SCHEME AND SYLLABUS

WEB 2.0 AND RICH INTERNET APPLICATIONS

Subject Code: 06CS832 IA Marks : 25

No. of Lecture Hrs./ Week : 04 Exam Hours : 03

Total No. of Lecture Hrs. : 52 Exam Marks : 100

PART - A

UNIT - 1 7 Hours

INTRODUCTION, WEB SERVICES: What is Web 2.0?, Folksonomies and Web 2.0, Software as a Service (SaaS), Data and Web 2.0, Convergence, Iterative development, Rich User experience, Multiple Delivery Channels, Social Networking. Web Services: SOAP, RPC Style SOAP, Document style SOAP, WSDL, REST services, JSON format, What is JSON?, Array literals, Object literals, Mixing literals, JSON Syntax, JSON Encoding and Decoding, JSON versus XML.

UNIT - 2 7 Hours

BUILDING RICH INTERNET APPLICATIONS WITH AJAX-1:

Building Rich Internet Applications with AJAX: Limitations of Classic Web application model, AJAX principles, Technologies behind AJAX, Examples of usage of AJAX, Dynamic web applications through Hidden frames for both GET and POST methods.

UNIT – 3 6 Hours

BUILDING RICH INTERNET APPLICATIONS WITH AJAX-2:

Frames, Asynchronous communication and AJAX application model, XMLHTTP Object – properties and methods, handling different browser implementations of XMLHTTP, The same origin policy, Cache control, AJAX Patterns (Only algorithms – examples not required): Predictive fetch pattern, Submission throttling pattern, Periodic refresh, Multi stage download, Fall back patterns.

UNIT – 4 6 Hours

BUILDING RICH INTERNET APPLICATIONS WITH FLEX - 1: lash player, Flex framework, MXML and Actionscript, Working with Data services, Understanding

differences between HTML and Flex applications, Understanding how Flex applications work, Understanding Flex and Flash authoring, MXML language, a simple example.

PART - B

UNIT – 5 6 Hours

BUILDING RICH INTERNET APPLICATIONS WITH FLEX - 2:

Using Actionscript, MXML and Actionscript correlations. Understanding Actionscript 3.0 language syntax: Language overview, Objects and Classes, Packages and namespaces, Variables & scope of variables, case sensitivity and general syntax rules, Operators, Conditional, Looping, Functions, Nested functions, Functions as Objects, Function scope, OO Programming in Actionscript: Classes, Interfaces, Inheritance, Working with String objects, Working with Arrays, Error handling in Actionscript: Try/Catch, Working with XML

UNIT - 6 6 Hours

BUILDING RICH INTERNET APPLICATIONS WITH FLEX - 3:

Framework fundamentals, Understanding application life cycle, Differentiating between Flash player and Framework, Bootstrapping Flex applications, Loading one flex application in to another, Understanding application domains, Understanding the preloader. Managing layout, Flex layout overview, Working with children, Container types, Layout rules,Padding, Borders and gaps, Nesting containers, Making fluid interfaces.

UNIT - 7 6 Hours

BUILDING RICH INTERNET APPLICATIONS WITH FLEX – 4:

Working with UI components: Understanding UI Components, Creating component instances, Common UI Component properties, Handling events, Button, Value selectors, Text components, List based controls, Data models and Model View Controller, Creating collection objects, Setting the data provider, Using Data grids, Using Tree controls, Working with selected values and items, Pop up controls, Navigators, Control bars Working with data: Using data models, Using XML, Using Actionscript classes, Data Binding.

UNIT – 8 8 Hours

BUILDING ADVANCED WEB 2.0 APPLICATIONS: Definition of mash up applications, Mash up Techniques, Building a simple mash up application with AJAX, Remote data

communication, strategies for data communication, Simple HTTPServices, URLLoader in Flex, Web Services in Flex, Examples: Building an RSS reader with AJAX, Building an RSS reader with Flex.

TEXT BOOKS:

1. Professional AJAX – Nicholas C Zakas et al, Wrox publications, 2006.

2. Programming Flex 2 – Chafic Kazoun, O'Reilly publications, 2007.

3. Mashups – Francis Shanahan, Wrox, 2007.

REFERENCE BOOKS:

1. Ajax: The Complete Reference – Thomas A. Powel, McGraw Hill, 2008.

2. Unleashing Web 2.0: From Concepts to Creativity – Gottfried Vossen, Stephan Hagemann, Elsevier, 2007.

3. Essential Actionscript 3.0 – Colin Moock, O'Reilly Publications, 2007.

4. Ajax Bible - Steven Holzner, Wiley India, 2007.

5. A Web 2.0 Primer Pragmatic Ajax – Justin Gehtland et al, SPD Publications, 2006.

6. Professional Web 2.0 Programming – Eric Van derVlist et al, Wiley India, 2007.

TABLE OF CONTENT

UNIT - 1 7 Hours

INTRODUCTION, WEB SERVICES

- What is Web 2.0?
- Folksonomies and Web 2.0
- Software as a Service (SaaS)
- Data and Web 2.0
- Convergence
- Iterative development
- Rich User experience
- Multiple Delivery Channels
- Social Networking
- Web Services:
    - SOAP
    - RPC Style SOAP
    - Document style SOAP
    - WSDL
    - REST services
- JSON
    - What is JSON?
    - Array literals
    - Object literals
    - Mixing literals
    - JSON Syntax
    - JSON Encoding and Decoding
    - JSON versus XML

INTRODUCTION, WEB SERVICES

What is Web 2.0?

- The term **Web 2.0** is commonly associated with web applications that facilitate interactive systemic biases, interoperability, user-centered design, and developing the World Wide Web.
- A Web 2.0 site allows users to interact and collaborate with each other in a social media dialogue as consumers of user-generated content in a virtual community, in contrast to websites where users (prosumers) are limited to the active viewing of content that they created and controlled.
- Examples of Web 2.0 include social networking sites, blogs, wikis, video sharing sites, hosted services, web applications, mashups and folksonomies.

Folksonomies and Web 2.0

- A **folksonomy** is a system of classification derived from the practice and method of collaboratively creating and managing tags to annotate and categorize content; this practice is also known as **collaborative tagging**, **social classification**, **social indexing**, and **social tagging**.
- *Folksonomy*, a term coined by Thomas Vander Wal, is a portmanteau of *folks* and *taxonomy*.
- Folksonomies became popular on the Web around 2004 as part of social software applications such as social bookmarking and photograph annotation. Tagging, which is one of the defining characteristics of Web 2.0 services, allows users to collectively classify and find information. Some websites include tag clouds as a way to visualize tags in a folksonomy.

Software as a Service (SaaS)

- **Software as a service** (**SaaS**), sometimes referred to as "software on demand," is software that is deployed over the internet and/or is deployed to run behind a firewall on a local area network or personal computer. With SaaS, a provider licenses an application to customers either as a service on demand, through a subscription, in a "pay-as-you-go" model, or (increasingly) at no charge.
- This approach to application delivery is part of the utility computing model where all of the technology is in the "cloud" accessed over the Internet as a service.

Advantages

- Anytime, anywhere accessibility
- Pay as you go
- Instant scalability
- Security

- Reliability
- APIs

- SaaS was initially widely deployed for sales force automation and Customer Relationship Management (CRM). Now it has become commonplace for many business tasks, including accounting software, computerized billing, ERP software, invoicing, human resource management, financials, content management, collaboration, document management, and service desk management.

Data and Web 2.0

- The Mantra "Content is king", has been rewritten as "data is king ".
- Allowing the users to consume data makes it possible to define an entirely a new business and functionality other than those that were originally intended.
- Independent developers are now capable of delivering applications that would be impossible without a large team of resources.

Convergence

- Convergence can be thought of as a trend in which different hardware devices such as televisions, computers and telephones merge and have similar functions.
- At present, applications are diverging from the desktop and being accessed from various device.
- The next logical step would be a convergence whereby these various access channels become integrated.
- One of the scenarios would be: A personal media center is basically a TV hooked up to a computer.
- You can view and record TV without the use of tapes. You can view enhanced programming guide with links out to internet content. You can view RSS and news headlines on this PC viewing them as TV, and so on.

Iterative development

- Web 2.0 companies operate in very short cycle of design, develop, launch, and get feedback, repeat. Here time to market is reduced.
- How does it work?
- Companies purposefully leave features out to achieve shorter cycle times. Rather than guess at what the users want, it's better to launch a small subset of functionality and then take real-world users' feedback this feedback is then used to drive feature definition in subsequent cycles.
- This constant loop of development and product releases is commonly referred to as perpetual beta. It's constantly being iterated on and refined.

Web Services: SOAP

- **SOAP**, originally defined as **Simple Object Access Protocol**, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.
- It relies on Extensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols, most notably Remote Procedure Call (RPC) and Hypertext Transfer Protocol (HTTP), for message negotiation and transmission.
- SOAP can form the foundation layer of a web services protocol stack, providing a basic messaging framework upon which web services can be built.
- This XML based protocol consists of three parts: an envelope, which defines what is in the message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing procedure calls and responses.
- As an example of how SOAP procedures can be used, a SOAP message could be sent to a web-service-enabled web site, for example, a real-estate price database, with the parameters needed for a search.
- The site would then return an XML-formatted document with the resulting data, e.g., prices, location, features. Because the data is returned in a standardized machine-parseable format, it could then be integrated directly into a third-party web site or application.
- The SOAP architecture consists of several layers of specifications: for message format, Message Exchange Patterns (MEP), underlying transport protocol bindings, message processing models, and protocol extensibility.
- SOAP is the successor of XML-RPC, though it borrows its transport and interaction neutrality and the envelope/header/body from elsewhere (probably from WDDX).

RPC Style SOAP and Document style SOAP

- There are two ways to structure a SOAP message. In the early versions of SOAP (before it was publicly published), SOAP was designed to support only RPC style. By the time the SOAP 1.0 spec was published, it was expanded to support both RPCs and unstructured messages (document).
- When using Document style, you can structure the contents of the SOAP Body any way you like.
- When using RPC style, the contents of the SOAP Body must conform to a structure that indicates the method name and contains a set of parameters. It looks like this:

```
<env:Body>
<m:&methodName xmlns:m="someURI">
<m:&param1>...</m:&param1>
<m:&param2>...</m:&param2>
...
</m:&methodName>
</env:Body>
```

- The response message has a similar structure containing the return value and any output parameters. Your parameters can be as complex as you like, so there's really not a huge amount of difference in what you can pass. For example, you can pass a purchase order as a document or as a parameter in a method called placeOrder. It's your choice:

Document style:

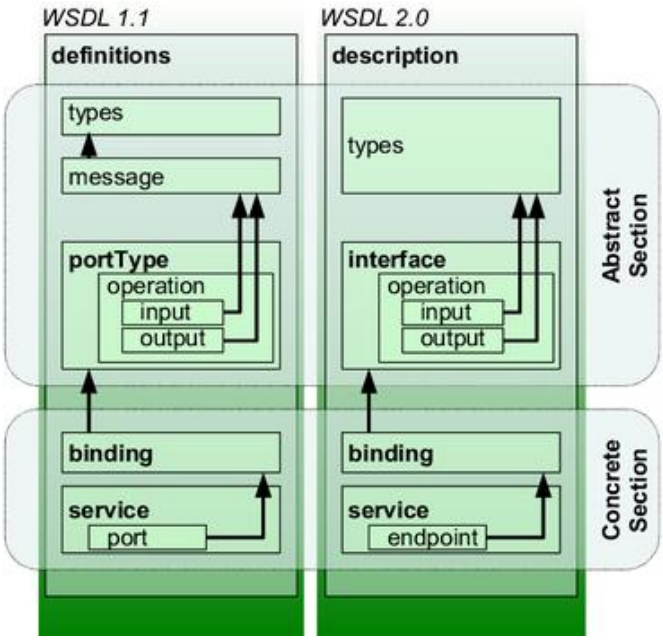
```
 <env:Body>
  <m:purchaseOrder xmlns:m="someURI">
   ...
</m:purchaseOrder>
</env:Body>
```

RPC style:

```
<env:Body>
<m:placeOrder xmlns:m="someURI">
<m:purchaseOrder>
...
</m:purchaseOrder>
</m:placeOrder>
 </env:Body>
```

WSDL

- The **Web Services Description Language** (WSDL, pronounced 'wiz-del') is an XML-based language that provides a model for describing Web services. The meaning of the acronym has changed from version 1.1 where the **D** stood for **Definition**.
- The current version of the specification is 2.0; version 1.1 has not been endorsed by the W3C but version 2.0 is a W3C recommendation. WSDL 1.2 was renamed WSDL 2.0 because of its substantial differences from WSDL 1.1.
- By accepting binding to all the HTTP request methods (not only GET and POST as in version 1.1), the WSDL 2.0 specification offers better support for RESTful web services, and is much simpler to implement.
- However support for this specification is still poor in software development kits for Web Services which often offer tools only for WSDL 1.1.[*dubious – discuss*] Furthermore, the latest version (version 2.0) of the Business Process Execution Language (BPEL) only supports WSDL 1.1.



Representation of concepts defined by WSDL 1.1 and WSDL 2.0 documents.

- The WSDL defines services as collections of network endpoints, or ports. The WSDL specification provides an XML format for documents for this purpose. The abstract definitions of ports and messages are separated from their concrete use or instance, allowing the reuse of these definitions.
- A port is defined by associating a network address with a reusable binding, and a collection of ports defines a service. Messages are abstract descriptions of the data being exchanged, and port types are abstract collections of supported operations.
- The concrete protocol and data format specifications for a particular port type constitutes a reusable binding, where the operations and messages are then bound to a concrete network protocol and message format. In this way, WSDL describes the public interface to the web service.
- WSDL is often used in combination with SOAP and an XML Schema to provide web services over the Internet.
- A client program connecting to a web service can read the WSDL file to determine what operations are available on the server. Any special datatypes used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the operations listed in the WSDL file.

REST services

- **Representational State Transfer** (**REST**) is a style of software architecture for distributed hypermedia systems such as the World Wide Web.
- The term *Representational State Transfer* was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. Fielding is one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification versions 1.0 and 1.1.
- Conforming to the REST constraints is referred to as being 'RESTful'.
- REST-style architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources.
- A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource.
- At any particular time, a client can either be in transition between application states or "at rest". A client in a rest state is able to interact with its user, but creates no load and consumes no per-client storage on the set of servers or on the network.
- The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. The representation of each application state contains links that may be used next time the client chooses to initiate a new state transition.
- REST was initially described in the context of HTTP, but is not limited to that protocol. RESTful architectures can be based on other Application Layer protocols if they already provide a rich and uniform vocabulary for applications based on the transfer of meaningful representational state.
- RESTful applications maximize the use of the pre-existing, well-defined interface and other built-in capabilities provided by the chosen network protocol, and minimize the addition of new application-specific features on top of it.

HTTP examples

- HTTP, for example, has a very rich vocabulary in terms of verbs (or "methods"), URIs, Internet media types, request and response codes, etc.
- REST uses these existing features of the HTTP protocol, and thus allows existing layered proxy and gateway components to perform additional functions on the network such as HTTP caching and security enforcement.
- An abbreviated list of claimed REST Examples is available.

SOAP RPC contrast

- SOAP RPC over HTTP, on the other hand, encourages each application designer to define a new and arbitrary vocabulary of nouns and verbs (for example getUsers(), savePurchaseOrder(...)), usually overlaid onto the HTTP 'POST' verb.
- This disregards many of HTTP's existing capabilities such as authentication, caching and content type negotiation, and may leave the application designer re-inventing many of these features within the new vocabulary. Examples of doing so may include the addition of methods such as getNewUsersSince(Date date), savePurchaseOrder(string customerLogon, string password, ...).

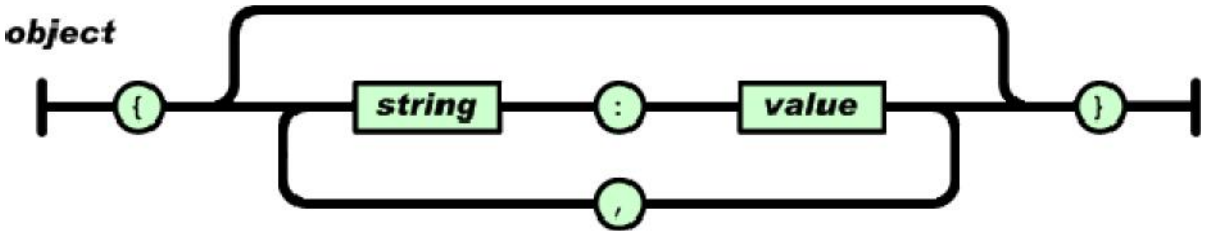
JSON format

- **JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.
- It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.
- JSON is built on two structures:
    - A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
    - An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

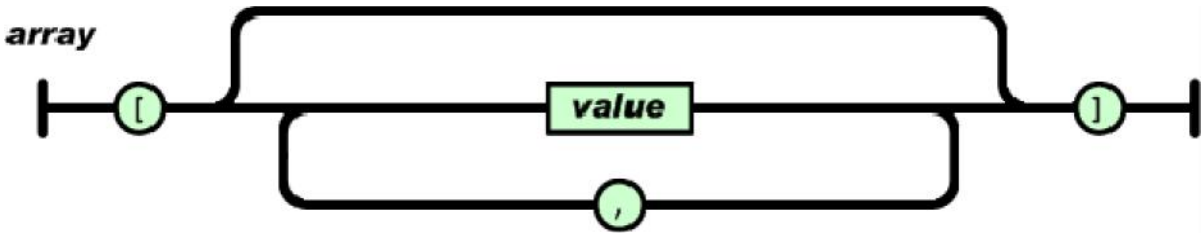
These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangable with programming languages also be based on these structures.

In JSON, they take on these forms:

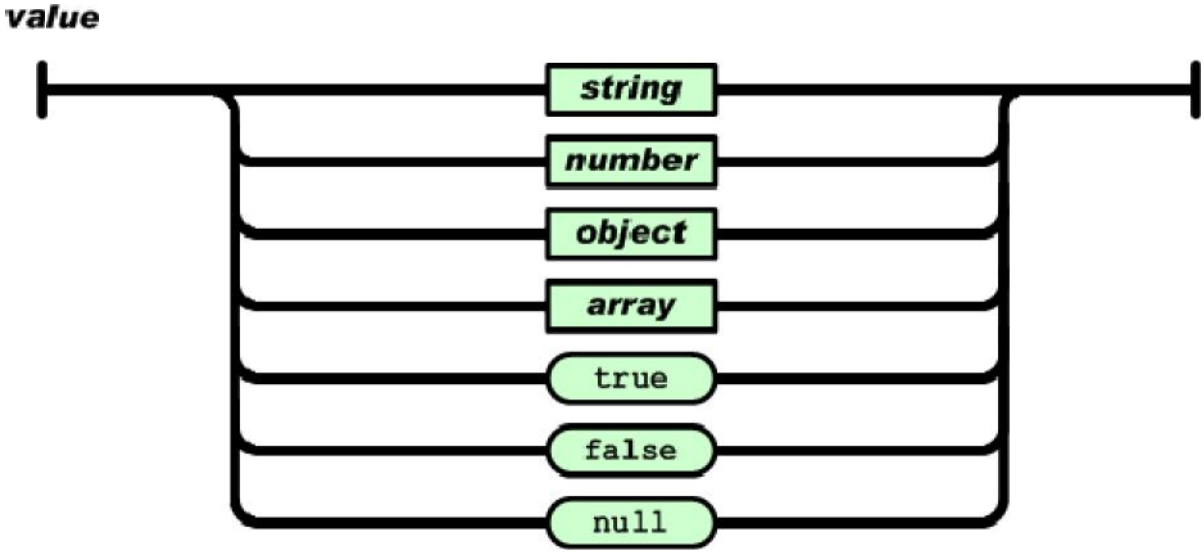
- An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).



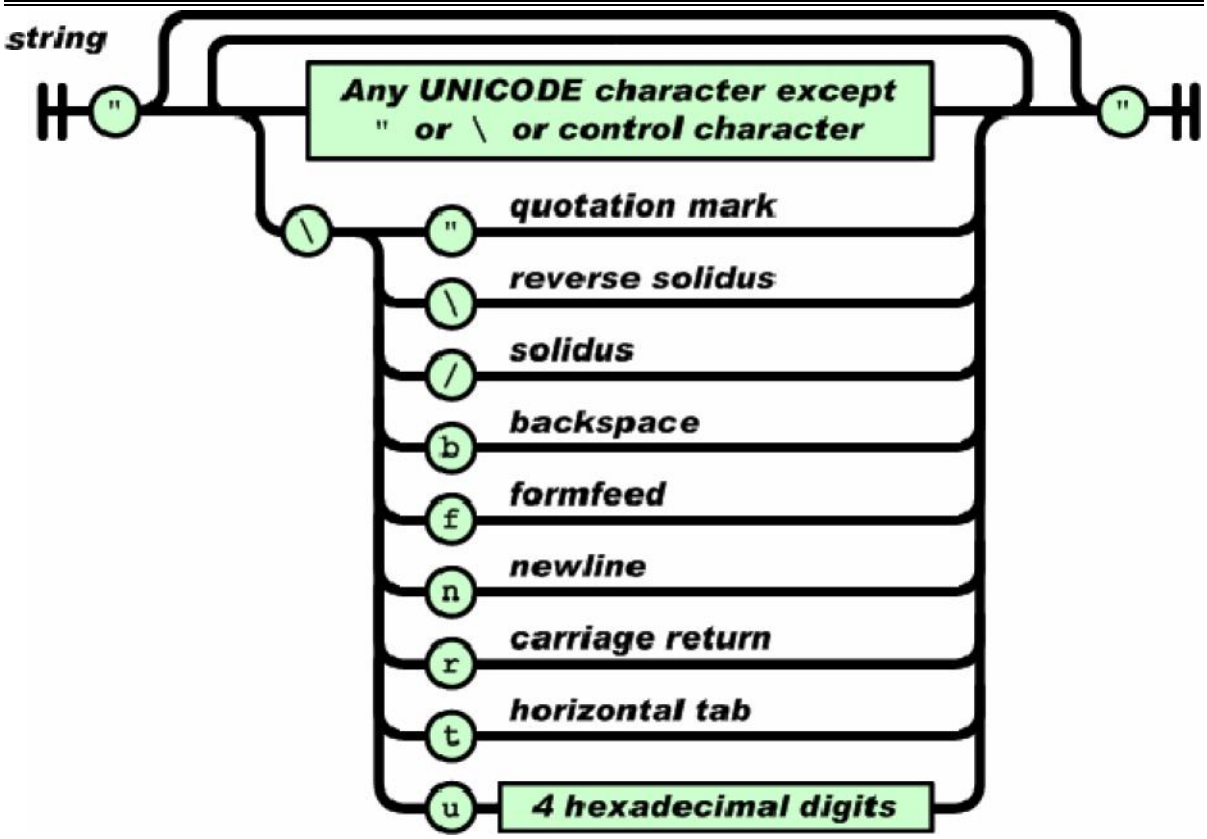
- An *array* is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).



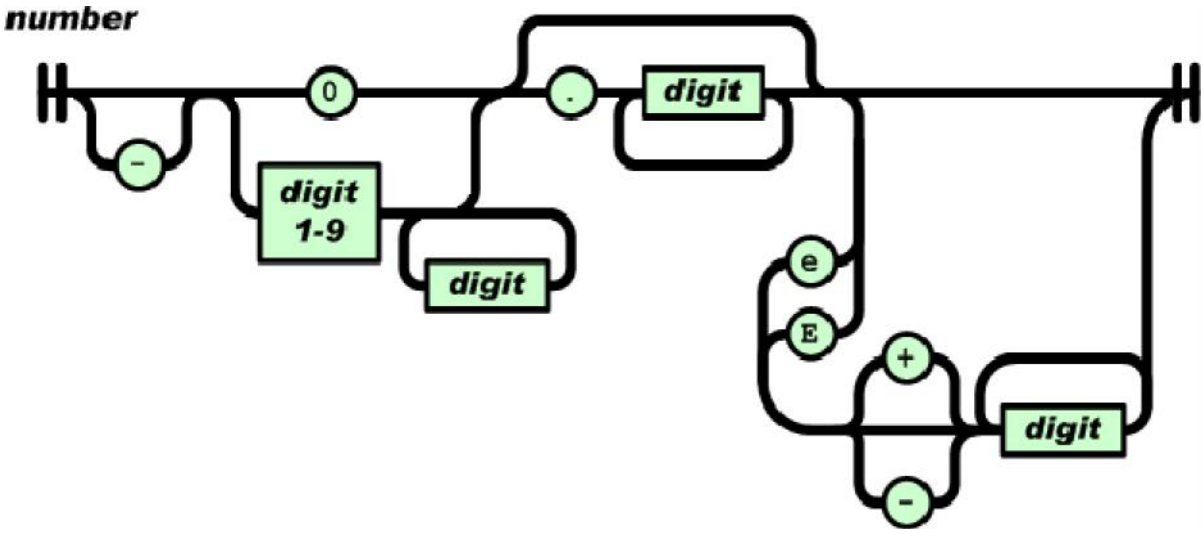
- A *value* can be a *string* in double quotes, or a *number*, or true or false or null, or an *object* or an *array*. These structures can be nested.



- A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.



- A *number* is very much like a C or Java number, except that the octal and hexadecimal formats are not used.



Whitespace can be inserted between any pair of tokens. Excepting a few encoding details, that completely describes the language.

What is JSON?

- **JSON** (an acronym for **JavaScript Object Notation**) is a lightweight text-based open standard designed for human-readable data interchange.
- It is derived from the JavaScript programming language for representing simple data structures and associative arrays, called objects. Despite its relationship to JavaScript, it is language-independent, with parsers available for virtually every programming language.
- The JSON format was originally specified by Douglas Crockford, and is described in RFC 4627. The official Internet media type for JSON is application/json. The JSON filename extension is .json.
- The JSON format is often used for serializing and transmitting structured data over a network connection. It is primarily used to transmit data between a server and web application, serving as an alternative to XML.

Array literals

- JSON is a lightweight format for exchanging data between the client and server. It is often used in Ajax applications because of its simplicity and because its format is based on JavaScript object literals.
- We will start this lesson by learning JavaScript's object-literal syntax and then we will see how we can use JSON in an Ajax application.

Object Literals

We saw earlier how to create new objects in JavaScript with the Object() constructor. We also saw how we could create our constructors for our own objects. In this lesson, we'll examine JavaScript's literal syntax for creating arrays and objects.

- Array literals are created with square brackets as shown below:

```
var Beatles = ["Paul","John","George","Ringo"];
```

- This is the equivalent of:

```
var Beatles = new Array("Paul","John","George","Ringo");
```

- Object literals are created with curly brackets:

```
var Beatles = {
 "Country" : "England",
 "YearFormed" : 1959,
 "Style" : "Rock'n'Roll"
}
```

This is the equivalent of:

```
var Beatles = new Object();
Beatles.Country = "England";
Beatles.YearFormed = 1959;
Beatles.Style = "Rock'n'Roll";
```

Just as with all objects in JavaScript, the properties can be references using dot notation or bracket notation.

```
alert(Beatles.Style); //Dot Notation
alert(Beatles["Style"]); //Bracket Notation
```

Mixing literals

- Arrays in Objects

  Object literals can contain array literals:

```
var Beatles = {
 "Country" : "England",
 "YearFormed" : 1959,
 "Style" : "Rock'n'Roll",
 "Members" : ["Paul","John","George","Ringo"]
}
```

- Objects in Arrays

  Array literals can contain object literals:

```
var Rockbands = [
 {
  "Name" : "Beatles",
  "Country" : "England",
  "YearFormed" : 1959,
  "Style" : "Rock'n'Roll",
  "Members" : ["Paul","John","George","Ringo"]
 },
 {
  "Name" : "Rolling Stones",
  "Country" : "England",
  "YearFormed" : 1962,
  "Style" : "Rock'n'Roll",
  "Members" : ["Mick","Keith","Charlie","Bill"]
 }
]
```

JSON Syntax

- The JSON syntax is like JavaScript's object literal syntax except that the objects cannot be assigned to a variable. JSON just represents the data itself. So, the Beatles object we saw earlier would be defined as follows:

```
{
 "Name" : "Beatles",
 "Country" : "England",
 "YearFormed" : 1959,
 "Style" : "Rock'n'Roll",
 "Members" : ["Paul","John","George","Ringo"]
}
```

JSON Encoding and Decoding

- Encoding

For encoding, the documentation's not all that bad. It will tell you to implement the default() method in a subclass (of json.JSONEncoder) which takes your obect as an argument, and returns a serializable object. By serializable, they just mean something in the form of one of the basic serializable types. So, say you have a class with a few attributes as follows:

```
class MyClass:
   def __init__ (my_int, my_list, my_dict):
      this.my_int = my_int
      this.my_list = my_list
      this.my_dict = my_dict
```

You could write a custom encode function by mapping all the class attributes you want to save as members of a dictionary. If there are helpful additional things you want to store as well, that's fine too. in this example, i use a string representation of a previously defined datetime object to make note of when the object was saved. Of course the only thing to remember is that when you later decode the object, you're going to be recreating a MyClass object from this data, and it will have to match (so, specifically, you'll either be discarding the date information or storing it elsewhere (or annotating your object with it on the fly)).

```
class MyEncoder(json.JSONEncoder):
    '' a custom JSON encoder for MyClass objects ''
    def default(self, my_class):
        if not isinstance (my_class, MyClass):
            print 'You cannot use the JSON custom MyClassEncoder for a non-MyClass object.'
            return
        return {'my_int': my_class.my_int, 'my_list': my_class.my_list,
                'my_dict': my_class.my_dict,
                'save date': the_date.ctime()}
```

- Decoding

Decoding is less clear than encoding. there are two ways you can customize the results returned by the json load() or loads() functions. One is by writing an object hook, and one is by subclassing JSONDecoder and overriding the decode() function.

When called, load/loads calls the decode() function on the json string or file pointer you pass to it. if object_hook is also specified, then the function passed to object_hook is called *after* the decode function is called.

the default behaviour of decode() is to return a python object FOR EVERY SIMPLE OBJECT in that string. this means that if you have a hierarchy of such objects, for example a dictionary which contains several lists, then although you only call load() once, *the decode() function gets called recursively for each python-like object in that string*. here's an example to convince yourself of this, using the previously encoded object:

```
fp = open ('myclass.json')
def custom_decode(json_thread):
   print json_thread
json.loads(fp, object_hook=custom_decode)
```

if what you want is to recover a custom object (such as the original MyClass object), this isnt terriby useful. at this point, it becomes clear we probably have to override the default loads() behaviour. as mentioned above, we do this by subclassing the JSONDecoder and overriding the decode() function. It's not clear why the lack of symmetry here with JSONEncode– we override default() in one, and decode() in the other. but, ok.

now, your custom decode function took a python object as argument, but the decode() function of course will receive the raw serialized string being decoded. the basic approach is to use the generic decode capability of the JSON module to parse the string that was stored on disk into a python dictionary object. but the decoder still doesnt know about your custom MyClass object, so what you do is actually create a new object, initializing it with the values in my_class_dict.

```
class ThreadDecoder(json.JSONDecoder):
   def decode (self, json_string):
      # use json's generic decode capability to parse the serialized string
      # into a python dictionary.
      my_class_dict = json.loads(json_string)
      return MyClass(my_class_dict['my_int'], my_class_dict['my_list'],
               my_class_dict['my_dict'])
```

And there you have it. This is a simple example, but objects and types can be nested arbitrarily; you just have to be willing to unravel them as appropriate such that you are encoding and decoding basic python types.

JSON versus XML

Why JSON?

- Essentially JSON is built for Data Structuring. When you take a look at it, JSON takes care of the Data Architecture of the Ajax based program. It has been widely accepted because it caters to the general need of Ajax.
- When you take a look at it, Ajax is a very broad program that needs to be harnessed. JSON is very easy to configure and it generally answers the need of every programmer.
- Compared to XML, JSON is also shorter to configure. In XML you could go as far as 7000 lines and could only go 75% of the data configuration. However, for JSON you can actually reduce it to a mere 1,500 lines.
- That's far easier and controllable. Debugging is also easy and as far as any developer is concerned, a thousand lines could even be manually checked.

Why XML?

- As we have noted, XML takes the longer route in developing the specific program. The reason for this is that XML is more concerned with the specifics of the program compared to JSON.
- Besides XML is a general purpose program so you have to configure it for JavaScript. The great thing about this program is that it gives you more freehand expression in your Application.
- Some see it as XML's disadvantage but the way I see it, XML's longer version of coding is a good thing since its full customization.
- Security is also better because surprisingly, XML is very simple. Because of its long lines it usually requires an expert developer.

UNIT - 2 7 Hours

BUILDING RICH INTERNET APPLICATIONS WITH AJAX-1

- Building Rich Internet Applications with AJAX
- Limitations of Classic Web application model
- AJAX principles
- Technologies behind AJAX
- Examples of usage of AJAX
- Dynamic web applications through Hidden frames for both GET and POST methods.

BUILDING RICH INTERNET APPLICATIONS WITH AJAX-1:

Building Rich Internet Applications with AJAX

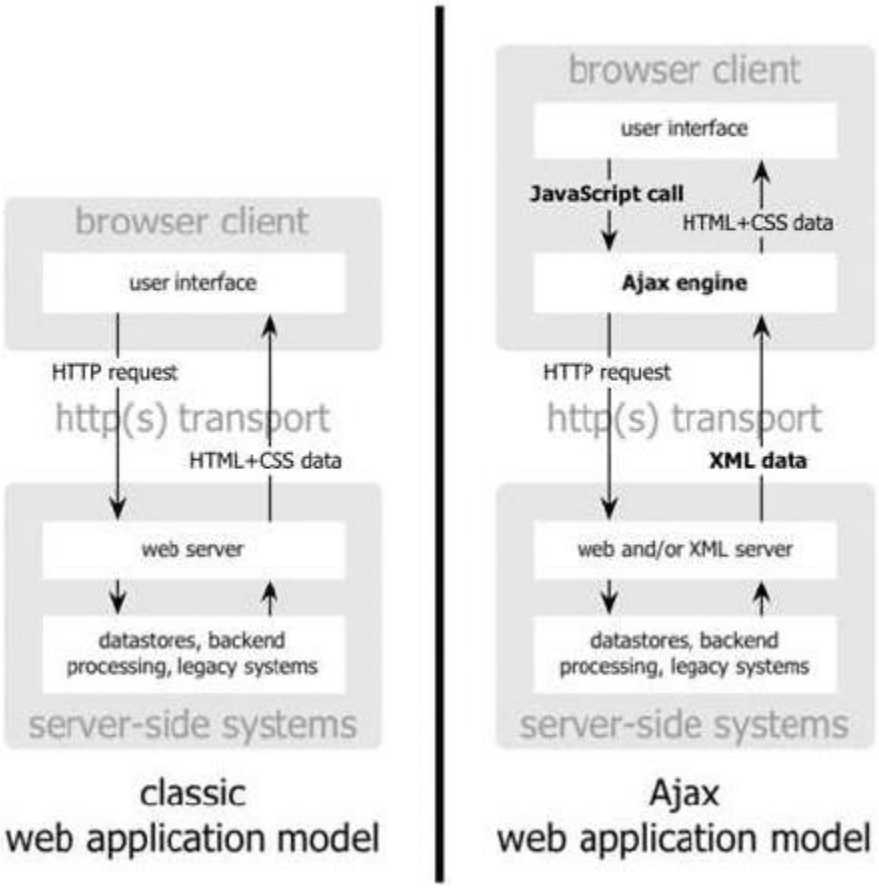
- **Ajax** (shorthand for Asynchronous JavaScript and XML) is a group of interrelated web development methods used on the client-side to create interactive web applications.
- With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page.
- Data is usually retrieved using the *XMLHttpRequest* object. Despite the name, the use of XML is not needed, and the requests need not be asynchronous.
- Like DHTML and LAMP, Ajax is not one technology, but a group of technologies. Ajax uses a combination of HTML and CSS to mark up and style information.
- The DOM is accessed with JavaScript to dynamically display, and to allow the user to interact with, the information presented. JavaScript and the XMLHttpRequest object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads.
- The term *Ajax* has come to represent a broad group of web technologies that can be used to implement a web application that communicates with a server in the background, without interfering with the current state of the page.
- In the article that coined the term Ajax, Jesse James Garrett explained that the following technologies are incorporated:

    - HTML or XHTML and CSS for presentation
    - the Document Object Model (DOM) for dynamic display of and interaction with data
    - XML for the interchange of data, and XSLT for its manipulation
    - the XMLHttpRequest object for asynchronous communication
    - JavaScript to bring these technologies together

- Since then, however, there have been a number of developments in the technologies used in an Ajax application, and the definition of the term Ajax.
- In particular, it has been noted that JavaScript is not the only client-side scripting language that can be used for implementing an Ajax application; other languages such as VBScript are also capable of the required functionality. (However, JavaScript is the most popular language for Ajax programming due to its inclusion in and compatibility with the majority of modern web browsers.)
- Also, XML is not required for data interchange and therefore XSLT is not required for the manipulation of data.
- JavaScript Object Notation (JSON) is often used as an alternative format for data interchange, although other formats such as preformatted HTML or plain text can also be used.

Limitations of Classic Web application model

- The classic web application model works like this: Most user actions in the interface trigger an HTTP request back to a web server.
- The server does some processing —retrieving data, crunching numbers, talking to various legacy systems —and then returns an HTML page to the client.
- It's a model adapted from the Web's original use as a hypertext medium, but as fans of The Elements of User Experience know, what makes the Web good for hypertext doesn't necessarily make it good for software applications.



- This approach makes a lot of technical sense, but it doesn't make for a great user experience. While the server is doing its thing, what's the user doing? That's right, waiting.
- And at every step in a task, the user waits some more. Obviously, if we were designing the Web from scratch for applications, we wouldn't make users wait around.

- Once an interface is loaded, why should the user interaction come to a halt every time the application needs something from the server? In fact, why should the user see the application go to the server at all?

AJAX principles

- As a new web application model, Ajax is still in its infancy. However, several web developers have taken this new development as a challenge.
- The challenge is to define what makes a good Ajax web application versus what makes a bad or mediocre one. Michael Mahemoff (http://mahemoff.com/), a software developer and usability expert, identified several key principles of good
- Ajax applications that are worth repeating:

    - **Minimal traffic:** Ajax applications should send and receive as little information as possible to and from the server. In short, Ajax can minimize the amount of traffic between the client and the server. Making sure that your Ajax application doesn't send and receive unnecessary information adds to its robustness.
    - **No surprises:** Ajax applications typically introduce different user interaction models than traditional web applications. As opposed to the web standard of click-and-wait, some Ajax applications use other user interface paradigms such as drag-and-drop or double-clicking. No matter what user interaction model you choose, be consistent so that the user knows what to do next.
    - **Established conventions:** Don't waste time inventing new user interaction models that your users will be unfamiliar with. Borrow heavily from traditional web applications and desktop applications so there is a minimal learning curve.
    - **No distractions:** Avoid unnecessary and distracting page elements such as looping animations, and blinking page sections. Such gimmicks distract the user from what he or she is trying to accomplish.
    - **Accessibility:** Consider who your primary and secondary users will be and how they most likely will access your Ajax application. Don't program yourself into a corner so that an unexpected new audience will be completely locked out. Will your users be using older browsers or special software? Make sure you know ahead of time and plan for it.
    - **Avoid entire page downloads:** All server communication after the initial page download should be managed by the Ajax engine. Don't ruin the user experience by downloading small amounts of data in one place, but reloading the entire page in others.
    - **User first:** Design the Ajax application with the users in mind before anything else. Try to make the common use cases easy to accomplish and don't be caught up with how you're going to fit in advertising or cool effects.

- The common thread in all these principles is usability. Ajax is, primarily, about enhancing the web experience for your users; the technology behind it is merely a means to that end.
- By adhering to the preceding principles, you can be reasonably assured that your Ajax application will be useful and usable.

Technologies behind AJAX

- JavaScript
    - Loosely typed scripting language
    - JavaScript function is called when an event in a page occurs
    - Glue for the whole AJAX operation
- DOM
    - API for accessing and manipulating structured documents
    - Represents the structure of XML and HTML documents
- CSS
    - Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript
- XMLHttpRequest
    - JavaScript object that performs asynchrous interaction with the server

Examples of usage of AJAX

```
<html>
<head>
<script type="text/javascript">
function loadXMLDoc()
{
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
```

```
 xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
 }
xmlhttp.onreadystatechange=function()
 {
 if (xmlhttp.readyState==4 && xmlhttp.status==200)
   {
   document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
   }
 }
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
}
</script>
</head>
<body>

<div id="myDiv"><h2>Let AJAX change this text</h2></div>
<button type="button" onclick="loadXMLDoc()">Change Content</button>

</body>
</html>
```

Dynamic web applications through Hidden frames for both GET and POST methods.

- HiddenFrameGET

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
   <title>Get Customer Data</title>
<?php

   //customer ID
   $sID = $_GET["id"];

   //variable to hold customer info
   $sInfo = "";
```

```
   //database information
   $sDBServer = "localhost";
   $sDBName = "ajax";
   $sDBUsername = "root";
   $sDBPassword = "";

   //create the SQL query string
   $sQuery = "Select * from Customers where CustomerId=".$sID;

   //make the database connection
   $oLink = mysql_connect($sDBServer,$sDBUsername,$sDBPassword);
   @mysql_select_db($sDBName) or $sInfo = "Unable to open database";

   if($sInfo == '') {
      if($oResult = mysql_query($sQuery) and mysql_num_rows($oResult) > 0) {
         $aValues = mysql_fetch_array($oResult,MYSQL_ASSOC);
         $sInfo = $aValues['Name']."<br />".$aValues['Address']."<br />".
               $aValues['City']."<br />".$aValues['State']."<br />".
               $aValues['Zip']."<br /><br />Phone: ".$aValues['Phone']."<br />".
               "<a href=\"mailto:".$aValues['E-mail']."\">".$aValues['E-mail']."</a>";
      } else {
         $sInfo = "Customer with ID $sID doesn't exist.";
      }
   }

   mysql_close($oLink);

?>

   <script type="text/javascript">
      window.onload = function () {
         var divInfoToReturn = document.getElementById("divInfoToReturn");
         top.frames["displayFrame"].displayCustomerInfo(divInfoToReturn.innerHTML);
      };

   </script>
```

```
</head>
<body>
   <div id="divInfoToReturn"><?php echo $sInfo ?></div>
</body>
</html>
```

- **HiddenFramePOST**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
   <title>Create New Customer</title>
<?php
   //get information
   $sName = $_POST["txtName"];
   $sAddress = $_POST["txtAddress"];
   $sCity = $_POST["txtCity"];
   $sState = $_POST["txtState"];
   $sZipCode = $_POST["txtZipCode"];
   $sPhone = $_POST["txtPhone"];
   $sEmail = $_POST["txtEmail"];

   //status message
   $sStatus = "";

   //database information
   $sDBServer = "localhost";
   $sDBName = "ajax";
   $sDBUsername = "root";
   $sDBPassword = "";

   //create the SQL query string
   $sSQL = "Insert into Customers(Name,Address,City,State,Zip,Phone,`E-mail`) ".
         " values ('$sName','$sAddress','$sCity','$sState', '$sZipCode'".
```

```
            ", '$sPhone', '$sEmail')";

    $oLink = mysql_connect($sDBServer,$sDBUsername,$sDBPassword);
    @mysql_select_db($sDBName) or $sStatus = "Unable to open database";

    if($sStatus == ''){
        if($oResult = mysql_query($sSQL)) {
            $sStatus = "Added customer; customer ID is ".mysql_insert_id();
         } else {
            $sStatus = "An error occurred while inserting; customer not saved.";
        }
    }

    mysql_close($oLink);
?>
    <script type="text/javascript">

        window.onload = function () {
            top.frames["displayFrame"].saveResult("<?php echo $sStatus ?>");
        }

    </script>
</head>
<body>
</body>
</html>
```

UNIT - 3 6 Hours

BUILDING RICH INTERNET APPLICATIONS WITH AJAX-2

- Frames
- Asynchronous communication and AJAX application model
- XMLHTTP Object – properties and methods
- Handling different browser implementations of XMLHTTP
- The same origin policy
- Cache control
- AJAX Patterns (Only algorithms – examples not required):
    - Predictive fetch pattern
    - Submission throttling pattern
    - Periodic refresh
    - Multi stage download
    - Fall back patterns.

Building Rich Internet Application with Ajax-2

Ajax Patterns

- A design pattern describes programming techniques to solve common problems.

Predictive Fetch Algorithm

- In case of traditional web applications, the application reacts only when there is an interaction.
- This is called “fetch on demand”. The user action tells the server what data should be retrieved.
- In the predictive fetch algorithm, the application guesses what the user is going to do next and retrieves the appropriate data.
- Determining the future action of the user is just or guess based on the users intentions.
- For eg: say a user is reading an online article of 3 pages. It can be assumed here that if the user is reading the 1st page for few seconds, the person will also be interested in reading the 2nd page.
- Hence the 2nd page can be downloaded at the background before the user explicitly clicks on the ‘Next’.
- Therefore when the user clicks on next the n2d page instantaneously appears reducing the response time.
- Similarly the 3rd page can be downloaded when the user reads 2nd page for a few seconds.
- This extra data being downloaded is cached on the client.
- Some approach can be applied in emails. If a person starts composing a mail, it is logical to anticipate that the mail would be sent to someone in the address book so this can be preloaded and kept.
- By using ajax to fetch information related to any possible next step, can overload the server.
- Therefore this algorithm has to be implemented only when it is logical to assume that information will be requisite to completing the user’s next request

Submission Throttling

- If retrieving data from the server is one part of the problem sending data to server is another.
- Seince in AJAX page refreshes needs to be avoided, it is important to know when to send user data to the server.
- One approach that could be taken is to send data to server on every user interaction. But this results in a lot of requests submitted to the server is a short period.

- In case of submission throttling, the data to be send to the server is buffered on the client.
- This data is then sent to the server at predefined times.
- The delay from typing to sending data is fine tuned such that it doent seem lika a delay to the user.
- Then a client side function is invoked that begins buffering the data.
- Then a client side function is invoked that begins buffering the data.
- It can be sent at a predefined time interval.
- This determination depends on the usecase being used.
- After the data is sent, the application continous to gather data.

Periodic Refresh

- This algorithm is basically used to nofify users of updated information.
- It describes the process of checking for new server information in specific intervals.
- This approach is called ‘polling’ and it requires the browser to keep track of when another request to the server should take place.

Multi-Stage Download

- In the modern web experience, a webpage is loaded with information, pictures, flash animations etc. Therefore to download a page all these elements need to be download leading to very slow speeds.
- Multistage download is an AJAX pattern wherein only the most basic functionality is loaded into a page initially.
- Upon completion of this, the page then begins to download other components that should appear on the page.
- If the user leaves the page befor all components all downloaded , then it is not much of consequence as all useful info was already displayed/
- If however the user stays on the page for some extemded period of time, the extra functionality is loaded at the background and available to the user.
- The developer decides what is downloaded at what point in time.
- This can be dealt with by providing a base and simple interface for the browser that don’t support AJAX and a richer interface for browsers that do.

Fall Back Patterns

- All the above methods work fine when there is no problem at the server side.
- The following problems can occur:

The request might never make it to the server.
An error might occur at the server.

Cancel Pending Requests

- If error occurs on the server like a file not fount or an internal server error , then it doesn't make sense to trying again after a few minutes.
- Such problems needs an administrator to fix it.
- In such a situation all the pending requests are simply cancelled.
- This can be done by setting a flag somewhere in the code that says "don't send any more request".
- This solution has maximum impact on the periodic refresh pattern.

Try Again

- Another option is to silently keep trying for either a specified amount of time or a particular number of files.
- This problem is handled behind the scenes without generally notifying the user.
- Unless the ajax functionality is key to the user's experience, it is not needed to notify her about the failures.

XMLHTTP Object

- The xmlhttp object was created to enable developers to initiate HTTP requests from anywhere in an application.
- These requests were intended to return an XML so the XMLHTTP object provided an easy way to access this info in the form of an XML document.
- This XMLHTTP Object was modified to suit various browsers and this version is known as the "XMLHTTP Request Object" or the XHR.

Creating an XHR object

- Since most of the Microsoft's implementation is on ActiveX control, the ActiveX object class should be used in the javascript.
  var oxhr=new ActiveXobject("Microsoft.xmlHttp");
- The signatures differ for various versions of the MSXML library.
- The various signatures are:
  - Microsoft.xmlHttp
  - MSXML2.xmlHttp
  - MSXML2.xmlHttp.3.0
  - MSXML2.xmlHttp.4.0
  - MSXML2.xmlHttp.5.0
  - MSXML2.xmlHttp .6.0

- The best way to find out which version is supported is to create all the probable ones. The ActiveX Control throws an error if it is not supported.

```
function createXHR() {
 var aVersions = [ "MSXML2.XMLHttp.6.0", "MSXML2.XMLHttp.3.0"];
 for (var i = 0; i < aVersions.length; i++)
   {
    try {
          var oXHR = new ActiveXObject(aVersions[i]);
           return oXHR;
           } catch (oError) {
     //Do nothing
           }
     }
 throw new Error("MSXML is not installed.");
   }
```

- In case of browsers apart from IE like Mozilla, Safari etc and also IE7 uses a simple code

```
var oxhr=new XMLHttpeRequest();
```

- In order to create a cross-browser way of creating XHR objects, both the methods are combined as shown:

```
function createXHR() {
      if (typeof XMLHttpRequest != "undefined") {
        return new XMLHttpRequest();
      } else if (window.ActiveXObject) {
        var aVersions = [ "MSXML2.XMLHttp.6.0", "MSXML2.XMLHttp.3.0"];
       for (var i = 0; i < aVersions.length; i++) {
          try {
                var oXHR = new ActiveXObject(aVersions[i]);
                return oXHR;
             } catch (oError) {
            //Do nothing
          }
       }
     }
throw new Error("XMLHttp object could not be created.");
 }
```

- Another way of creating corss-browser XHR Objects is to use the ZXML library which already has a cross-browser code written.
- Therefore a single function can be used to any browser.

```
var oxhr = zxmlhttp.createRequest();
```

Using XHR

- In order to send the http request from javascript, the first step is to call the open() method which initializes the object.
- The following are the 3 arguments for this method.

Request Type: A string indicating the request type to be made-typically,GET or POST
URL: A string indicating the URL to send the request to
Async:A Boolean value indicating whether the request should be made asynchronously

- The next step is to define an 'onreadyStateChange' event handler.
- The XHR object has a property called 'readyState' that changes as the request goes through and the response is received.
- Every time the readyState property changes from one value to another, the readyStateChange event fires and the onreadyStateChange event handler is called.

```
var oXHR = zXmlHttp.createRequest();
oXHR.open("get", "info.txt", true);
oXHR.onreadystatechange = function ()
{
    if (oXHR.readyState == 4)
   {
       alert("Got response.");
      }
};
oXHR.send(null);
```

- The send method is called at teh end which actually sends the request.
- If the request doesn't have a body, then a null should be passed in.
- In the above code, after response is received, an alert is dispalayed.
- This returns the context of info.txt. if there are any errors like the file didint exist and so on, it needs to be handled.

```
if (oXHR.status == 200) {

  alert("Data returned is: " + oXHR.responseText);
    } else {
  alert("An error occurred: " + oXHR.statusText);
    }
```

- The response Header can be accessed using the 'getResponseHeader()' method and passing the name of header to be retrieved.

```
var sContentType = oXHR.getResponseHeader("Content-Type");
 if (sContentType == "text/xml") {
 alert("XML content received.");
 } else if (sContentType == "text/plain") {
 alert("Plain text content received.");
```

```
} else {
alert("Unexpected content received.");
}
```

- The getAllResponseHeader() method can be used to return a string containing all the headers. Each header in this string is separated by '\n' or '\r\n'.

```
var sHeaders = oXHR.getAllResponseHeaders();

var aHeaders = sHeaders.split(/\r?\n/);
for (var i=0; i < aHeaders.length; i++) {
alert(aHeaders[i]);
}
```

- The 'setRequestHeader()' method can be set headon on the request before it is sent.

```
  var oXHR = zXmlHttp.createRequest();

oXHR.open("get", "info.txt", true);
oXHR.onreadystatechange = function ()
 {
   if (oXHR.readyState == 4)
      {
        alert("Got response.");
      }
};
oXHR.setRequestHeader("myheader", "myvalue");
oXHR.send(null);
```

Synchronous Requests

- In case of synchronous requests, it is not required to assign the onready statechange event handler.
- The response will have been received by the time the send() method returns.

```
 var oXHR = zXmlHttp.createRequest();
oXHR.open("get", "info.txt", false);
oXHR.send(null);
if (oXHR.status == 200) {
alert("Data returned is: " + oXHR.responseText);
} else
alert("An error occurred: " + oXHR.statusText);
    }
```

Cache Control

- Whenever repeated calls to the same page all dealt with browser caching needs to be considered.
- Web browsers tend to cache certain resources to improve the speed with which sites are downloaded and displayed.
- This increases the speed on frequently visited web sites.
- Caching can be dealt with by including a header with caching information on any data being sent from the server to the browser.

  Cache_Control : no_cache
  Expires : Fri, 30 Oct 1998 14:19:41 GMT
- This tells the browser not to cache the data coming from the specific URL. Instead the browser always calls a new version from the server instead of a saved version from its own cache.

UNIT - 4 6 Hours

BUILDING RICH INTERNET APPLICATIONS WITH FLEX - 1:

- Flash player
- Flex framework
- MXML and Actionscript
- Working with Data services
- Understanding differences between HTML and Flex applications
- Understanding how Flex applications work
- Understanding Flex and Flash authoring
- MXML language, a simple example.

BUILDING RICH INTERNET APPLICATIONS WITH FLEX - 1:

Flash player

- Flex is part of the Adobe Flash Platform, which is a set of technologies with Flash Player at the core.
- Flex applications are intended to be deployed to Flash Player, meaning ***Flash Player runs all Flex applications.***
- Flash Player is one of the most ubiquitous pieces of software anywhere. Because all the computers that have internet will be having some version of Flash Player installed and some of the mobile devices being Flash – enabled.

Flex framework

- The Flex framework is synonymous with the Flex class library and is a collection of ActionScript classes used by Flex applications.
- The Flex framework is written entirely in ActionScript classes, and defines controls, containers, and managers designed to simplify building rich Internet applications.
- The Flex class library.

MXML and Actionscript

- The Flex framework provides two programming languages: ActionScript and MXML. ActionScript 3.0 is an ECMA-compliant scripting language similar in syntax to JavaScript and Java. MXML is an XML-based declarative language similar to CFML.
- This is an example of an ActionScript function with variable declarations:

```
protected function addEmployee_clickHandler(event:MouseEvent):void { private var firstName:String; private var lastName:String; }
```

- This is an example of a Button UI control declared as an MXML tag:

```
<s:Button id="addEmployee" label="Add Employee" click="addEmployee_clickHandler(event)" />
```

- MXML tags are actually created with ActionScript under the hood. When you compile your Flex application, the MXML is converted into ActionScript which is then compiled into a SWF file.
- This means that your entire application could be written in ActionScript. However, you will primarily use MXML to define your Flex application UI and ActionScript to program your business logic.

Introducing ActionScript classes

- ColdFusion developers who are not familiar with object-oriented programming (OOP) will need to establish a foundation in OOP concepts.

Note: The free Adobe Flex in a Week training series covers an introduction to object-oriented programming that is specifically designed for developers to learn OOP in the context of Flex application development.

- ActionScript is an OOP language that encapsulates all of its functionality in classes. The Flex framework includes libraries of pre-built classes that provide data retrieval and handling, animation, UI elements and layout, and much more. Figure 1 shows some ActionScript 3.0 classes in a common OOP documentation format.

Working with Data services

- **Data synchronization** - Remove the complexity and error potential from the rich-client data synchronization process by using a robust, high-performance data synchronization engine between client and server.
- **Publish/subscribe messaging** - Publish and subscribe to message topics in real time with the same reliability, scalability, and overall quality of service as traditional thick client applications, enabling the creation of critical, more complex applications such as logistics handling, inventory control, stock trading, and more.
- **Data paging** - Easily manage and use large record sets using a built-in, efficient paging engine.
- **Data push** - Push data to thousands of concurrent users without polling, providing up-to-the-second views of critical data such as stock trader applications, live resource monitoring, shop floor automation, and more.
- **In-context collaboration** - Create applications that concurrently share in-context information with other users, enabling new application concepts such as "co-browsing," which allows users to share experiences and collaborate in real time with other users.

Understanding differences between HTML and Flex applications

- What works for HTML may or may not work for Flex.
- Both traditional and Flex applications are n-tiered.
- The traditional applications have, at a minimum, a data tier , a business tier and a presentation tier. Data tier – databases or similar resources.
- Business tier – The core business logic. Eg: accept request from client or presentation tier, query the data tier.Presentation tier – HTML, CSS, JavaScript, JSP or similar documents.
- A new tier called Client tier. Client tier – enables clients to offload computation from the server. It helps in network latency and making for responsive and highly interactive user interfaces. In case of Traditional web app client tier is state less and non responsive.

- The flex app client is stateful, which means that it can make changes to view without having to make a request to the server.

Understanding how Flex applications work

- Flex applications deployed on the web works differently than html applications.
- Every Flex app deployed on the web utilizes Flash Player as the deployment platform. Should understand Flash Player to understand Flex. All Flex app use the Flex Frame work at a minimum to compile the app.
- It is Imp to understand the relationship between the Source code files, the complier and Flash player All Flex App require MXML or Action script class files or both.

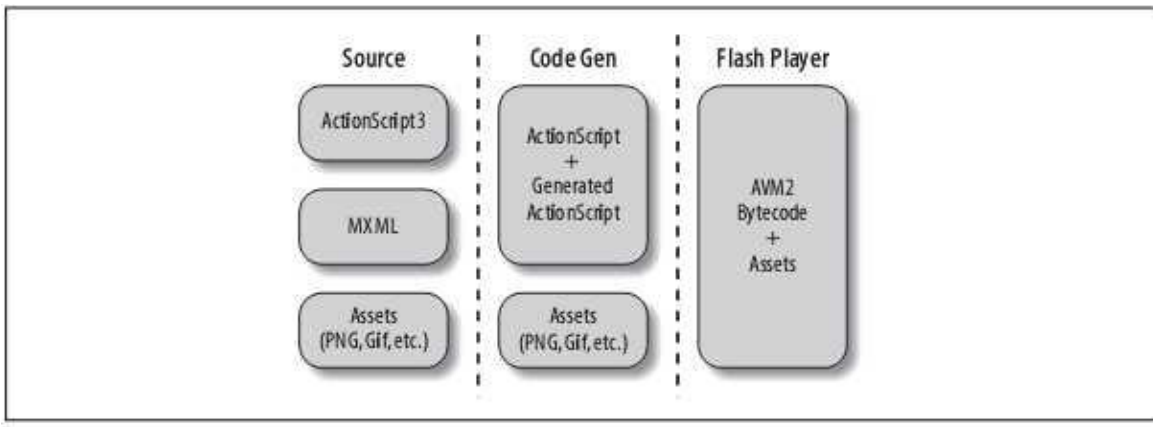


Figure 1-1. Understanding Flex application source-compile-deploy workflow

- Flash Player does not how to interpret MXML or uncompiled Action script class files.
- It is necessary to compile the source code files to the .swf format, which flash player can interpret. A typical Flex application compiles to just one .swf file. Deploy this .swf file to server and when a requested, it plays back in flash player. Assets can be embedded within a .swf file or they can be loaded at runtime.
    - Embedding makes less streamlined downloading experience and less dynamic app.
    - The assets are loaded into flash player when requested by the .swf at runtime.
    - In this case asset file must be deployed to a valid URL.

Understanding Flex and Flash authoring

- Flex Authoring is a traditional tool for creating content for Flash Player. Flash Authoring is a product that was first created in 1996 as a vector animation tool primarily aimed at creating animation content for the web.

- While flash authoring is a fantastic tool for creating animations, it is not the ideal tool for creating applications.The metaphors that Flash Authoring uses at its core are simply not applicable to app development.
- Flex 2 is a product aimed primarily at creating app.Both Flex and Flash authoring allow you to create .swf content that runs in Flash player.
- In theory we can achieve the same things using both products. Flash authoring allows to create timeline-based animations.Flex allows to much more rapidly assemble screens of content with transitions and data communication behaviors

MXML language, a simple example

- The structure of MXML is similar to XML and HTML.
- MXML is a Markup language used to create the user interface and to view portions of Flex Applications.
- It uses tags to create components such as user interface controls (buttons, menus, etc.), and to specify how those components interact with one another and with the rest of the application.
- All MXML must appear within MXML documents, which are plain text documents.We can use text editor to code MXML or Flex builder. It is stored as extension .mxml.
- Structure:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
<mx:Button label="Example Button"></mx:Button>
</mx:Application>
Or
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
<mx:Button label="Example Button" />

</mx:Application>
```

UNIT – 5 6hrs

BUILDING RICH INTERNET APPLICATIONS WITH FLEX - 2:

- Using Actionscript
- MXML and Actionscript correlations.
- Understanding Actionscript 3.0 language syntax:
    - Language overview
    - Objects and Classes
    - Packages and namespaces
    - Variables & scope of variables, case sensitivity and general syntax rules
    - Operators, Conditional, Looping
    - Functions, Nested functions, Functions as Objects, Function scope,
- OO Programming in Actionscript:
    - Classes, Interfaces, Inheritance
    - Working with String objects
    - Working with Arrays
- Error handling in Actionscript:
    - Try/Catch
    - Working with XML

Building Rich Internet Applications With Flex-2

Actionscript introduction

- **ActionScript** is an object-oriented language originally developed by Macromedia Inc.
- used primarily for the development of websites and software targeting the Adobe Flash Player platform, used on Web pages in the form of embedded SWF files.

ActionScript was initially designed for controlling simple 2D vector animations made in Adobe Flash (formerly Macromedia Flash). Initially focused on animation, early versions of Flash content offered few interactivity features and thus had very limited scripting capability. Later versions added functionality allowing for the creation of Web-based games and rich Internet applications with streaming media (such as video and audio). Today, ActionScript is suitable for use in some database applications, and in basic robotics, as with the Make Controller Kit.

MXML and Actionscript correlations

- MXML is a powerful way to simplify the creation of user interfaces. In most cases, it is far better to use MXML for layout than to attempt the same thing with ActionScript.
- ActionScript is far better suited for business logic and data models.
- When you use an MXML tag to create a component instance, it is the equivalent to calling the component class's constructor as part of a new statement. For example, the following MXML tag creates a new button:

```
<mx:Button id="button" />
```

That is equivalent to the following piece of ActionScript code:

```
var button:Button = new Button();
```

- If you assign property values using MXML tag attributes, that's equivalent to setting the object properties via ActionScript. For example, the following creates abuttonand sets thelabel:

```
<mx:Button id="button" label="Click" />
```

The following code is the ActionScript equivalent:

```
Var button:Button = new Button();
button.label = "Click";
```

For example, the following creates a new class that extendsmx.core.Applicationand creates one property calledButtonof typemx.controls.Button:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
  <mx:Button id="Button" />
/mx:Application>
```

The preceding example is essentially the same as the following ActionScript class:

```
package {
  import mx.core.Application;
  import mx.controls.Button;
  public class Example extends Application {
    internal var button:Button;
    public function Example() {
      super();
      button = new Button();
      addChild(button);
    }
  }
}
```

Actionscript 3.0 language syntax:

Language overview

- Objects lie at the heart of the ActionScript 3.0 language—they are its fundamental building blocks. Every variable you declare, every function you write, and every class instance you create is an object. You can think of an ActionScript 3.0 program as a group of objects that carry out tasks, respond to events, and communicate with one another.
- In ActionScript 3.0, objects are simply collections of properties.
- These properties are containers that can hold not only data, but also functions or other objects. If a function is attached to an object in this way, it is called a method.

Objects and classes

- In ActionScript 3.0, every object is defined by a class. A class can be thought of as a template or a blueprint for a type of object. Class definitions can include variables and constants, which hold data values, and methods, which are functions that encapsulate behavior bound to the class. The values stored in properties can be primitive values or other objects. Primitive values are numbers, strings, or Boolean values.
- ActionScript contains a number of built-in classes that are part of the core language. Some of these built-in classes, such as Number, Boolean and String, represent the primitive values available in ActionScript. Others, such as the Array, Math, and XML classes, define more complex objects.
- All classes, whether built-in or user-defined, derive from the Object class.

```
var someObj:Object;

var someObj;.
```

- You can define your own classes using the class keyword. You can declare class properties in three ways: constants can be defined with the const keyword, variables are defined with the var keyword, and getter and setter properties are defined by using the get and set attributes in a method declaration. You can declare methods with the functionkeyword.
- You create an instance of a class by using the new operator. The following example creates an instance of the Date class called myBirthday .

  ```
  var myBirthday:Date = new Date();
  ```

Packages and namespaces

- Packages in ActionScript 3.0 are implemented with namespaces, When you declare a package, you are implicitly creating a special type of namespace that is guaranteed to be known at compile time. Namespaces, when created explicitly, are not necessarily known at compile time.
- The following example uses the package directive to create a simple package containing one class:

```
package samples

{
   public class SampleCode
   {
      public var sampleGreeting:String;
```

```
        public function sampleFunction()
        {
            trace(sampleGreeting + " from sampleFunction()");
        }
    }
}
```

- The use of packages also helps to ensure that the identifier names that you use are unique and do not conflict with other identifier names.
- Importing a package : If the class resides in a package named samples, you must use one of the following import statements before using the SampleCode class:

```
import samples.*;

import samples.SampleCode;
```

- Fully qualified names are useful when identically named classes, methods, or properties result in ambiguous code, but can be difficult to manage if used for all identifiers. For example, the use of the fully qualified name results in verbose code when you instantiate a SampleCode class instance:

```
var mySample:samples.SampleCode = new samples.SampleCode();
```

- In situations where you are confident that ambiguous identifiers will not be a problem, you can make your code easier to read by using simple identifiers.

```
var mySample:SampleCode = new SampleCode();
```

- When a package is created, the default access specifier for all members of that package is internal , which means that, by default, package members are only visible to other members of that package. If you want a class to be available to code outside the package, you must declare that class to be public . For example, the following package contains two classes, SampleCode and CodeFormatter:

```
// SampleCode.as file
package samples
{
    public class SampleCode {}
}


// CodeFormatter.as file
```

```
package samples
{
   class CodeFormatter {}
}
```

The SampleCode class is visible outside the package because it is declared as a public class. The CodeFormatter class, however, is visible only within the samples package itself. If you attempt to access the CodeFormatter class outside the samples package, you will generate an error, as the following example shows:

```
import samples.SampleCode;
import samples.CodeFormatter;
var mySample:SampleCode = new SampleCode(); // okay, public class
var myFormatter:CodeFormatter = new CodeFormatter(); // error
```

- If you want both classes to be available outside the package, you must declare both classes to be public . You cannot apply the public attribute to the package declaration.
- Fully qualified names are useful for resolving name conflicts that may occur when using packages. Such a scenario may arise if you import two packages that define classes with the same identifier. For example, consider the following package, which also has a class named SampleCode:

```
package langref.samples
{
   public class SampleCode {}
}
```

- If you import both classes, as follows, you will have a name conflict when referring to the SampleCode class:

```
import samples.SampleCode;
import langref.samples.SampleCode;
var mySample:SampleCode = new SampleCode(); // name conflict
```

- The compiler has no way of knowing which SampleCode class to use. To resolve this conflict, you must use the fully qualified name of each class, as follows:

```
var sample1:samples.SampleCode = new samples.SampleCode();
var sample2:langref.samples.SampleCode = new langref.samples.SampleCode();
```

Variables & their scope

- Variables allow you to store values that you use in your program. To declare a variable, you must use the varstatement with the variable name. For example, the following line of ActionScript declares a variable named i :

  var i;

- If you omit the var statement when declaring a variable, you will get a compiler error in strict mode and run-time error in standard mode. For example, the following line of code will result in an error if the variable i has not been previously defined:

- i; // error if i was not previously defined

- To associate a variable with a data type, you must do so when you declare the variable. Declaring a variable without designating the variable's type is legal, but will generate a compiler warning in strict mode. You designate a variable's type by appending the variable name with a colon (:), followed by the variable's type. For example, the following code declares a variable i that is of type int:

  var i:int;

- You can assign a value to a variable using the assignment operator (=). For example, the following code declares a variable i and assigns the value 20 to it:

  var i:int;

  i = 20;

- You may find it more convenient to assign a value to a variable at the same time that you declare the variable, as in the following example:

  var i:int = 20;

- The technique of assigning a value to a variable at the time it is declared is commonly used not only when assigning primitive values such as integers and strings, but also when creating an array or instantiating an instance of a class. The following example shows an array that is declared and assigned a value using one line of code.

  var numArray:Array = ["zero", "one", "two"];

- You can create an instance of a class by using the new operator. The following example creates an instance of a named CustomClass , and assigns a reference to the newly created class instance to the variable namedcustomItem :

  var customItem:CustomClass = new CustomClass();

- If you have more than one variable to declare, you can declare them all on one line of code by using the comma operator (,) to separate the variables. For example, the following code declares three variables on one line of code:

  var a:int, b:int, c:int;

- You can also assign values to each of the variables on the same line of code. For example, the following code declares three variables (a , b, and c) and assigns each a value:

  var a:int = 10, b:int = 20, c:int = 30;

  Understanding variable scope

- The scope of a variable is the area of your code where the variable can be accessed by a lexical reference. The example shows that a global variable is available both inside and outside the function definition.

```
var strGlobal:String = "Global";
function scopeTest()
{
   trace(strGlobal); // Global
}
scopeTest();
trace(strGlobal); // Global
```

- You declare a local variable by declaring the variable inside a function definition. For example, if you declare a variable named str2 within a function named localScope() , that variable will not be available outside the function.

```
function localScope()
{
   var strLocal:String = "local";
}
localScope();
trace(strLocal); // error because strLocal is not defined globally
```

- If the variable name you use for your local variable is already declared as a global variable, the local definition hides (or shadows) the global definition while the local variable is in scope

```
var str1:String = "Global";
```

```
function scopeTest ()
{
   var str1:String = "Local";
   trace(str1); // Local
}
scopeTest();
trace(str1); // Global
```

Conditions and looping

if..else

- The if..else conditional statement allows you to test a condition and execute a block of code if that condition exists, or execute an alternative block of code if the condition does not exist. For example, the following code tests whether the value of x exceeds 20, generates a trace() function if it does, or generates a different trace() function if it does not:

```
if (x > 20)
{
   trace("x is > 20");
}
else
{
   trace("x is <= 20");
}
```

- If you do not want to execute an alternative block of code, you can use the if statement without the elsestatement.

if..else if

- You can test for more than one condition using the if..else if conditional statement. For example, the following code not only tests whether the value of x exceeds 20, but also tests whether the value of x is negative:

```
if (x > 20)
{
   trace("x is > 20");
}
```

```
else if (x < 0)
{
   trace("x is negative");
}
```

- If an if or else statement is followed by only one statement, the statement does not need to be enclosed in braces. For example, the following code does not use braces:

```
if (x > 0)
   trace("x is positive");
else if (x < 0)
   trace("x is negative");
else
   trace("x is 0");
```

- However, Adobe recommends that you always use braces, because unexpected behavior can occur if statements are later added to a conditional statement that lacks braces. For example, in the following code the value ofpositiveNums increases by 1 whether or not the condition evaluates to true :

```
var x:int;
var positiveNums:int = 0;

if (x > 0)
   trace("x is positive");
   positiveNums++;

trace(positiveNums); // 1
```

switch

- The switch statement is useful if you have several execution paths that depend on the same condition expression. It provides functionality similar to a long series of if..else if statements, but is somewhat easier to read
- For example, the following switch statement prints the day of the week, based on the day number returned by the Date.getDay() method:

```
var someDate:Date = new Date();
var dayNum:uint = someDate.getDay();
switch(dayNum)
```

```
{
    case 0:
        trace("Sunday");
        break;
    case 1:
        trace("Monday");
        break;
    case 2:
        trace("Tuesday");
        break;
    case 3:
        trace("Wednesday");
        break;
    case 4:
        trace("Thursday");
        break;
    case 5:
        trace("Friday");
        break;
    case 6:
        trace("Saturday");
        break;
    default:
        trace("Out of range");
        break;
}
```

Functions

Functions are blocks of code that carry out specific tasks and can be reused in your program. There are two types of functions in ActionScript 3.0: methods and function closures . Whether a function is a called a method or a function closure depends on the context in which the function is defined. A function is called a method if you define it as part of a class definition or attach it to an instance of an object. A function is called a function closure if it is defined in any other way.

- Functions passed as arguments to another function are passed by reference and not by value. When you pass a function as an argument, you use only the identifier and not the

parentheses operator that you use to call the method. For example, the following code passes a function named clickListener() as an argument to theaddEventListener() method:

addEventListener(MouseEvent.CLICK, clickListener);

- For example, the following code creates two functions: foo() , which returns a nested function named rectArea()that calculates the area of a rectangle, and bar() , which calls foo() and stores the returned function closure in a variable named myProduct . Even though the bar() function defines its own local variable x (with a value of 2), when the function closure myProduct() is called, it retains the variable x (with a value of 40) defined in functionfoo(). The bar() function therefore returns the value 160 instead of 8 .

```
function foo():Function
{
   var x:int = 40;
   function rectArea(y:int):int // function closure defined
   {
      return x * y
   }
   return rectArea;
}
function bar():void
{
   var x:int = 2;
   var y:int = 4;
   var myProduct:Function = foo();
   trace(myProduct(4)); // function closure called
}
bar(); // 160
```

OO Programing in Actionscript

Classes

- A class is an abstract representation of an object. A class stores information about the types of data that an object can hold and the behaviors that an object can exhibit.
- The usefulness of such an abstraction may not be apparent when you write small scripts that contain only a few objects interacting with one another

- ActionScript 3.0 class definitions use syntax that is similar to that used in ActionScript 2.0 class definitions

```
public class Shape
{
   var visible:Boolean = true;
}
```

- One significant syntax change involves class definitions that are inside a package. For example, the following class declarations show how the BitmapData class, which is part of the flash.display package, is defined in ActionScript 2.0 and ActionScript 3.0:

```
// ActionScript 2.0
class flash.display.BitmapData {}


// ActionScript 3.0
package flash.display
{
   public class BitmapData {}
}
```

Class attributes

- ActionScript 3.0 allows you to modify class definitions using one of the following four attributes:

• Attribute	• Definition
• dynamic	• Allow properties to be added to instances at run time.
• final	• Must not be extended by another class.
• internal (default)	• Visible to references inside the current package.
• public	• Visible to references everywhere.

Class body

- The class body, which is enclosed by curly braces, is used to define the variables, constants, and methods of your class. The following example shows the declaration for the Accessibility class in the Adobe Flash Player API:

```
public final class Accessibility
{
   public static function get active():Boolean;
   public static function updateProperties():void;
}
```

Interfaces

- An interface is a collection of method declarations that allows unrelated objects to communicate with one another. For example, ActionScript 3.0 defines the IEventDispatcher interface, which contains method declarations that a class can use to handle event objects. The IEventDispatcher interface establishes a standard way for objects to pass event objects to one another. The following code shows the definition of the IEventDispatcher interface:

```
public interface IEventDispatcher
{
   function addEventListener(type:String, listener:Function,
         useCapture:Boolean=false, priority:int=0,
         useWeakReference:Boolean = false):void;
   function removeEventListener(type:String, listener:Function,
         useCapture:Boolean=false):void;
   function dispatchEvent(event:Event):Boolean;
   function hasEventListener(type:String):Boolean;
   function willTrigger(type:String):Boolean;
}
```

Inheritance

- Inheritance is a form of code reuse that allows programmers to develop new classes that are based on existing classes. The existing classes are often referred to as base classes or superclasses, while the new classes are usually called subclasses.

```
class Shape
{
   public function area():Number
```

```
    {
        return NaN;
    }
}

class Circle extends Shape
{
    private var radius:Number = 1;
    override public function area():Number
    {
        return (Math.PI * (radius * radius));
    }
}

class Square extends Shape
{
    private var side:Number = 1;
    override public function area():Number
    {
        return (side * side);
    }
}

var cir:Circle = new Circle();
trace(cir.area()); // output: 3.141592653589793
var sq:Square = new Square();
trace(sq.area()); // output: 1
```

- Because each class defines a data type, the use of inheritance creates a special relationship between a base class and a class that extends it.
- A subclass is guaranteed to possess all the properties of its base class, which means that an instance of a subclass can always be substituted for an instance of the base class.

Working with strings

- The String class contains methods that let you work with text strings.

- Strings are important in working with many objects. The methods described in this chapter are useful in working with strings used in objects such as TextField, StaticText, XML, ContextMenu, and FileReference objects.
- The String class is used to represent string (textual) data in ActionScript 3.0. ActionScript strings support both ASCII and Unicode characters. The simplest way to create a string is to use a string literal. To declare a string literal, use straight double quotation mark (") or single quotation mark (') characters. For example, the following two strings are equivalent:

```
var str1:String = "hello";
var str2:String = 'hello';
```

You can also declare a string by using the new operator, as follows:

```
    var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();      // str3 == ""
```

The following two strings are equivalent:

```
var str1:String = "hello";
var str2:String = new String("hello");
```

- To use single quotation marks (') within a string literal defined with single quotation mark (') delimiters, use the backslash escape character (\). Similarly, to use double quotation marks (") within a string literal defined with double quotation marks (") delimiters, use the backslash escape character (\). The following two strings are equivalent:

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

Working with Arrays

- To access an individual element of an indexed array, you use the array access ([]) operator to specify the index position of the element you wish to access. For example, the following code represents the first element (the element at index 0) in an indexed array named songTitles :

```
songTitles[0]
```

- The combination of the array variable name followed by the index in square brackets functions as a single identifier. (In other words, it can be used in any way a variable name

can). You can assign a value to an indexed array element by using the name and index on the left side of an assignment statement:

```
songTitles[1] = "Symphony No. 5 in D minor";

var oddNumbers:Array = [1, 3, 5, 7, 9, 11];

var len:uint = oddNumbers.length;

for (var i:uint = 0; i < len; i++)

{

   trace("oddNumbers[" + i.toString() + "] = " + oddNumbers[i].toString());

}
```

Handling errors

- ActionScript 3.0 includes many tools for error handling, including:
- Error classes. ActionScript 3.0 includes a broad range of Error classes to expand the scope of situations that may produce error objects. Each Error class helps applications handle and respond to specific error conditions, whether they are related to system errors (like a MemoryError condition), coding errors (like an ArgumentError condition), networking and communication errors (like a URIError condition), or other situations. For more information on each class, see Comparing the Error classes.
- Fewer silent failures. In earlier versions of Flash Player, errors were generated and reported only if you explicitly used the throw statement. For Flash Player 9 and later and Adobe AIR, native ActionScript methods and properties throw run-time errors that allow you to handle these exceptions more effectively when they occur, and then individually react to each exception.
- Clear error messages displayed during debugging. When you are using the debugger version of Flash Player or Adobe AIR, problematic code or situations will generate robust error messages, which help you easily identify reasons why a particular block of code fails. This makes fixing errors more efficient. For more information, seeWorking with the debugger versions of Flash Player and AIR.
- Precise errors allow for clear error messages displayed to users at run time. In previous versions of Flash Player, the FileReference.upload() method returned a Boolean value of false if the upload() call was unsuccessful, indicating one of five possible errors. If an

error occurs when you call the upload() method in ActionScript 3.0, you can throw one of four specific errors, which helps you display more accurate error messages to end users.

- Refined error handling. Distinct errors are thrown for many common situations. For example, in ActionScript 2.0, before a FileReference object has been populated, the name property has the value null (so, before you can use or display the name property, you need to ensure that the value is set and not null). In ActionScript 3.0, if you attempt to access the name property before it has been populated, Flash Player or AIR throws an IllegalOperationError, which informs you that the value has not been set, and you can usetry..catch..finally blocks to handle the error. For more information see Using try..catch..finally statements.
- No significant performance drawbacks. Using try..catch..finally blocks to handle errors takes little or no additional resources compared to previous versions of ActionScript.
- An ErrorEvent class that allows you to build listeners for specific asynchronous error events. For more information see Responding to error events and status.

Working with XML

- ActionScript 3.0 includes a group of classes based on the ECMAScript for XML (E4X) specification (ECMA-357 edition 2). These classes include powerful and easy-to-use functionality for working with XML data. Using E4X, you will be able to develop code with XML data faster than was possible with previous programming techniques. As an added benefit, the code you produce will be easier to read.
- You can assign an XML literal to an XML object, as follows:

```
var myXML:XML =
  <order>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
```

```
    </item>
  </order>
```

- As the following snippet shows, you can also use the new constructor to create an instance of an XML object from a string that contains XML data:

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

- If the XML data in the string is not well formed (for example, if a closing tag is missing), you will see a run-time error.
- You can also pass data by reference (from other variables) into an XML object, as the following example shows:

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = "5";
var content:String = "Chicken";
var x:XML = <{tagname} {attributename}={attributevalue}>{content}</{tagname}>;
trace(x.toXMLString())
  // Output: <item id="5">Chicken</item>
```

- To load XML data from a URL, use the URLLoader class, as the following example shows:

```
import flash.events.Event;
import flash.net.URLLoader;
import flash.net.URLRequest;

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
```

```
loader.addEventListener(Event.COMPLETE, onComplete);


function onComplete(event:Event):void
{
   var loader:URLLoader = event.target as URLLoader;
   if (loader != null)
   {
      externalXML = new XML(loader.data);
      trace(externalXML.toXMLString());
   }
   else
   {
      trace("loader is not a URLLoader!");
   }
}
```

UNIT 6 6 hrs

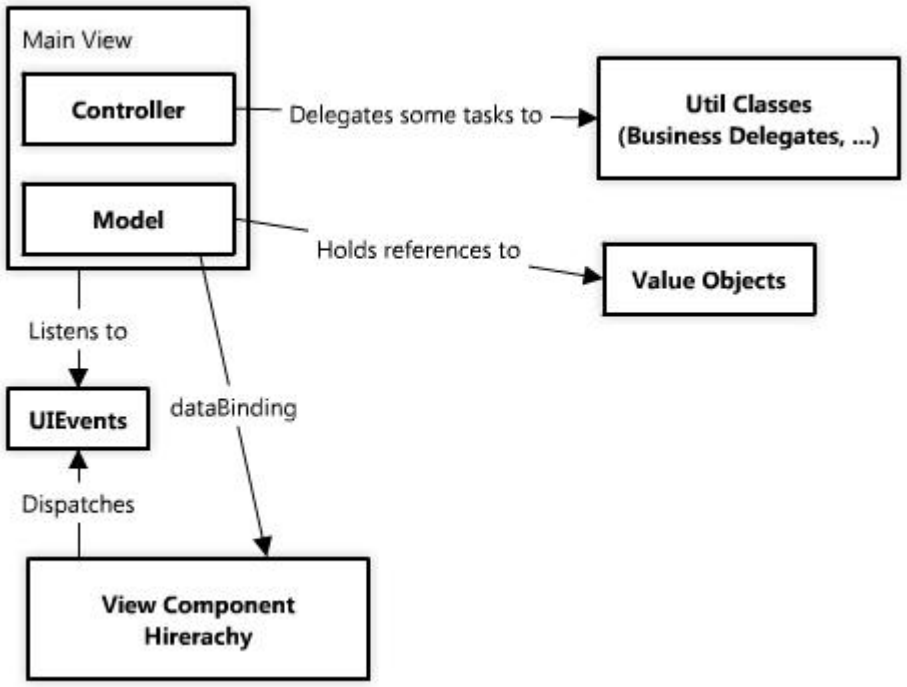
BUILDING RICH INTERNET APPLICATIONS WITH FLEX - 3:

- Framework fundamentals
- Understanding application life cycle
- Differentiating between Flash player and Framework
- Bootstrapping Flex applications
- Loading one flex application in to another
- Understanding application domains
- Understanding the preloader. Managing layout
- Flex layout overview
- Working with children
- Container types
- Layout rules,Padding, Borders and gaps
- Nesting containers
- Making fluid interfaces.

BUILDING RICH INTERNET APPLICATIONS WITH FLEX - 3:

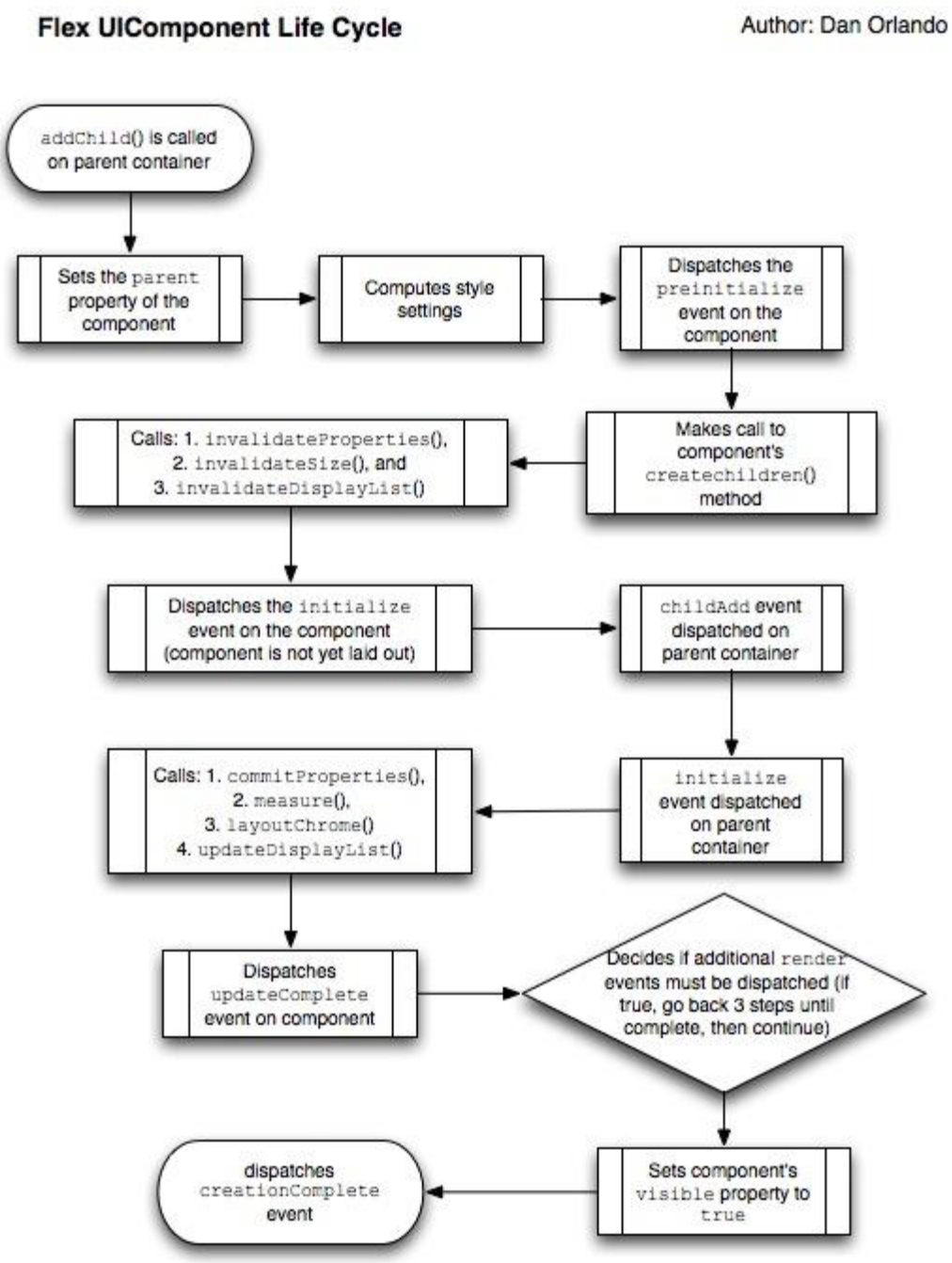
Framework fundamentals

Flex applications are designed as follows :



Understanding application life cycle

The following flow diagram represents the process that is abstracted within the Flex framework when the addChild() command is made to create and make a visual component visible on the stage, or when the properties of the component and/or it parent



change

Differentiating between Flash player and Framework

- Flex provides the ability to create a SWF file (flash player files) that runs on the Adobe Flash Player in any web browser.

- Just like the Flash was created to enable animators and illustrators to provide visually appealing experiences on the web, Flex was design for the same purpose, but Flex is Flash's "big brother", it's the same "technology" but the way applications are built is different and allow much more complicated applications to be built by software engineers instead of animators.
- Flex is not a new language (it does use MXML, but that is converted to Actionscript and most of the hard work is done in Actionscript), it uses Actionscript just like flash does, but it comes with a lot of extra features, to make more intelligent applications.

Loading one flex application in to another

- The SWFLoader control lets you load one Flex application into another Flex application as a SWF file.
- It has properties that let you scale its contents. It can also resize itself to fit the size of its contents.
- By default, content is scaled to fit the size of the SWFLoader control. The SWFLoader control can also load content on demand programmatically, and monitor the progress of a load operation.
- The SWFLoader control also lets you load the contents of a GIF, JPEG, PNG, SVG, or SWF file into your application, where the SWF file does not contain a Flex application, or a ByteArray representing a SWF, GIF, JPEG, or PNG.
- Creating a SWFLoader control

  SWFLoader control is defined in MXML by using the `<mx:SWFLoader>` tag, as the following example shows. Specify an `id` value if you intend to refer to a component elsewhere in your MXML, either in another tag or in an ActionScript block:

```
<?xml version="1.0"?>
<!-- controls\swfloader\SWFLoaderSimple.mxml-->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

   <mx:SWFLoader id="loader1" source="FlexApp.swf"/>
</mx:Application>
```

- The following example, in the file FlexApp.mxml, shows a simple Flex application that defines two Label controls, a variable, and a method to modify the variable:

```
<?xml version="1.0"?>
<!-- controls\swfloader\FlexApp.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   height="200" width="200">

   <mx:Script>
      <![CDATA[
         [Bindable]
         public var varOne:String = "This is a public variable.";
```

```
            public function setVarOne(newText:String):void {
               varOne=newText;
            }
         ]]>
      </mx:Script>

      <mx:Label id="lblOne" text="I am here."/>
      <mx:Label text="{varOne}"/>

      <mx:Button label="Nested Button" click="setVarOne('Nested button pressed.');"/>

   </mx:Application>
```

Understanding application domains

- The ApplicationDomain class is a container for discrete groups of class definitions.
- Application domains are used to partition classes that are in the same security domain.
- They allow multiple definitions of the same class to exist and allow children to reuse parent definitions.
- Application domains are used when an external SWF file is loaded through the Loader class. All ActionScript 3.0 definitions in the loaded SWF file are stored in the application domain, which is specified by the `applicationDomain` property of the LoaderContext object that you pass as a `context` parameter of the Loader object's `load()` or `loadBytes()` method. The LoaderInfo object also contains an `applicationDomain` property, which is read-only.
- All code in a SWF file is defined to exist in an application domain.
- The current application domain is where your main application runs. The system domain contains all application domains, including the current domain, which means that it contains all Flash Player classes.
- Every application domain, except the system domain, has an associated parent domain.
- The parent domain of your main application's application domain is the system domain. Loaded classes are defined only when their parent doesn't already define them. You cannot override a loaded class definition with a newer definition.

Understanding the preloader

Flex layout overview

- One of the key features of Flex is its ability to simplify application layout. Traditional application development requires writing layout code, or working with layout components in a nonintuitive manner.
- With MXML and Flex's layout containers, you can produce most applications without having to write a single line of custom layout code.
- Container components are the basis of how Flex provides layout logic. At the most
- basic level, the Application class is a container, and subitems within the Application
- class (tag) are called children. In MXML, placing nodes within a container declaration
- signifies that the objects are instantiated and are added to the container as children,
- and the container automatically handles their positioning and sizing.
- For example, in the following code two children are added to the Application container
- —a TextInput instance and a Button instance:

```
<?xml version="1.0" encoding="utf-8"?>
<mx: Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx: TextInput/>
   <mx: Button label="Submit"/>
</mx: Application>
```

In the preceding code, you added two children to the Application container by simply splacing the children as subnodes of the container using MXML.

In the example added to the container's display list, which, under the hood, is the same display list Flash Player uses. Containers allow for several different types of layout management

Working with children, Container types

- Containers provide a hierarchical structure that lets you control the layout characteristics of child components.
- . There are two types of containers: layout and navigator.
- Containers have predefined navigation and layout rules, so you do not have to spend time defining these. Instead, you can concentrate on the information that you deliver, and the options that you provide for your users, and not worry about implementing all the details of user action and application response.
- Predefined layout rules also offer the advantage that your users soon grow accustomed to them. That is, by standardizing the rules of user interaction, your users do not have to

think about how to navigate the application, but can instead concentrate on the content that the application offers.

- Different containers support different layout rules:

  - All containers, except the Canvas container, support *automatic* layout
  - The Canvas container, and optionally the Application and Panel containers, use *absolute* layout, where you explicitly specify the children's *x* and *y* positions.
  - Absolute layout provides a greater level of control over sizing and positioning than does automatic layout; for example,

About layout containers and navigator containers

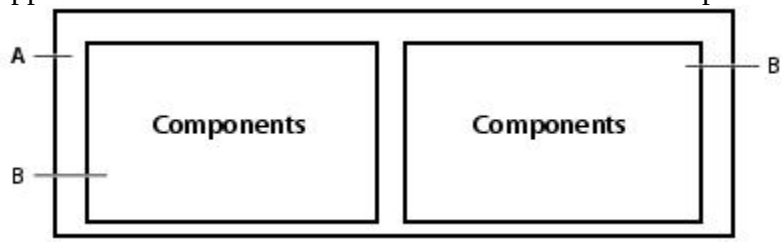
Flex defines two types of containers:

Layout containers

- Control the sizing and positioning of the child controls and child containers defined within them. For example, a Grid layout container sizes and positions its children in a layout similar to an HTML table.
- Layout containers also include graphical elements that give them a particular style or reflect their function.
- The DividedBox container, for example, has a bar in the center that users can drag to change the relative sizes of the two box divisions.
- The TitleWindow control has an initial bar that can contain a title and status information.

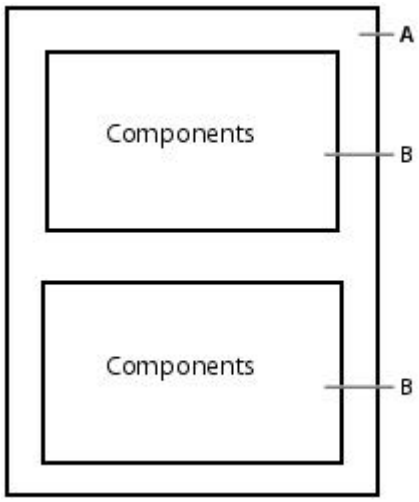
Navigator containers

- Control user movement, or navigation, among multiple child containers.
- The individual child containers, not the navigator container, control the layout and positioning of their children.
- For example, an Accordion navigator container lets you construct a multipage form from multiple Form layout containers.
- Although you can create an entire Flex application by using a single container, typical applications use multiple containers.



A. Parent HBox layout container *B. Child VBox layout container*

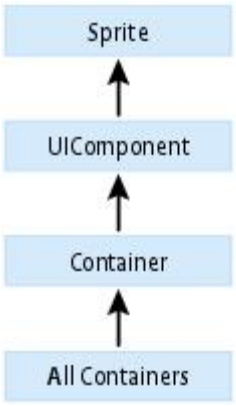
- A VBox container arranges its children in a single vertical stack, or column, and oversees the layout of its own children. The following image shows the preceding example with the outermost container changed to a VBox layout container:



A. Parent VBox layout container *B. Child VBox layout container*

Class hierarchy for containers

Flex containers are implemented as a hierarchy in an ActionScript class library, as the following image shows:



Container example

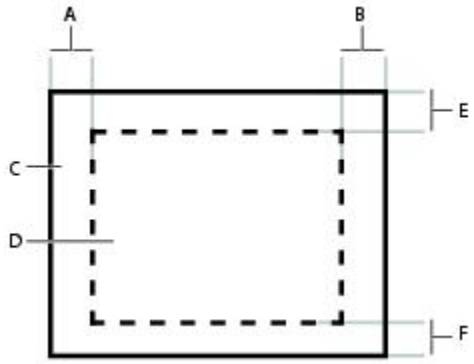
```
<?xml version="1.0"?>
<!-- containers\intro\Panel3Children.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Panel title="My Application"
        layout="vertical" horizontalAlign="center"
        paddingLeft="10" paddingRight="10"
        paddingTop="10" paddingBottom="10">
            <mx:TextInput id="myinput" text="enter zip code"/>
```

```
            <mx:Button id="mybutton" label="GetWeather"/>
            <mx:TextArea id="mytext" height="20"/>
    </mx:Panel>
</mx:Application>
```

Padding, Borders and gaps, Nesting containers

- The rectangular region of a container encloses its content area, the area that contains its child components.
- The size of the region around the content area is defined by the container padding and the width of the container border.
- A container has top, bottom, left, and right padding, each of which you can set to a pixel width. A container also has properties that let you specify the type and pixel width of the border.
- The following image shows a container and its content area, padding, and borders:



A. Left padding *B.* Right padding *C.* Container *D.* Content area *E.* Top padding *F. Bottom padding*

Making fluid interfaces

- Fluid layout makes your application resize and fit to the size of the browser everytime the browser is resized. This is also called elastic or liquid layout sometimes.

  One way of achieving it is by using percentage for the dimensions of your container objects viz. HBox.

- Use percentage for the children of this container too as this will help the layout manager in auto resizing the child container according to the new size of its parent container. e.g.

```
<mx:vbox id="parentContainer"width="100%" height="100%"
horizontalscrollpolicy="off" >
     <mx:hbox id="childContainer" width="100%" height="20%"
horizontalscrollpolicy="off" stylename="titleBox"/>
</mx:vbox>
```

To make the repainting much smoother, we are going to add the function mentioned below

```
private function resizeHandler():void
{
 stage.addEventListener(Event.RESIZE, onStageResize);
}
private function onStageResize(e:Event):void
{
   validateNow();
}
```

UNIT – 7 6hrs

BUILDING RICH INTERNET APPLICATIONS WITH FLEX – 4:

- Working with UI components:
  - Understanding UI Components
  - Creating component instances
  - Common UI Component properties
  - Handling events
  - Button, Value selectors, Text components, List based controls
  - Data models and Model View Controller
  - Creating collection objects, Setting the data provider
  - Using Data grids
  - Using Tree controls
  - Working with selected values and items
  - Pop up controls, Navigators, Control bars
- Working with data:
  - Using data models
  - Using XML
  - Using Actionscript classes
  - Data Binding.

- **BUILDING RICH INTERNET APPLICATIONS WITH FLEX – 4:**

Working with UI components:

Understanding UI Components

- Flex includes a component-based development model that you use to develop your application and its user interface. You can use the prebuilt visual components included with Flex, extend components to add new properties and methods, and create components as required by your application.
- Visual components are extremely flexible and provide you with a great deal of control over the component's appearance, how the component responds to user interactions, the font and size of any text included in the component, the size of the component in the application, and many other characteristics.
- The characteristics of visual components include the following:
  - **Size** - Height and width of a component. All visual components have a default size. You can use the default size, specify your own size, or let Flex resize a component as part of laying out your application.
  - **Events** - Application or user actions that require a component response. Events include component creation, mouse actions such as moving the mouse over a component, and button clicks.
  - **Styles** - Characteristics such as font, font size, and text alignment. These are the same styles that you define and use with Cascading Style Sheets (CSS).
  - **Behaviors** - Visible or audible changes to the component triggered by an application or user action. Examples of behaviors are moving or resizing a component based on a mouse click.
  - **Skins** - Classes that control a visual component's appearance.

Commonly used UIComponent properties

The following table lists only the most commonly used properties of components that extend the UIComponent class:

Property	Type	Description
`doubleClickEnabled`	Boolean	Setting to `true` lets the component dispatch a `doubleClickEvent` when a user presses and releases the mouse button twice in rapid succession over the component.
`enabled`	Boolean	Setting to `true` lets the component accept keyboard focus and mouse input. The default value is `true`. If you set `enabled` to `false` for a container, Flex dims the color of the container and all of its children, and blocks user input to the container

		and to all its children.
`height`	Number	The height of the component, in pixels. In MXML tags, but not in ActionScript, you can set this property as a percentage of available space by specifying a value such as 70%; in ActionScript, you must use the `percentHeight` property. The property always returns a number of pixels. In ActionScript, you use the `perCent`
`id`	String	Specifies the component identifier. This value identifies the specific instance of the object and should not contain any white space or special characters. Each component in a Flex document must have a unique `id` value. For example, if you have two custom components, each component can include one, and only one Button control with the id "okButton".
`percentHeight`	Number	The height of the component as a percentage of its parent container, or for `<mx:Application>` tags, the full height of the browser. Returns `NaN` if a percent-based width has never been set or if a `width` property was set after the `percentWidth` was set.
`percentWidth`	Number	The width of the component as a percentage of its parent container, or for `<mx:Application>` tags, the full span of the browser. Returns `NaN` if a percent-based width has never been set or if a `width` property was set after the `percentWidth` was set.
`styleName`	String	Specifies the style class selector to apply to the component.
`toolTip`	String	Specifies the text string displayed when the mouse pointer hovers over that component.
`visible`	Boolean	Specifies whether the container is visible or invisible. The default value is `true`, which specifies that the container is visible.
`width`	Number	The width of the component, in pixels. In MXML tags, but not in ActionScript, you can set this property as a percentage of available space by specifying a value such as 70%; in ActionScript, you must use the `percentWidth`

		property. The property always returns a number of pixels.
x	Number	The component's *x* position within its parent container. Setting this property directly has an effect only if the parent container uses absolute positioning.
y	Number	The component's *y* position within its parent container. Setting this property directly has an effect only if the parent container uses absolute positioning.

Handling events

EventDispatcher class

- Event handling in ActionScript 3.0 depends heavily on the EventDispatcher class.
- Although this class isn't entirely new to ActionScript, it is the first time it has been included as a core part of the ActionScript language
- ActionScript 2.0, you define the event handler within the object receiving the event—giving the function the name of the event being received. For example, to react to an "onPress" event for a button named `submitButton` in ActionScript 2.0, you would use:

```
submitButton.onPress = function() { ... }
```

- Using EventDispatcher, the same elements are at play; an object receiving an event, an event name, and a function that reacts to an event—only the process is slightly different. The code using EventDispatcher looks like this:

```
function pressHandler(){ ... }
submitButton.addEventListener("onPress", pressHandler);
```

- This process adds what appears to be an extra step, but it allows for more flexibility. Since you are using a function to add event handlers instead of defining them directly on the target object itself, you can now add as many handlers as you like to "listen" to a single event.
- Removing events in ActionScript 2.0 just meant deleting the handler:

```
delete submitButton.onPress;
```

EventDispatcher methods

Here is a summary of the methods in EventDispatcher for ActionScript 3.0. Many of these methods are similar to the methods in the ActionScript 2.0 version:

- addEventListener(type:String, listener:Function, useCapture:Boolean = false, priority:int = 0, useWeakReference:Boolean = false):void
- removeEventListener(type:String, listener:Function, useCapture:Boolean = false):void
- dispatchEvent(event:Event):Boolean
- hasEventListener(type:String):Boolean
- willTrigger(type:String):Boolean

- **addEventListener():** Adds an event handler function to listen to an event so that when that event occurs, the function will be called.
- **removeEventListener():** Removes an event handler added to a listeners list using addEventListener. The same first 3 arguments used in addEventListener must be used in removeEventListener to remove the correct handler.
- **dispatchEvent():** Sends the passed event to all listeners in the listeners list of an object that relates to the event type. This method is most commonly used when creating custom events.
- **hasEventListener():** Determines whether or not an object has listeners for a specific type of event.
- **willTrigger():** Determines whether or not an object or any of its parent containers have listeners for a specific type event. This is much like hasEventListener but this method checks the current object as well as all objects that might be affected from the propagation of the event.

Button, Value selectors, Text components, List based controls

- The Button control is a commonly used rectangular button.
- Button controls look like they can be pressed, and have a text label, an icon, or both on their face. You can optionally specify graphic skins for each of several Button states.
- You can create a normal Button control or a toggle Button control. A normal Button control stays in its pressed state for as long as the mouse button is down after you select it. A toggle Button controls stays in the pressed state until you select it a second time.
- Buttons typically use event listeners to perform an action when the user selects the control. When a user clicks the mouse on a Button control, and the Button control is enabled, it dispatches a click event and a buttonDown event. A button always dispatches events such as the mouseMove, mouseOver, mouseOut, rollOver, rollOut, mouseDown, and mouseUp events whether enabled or disabled.
- You define a Button control in MXML by using the <mx:Button> tag, as the following example shows. Specify an id value if you intend to refer to a component elsewhere in your MXML, either in another tag or in an ActionScript block. The following code creates a Button control with the label "Hello world!":

```
<?xml version="1.0"?>
<!-- controls\button\ButtonLabel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

   <mx:Button id="button1" label="Hello world!" width="100"/>
</mx:Application>
```

- You use Flex text-based controls to display text and to let users enter text into your application. The following table lists the controls, and indicates whether the control can have multiple lines of input instead of a single line of text, and whether the control can accept user input:

Control	Multiline	Allows user Input
Label	No	No
TextInput	No	Yes
Text	Yes	No
TextArea	Yes	Yes
RichTextEditor	Yes	Yes

- All controls except the RichTextEditor control are single components with a simple text region; for example, the following image shows a TextInput control in a simple form:

```
<?xml version="1.0"?>
<!-- textcontrols/FormItemLabel.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" >
  <mx:Form id="myForm" width="500" backgroundColor="#909090">
    <!-- Use a FormItem to label the field. -->
    <mx:FormItem label="First Name">
      <mx:TextInput id="ti1" width="150"/>
    </mx:FormItem>
  </mx:Form>
</mx:Application>
```

Using Data grids and Using Tree controls

The DataGrid control provides the following features:

- Resizable, sortable, and customizable column layouts, including hidable columns
- Optional customizable column and row headers, including optionally wrapping header text
- Columns that the user can resize and reorder at run time
- Selection events
- Ability to use a custom item renderer for any column
- Support for paging through data
- Locked rows and columns that do not scroll

```
<?xml version="1.0"?>
<!-- dpcontrols/DataGridSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:DataGrid>
    <mx:ArrayCollection>
      <mx:Object>
        <mx:Artist>Pavement</mx:Artist>
        <mx:Price>11.99</mx:Price>
        <mx:Album>Slanted and Enchanted</mx:Album>
```

```
        </mx:Object>
        <mx:Object>
          <mx:Artist>Pavement</mx:Artist>
          <mx:Album>Brighten the Corners</mx:Album>
          <mx:Price>11.99</mx:Price>
        </mx:Object>
      </mx:ArrayCollection>
    </mx:DataGrid>
  </mx:Application>
```

- You define a Tree control in MXML by using the `<mx:Tree>` tag.
- The Tree class extends the List class and Tree controls take all of the properties and methods of the List control
- The Tree control normally gets its data from a hierarchical data provider, such as an XML structure. If the Tree represents dynamically changing data, you should use a collection, such as the standard ArrayCollection or XMLListCollection object, as the data provider.
- The Tree control uses a data descriptor to parse and manipulate the data provider content. By default, the Tree control uses a DefaultDataDescriptor instance, but you can create your own class and specify it in the Tree control's `dataDescriptor` property.

The following code contains a single Tree control that defines the tree shown in the image in Tree control.

```
<?xml version="1.0"?>
<!-- dpcontrols/TreeSimple.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Tree id="tree1" labelField="@label" showRoot="true" width="160">
    <mx:XMLListCollection id="MailBox">
      <mx:XMLList>
        <folder label="Mail">
          <folder label="INBOX"/>
          <folder label="Personal Folder">
            <Pfolder label="Business" />
            <Pfolder label="Demo" />
              <Pfolder label="Personal" isBranch="true" />
              <Pfolder label="Saved Mail" />
            </folder>
            <folder label="Sent" />
            <folder label="Trash" />
        </folder>
      </mx:XMLList>
    </mx:XMLListCollection>
  </mx:Tree>
</mx:Application>
```

Working with data

- *Data binding* is the process of tying the data in one object to another object. It provides a convenient way to pass data between the different layers of the application.
- Data binding requires a source property, a destination property, and a triggering event that indicates when to copy the data from the source to the destination.
- An object dispatches the triggering event when the source property changes.
- Adobe Flex provides three ways to specify data binding: the curly braces ({}) syntax in MXML, the <mx:Binding> tag in MXML, and the BindingUtils methods in ActionScript.
- The following example uses the curly braces ({}) syntax to show a Text control that gets its data from a TextInput control's text property:

```
<?xml version="1.0"?>
<!-- binding/BasicBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

   <mx:TextInput id="myTI" text="Enter text here"/>
   <mx:Text id="myText" text="{myTI.text}"/>
</mx:Application>
```

Using ActionScript expressions in curly braces

Binding expressions in curly braces can contain an ActionScript expression that returns a value. For example, you can use the curly braces syntax for the following types of binding.

- A single bindable property inside curly braces
- To cast the data type of the source property to a type that matches the destination property
- String concatenation that includes a bindable property inside curly braces
- Calculations on a bindable property inside curly braces
- Conditional operations that evaluate a bindable property value

The following example shows a data model that uses each type of binding expression:

```
<?xml version="1.0"?>
<!-- binding/AsInBinding.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

   <mx:Model id="myModel">
      <myModel>
        <!-- Perform simple property binding. -->
        <a>{nameInput.text}</a>
        <!-- Perform string concatenation. -->
        <b>This is {nameInput.text}</b>
        <!-- Perform a calculation. -->
        <c>{(Number(numberInput.text)) * 6 / 7}</c>
```

```
        <!-- Perform a conditional operation using a ternary operator. -->
        <d>{(isMale.selected) ? "Mr." : "Ms."} {nameInput.text}</d>
      </myModel>
    </mx:Model>

    <mx:Form>
      <mx:FormItem label="Last Name:">
        <mx:TextInput id="nameInput"/>
      </mx:FormItem>
      <mx:FormItem label="Select sex:">
        <mx:RadioButton id="isMale"
          label="Male"
          groupName="gender"
          selected="true"/>
        <mx:RadioButton id="isFemale"
          label="Female"
          groupName="gender"/>
      </mx:FormItem>
      <mx:FormItem label="Enter a number:">
        <mx:TextInput id="numberInput" text="0"/>
      </mx:FormItem>
    </mx:Form>

    <mx:Text
      text="{'Calculation: '+numberInput.text+' * 6 / 7 = '+myModel.c}"/>
    <mx:Text text="{'Conditional: '+myModel.d}"/>
  </mx:Application>
```

UNIT – 8 8hrs

BUILDING ADVANCED WEB 2.0 APPLICATIONS

- Definition of mash up applications
- Mash up Techniques
- Building a simple mash up application with AJAX
- Remote data communication, strategies for data communication
- Simple HTTPServices
- URLLoader in Flex
- Web Services in Flex
- Examples:
    - Building an RSS reader with AJAX
    - Building an RSS reader with Flex.

BUILDING ADVANCED WEB 2.0 APPLICATIONS

Definition of Mash up applications

- A *mashup* is similar to a remix. You might have heard examples again from the music world where elements of Led Zeppelin are combined with Jayzee, for example, to form a weird rap/rock song. That's a mashup. The same can be done with data from the Internet.

Mashup Techniques - Mashing on the Web Server

- Every site sits on a Web server. It's the thing that serves up the page, typically Internet Information Server (IIS) in the Microsoft world.

Understanding the Architecture How it works

This use case is definitely the most straightforward:

- The Web browser communicates with the server, requesting a page using straight HTTP or HTTPS.
- That page is constructed by the Web server, which reaches out to what I'll call the *source* or *partner sites* (for example, Amazon, Yahoo, or Google, and so on). The first request in this example is to Amazon using the Simple Object Access Protocol (SOAP) over HTTP.
- Amazon returns back a SOAP response.
- The second request in this example is to Yahoo using a Representational State Transfer style approach.
- Yahoo responds with Plain Old XML over HTTP.
- Lastly, the Web server aggregates the responses, combining and rationalizing the data in whatever manner makes sense.
- The resulting data is bound to the HTML and inserted into the response, which is sent back to the browser.

Pros and Cons

- The benefits of this approach are that the browser is decoupled entirely from the partner sites supplying the data. The Web server acts as a proxy and aggregator for the responses.
- Disadvantages of this approach are that the browser requests an entire page, which typically is acceptable.
- Second, the Web server is doing all the work in terms of data manipulation. Though this is good in terms of maintenance, it's not so good in terms of scalability. When your mashup gains popularity and starts being viewed by thousands of users, the amount of work the server's doing increases, while the browser residing at the client is relatively idle.

Remote Data Communication

- Remote data communication occurs at runtime.
- Flex applications support a variety of remote data communication techniques built on standards.

- There are three basic categories of Flex application remote data communication:

HTTP request/response-style communication

- This category consists of several overlapping techniques. Utilizing the Flex framework HTTPService component or the Flash Player API URLLoader class, you can send and load uncompressed data such as text blocks, URL encoded data, and XML packets Each technique achieves the similar goal of sending requests and receiving responses using HTTP or HTTPS.

Real-time communication

- This category consists of persistent socket connections. Flash Player supports two types of socket connections: those that require a specific format for packets (XMLSocket) and those that allow raw socket connections (Socket)

File upload/download communication

- This category consists of the FileReference API which is native to Flash Player and allows file upload and download directly within Flex applications.

Understanding Strategies for Data Communication

- When you build Flex applications that utilize data communication, it's important to understand the strategies available for managing those communications and how to select the right strategy for an application. All Flex applications run in Flash Player. With the exception of some Flex applications created using Flex Data Services, almost all Flex applications are composed of precompiled *.swf* files that are loaded in Flash Player on the client.
- Because Flex applications are stateful and self-contained, they don't require new page requests and wholesale screen refreshes to make data requests and handle responses.
- The Flex framework provides components for working with data communication using standard HTTP requests as well as SOAP requests.

Working with Request/Response Data Communication

- You can work with request/response data communication in three basic ways: via simple HTTP services, web services, and Flash Remoting.

Simple HTTP Services

- The most basic type of HTTP request/response communication uses what we call *simple HTTP services*. These services include things such as text and XML resources, either in static documents or dynamically generated by something such as a ColdFusion page, a servlet, or an ASP.NET page.

HTTPService

- HTTPService is a component that allows you to make requests to simple HTTP services such as text files, XML files, or scripts and pages that return dynamic data. You must always define a value for the url property of an HTTPService object.
- The following example uses MXML to create an HTTPService object that loads text from a file called *data.txt* saved in the same directory as the compiled *.swf* file:
  <mx:HTTPService id="textService" url="data.txt" />

Sending requests

- Creating an HTTPService object does not automatically make the request to load the data. In order to make the request, you must call the send( ) method. If you want to load the data when the use clicks a button, you can call the send( ) method in response to a click event:

```
textService.send( );
```

Handling results

- The send( ) method makes the request, but a response is not likely to be returned instantaneously. Instead, the application must wait for a result event. The following example displays an alert when the data loads:

```
<mx:HTTPService id="textService" url="data.txt"  result="mx.controls.Alert.show('Data loaded')" />
```

Sending parameters

- When you want to pass parameters to the service, you can use the request property of the HTTPService instance. The request property requires an Object value. By default, the name/value pairs of the object are converted to URL-encoded format and are sent to the service using HTTP GET.
- The default value is object, which yields the default behavior you've already seen. You can optionally specify any of the following values:

text
The data is not parsed at all, but is treated as raw text.

flashvars
The data is assumed to be in URL-encoded format, and it will be parsed into an object with properties corresponding to the name/value pairs.

array
The data is assumed to be in XML format, and it is parsed into objects much the same as with the object settings. However, in this case, the result is always an array. If the returned data does not automatically parse into an array, the parsed data is placed into an array.

xml
The data is assumed to be in XML format, and it is interpreted as XML using the legacy XMLNode ActionScript class.

e4x
The data is assumed to be in XML format, and it is interpreted as XML using the ActionScript 3.0 XML class (E4X).

Using HTTPService with ActionScript

- Although the simplest and quickest way to use an HTTPService object is to primarily use MXML, this technique is best-suited to nonenterprise applications in which the data communication scenarios are quite simple.
- Because HTTPService components provide significant data conversion advantages (such as automatic serialization of data), it is still frequently a good idea to use an HTTPService object within a remote proxy. However, it is generally necessary to

then work with the HTTPService component entirely with ActionScript, including constructing the object and handling the responses.

URLLoader

- HTTPService allows you to use requests and handle responses to and from simple HTTP services. You can optionally use the Flash Player class called flash.net.URLLoader to accomplish the same tasks entirely with ActionScript, but at a slightly lower level.
- The first step when working with a URLLoader object is always to construct the object using the constructor method, as follows:
  var loader:URLLoader = new URLLoader( );

- Once you've constructed the object, you can do the following:
  - Send requests.
  - Handle responses.
  - Send parameters.

Sending requests

- You can send requests using the load( ) method of a URLLoader object. The load( ) method requires that you pass it a flash.net.URLRequest object specifying at a minimum what URL to use when making the request. The following makes a request to a text file called *data.txt*:
  loader.load(new URLRequest("data.txt"));

Handling responses

- URLLoader objects dispatch complete events when a response has been returned. Any return value is stored in the data property of the URLLoader object.

Sending parameters

- You can send parameters using URLLoader as well. In order to send parameters, you assign a value to the data property of the URLRequest object used to make the request. The URLRequest object can send binary data or string data.

Web Services

- Flash Player has no built-in support for SOAP web services. However, Flex provides a WebService component that uses built-in HTTP request/response support as well as XML support to enable you to work with SOAP-based web services. There are two ways you can work with the WebService components: using MXML and using ActionScript.

Using WebService Components with MXML

- You can create a WebService component instance using MXML. When you do, you should specify an id and a value for the wsdl property.\
- Eg: <mx:WebService id="statesService" wsdl="http://www.rightactionscript.com/states/webservice/StatesService.php?wsdl" />

- Web services define one or more methods or operations. You must define the WebService instance so that it knows about the operations using nested operation tags. The operation tag requires that you specify the name at a minimum.

Calling web service methods

- All operations that you define for a WebService component instance are accessible as properties of the instance. For example, in the preceding section, we created a WebService instance called statesService with an operation called getCountries. That means you can use ActionScript to reference the operation as statesService. getCountries.
- You can then call getCountries just as though it were a method of statesService:
  statesService.getCountries( );

Handling results

- When a web service operation returns a result, you can handle it in one of two ways: explicitly handle the result event or use data binding. Then, once a result is returned, you can retrieve the result value from the lastResult property of the operation.

Using WebService Components with ActionScript

- You can use a WebService component using ActionScript instead of MXML. This is useful in cases where you want to fully separate the view from the controller and the model, such as in the recommended remote proxy approach.
- The MXML version of the WebService component is an instance of mx.rpc.soap.mxml.WebService, which is a subclass of mx.rpc.soap.WebService. When you use the component directly from ActionScript you should instantiate mx.rpc.soap.WebService directly:

```
// Assume the code already has an import statement for mx.rpc.soap.WebService.
var exampleService:WebService = new WebService( );
```

- Next, you must call a method called loadWSDL( ). You must call the method prior to calling any of the web service operations. Assuming you set the wsdl property, you don't need to pass any parameters to loadWSDL( ):
  exampleService.loadWSDL( );