USN | | | | | | | | | | |

Internal Assessment Test 2 – Nov. 2017

Sub:	WEB 2.0 & RICH INTERNET APPLICATIONS					Sub Code:	13MCA552	Branch:	MCA
Date:	09-11-2017	Duration:	90 min's	Max Marks:	50	Sem / Sec:	V		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1 (a)	What is Flex Framework. Explain any six components of Flex Framework. What is Flex Framework? Explain any six components of Flex Framework? Flex is a collection of technologies that enables you to rapidly build applications deployed to Flash Player, a runtime environment for delivering sophisticated user interfaces and interactivity. Flex leverages existing, matured technologies and standards such as XML, web services, HTTP, Flash Player, and ActionScript. Even though Flex allows you to create complete rich Internet applications, it does so in a relatively simple and intuitive manner. While Flex does allow you to get under the hood for more granular control over all the elements, it significantly lowers the learning curve in that it allows you to compose applications rapidly by assembling off-theshelf components, including UI controls, layout containers, data models, and data communication components. 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. **Form controls** Form controls are standard controls such as buttons, text inputs, text areas, lists, radio buttons, checkboxes, and combo boxes. In addition to the standard form controls familiar to most HTML developers, the Flex class library also includes controls such as a rich text editor, a color selector, a date selector, and more. **Menu controls** Flex provides a set of menu controls such as pop-up menus and menu bars. **Media components** One of the hallmarks of Flex applications is rich media support. The Flex class library provides a set of components for working with media such as images, audio, and video. **Layout containers** Flex applications enable highly configurable screen layout. You can use the layout containers to place contents within a screen and determine how they will change over time or when the user changes the dimensions of Flash Player. With a diverse set of container components you can create sophisticated layouts using grids, forms, boxes, canvases, and more. You can place elements with absolute or relative coordinates so that they can adjust correctly to different dimensions within Flash Player. **Data components and data binding** Flex applications are generally distributed applications that make remote procedure calls to data services residing on servers. The data components consist of connectors that simplify the procedure calls, data models to hold the data that is returned, and	[10]	CO3	L3

data binding functionality to automatically associate form control data with data models.

Formatters and validators
Data that is returned from remote procedure calls often needs to be formatted before getting displayed to the user. The Flex class library includes a robust set of formatting features (format a date in a variety of string representations, format
a number with specific precision, format a number as a phone number string, etc.) to accomplish that task. Likewise, when sending data to a data service from user input, you'll frequently need to validate the data beforehand to
ensure it is in the correct form. The Flex class library includes a set of validators for just that purpose.

Cursor management
Unlike traditional web applications, Flex applications are stateful, and they don't have to do a complete screen refresh each time data is sent or requested from a data service. However, since remote procedure calls often incur network
and system latency, it's important to notify the user when the client is waiting on a response from the data service. Cursor management enables Flex applications to change the cursor appearance in order to notify the user of such changes.

State management
A Flex application will frequently require many state changes. For example, standard operations such as registering for a new account or making a purchase usually require several screens. The Flex class library provides classes for managing those changes in state. State management works not only at the macro level for screen changes, but also at the micro level for state changes within individual components. For example, a product display component could have several states: a base state displaying just an image and a name, and a details state that adds a description, price, and shipping availability. Furthermore, Flex provides the ability to easily apply transitions so that state changes are animated.

Effects
Flex applications aren't limited by the constraints of traditional web applications. Since Flex applications run within Flash Player, they can utilize the animation features of Flash. As such, the Flex class library enables an assortment of
effects such as fades, zooms, blurs, and glows.

History management
As states change within a Flex application, the history management features of the Flex class library enable you to navigate from state to state using the back and forward buttons of the web browser.

Drag and drop management
The Flex class library simplifies adding drag and drop functionality to components with built-in drag and drop functionality on select components and a manager class that allows you to quickly add drag and drop behaviors to components.

Tool tips
Use this feature of the Flex class library to add tool tips to elements as the user moves the mouse over them.

Style management
The Flex class library enables a great deal of control over how nearly every aspect of a Flex application is styled. You can apply style changes such as color and font settings to most controls and containers directly to the objects or via
CSS.

	Flex Builder 2 Flex Builder 2 is the official Adobe IDE for building and debugging Flex applications. Flex Builder 2 is the official Adobe IDE for building and debugging Flex applications. you can opt to install the free Flex SDK, which includes the compiler and the Flex framework. You can then integrate the Flex framework with a different IDE, or you can use any text editor to edit the MXML and ActionScript files, and you can run the compiler from the command line. Data components and data binding Formatters and validators			

2 (a)	Explain two types of runtime error handling in ActionScript.
	Explain two types of runtime error handling in ActionScript.
	Asynchronous and synchronous errors
	Handling Synchronous Errors
	Synchronous errors occur immediately when trying to execute a statement. You can use try/catch/finally to handle synchronous errors.
	When you have some code that may throw runtime errors, surround it with a try statement:
	try {
	// Code that might throw errors
	}
	You must then include one or more catch blocks following a try. If the code in the try block throws an error, the application attempts to match the error to the catch blocks in the order in which they appear. Every catch block must specify the specific type of error that it handles. The application runs the first catch block that it encounters to see if it matches the type of error thrown. All error types are either flash.errors.Error types or subclasses of Error. Therefore, you should try to catch more specific error types first, and more generic types (e.g., Error) later; for example:
	try {
	// Code that might throw errors
	}
	catch (error:IOError) {
	// Code in case the specific error occurs
	}
	catch (error:Error) {
	// Code in case a non-specific error occurs
	}
	In addition, you can add a finally clause that runs regardless of whether the try statement is successful:
	try {
	// Code that might throw errors
	}
	catch (error:IOError) {
	// Code in case the specific error occurs
	}
	catch (error:Error) {
	// Code in case a non-specific error occurs
	}
	finally {
	// Code to run in any case
	}
	Handling Asynchronous Errors
	Many objects in ActionScript can potentially throw *asynchronous errors*. Asynchronous errors are those that occur in response to network operations. For example, if a
		[10]	CO3	L3

requested file is not found, the network operation fails asynchronously, and an asynchronous
error is thrown. All asynchronous errors are in the form of events, and they
use the same event model as standard events. For example, if a URLLoader object
attempts to load data outside the Flash Player security sandbox, it dispatches a
securityError event. The following example illustrates how to handle error events:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
initialize="initializeHandler(event)">
<mx:Script>
<![CDATA[
private function initializeHandler(event:Event):void {
        var loader:URLLoader = new URLLoader( );
        // In order to test this you'll need to specify a URL of a file that
        // exists outside of the security sandbox.
        loader.load(new URLRequest("data.xml"));
        loader.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
        securityErrorHandler);
}

private function securityErrorHandler(event:SecurityErrorEvent):void
 {
        errors.text += event + "\n";
}
]]>
</mx:Script>
<mx:TextArea id="errors" />
</mx:Application>
```

3 (a)	What is Action Script.Discuss the different ways for embedding ActionScript within FLEX. What is Action Script? Discuss the different ways for embedding ActionScript within FLEX? ActionScript is the programming language understood by Flash Player and is the fundamental engine of all Flex applications. MXML simplifies screen layout and many basic tasks, but all of what MXML does is made possible by ActionScript, and ActionScript can do many things that MXML cannot do. For example, you need ActionScript to respond to events such as mouse clicks. Although it is possible to build an application entirely with MXML or entirely with ActionScript, it is more common and more sensible to build applications with the appropriate balance of both MXML and ActionScript. Each offers benefits, and they work well together. MXML is best suited for screen layout and basic data features. ActionScript is best suited for user interaction, complex data functionality, and any custom functionality not included in the Flex class library. ActionScript is supported natively by Flash Player, and does not require any additional libraries to run. All the native ActionScript classes are packaged in the flash package or in the top-level package. In contrast, the Flex framework is written in ActionScript, but those classes are included in a *.swf* file at compile time. All the Flex framework classes are in the mx package. There are three tiers of ActionScript APIs: Flash Player APIs	[10]	CO3	L3

These APIs are part of the Flash Player itself, and they run natively in that runtime environment. Flash Player APIs consist of core classes such as String, Number, Date, and Array as well as Flash Player-specific classes such as DisplayObject, URLLoader, NetConnection, Video, and Sound.

Flex framework APIs
Flex framework is effectively a layer on top of the Flash Player APIs. The Flex framework APIs consist of all the Flex containers (Application, VBox, etc.), controls (Button, TextInput, etc.), and other assorted data, manager, and utility classes

Custom APIs
These APIs are for the classes you build for use in custom applications.

How to use Action Script with Flex

When you want to use ActionScript within Flex, you have four basic options for where to place the code:

- Inline within MXML tags
- Nested within MXML tags
- In MXML scripts
- Within ActionScript classes

InlineActionScript
Inline ActionScript appears within MXML tags. Inline event handling and data binding using curly brace syntax necessarily uses basic ActionScript. The following example uses ActionScript to display an alert dialog box when the user clicks on a button:

```
<mx:Button id="alertButton" label="Show Alert"
click="mx.controls.Alert.show('Example')" />
```

The following example uses ActionScript to display an alert dialog box when the user clicks on a button:

```
<mx:Button id="alertButton" label="Show Alert"
click="mx.controls.Alert.show('Example')" />
```

Example for Inline Data Binding

```
<mx:VBox>
<mx:TextInput id="input" />
<mx:Text id="output" text="{input.text}" />
</mx:VBox>
```

Nested ActionScript
Nest ActionScript code within MXML tags is by placing the code in CDATA block.

```
<mx:Button>
<mx:click>
<![CDATA[
mx.controls.Alert.show("Example");
]]>
</mx:click>
</mx:Button>
```

MXML Scripts

The second way to add ActionScript code to an application is to place it within an MXML script. An MXML script appears in an MXML document within a Script element:

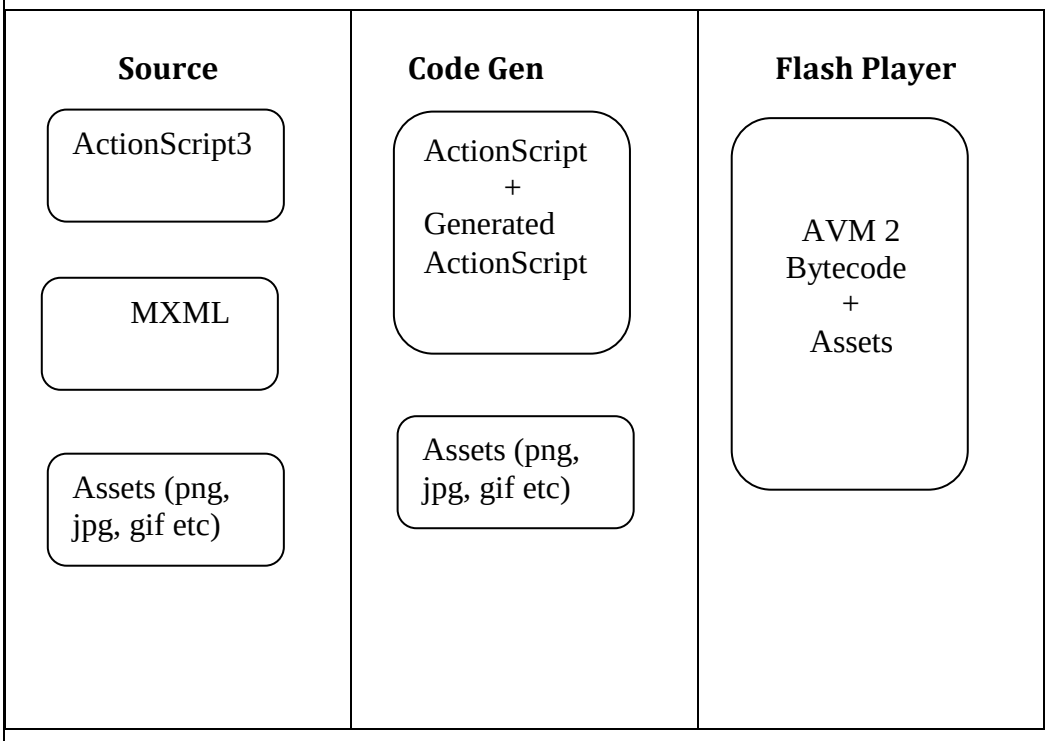
```
<mx:Script>
<![CDATA[
import mx.controls.Alert;
        private function example( ):void {
                Alert.show("Example");
}
]]>
</mx:Script>

<mx:Script source="code.as" />
```

Within MXML scripts, you can import classes and declare properties and methods.

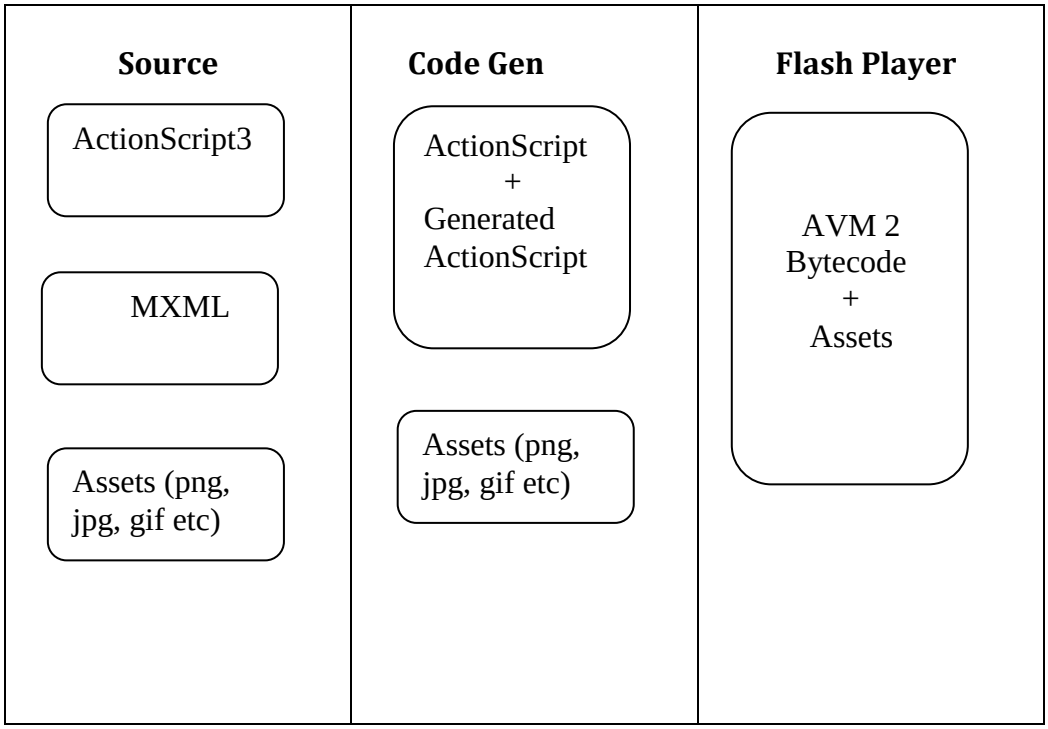
Classes

Classes are the most sophisticated and powerful use of ActionScript. ActionScript class code exists within separate documents, apart from the MXML application and component documents. ActionScript class files are text files that use the file extension *.as*.

4 (a)	Explain with diagram, how flex application works when deployed on the web.	[10]	CO3	L3



Explain with diagram, how flex application works when deployed on the web.



Flex applications deployed on the Web work differently than HTML-based applications. Every Flex application deployed on the Web utilizes Flash Player as the deployment platform. all Flex applications use the Flex framework at a minimum to compile the application. All Flex applications require at least one MXML file or ActionScript class file, and most Flex applications utilize both MXML and ActionScript files. The MXML and ActionScript class files comprise the source code files for the application. Flash Player does not know how to interpret MXML or uncompiled ActionScript class files. Instead, 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. You then deploy that one *.swf* to the server,

and when requested, it plays back in Flash Player. That means that unlike HTML-based applications, the source code files remain on the development machine, and you do not deploy them to the production server.

Asset files such as MP3s, CSS documents, and PNGs can be embedded within a *.swf*, or they can be loaded at runtime. When an asset is embedded within a *.swf*, it is not necessary to deploy the file to the production server, because it is compiled within the *.swf* file. However, since embedding assets within the *.swf* often makes for a less streamlined downloading experience and a less dynamic application, it is far more common to load such assets at runtime. That means that the asset files are not compiled into the *.swf*, and much like an HTML page, the assets are loaded into Flash Player when requested by the *.swf* at runtime. In that case, the asset files must be deployed to a valid URL when the *.swf* is deployed.
Data services are requested at runtime. That means that the services must be available at a valid URL when requested at runtime. For example, if a Flex application utilizes a web service, that web service must be accessible from the client when requested. Media servers and Flex Enterprise Services must also be accessible when used by Flex applications.

5 (a) Explain the differences between the traditional web applications and flex applications. [10] CO3 L3

Differences between Traditional and Flex Web Applications

Traditional Web Application	Flex Web Application
1. Traditional web applications have data tier, business tier, and presentation tier. The presentation tier consists of HTML, CSS, JavaScript, JSP, ASP, PHP, or similar documents. A request is made from the user's web browser for a specific presentation tier resource, and the web server runs any necessary interpreters to convert the resource to HTML and JavaScript, which is then returned to the web browser running on the client computer. Technically the HTML rendered in the browser is a client tier in a traditional web application.	Flex applications have data tier, business tier and client tier. The client tier of flex applications enables enables clients to offload computation from the server, freeing up network latency and making for responsive and highly interactive user interfaces. Data tiers generally consist of databases or similar resources. Business tiers consist of the core application business logic. As an example, a business tier may accept requests from a client or presentation tier, query the data tier, and return the requested data. Flex applications generally reside embedded within the presentation tier. In addition, Flex applications can integrate with the presentation tier to create tightly coupled client-side systems. Flex applications use Flash Player to run sophisticated client-tier portions of the application.

	2. The client tier of a traditional web application is stateless and fairly nonresponsive, it is generally not considered a full-fledged tier.	2. The Flex application client is *stateful,* which means that it can make changes to the view without having to make a request to the server. the Flex application client is responsive. For example, Flash Player can respond to user interaction such as mouse movement, mouse clicks, and keyboard presses, and it can respond to events such as notifications from the business tier when data is returned or pushed to the client. Flash Player also can respond to timer events. Flex applications can walk the user through a step-based or wizard-like interface, collect and validate data, and allow the user to update and edit previous steps, all without having to make requests to the business tier until the user wants to submit the data. All this makes **Flex clients potentially far more compelling, responsive, and engaging than traditional web applications**.			
6 (a)	Explain in detail MXML and Actionscript correlations MXML is a powerful way to simplify the creation of user interfaces. 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. MXML simplifies writing these classes because the MXML tags automatically translate into many lines of ActionScript code that handle important Flex framework tasks such as initialization, layout rules, and so forth. When you create components with IDs in an MXML document, those are really *properties* of the class formed by the document. Variable declarations within MXML scripts are treated as properties of the class, and functions are methods of the class. 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 Action-Script. ActionScript is far better suited for business logic and data models. However, MXML and ActionScript are not really so different. In fact, MXML actually gets converted to ActionScript during compilation, and the MXML structure can be understood in terms of an ActionScript class. This can be useful because it allows you to better understand how MXML works and how it relates to ActionScript. 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 a button and sets the label: <mx:Button id="button" label="Click" /> The following code is the ActionScript equivalent: var button:Button = new Button( ); button.label = "Click";	[5]	CO3	L3

This demonstrates that MXML component tags correspond to ActionScript classes. Furthermore, MXML documents themselves are essentially ActionScript classes, simply authored in a different syntax. This is an extremely important point to understand. An application document is a class that extends the mx.core.Application, and component documents are classes that extend the corresponding component class (e.g., mx.containers.VBox).

MXML simplifies writing these classes because the MXML tags automatically translate into many lines of ActionScript code that handle important Flex framework tasks such as initialization, layout rules, and so forth.
When you create components with IDs in an MXML document, those are really *properties* of the class formed by the document. For example, the following creates a new class that extends mx.core.Application and creates one property called Button of type mx.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);
                }
        }
}
```

(b)	Differentiate between Flex Framework and Flash Player Flash Player is a runtime environment for Flash and Flex applications. Flex applications run in the Flash Player as Flash applications. It can run *.swf* files, which contain bytecode that can communicate with Flash Player, instructing it to perform operations such as loading images, drawing graphics, making HTTP requests, and so on. The *.swf* files for Flex applications cannot contain anything that a standard Flash application can't contain, and therefore, both applications have the same behaviors. This is because the applications contain only the instructions, and Flash Player is what runs the instructions. **The main difference between Flash and Flex is not the content, but how the content is created.** Flex consists of a compiler that is capable of compiling MXML and ActionScript. The entire Flex framework is written in ActionScript and MXML. When an application is developed in Flex framework, the compiler will include the necessary libraries in the *.swf* files. • If the class is in a package starting with the letters *mx* (e.g., mx.controls.Button), it is part of the Flex framework. • MXML tags almost always (with few exceptions) correspond to Flex framework classes. • If the class is in a package starting with the word *flash* (e.g., flash.net.URLLoader), it is part of Flash Player.	[5]	CO3	L3

7 (a)	Explain the various types of data binding with an example.
	Data Binding is a process in which data of one object is tied to another object. Data binding requires a source property, a destination property and a triggering event which indicates when to
	copy the data from source to destination.
	Flex provides three ways to do Data Binding
	Curly brace syntax in MXML Script
	<fx:binding> tag in MXML
	BindingUtils in ActionScript
	Data Binding - Using Curly Braces in MXML
	Following example demonstrates using curly braces to specify data binding of a source to destination.
	<s:TextInput />
	<s:TextInput />
	Data Binding - Using <fx:Binding> tag in MXML
	Following example demonstrates using <fx:Binding> tag to specify data binding of a source to
	destination.
	<fx:Binding source="txtInput1.text" destination="txtInput2.text" />
	<s:TextInput />
	<s:TextInput />
	Data Binding - Using BindingUtils in ActionScript
	Following example demonstrates using BindingUtils to specify data binding of a source to destination.
	<fx:Script>
	<![CDATA[
	im port m x.binding.utils.BindingUtils;
	im port m x.events.FlexEvent;
	protected function txtInput2_preinitializeHandler(event:FlexEvent):void
	{
	BindingUtils.bindProperty(txtInput2,"text",txtInput1, "text");
	}
	]]>
	</fx:Script>
	<s:TextInput />
	<s:TextInput
	preinitialize="txtInput2_preinitializeHandler(event)"/>
		[10]	CO4	L3

8 (a)	Write a MXML code to input a text and bind two text controls so that , as the user changes the value in the text input , the value displayed in the text control also changes. <?xml version=”1.0” encoding =”utf-8”?> <mx:Application xmlns:mx=www.adobe.com/mxml layout=”absolute”> <mx:VBox> <mx:TextInput id=”txtInput1”/> < mx:TextInput id=”txtInput2”/> <mx:Binding source="txtInput1.text" destination="txtInput2.text" /> </mx:VBox> </mx:Application>	[10]	CO3	L3