **models**. These models are often related to rows in the site's database and can be loaded, updated, and eventually saved back to the database using a REST interface.

Rather than writing the code to connect listeners and event handlers, Backbone allows user interface components to be notified of changes so they can update themselves just by setting everything up correctly. You must download the source for these libraries to your server, and reference them.

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



Backbone Models

The term models can be a challenging one to apply, since authors of several frameworks and software engineering patterns already use the term.

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.

Collections

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. In the snippet below is a collection of Albums, AlbumList, is defined by extending from Backbone’s Collection object. In addition an initial list of TravelAlbums, named albums, is instantiated to illustrate the creation of some model objects inside a Collection.

```
// 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}),
]);
```

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.

For our example we extend a View as shown below. In that code we attach our view to a particular tagName (in our case the <li> element) and then associate the click event with a new method named toggleAlbum().

```
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();
}
});
```

Always override the render() method since it defines the HTML that is output.