



**ADOB E FLEX**  
mobile application framework

**tutorialspoint**  
SIMPLY EASY LEARNING

[www.tutorialspoint.com](http://www.tutorialspoint.com)

## About the Tutorial

---

Flex is a powerful, open source application framework that allows you to build mobile applications for iOS, Android, and BlackBerry Tablet OS devices, as well as traditional applications for browsers and desktops using the same programming model, tool, and codebase.

You can build Flex applications using Adobe Flash Builder which is an enterprise-class Eclipse based IDE. This tutorial will provide you an in-depth understanding on Flex concepts needed to get a web and mobile application up and running.

## Audience

---

This tutorial is meant for software professionals who would like to learn Flex Programming in simple and easy steps. After completing this tutorial successfully, you should be at an intermediate level of expertise from where you can take yourself to higher level of proficiency.

## Prerequisites

---

Before proceeding with this tutorial, it is advisable to have a basic understanding of other web technologies like HTML, CSS, or AJAX for better understanding.

## Copyright & Disclaimer

---

© Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com)

## Table of Contents

---

<b>About the Tutorial .....</b>	i
<b>Audience .....</b>	i
<b>Prerequisites .....</b>	i
<b>Copyright &amp; Disclaimer .....</b>	i
<b>Table of Contents .....</b>	ii
<b>1. FLEX – OVERVIEW .....</b>	1
<b>What is Flex? .....</b>	1
<b>Advantages of Flex .....</b>	1
<b>Disadvantages of Flex .....</b>	2
<b>2. FLEX – ENVIRONMENT SETUP .....</b>	3
<b>System Requirement .....</b>	3
Step 1 – Verify Java installation on your machine .....	3
Step 2 - Setup Java Development Kit (JDK) .....	4
Step 3 - Setup Adobe Flash Builder 4.5 .....	4
Step 4 – Setup Apache Tomcat .....	5
<b>3. FLEX – APPLICATIONS.....</b>	7
<b>Application Build Process .....</b>	7
<b>Application Launch Process .....</b>	7
Flex Framework Libraries.....	8
<b>Client-side Code.....</b>	8
Public Resources .....	9
HelloWorld.mxml .....	9
<b>Server-side Code.....</b>	11
<b>4. FLEX – CREATE APPLICATION .....</b>	12
<b>Step 1 – Create Project .....</b>	12
<b>Step 2 – Create External CSS File.....</b>	13
<b>Step 3 – Modify Wrapper HTML page template.....</b>	14

<b>Step 4 – Create Internal CSS file.....</b>	<b>18</b>
<b>Step 5 – Modify Entry Level Class .....</b>	<b>19</b>
<b>Step 6 – Build Application.....</b>	<b>20</b>
<b>Step 7 – Run Application.....</b>	<b>21</b>
<b>5. FLEX – DEPLOY APPLICATION .....</b>	<b>23</b>
Create WAR File .....	27
Deploy WAR file .....	27
Run Application.....	27
<b>6. FLEX – LIFE CYCLE PHASES.....</b>	<b>29</b>
<b>Life Cycle of Flex Application .....</b>	<b>29</b>
<b>Flex Life Cycle Example .....</b>	<b>30</b>
<b>7. FLEX – STYLE WITH CSS .....</b>	<b>33</b>
<b>Way # 1: Using External Style Sheet File .....</b>	<b>33</b>
<b>Way # 2: Using Styles Within Ui Container Component .....</b>	<b>33</b>
Class Level Selector .....	33
Id Level Selector .....	34
Type Level Selector .....	34
<b>Flex Style with CSS Example .....</b>	<b>35</b>
<b>8. FLEX – STYLE WITH SKIN.....</b>	<b>39</b>
<b>What is Skinning? .....</b>	<b>39</b>
Step 1 – Create a Skin .....	39
<b>Step 2 – Apply Skin .....</b>	<b>40</b>
Apply skin in MXML script (statically) .....	40
Apply skin in ActionScript (dynamically) .....	40
<b>Flex Style with Skin Example .....</b>	<b>40</b>
<b>9. FLEX – DATA BINDING .....</b>	<b>46</b>
<b>What is Data Binding? .....</b>	<b>46</b>
Data Binding – Using Curly Braces in MXML.....	46
Data Binding – Using <fx:Binding> tag in MXML.....	46
Data Binding – Using BindingUtils in ActionScript .....	46
<b>Flex Data Binding Example .....</b>	<b>47</b>

<b>10. FLEX – BASIC CONTROLS .....</b>	<b>51</b>
<b>Flex UI Elements .....</b>	<b>51</b>
<b>Flex – Event Dispatcher Class.....</b>	<b>52</b>
Introduction .....	52
What is an Event? .....	52
What is an Event Target .....	52
Class Declaration.....	53
Public Methods .....	53
Events.....	54
Methods Inherited .....	54
<b>Flex – UIComponent Class .....</b>	<b>54</b>
Introduction .....	54
Class Declaration.....	54
Public Properties .....	55
Protected Properties.....	65
Public Methods .....	66
Protected Methods .....	75
Events.....	77
Methods Inherited .....	81
<b>Basic Controls .....</b>	<b>81</b>
<b>Flex - Label Control.....</b>	<b>81</b>
Introduction .....	81
Class Declaration.....	82
Public Methods .....	82
Methods Inherited .....	82
<b>Flex Label Control Example.....</b>	<b>82</b>
<b>Flex – Text Control.....</b>	<b>84</b>
Introduction .....	84
Class Declaration.....	84
Public Methods .....	84
Methods Inherited .....	85
<b>Flex Text Control Example .....</b>	<b>85</b>
<b>Flex – Image Control.....</b>	<b>87</b>
Introduction .....	87
Class Declaration.....	88
Public Properties .....	88
Public Methods .....	89
Events.....	90
Methods Inherited .....	90
<b>Flex Image Control Example .....</b>	<b>90</b>

<b>Flex – LinkButton Control .....</b>	<b>93</b>
Class Declaration.....	93
Public Methods .....	94
Methods Inherited .....	94
 <b>Flex LinkButton Control Example .....</b>	 94
 <b>11. FLEX – FORM CONTROLS.....</b>	 97
 <b>Flex – Event Dispatcher Class .....</b>	 97
Introduction .....	97
What is an Event? .....	97
What is an Event Target.....	97
Class Declaration.....	98
Public Methods .....	98
Events.....	99
Methods Inherited .....	99
 <b>Flex - UIComponent Class .....</b>	 99
Class Declaration.....	99
Public Properties.....	100
Protected Properties.....	110
Public Methods .....	111
Protected Methods .....	120
Events.....	122
Methods Inherited .....	126
 <b>Form Controls.....</b>	 126
 <b>Flex – Button Control.....</b>	 127
Class Declaration.....	127
Public Properties.....	127
Public Methods .....	127
Methods Inherited .....	128
 <b>Flex Button Control Example .....</b>	 128
 <b>Flex – ToggleButton Control .....</b>	 130
Class Declaration.....	130
Public Methods .....	131
Methods Inherited .....	131
 <b>Flex ToggleButton Control Example .....</b>	 131
 <b>Flex - CheckBox Control .....</b>	 133
Class Declaration.....	133
Public Methods .....	134
Methods Inherited .....	134
 <b>Flex CheckBox Control Example .....</b>	 134

<b>Flex – ColorPicker Control.....</b>	<b>136</b>
Introduction .....	136
Class Declaration.....	136
Public Properties.....	136
Protected Properties.....	137
Public Methods .....	137
Events.....	138
Methods Inherited .....	138
<b>Flex ColorPicker Control Example .....</b>	<b>139</b>
<b>Flex – ComboBox Control .....</b>	<b>140</b>
Class Declaration.....	141
Public Properties.....	141
Public Methods .....	142
Methods Inherited .....	142
<b>Flex ComboBox Control Example .....</b>	<b>142</b>
<b>Flex – DateChooser Control .....</b>	<b>145</b>
Class Declaration.....	146
Public Properties.....	146
Protected Properties.....	148
Public Methods .....	148
Events.....	148
Methods Inherited .....	149
<b>Flex DateChooser Control Example.....</b>	<b>149</b>
<b>Flex – RadioButton Control.....</b>	<b>151</b>
Class Declaration.....	152
Public Properties.....	152
Public Methods .....	152
Methods Inherited .....	152
<b>Flex RadioButton Control Example .....</b>	<b>153</b>
<b>Flex – TextArea Control .....</b>	<b>156</b>
Class Declaration.....	156
Public Properties.....	156
Public Methods .....	157
Methods Inherited .....	157
<b>Flex TextArea Control Example.....</b>	<b>158</b>
<b>Flex – TextInput Control .....</b>	<b>160</b>
Class Declaration.....	160
Public Properties.....	160
Public Methods .....	161
Methods Inherited .....	161

<b>Flex TextInput Control Example .....</b>	<b>161</b>
<b>Flex – DropDownList Control .....</b>	<b>163</b>
Class Declaration.....	163
Public Properties.....	163
Public Methods .....	164
Methods Inherited .....	164
<b>Flex DropDownList Control Example.....</b>	<b>164</b>
<b>Flex – NumericStepper Control.....</b>	<b>167</b>
Class Declaration.....	167
Public Properties.....	168
Public Methods .....	168
Methods Inherited .....	168
<b>Flex DropDownList Control Example.....</b>	<b>169</b>
<b>12. FLEX – COMPLEX CONTROLS .....</b>	<b>172</b>
<b>Flex – Event Dispatcher Class .....</b>	<b>172</b>
Introduction .....	172
What is an Event? .....	172
What is an Event Target.....	172
Class Declaration.....	173
Public Methods .....	173
Events.....	174
Methods Inherited .....	174
<b>Flex - UIComponent Class .....</b>	<b>174</b>
Introduction .....	174
Class Declaration.....	174
Public Properties.....	175
Protected Properties.....	185
Public Methods .....	187
Protected Methods .....	195
Events.....	197
Methods Inherited .....	201
<b>Complex Controls .....</b>	<b>201</b>
<b>Flex - DataGrid Control .....</b>	<b>202</b>
Class Declaration.....	202
Public Properties.....	203
Public Methods .....	206
Protected Methods .....	208
Events.....	209
Methods Inherited .....	210
<b>Flex DataGrid Control Example.....</b>	<b>211</b>

<b>Flex – AdvancedDataGrid Control.....</b>	<b>213</b>
Class Declaration.....	213
Public Properties.....	214
Protected Properties.....	215
Public Methods .....	216
Protected Methods .....	217
Events.....	218
Methods Inherited .....	219
<b>Flex AdvancedDataGrid Control Example .....</b>	<b>219</b>
<b>Flex – Menu Control .....</b>	<b>222</b>
Class Declaration.....	222
Public Properties.....	222
Public Methods .....	223
Protected Methods .....	223
Events.....	224
Methods Inherited .....	224
<b>Flex Menu Control Example.....</b>	<b>225</b>
<b>Flex – ProgressBar Control.....</b>	<b>228</b>
Class Declaration.....	228
Public Properties.....	228
Public Methods .....	230
Events.....	230
Methods Inherited .....	230
Flex ProgressBar Control Example .....	231
<b>Flex – RichTextEditor Control.....</b>	<b>233</b>
Class Declaration.....	233
Public Properties.....	233
Public Methods .....	235
Events.....	235
Methods Inherited .....	235
<b>Flex RichTextEditor Control Example .....</b>	<b>236</b>
<b>Flex – TileList Control.....</b>	<b>238</b>
Class Declaration.....	238
Public Methods .....	238
Methods Inherited .....	239
<b>Flex TileList Control Example .....</b>	<b>239</b>
<b>Flex – Tree Control.....</b>	<b>242</b>
Class Declaration.....	242
Public Properties.....	242
Public Methods .....	243
Protected Methods .....	244
Events.....	244

Methods Inherited .....	245
<b>Flex Tree Control Example .....</b>	<b>245</b>
<b>Flex – VideoPlayer Control.....</b>	<b>248</b>
Class Declaration.....	248
Public Properties.....	248
Public Methods .....	250
Protected Methods .....	251
Events.....	251
Methods Inherited .....	252
<b>Flex VideoPlayer Control Example .....</b>	<b>252</b>
<b>Flex – Accordion Control.....</b>	<b>254</b>
Class Declaration.....	254
Public Properties.....	254
Protected Properties.....	255
Public Methods .....	255
Events.....	256
Methods Inherited .....	256
<b>Flex Accordion Control Example .....</b>	<b>256</b>
<b>Flex – TabNavigator Control .....</b>	<b>259</b>
Class Declaration.....	259
Protected Properties.....	259
Public Methods .....	260
Protected Methods .....	260
Methods Inherited .....	260
<b>Flex TabNavigator Control Example .....</b>	<b>261</b>
<b>Flex – ToggleButtonBar Control .....</b>	<b>263</b>
Class Declaration.....	263
Public Properties.....	264
Public Methods .....	264
Methods Inherited .....	264
<b>Flex ToggleButtonBar Control Example .....</b>	<b>264</b>
<b>13. FLEX – LAYOUT PANELS.....</b>	<b>268</b>
<b>Flex – Event Dispatcher Class.....</b>	<b>268</b>
Introduction .....	268
What is an Event? .....	268
What is an Event Target .....	268
Class Declaration.....	269
Public Methods .....	269
Events.....	270
Methods Inherited .....	270

<b>Flex - UIComponent Class .....</b>	<b>270</b>
Class Declaration.....	270
Public Properties.....	271
Protected Properties.....	281
Public Methods .....	283
Protected Methods .....	291
Events.....	293
Methods Inherited .....	297
<b>Layout Panels .....</b>	<b>297</b>
<b>Flex – BorderContainer .....</b>	<b>298</b>
Class Declaration.....	298
Public Properties.....	298
Public Methods .....	299
Methods Inherited .....	299
<b>Flex BorderContainer Example .....</b>	<b>299</b>
<b>Flex – Form .....</b>	<b>302</b>
Class Declaration.....	302
Public Properties.....	302
Public Methods .....	303
Methods Inherited .....	303
<b>Flex Form Example .....</b>	<b>303</b>
<b>Flex – VGroup .....</b>	<b>305</b>
Introduction .....	305
Class Declaration.....	305
Public Properties.....	306
Public Methods .....	307
Methods Inherited .....	307
<b>Flex VGroup Example .....</b>	<b>308</b>
<b>Flex – HGroup .....</b>	<b>311</b>
Class Declaration.....	311
Public Properties.....	311
Public Methods .....	313
Methods Inherited .....	313
<b>Flex HGroup Example .....</b>	<b>313</b>
<b>Flex – Panel .....</b>	<b>316</b>
Class Declaration.....	317
Public Properties.....	317
Public Methods .....	317
Methods Inherited .....	317
<b>Flex Panel Example .....</b>	<b>318</b>

<b>Flex – SkinnableContainer .....</b>	<b>320</b>
Class Declaration.....	320
Public Properties.....	321
Public Methods .....	321
Protected Methods.....	322
Events.....	323
Methods Inherited .....	323
<b>Flex SkinnableContainer Example.....</b>	<b>324</b>
<b>Flex – TabBar .....</b>	<b>327</b>
Class Declaration.....	327
Public Methods .....	327
Methods Inherited .....	328
<b>Flex TabBar Example .....</b>	<b>328</b>
<b>Flex – TitleWindow.....</b>	<b>330</b>
Class Declaration.....	330
Public Methods .....	330
Events.....	331
Methods Inherited .....	331
<b>Flex TitleWindow Example .....</b>	<b>332</b>
Step 1 – Create a Project.....	332
Step 2 – Create a custom Title WIndow component .....	332
Step 3 – Modify HelloWorld.mxml.....	333
Step 4 – Complie and Run application .....	334
<b>14. FLEX – VISUAL EFFECTS .....</b>	<b>336</b>
<b>Flex - Effect.....</b>	<b>336</b>
Class Declaration.....	336
Public Properties.....	336
Protected Properties.....	338
Public Methods .....	338
Protected Methods .....	340
Events.....	340
Methods Inherited .....	341
<b>Basic Effects.....</b>	<b>341</b>
<b>Flex – Fade Effect.....</b>	<b>342</b>
Class Declaration.....	342
Public Properties.....	342
Public Methods .....	342
Methods Inherited .....	343
<b>Flex Fade Effect Example .....</b>	<b>343</b>
<b>Flex – WipeLeft Effect .....</b>	<b>345</b>

Class Declaration.....	345
Public Methods .....	346
Methods Inherited .....	346
<b>Flex WipeLeft Effect Example .....</b>	<b>346</b>
<b>Flex – WipeRight Effect.....</b>	<b>348</b>
Class Declaration.....	348
Public Methods .....	349
Methods Inherited .....	349
<b>Flex WipeRight Effect Example .....</b>	<b>349</b>
<b>Flex – Move3D Effect .....</b>	<b>351</b>
Class Declaration.....	351
Public Properties.....	351
Public Methods .....	352
Methods Inherited .....	352
<b>Flex Move3D Effect Example .....</b>	<b>353</b>
<b>Flex – Scale3D Effect.....</b>	<b>355</b>
Class Declaration.....	355
Public Properties.....	356
Public Methods .....	356
Methods Inherited .....	357
<b>Flex Scale3D Effect Example .....</b>	<b>357</b>
<b>Flex – Rotate3D Effect .....</b>	<b>360</b>
Class Declaration.....	360
Public Properties.....	360
Public Methods .....	361
Methods Inherited .....	361
<b>Flex Rotate3D Effect Example .....</b>	<b>361</b>
<b>Flex – AnimateProperties Effect .....</b>	<b>364</b>
Class Declaration.....	364
Public Properties.....	364
Public Methods .....	365
Events.....	365
Methods Inherited .....	366
<b>Flex Animate Effect Example .....</b>	<b>366</b>
<b>15. FLEX – EVENT HANDLING .....</b>	<b>369</b>
<b>Event Flow Phases .....</b>	<b>369</b>

<b>16. FLEX – CUSTOM CONTROLS .....</b>	<b>374</b>
<b>Using ActionScript .....</b>	<b>374</b>
<b>Using MXML .....</b>	<b>375</b>
<b>17. FLEX – RPC SERVICES.....</b>	<b>380</b>
Items.xml .....	380
HTTPService Declaration.....	380
RPC Call .....	381
<b>RPC Service Call Example .....</b>	<b>381</b>
<b>18. FLEX – FLEXUNIT INTEGRATION .....</b>	<b>384</b>
<b>Create a Test Case Class.....</b>	<b>384</b>
<b>Flex Unit Integration Example .....</b>	<b>385</b>
<b>Running Test cases .....</b>	<b>387</b>
<b>19. FLEX – DEBUG APPLICATION .....</b>	<b>389</b>
<b>Debugging Example .....</b>	<b>389</b>
Step 1 - Place Breakpoints .....	390
Step 2 - Debug Application .....	391
<b>20. FLEX – INTERNATIONALIZATION.....</b>	<b>398</b>
<b>Workflow of internationalizing a Flex Application .....</b>	<b>398</b>
Step 1 – Create folder structure .....	398
Step 2 – Create properties files.....	398
Step 3 – Specify Compiler options .....	399
<b>Internalization Example .....</b>	<b>399</b>
<b>21. FLEX – PRINTING SUPPORT .....</b>	<b>404</b>
<b>Prepare and Send a Print Job.....</b>	<b>404</b>
<b>Printing Example .....</b>	<b>404</b>

# 1. Flex – Overview

## What is Flex?

Flex is a powerful, open source application framework that allows you to build traditional applications for browser, mobile and desktop using the same programming model, tool, and codebase.

Flex provides FLEX SDK consisting of the Flex class library (ActionScript classes), the Flex compilers, the debugger, the MXML and ActionScript programming languages, and other utilities to build expressive and interactive rich internet applications (RIA)

Flex takes care of the user interface (UI) or the client-side functionality of a web application. Server-side functionality is dependent on server-side components written in a traditional scripting language (Java/ PHP etc.)

A Flex based application actually delivered as a SWF file and it closely resembles the HTML / JavaScript portion of a traditional web application.

Flex application is deployed as SWF file(s) plus an HTML wrapper, the CSS file(s) and any server-side script files (i.e. Java, .CFM, .PHP, etc.) to the server. Like traditional web applications.

These resources are delivered from a server to the client's browser using the customary HTTP request / response fashion and Flash Player which runs the application in a browser.

## Advantages of Flex

- Flex applications are usually Flash Player based which can access device capabilities like GPS, camera, local database, graphics accelerometer.
- Flex applications can run on Android, BlackBerry Tablet OS, and iOS devices.
- Flex applications can run on Browsers as well as on Desktop.
- Flex applications are platform independent. UI can be native to platform or can be made same on each platform.
- Flex applications can interact with server with all major server side technologies like Java, Spring, Hibernate, PHP, Ruby, .NET, Adobe ColdFusion, and SAP using industry standards such as REST, SOAP, JSON, JMS, and AMF.
- Flex Applications assures rich user experience through intuitive interaction with the application and presenting information in a visually richer interface.
- Flex application is a single page application where states can transition from one state to other state without having to fetch a new page from the server or to refresh the browser.

- Flex application reduces the load on the server to great extent because it is only required to return the application once, rather than a new page every time when the user changes views.

## Disadvantages of Flex

---

- Flex applications are single threaded applications but Flex provides an asynchronous programming model to mitigate this concern.
- Flex is ActionScript and XML based. Learning of these two is a must to work in Flex.

## 2. Flex – Environment Setup

This tutorial will guide you on how to prepare a development environment to start your work with Adobe Flex Framework. This tutorial will also teach you how to setup JDK and Adobe Flash Builder on your machine before you setup Flex Framework.

### System Requirement

FLEX requires JDK 1.4 or higher, so the very first requirement is to have JDK installed in your machine.

JDK	1.4 Or above.
Memory	No minimum requirement.
Disk Space	No minimum requirement.
Operating System	No minimum requirement.

Follow the given steps to setup your environment to start with Flex application development.

### Step 1 – Verify Java installation on your machine

Now open the console and execute the following **java** command.

OS	Task	Command
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	\$ java -version
Mac	Open Terminal	machine:~ joseph\$ java -version

Let's verify the output for all the operating systems:

OS	Generated Output
Windows	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Linux	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Mac	java version "1.6.0_21"

	Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM)64-Bit Server VM (build 17.0-b17, mixed mode, sharing)
--	--

## Step 2 - Setup Java Development Kit (JDK)

If you do not have Java installed, then you can install the Java Software Development Kit (SDK) from Oracle's Java site: [Java SE Downloads](#). You will find instructions for installing JDK in downloaded files, then follow the given instructions to install and configure the setup. Finally set PATH and JAVA\_HOME environment variables to refer to the directory that contains java and javac, typically java\_install\_dir/bin and java\_install\_dir respectively.

Set the **JAVA\_HOME** environment variable to point to the base directory location where Java is installed on your machine. For example:

OS	Output
Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.6.0_21
Linux	export JAVA_HOME=/usr/local/java-current
Mac	export JAVA_HOME=/Library/Java/Home

Append the Java compiler location to the System Path.

OS	Output
Windows	Append the string ;%JAVA_HOME%\bin to the end of the system variable, Path.
Linux	export PATH=\$PATH:\$JAVA_HOME/bin/
Mac	not required

## Step 3 - Setup Adobe Flash Builder 4.5

All the examples in this tutorial has been written using Adobe Flash Builder 4.5 Professional IDE Trial Version. Hence, suggest you to have latest version of Adobe Flash Builder installed on your machine. Also, check compatibility of operating system.

To install Adobe Flash Builder IDE, download the latest Adobe Flash Builder binaries from <http://www.adobe.com/in/products/flash-builder.html>. Once you downloaded the installation, unpack the binary distribution into a convenient location. For example, in C:\flash-builder on windows, or /usr/local/flash-builder on Linux/Unix and finally set PATH variable appropriately.

Flash Builder will start, when you execute the following commands on windows machine, or when you can simply double click on FlashBuilder.exe

```
%C:\flash-builder\FlashBuilder.exe
```

Flash Builder can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine:

```
$/usr/local/flash-builder/FlashBuilder
```

Adobe Flash Builder Trial Version can be used for 60 days. Just accept the terms and conditions, and skip the initial registration steps to continue with the IDE. For our understanding, we're using the trial version for teaching purpose.

After a successful startup, if everything is fine then it should display the following result:

Adobe Flash Builder comes pre-configured with FLEX SDKs. We're using FLEX SDK 4.5 in our examples which come bundled with Adobe Flash Builder 4.5.

#### **Step 4 – Setup Apache Tomcat**

You can download the latest version of Tomcat from <http://tomcat.apache.org/>. Once, you downloaded the installation, unpack the binary distribution into a convenient location. For example, in C:\apache-tomcat-6.0.33 on windows, or /usr/local/apache-tomcat-6.0.33 on Linux/Unix and set CATALINA\_HOME environment variable pointing to the installation locations.

Tomcat can be started by executing the following commands on windows machine, or you can simply double click on startup.bat

```
%CATALINA_HOME%\bin\startup.bat
or
C:\apache-tomcat-6.0.33\bin\startup.bat
```

Tomcat can be started by executing the following commands on UNIX (Solaris, Linux, etc.) machine:

```
$CATALINA_HOME/bin/startup.sh
Or
/usr/local/apache-tomcat-6.0.33/bin/startup.sh
```

After a successful startup, the default web applications included with Tomcat will be available by visiting <http://localhost:8080/>. If everything is fine, then it should display the following result:

Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat web site: <http://tomcat.apache.org>

Tomcat can be stopped by executing the following commands on windows machine:

```
%CATALINA_HOME%\bin\shutdown
or
```

```
C:\apache-tomcat-5.5.29\bin\shutdown
```

Tomcat can be stopped by executing the following commands on UNIX (Solaris, Linux, etc.) machine:

```
$CATALINA_HOME/bin/shutdown.sh  
Or  
/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh
```

### 3. Flex – Applications

Before we start creating actual “*HelloWorld*” application using Flash Builder, let us see what the actual parts of a Flex application are:

A Flex application consists of the following four important parts, out of which last part is optional but first three parts are mandatory.

- Flex Framework Libraries
- Client-side code
- Public Resources (HTML/JS/CSS)
- Server-side code

Sample locations of different parts of a typical Flex application like **HelloWord** will be as shown below:

Name	Location
Project root	HelloWorld/
Flex Framework Libraries	Build Path
Public resources	html-template
Client-side code	table table-bordered/com/tutorialspoint/client
Server-side code	table table-bordered/com/tutorialspoint/server

#### Application Build Process

To start with, Flex application requires Flex framework libraries. Later, Flash Builder automatically adds the libraries to build path.

When we build our code using Flash builder, Flash builder will do the following tasks:

- Compiles the source code to *HelloWorld.swf* file.
- Compiles a *HelloWorld.html* (a wrapper file for *swf* file) from a file *index.template.html* stored in *html-template* folder
- Copies *HelloWorld.swf* and *HelloWorld.html* files in target folder, *bin-debug*.
- Copies *swfobject.js*, a JavaScript code responsible to load *swf* file dynamically in *HelloWorld.html* in target folder, *bin-debug*
- Copies framework libraries in form of *swf* file named *frameworks\_xxx.swf* in target folder, *bin-debug*
- Copies other flex modules (*.swf* files such as *sparkskins\_xxx.swf*, *textLayout\_xxx.swf*) in target folder.

#### Application Launch Process

- Open the *HelloWorld.html* file available in *\HelloWorld\bin-debug* folder in any web-browser.
- *HelloWorld.swf* will load automatically and application will start running.

## Flex Framework Libraries

Following is the brief detail about few important framework libraries. Please note that, Flex libraries are denoted using .swc notation

S.No	Nodes & Description
1	<b>playerglobal.swc</b> This library is specific to FlashPlayer installed on your machine and contains native methods supported by flash player.
2	<b>textlayout.swc</b> This library supports the text layout related features.
3	<b>framework.swc</b> This is the flex framework library contains the core features of Flex.
4	<b>mx.swc</b> This library stores the definitions of mx UI controls.
5	<b>charts.swc</b> This library supports the charting controls.
6	<b>spark.swc</b> This library stores the definitions of spark UI controls.
7	<b>sparkskins.swc</b> This library supports the skinning of spark UI controls.

## Client-side Code

Flex application code can be written in **MXML** as well as **ActionScript**.

S.No.	Type & Description
1	<b>MXML</b> MXML is an XML markup language that we'll use to lay out user interface components. MXML is compiled into ActionScript during build process.
2	<b>ActionScript</b> ActionScript is an object-oriented procedural programming language and is based on the ECMAScript (ECMA-262) edition 4 draft language specification.

In Flex, we can mix ActionScript and MXML, to do the following:

- Layout user interface components using MXML tags
- Use MXML to declaratively define nonvisual aspects of an application, such as access to data sources on the server
- Use MXML to create data bindings between user interface components and data sources on the server.

- Use ActionScript to define event listeners inside MXML event attributes.
- Add script blocks using the <mx:Script> tag.
- Include external ActionScript files.
- Import ActionScript classes.
- Create ActionScript components.

## Public Resources

These are help files referenced by Flex application, such as Host HTML page, CSS or images located under html-template folder. It contains following files:

S.N.	File Name & Description
1	<b>index.template.html</b> Host HTML page, with place holders. Flash Builder uses this template to build actual page HelloWorld.html with HelloWorld.swf file.
2	<b>playerProductInstall.swf</b> This is a flash utility to install Flash Player in express mode.
3	<b>swfobject.js</b> This is the JavaScript responsible to check version of flash player installed and to load HelloWorld.swf in HelloWorld.html page.
4	<b>html-template/history</b> This folder contains resources for history management of the application.

## HelloWorld.mxml

This is the actual MXML/AS (ActionScript) code written implementing the business logic of the application and that the Flex compiler translates into SWF file which will be executed by flash player in the browser.

A sample HelloWorld Entry class will be as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%"
    minWidth="500" minHeight="500"
    initialize="application_initializeHandler(event)">

    <fx:Script>
        <![CDATA[
```

```

import mx.controls.Alert;
import mx.events.FlexEvent;
protected function btnClickMe_clickHandler(event:MouseEvent):void
{
    Alert.show("Hello World!");
}

protected function application_initializeHandler(event:FlexEvent):void
{
    lblHeader.text = "My Hello World Application";
}

]]>
</fx:Script>
<s:VGroup horizontalAlign="center" width="100%" height="100%" paddingTop="100" gap="50">
    <s:Label id="lblHeader" fontSize="40" color="0x777777"/>
    <s:Button label="Click Me!" id="btnClickMe"
        click="btnClickMe_clickHandler(event)" />
</s:VGroup>
</s:Application>

```

Following table gives the description of all the tags used in the above code script.

S.N.	Node & Description
1	<b>Application</b> Defines the Application container that is always the root tag of a Flex application.
2	<b>Script</b> Contains the business logic in ActionScript language.
3	<b>VGroup</b> Defines a Vertical Grouping Container which can contain Flex UI controls in vertical fashion.
4	<b>Label</b> Represents a Label control, a very simple user interface component that displays text.
5	<b>Button</b> Represents a Button control, which can be clicked to do some action.

## Server-side Code

---

This is the server side part of your application and it's very much optional. If you are not doing any backend processing within your application, then you do not need this part but if there is some processing required at backend and your client-side application interacts with the server, then you will have to develop these components.

In the next chapter, we will use all the above-mentioned concepts to create a **HelloWorld** application using Flash Builder.

## 4. Flex – Create Application

We'll use Flash Builder 4.5 to create Flex Applications. Let's start with a simple **HelloWorld** application.

### Step 1 – Create Project

The first step is to create a simple Flex Project using Flash Builder IDE. Launch project wizard using the option **File > New > Flex Project**. Now name your project as *HelloWorld* using the wizard window as follows:

Select Application Type **Web (runs in Adobe Flash Player)**. However, if this is not selected, then leave other default values as such and click Finish Button. Once your project is created successfully, then you will have the following content in your Project Explorer:

Here is a brief description of all the important folders:

Folder	Location
table table-bordered	Source code (mxml / as classes) files. We've created com/tutorialspoint/client folder structure containing the client-side specific java classes responsible for client UI display.
bin-debug	This is the output part, it represents the actual deployable web application. History folder contains support files for history management of Flex application. framework_xxx.swf, flex framework files should be used by flex application. HelloWorld.html, wrapper/host HTML File for flex application. HelloWorld.swf, our flex based application. playerProductInstall.swf, flash player express installer. spark_xxx.swf, library for spark component support. swfobject.js, JavaScript responsible to load HelloWorld.swf in HelloWorld.html. It checks flash player version and passes initialization parameter to HelloWorld.swf file. textLayout_xxx.swf, library for text component support.
	This represents the configurable web application. Flash Builder compiles files from html-template to bin-debug folder. History folder contains support files for history management of Flex application.

html-template	<p>index.template.html, wrapper/host HTML File for flex application having place holders for Flash Builder specific configuration. Gets compiled to HelloWorld.html in bin-debug folder during build.</p> <p>playerProductInstall.swf, flash player express installer gets copied to bin-debug folder during build.</p> <p>swfobject.js, JavaScript responsible to load HelloWorld.swf in HelloWorld.html. It checks flash player version and passes initialization parameter to HelloWorld.swf file gets copied to bin-debug folder during build.</p>
---------------	--

## Step 2 – Create External CSS File

---

Create a CSS file **styles.css** for Wrapper HTML page in **html-template** folder.

```

html, body {
    height:100%;
}

body {
    margin:0;
    padding:0;
    overflow:auto;
    text-align:center;
}

object:focus {
    outline:none;
}

#flashContent {
    display:none;
}

.pluginHeader {
    font-family:Arial, Helvetica, sans-serif;
    font-size:14px;
    color:#9b1204;
    text-decoration:none;
    font-weight:bold;
}

.pluginInstallText {

```

```

font-family:Arial, Helvetica, sans-serif;
font-size:12px;
color:#000000;
line-height:18px;
font-style:normal;
}

.pluginText {
    font-family:Arial, Helvetica, sans-serif;
    font-size:12px;
    color:#000000;
    line-height:18px;
    font-style:normal;
}

```

## Step 3 – Modify Wrapper HTML page template

Modify Wrapper HTML page template **index.template.html** in **html-template** folder. Flash Builder will create a default Wrapper HTML page template *html-template/index.template.html*, which will be compiled to HelloWorld.html.

This file contains placeholders which Flash Builder replaces during the compilation process. For example, flash player version, application name, etc.

Let us modify this file to display custom messages in case flash plugin is not installed.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
<title>${title}</title>
<meta name="google" value="notranslate" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link rel="stylesheet" href="styles.css" type="text/css"></link>
<link rel="stylesheet" type="text/css" href="history/history.css" />
<script type="text/javascript" table table-
bordered="history/history.js"></script>
<script type="text/javascript" table table-bordered="swfobject.js"></script>
<script type="text/javascript">
    // For version detection, set to min. required Flash Player version,
    // or 0 (or 0.0.0), for no version detection.

```

```

var swfVersionStr = "${version_major}.${version_minor}.${version_revision}";
// To use express install, set to playerProductInstall.swf,
//otherwise the empty string.
var xiSwfUrlStr = "${expressInstallSwf}";
var flashvars = {};
var params = {};
params.quality = "high";
params.bgcolor = "${bgcolor}";
params.allowscriptaccess = "sameDomain";
params.allowfullscreen = "true";
var attributes = {};
attributes.id = "${application}";
attributes.name = "${application}";
attributes.align = "middle";
swfobject.embedSWF(
"${swf}.swf", "flashContent",
"${width}", "${height}",
swfVersionStr, xiSwfUrlStr,
flashvars, params, attributes);
// JavaScript enabled so display the flashContent div in case
//it is not replaced with a swf object.
swfobject.createCSS("#flashContent", "display:block;text-align:left;");
</script>
</head>
<body>
<div id="flashContent">
<p style="margin:100px;">
<table width="700" cellpadding="10" cellspacing="2" border="0">
<tr><td class="pluginHeader">Flash Player Required</td></tr>

<tr><td class="pluginText">The Adobe Flash Player version
10.2.0 or greater is required.</td></tr>
<tr><td class = "pluginInstallText" align="left">
<table border="0" width="100%">
<tr class = "pluginInstallText" >
<td>Click here to download and install Adobe Flash Player:</td>

```

```

<td> </td>
<td align="right">      <script type="text/javascript">
var pageHost
=((document.location.protocol == "https:") ? "https://" : "http://");
document.write("<a target='_blank'"
+" href='http://get.adobe.com/flashplayer/'><" 
+"img style='border-style: none' table table-bordered='"
+pageHost
+"www.adobe.com/images/shared/download_buttons/get_flash_player.gif'"
+" alt='Get Adobe Flash player' /></a>" );
</script>
</td>
</tr>
</table>
</td>
</tr>
</table>
</p>
</div>
<noscript>
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
width="${width}" height="${height}" id="${application}">
<param name="movie" value="${swf}.swf" />
<param name="quality" value="high" />
<param name="bgcolor" value="${bgcolor}" />
<param name="allowScriptAccess" value="sameDomain" />
<param name="allowFullScreen" value="true" />
<!--[if !IE]-->
<object type="application/x-shockwave-flash" data="${swf}.swf"
width="${width}" height="${height}">
<param name="quality" value="high" />
<param name="bgcolor" value="${bgcolor}" />
<param name="allowScriptAccess" value="sameDomain" />
<param name="allowFullScreen" value="true" />
<!--<![endif]-->
<!--[if gte IE 6]-->
<p>

```

```

<p style="margin:100px;">
<table width="700" cellpadding="10" cellspacing="2" border="0">
<tr><td class="pluginHeader">Flash Player Required</td></tr>

<tr><td class="pluginText">The Adobe Flash Player version
10.2.0 or greater is required.</td></tr>
<tr><td class = "pluginInstallText" align="left">
<table border="0" width="100%">
<tr class = "pluginInstallText" >

<td>Click here to download and install Adobe Flash Player:</td>

<td> </td>
<td align="right"> <script type="text/javascript">
var pageHost
= ((document.location.protocol == "https:") ? "https://" : "http://");
document.write("<a target='_blank'"
+" href='http://get.adobe.com/flashplayer/'><"
+"img style='border-style: none' table table-bordered='"
+pageHost
+"www.adobe.com/images/shared/download_buttons/get_flash_player.gif'"
+" alt='Get Adobe Flash player' /></a>" );
</script>
</td>
</tr>
</table>
</td>
</tr>
</table>
</p>
</p>
<!--<![endif]-->
<p style="margin:100px;">
<table width="700" cellpadding="10" cellspacing="2" border="0">
<tr><td class="pluginHeader">Flash Player Required</td></tr>

<tr><td class="pluginText">The Adobe Flash Player version
10.2.0 or greater is required.</td></tr>

```

```

<tr><td class = "pluginInstallText" align="left">
    <table border="0" width="100%">
        <tr class = "pluginInstallText" >
            <td>Click here to download and install Adobe Flash Player:</td>
            <td> </td>
            <td align="right">      <script type="text/javascript">
                var pageHost
                = ((document.location.protocol == "https:") ? "https://":
                "http://");
                document.write("<a target='_blank' "
                +" href='http://get.adobe.com/flashplayer/'><" +
                +"img style='border-style: none' table table-bordered='"
                +pageHost
                +"www.adobe.com/images/shared/download_buttons/get_flash_player.gif'" +
                +" alt='Get Adobe Flash player' /></a>" );
            </script>
        </td>
    </tr>
    </table>
</td>
</tr>
</table>
</p>
<!--[if !IE]>-->
</object>
<!--<![endif]-->
</object>
</noscript>
</body>
</html>

```

## Step 4 – Create Internal CSS file

Create a CSS file **Style.css** for **HelloWorld.mxml** in **table-table-bordered/com/tutorialspoint** folder. Flex provides similar css styles for its UI Controls as there are css styles for HTML UI controls.

```

/* CSS file */

@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/mx";


.heading
{
    fontFamily: Arial, Helvetica, sans-serif;
    fontSize: 17px;
    color: #9b1204;
    textDecoration:none;
    fontWeight:normal;
}

.button {
    fontWeight: bold;
}

.container {
    borderRadius :10;
    horizontalCenter :0;
    borderColor: #777777;
    verticalCenter:0;
    backgroundColor: #efefef;
}

```

## Step 5 – Modify Entry Level Class

Flash Builder will create a default mxml file *table-table-bordered/com.tutorialspoint/HelloWorld.mxml*, which is having root tag `<application>` container for the application. Let us modify this file to display "Hello,World!":

```

<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%"
    minWidth="500" minHeight="500"
    initialize="application_initializeHandler(event)">

```

```

<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<fx:Script>
<![CDATA[
import mx.controls.Alert;
import mx.events.FlexEvent;
protected function btnClickMe_clickHandler(event:MouseEvent):void
{
    Alert.show("Hello World!");
}

protected function application_initializeHandler(event:FlexEvent):void
{
    lblHeader.text = "My Hello World Application";
}
]]>
</fx:Script>
<s:BorderContainer width="500" height="500" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="50" horizontalAlign="center"
verticalAlign="middle">
<s:Label id="lblHeader" fontSize="40" color="0x777777"
styleName="heading"/>
<s:Button label="Click Me!" id="btnClickMe"
click="btnClickMe_clickHandler(event)" styleName="button" />
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

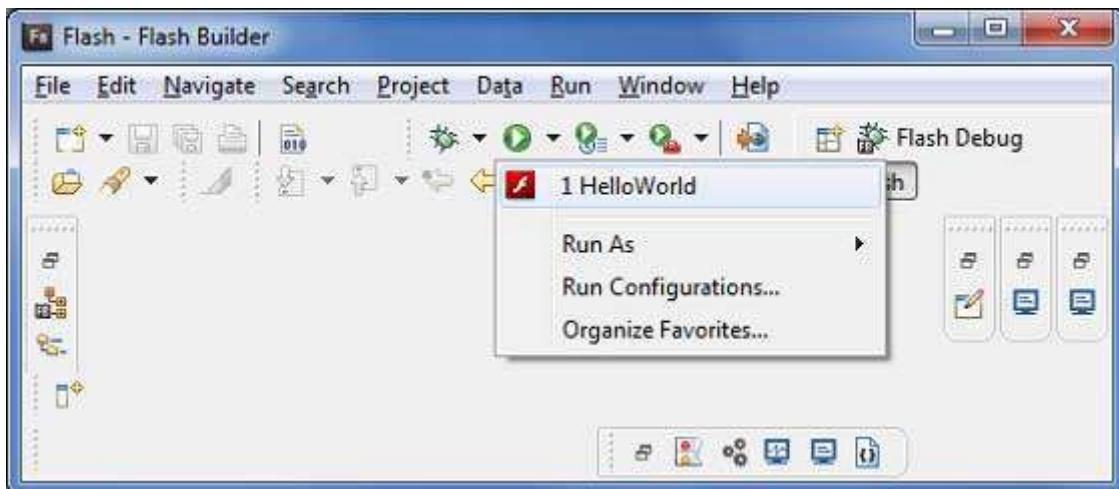
You can create more mxml or actionscript files in the same source directory to define either new applications or to define helper routines.

## Step 6 – Build Application

Flash Builder has **Build Automatically** by default checked. Just check the **Problems** View if there is any error. Once you are done with the changes, you will not see any errors.

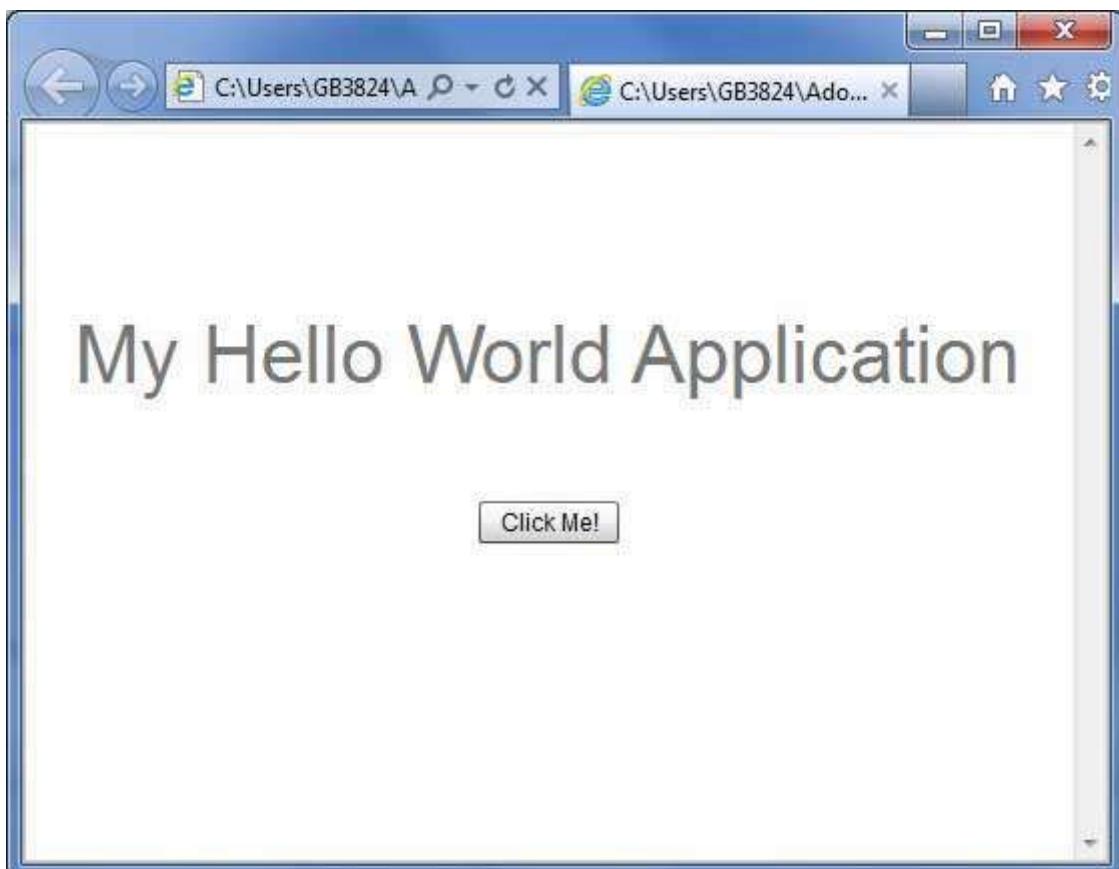
## Step 7 – Run Application

Now click on  Run application menu and select **HelloWorld** application to run the application.



If everything is fine, you must see browser pop up, application up, and running. If everything is fine with your application, it will produce the following result: [\[Try it online\]](#)

Because, you are running your application in flash player, then it will need to install Flash Player plugin for your browser. Simply follow the onscreen instructions to install the plugin. If you already have Flash Player plugin set for your browser, then you should be able to see the following output:



Congratulations! You have implemented your first application using **Flex**.

## 5. Flex – Deploy Application

This tutorial will explain you how to create an application **war** file and how to deploy that in Apache Tomcat Web server root.

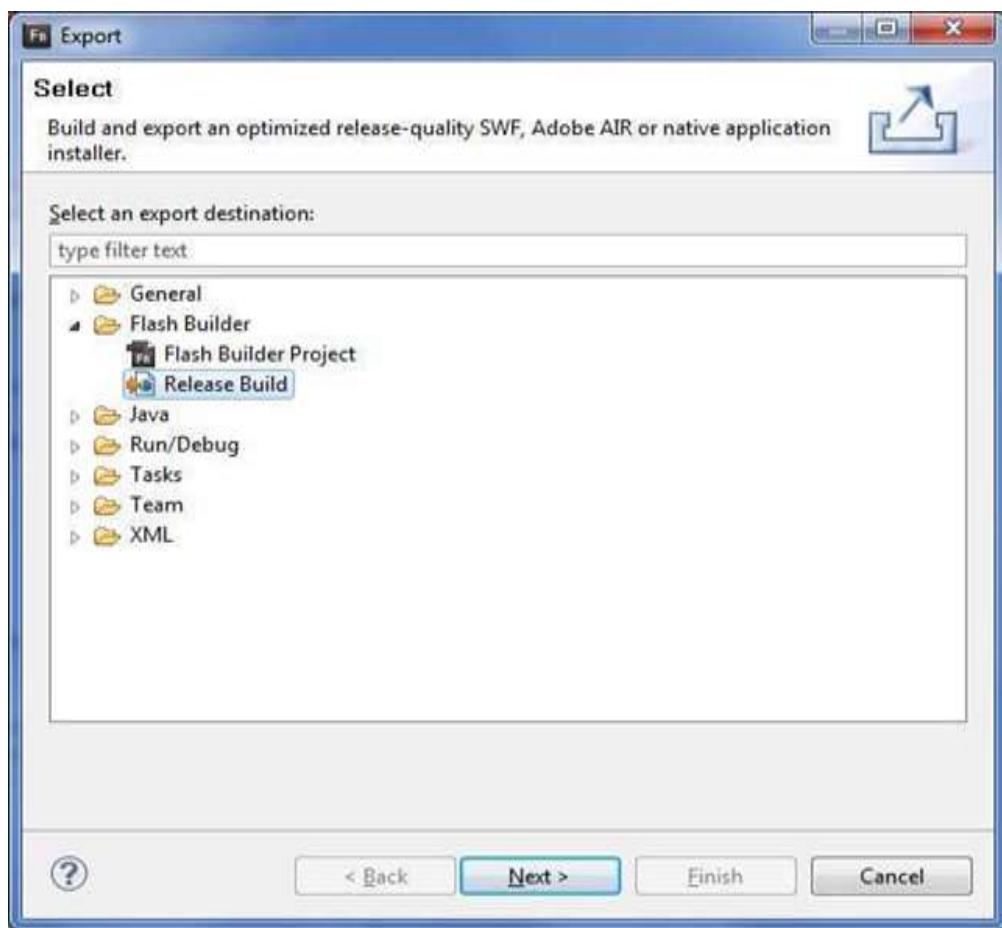
If you understood this simple example then you will also be able to deploy a complex Flex application following the same steps.

Let us follow the following steps to create a Flex application:

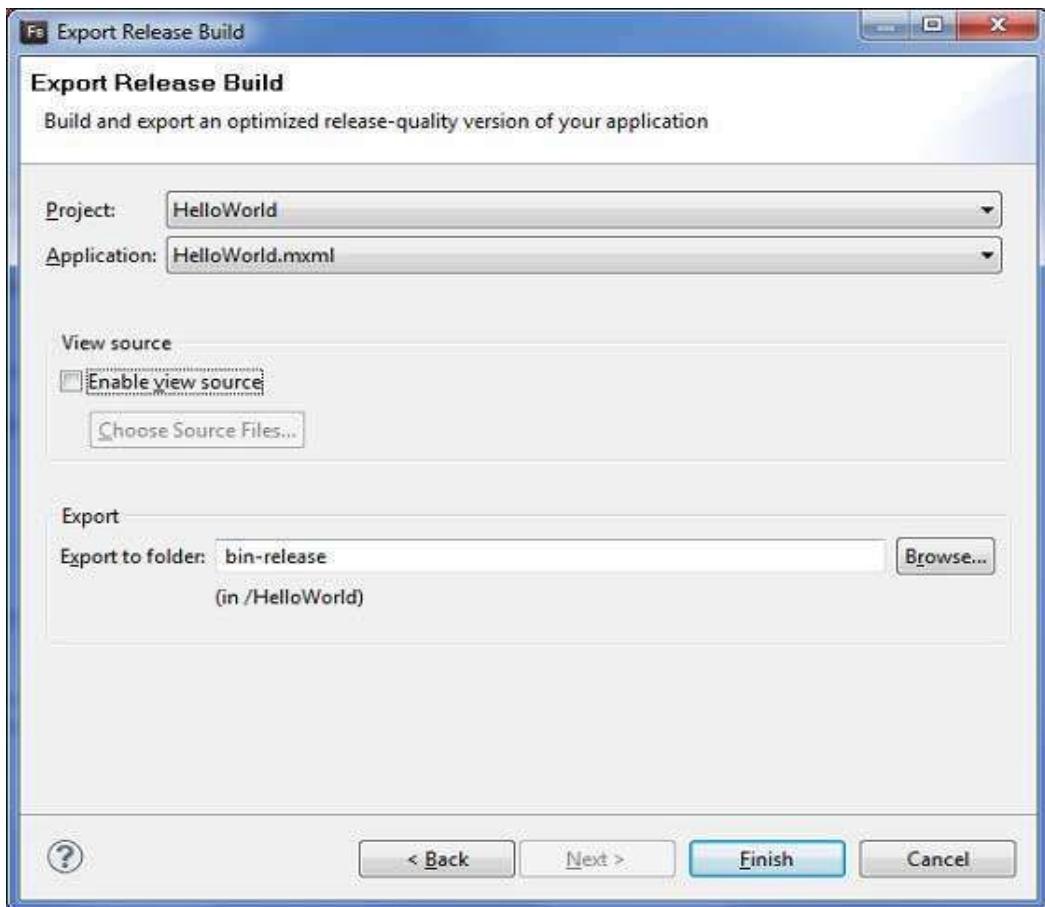
Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the Flex - Create Application chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Follow the steps given below to create a release build of a Flex application and then deploy it to tomcat server:

The first step is to create a release build using Flash Builder IDE. Launch release build wizard using the option **File > Export > Flash Builder > Release Build**



Select project as *HelloWorld* using the wizard window as follows



Leave other default values as such and click Finish Button. Now, Flash Builder will create a bin-release folder containing the project's release build.

Now our release build is ready, let us follow the following steps to deploy a Flex application:

Step	Description
1	Zip the content of the bin-release folder of the application in the form of HelloWorld.war file and deploy it in Apache Tomcat Webserver.
2	Launch your web application using appropriate URL as explained below in the last step.

Following is the content of the modified mxml file table

```
table-bordered/com.tutorialspoint/HelloWorld.mxml.
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%"
    minWidth="500" minHeight="500"
```

```

initialize="application_initializeHandler(event)">
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<fx:Script>
<![CDATA[
    import mx.controls.Alert;
    import mx.events.FlexEvent;
    protected function
    btnClickMe_clickHandler(event:MouseEvent):void
    {
        Alert.show("Hello World!");
    }

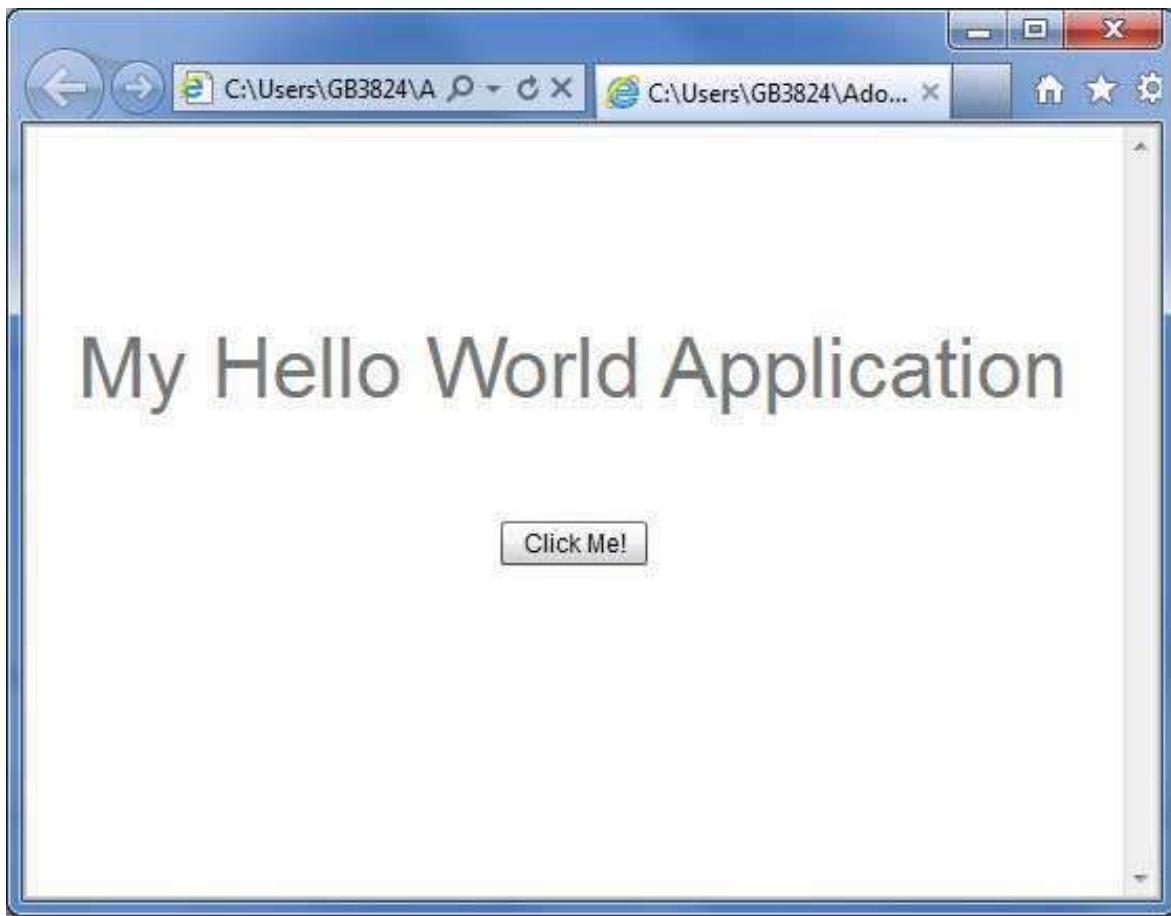
    protected function
    application_initializeHandler(event:FlexEvent):void
    {
        lblHeader.text = "My Hello World Application";

    }
]]>
</fx:Script>
<s:BorderContainer width="500" height="500" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center"
        verticalAlign="middle">
        <s:Label id="lblHeader" fontSize="40" color="0x777777"
            styleName="heading"/>
        <s:Button label="Click Me!" id="btnClickMe"
            click="btnClickMe_clickHandler(event)" styleName="button" />
    </s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter.

If everything is fine with your application, then it will produce the following result:



## Create WAR File

Now our application is working fine and we are ready to export it as a war file. Follow the following steps:

- Go into your project's bin-release directory C:\workspace\HelloWorld\bin-release
- Select all the files & folders available inside bin-release directory.
- Zip all the selected files & folders in a file called *HelloWorld.zip*.
- Rename *HelloWorld.zip* to *HelloWorld.war*.

## Deploy WAR file

Stop the tomcat server.

- Copy the *HelloWorld.war* file to tomcat installation directory > webapps folder.
- Start the tomcat server.
- Look inside webapps directory, there should be a folder *HelloWorld* got created.
- Now *HelloWorld.war* is successfully deployed in Tomcat Webserver root.

## Run Application

Enter a URL in web browser:

<http://localhost:8080/HelloWorld>HelloWorld.html>

to launch the application

Server name (localhost) and port (8080) may vary as per your tomcat configuration.

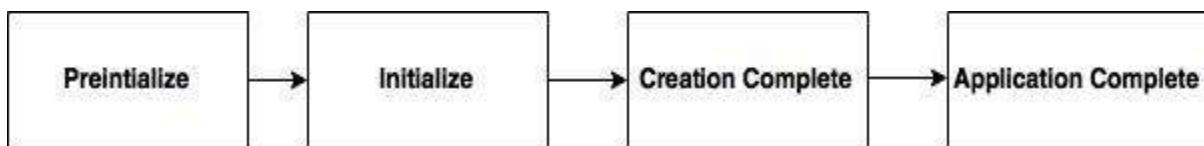


# 6. Flex – Life Cycle Phases

## Life Cycle of Flex Application

Although, you can build Flex applications without understanding the life cycle phases of an application, it is good to know the basic mechanism; the order in which things occur. It will help you to configure features such as loading other Flex applications at runtime, and manage the process of loading and unloading class libraries and assets at runtime.

A good understanding of the Flex application life cycle will enable you to build better applications and optimize them because you will know where to optimally run code. For example, if you need to ensure that some code runs during a pre-loader, you need to know where to place the code for that event.



When we load flex application in a browser, the following events occurs during the lifecycle of flex application.

Following is the brief detail about different Flex Life cycle events.

S.N.	Event & Description
1	<p>preInitialize: mx.core.UIComponent.preinitialize Event Type: mx.events.FlexEvent.PREINITIALIZE</p> <p>This event is dispatched at the beginning of the component initialization sequence. The component is in a very raw state when this event is dispatched. Many components, such as Button control creates internal child components to implement functionality. For example, the Button control creates an internal UI TextField component to represent its label text.</p> <p>When Flex dispatches the pre-initialize event, the children, including all the internal children, of a component have not yet been created.</p>
2	<p>initialize: mx.core.UIComponent.initialize Event Type: mx.events.FlexEvent.INITIALIZE</p> <p>This event is dispatched after pre-initialize phase. Flex framework initializes the internal structure of this component during this phase. This event automatically fires when the component is added to a parent.</p> <p>You do not need to call initialize() generally.</p>
	<p>creationComplete: mx.core.UIComponent.creationComplete Event Type: mx.events.FlexEvent.CREATION_COMPLETE</p>

3	This event is dispatched when the component has finished its construction, property processing, measuring, layout, and drawing.  At this point, depending on its visible property, the component is not visible even though it has been drawn.
4	applicationComplete: spark.components.Application.applicationComplete  Event Type:mx.events.FlexEvent.APPLICATION_COMPLETE  Dispatched after the Application has been initialized, processed by the LayoutManager, and attached to the display list.  This is the last event of the application creation life cycle and signifies that application has been loaded completely.

## Flex Life Cycle Example

---

Let us follow the steps to understand test life cycle of a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    initialize="reportEvent(event)"
    preinitialize="reportEvent(event)"
    creationComplete="reportEvent(event)"
    applicationComplete="reportEvent(event)">
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
```

```

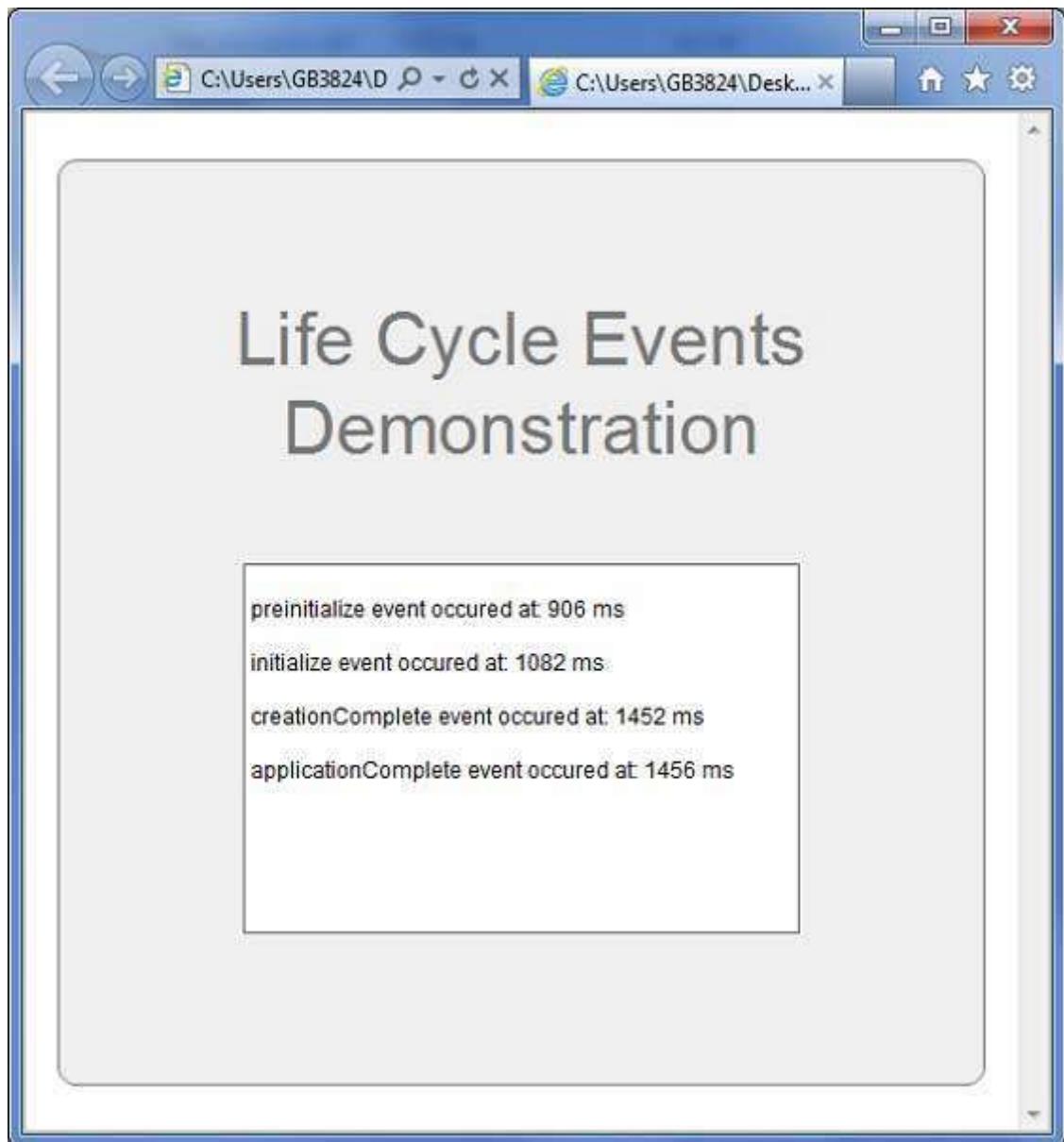
import mx.controls.Alert;
import mx.events.FlexEvent;

[Bindable]
private var report:String = "";

private function reportEvent(event:FlexEvent):void{
    report += "\n" + (event.type + " event occurred at: "
    + getTimer() + " ms" + "\n");
}
]]>
</fx:Script>
<s:BorderContainer width="500" height="500" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center" verticalAlign="middle">
<s:Label textAlign="center" width="100%" id="lblHeader"
fontSize="40" color="0x777777" styleName="heading"
text="Life Cycle Events Demonstration"/>
<s:TextArea id="reportText" text="{report}" editable="false"
width="300" height="200">
</s:TextArea>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## 7. Flex – Style with CSS

Flex supports the use of CSS syntax and styles to apply to its UI controls in the same way as CSS to HTML components.

### Way # 1: Using External Style Sheet File

You can refer to a style sheet available in the class path of the application. For example consider Style.css file in **com/tutorialspoint/client folder** where HelloWorld.mxml file also lies.

```
/* CSS file */  
@namespace s "library://ns.adobe.com/flex/spark";  
@namespace mx "library://ns.adobe.com/flex/mx";  
...  
.container {  
    cornerRadius :10;  
    horizontalCenter :0;  
    borderColor: #777777;  
    verticalCenter:0;  
    backgroundColor: #efefef;  
}
```

Then css file can be referred by following code snippet

```
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
```

Assign styles to UI component using `styleName` property

```
<s:BorderContainer width="500" height="500" id="mainContainer"  
    styleName="container">  
...  
</s:BorderContainer>
```

### Way # 2: Using Styles Within Ui Container Component

You can define styles within UI container component using `<fx:Style>` tag

#### Class Level Selector

```
<fx:Style>
```

```

@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/mx";

/* class level selector */
.errorLabel {
    color: red;
}
</fx:Style>

```

Assign styles to UI component using styleName property.

```
<s:Label id="errorMsg" text="This is an error message" styleName="errorLabel"/>
```

## Id Level Selector

Style UI component using id selector.

```

<fx:Style>
/* id level selector */
#msgLabel {
    color: gray;
}
</fx:Style>
<s:Label id="msgLabel" text="This is a normal message" />

```

## Type Level Selector

Style one type of UI Component in one GO.

```

<fx:Style>
/* style applied on all buttons */
s|Button {
    fontSize: 15;
    color: #9933FF;
}
</fx:Style>
<s:Button label="Click Me!" id="btnClickMe"
click="btnClickMe_clickHandler(event)" />

```

## Flex Style with CSS Example

Let us follow the steps to check CSS styling of a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>Style.css</i> , <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified CSS file **src/com.tutorialspoint/Style.css**.

```
/* CSS file */
@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/mx";

.heading
{
    fontFamily: Arial, Helvetica, sans-serif;
    fontSize: 17px;
    color: #9b1204;
    textDecoration:none;
    fontWeight:normal;
}

.button {
    fontWeight: bold;
}

.container {
    cornerRadius :10;
    horizontalCenter :0;
    borderColor: #777777;
    verticalCenter:0;
    backgroundColor: #efefef;
```

```
}
```

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    initialize="application_initializeHandler(event)">
    <!--Add reference to style sheet -->
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <!--Using styles within mxml file -->
    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        @namespace mx "library://ns.adobe.com/flex/mx";

        /* class level selector */
        .errorLabel {
            color: red;
        }

        /* id level selector */
        #msgLabel {
            color: gray;
        }

        /* style applied on all buttons */
        s|Button {
            fontSize: 15;
            color: #9933FF;
        }
    </fx:Style>
    <fx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.events.FlexEvent;
```

```

protected function btnClickMe_clickHandler(event:MouseEvent)
:void {
    Alert.show("Hello World!");
}

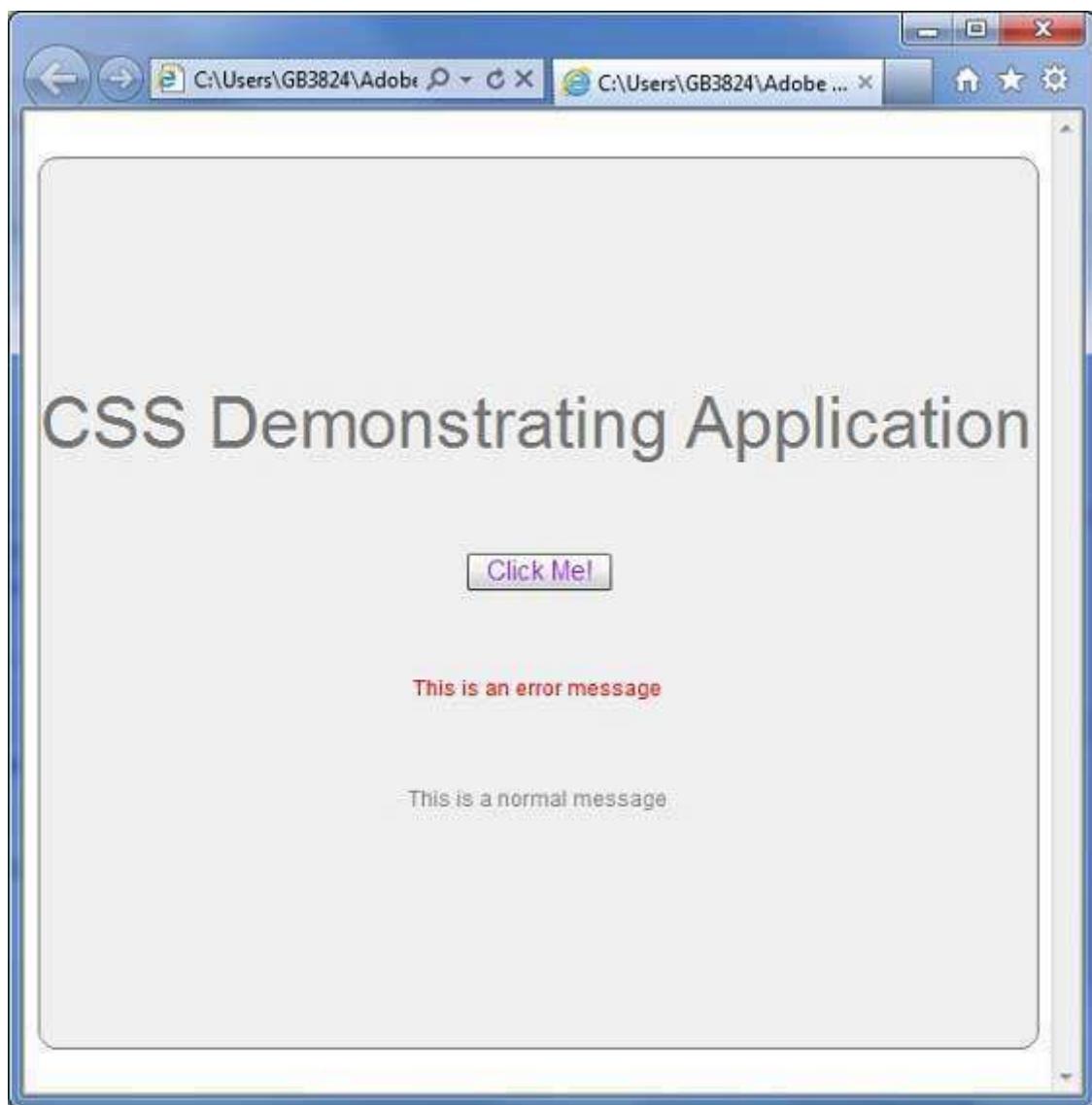
protected function application_initializeHandler(event:FlexEvent)
:void {
    lblHeader.text = "CSS Demonstrating Application";

}
]]>
</fx:Script>
<s:BorderContainer width="560" height="500" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center" verticalAlign="middle">
<s:Label width="100%" id="lblHeader" fontSize="40"
color="0x777777" styleName="heading"/>
<s:Button label="Click Me!" id="btnClickMe"
click="btnClickMe_clickHandler(event)" />
<s:Label id="errorMsg"
text="This is an error message" styleName="errorLabel" />

<s:Label id="msgLabel" text="This is a normal message" />
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



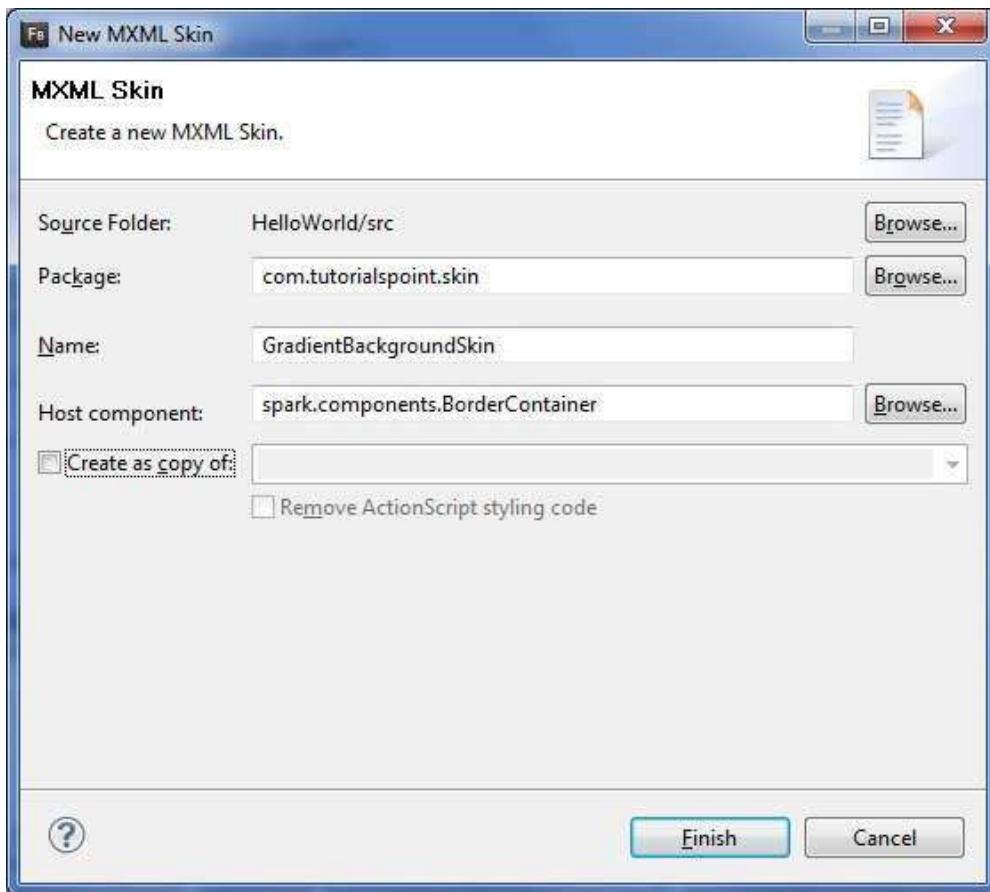
# 8. Flex – Style with Skin

## What is Skinning?

- Skinning in Flex, is a process of customizing look and feel of a UI Component completely.
- A Skin can define text, image, filters, transitions and states of a component.
- A Skin can be created as a separate mxml or ActionScript component.
- Using skin, we can control all visual aspects of a UI component.
- The process of defining skin is same for all the UI component.

### Step 1 – Create a Skin

Launch Create MXML Skin wizard using the option **File > New > MXML Skin**.



Enter Package as **com.tutorialspoint.skin**, name as **GradientBackgroundSkin** and choose host component as existing flex BorderContainer control **spark.component.BorderContainer**.

Now you've created a skin for a BorderContainer. Modify content of the mxml skin file **src/com.tutorialspoint/skin/GradientBackgroundSkin.mxml**.

Update fill layer as follows:

```
<!-- fill -->
<s:Rect id="backgroundRect" left="0" right="0" height="100%" top="0">
    <s:fill>
        <s:LinearGradient rotation="90">
            <s:GradientEntry color="0x888888" ratio="0.2"/>
            <s:GradientEntry color="0x111111" ratio="1"/>
        </s:LinearGradient>
    </s:fill>
</s:Rect>
```

## Step 2 – Apply Skin

You can apply skin over a component in two ways:

### Apply skin in MXML script (statically)

Apply **GradientBackgroundSkin** to a BorderContainer with id **mainContainer** using its **skinClass** attribute.

```
<s:BorderContainer width="560" height="500" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle"
        skinClass="com.tutorialspoint.skin.GradientBackgroundSkin">
```

### Apply skin in ActionScript (dynamically)

Apply **GradientBackgroundSkin** to a BorderContainer with id **mainContainer** using its **skinClass** property.

```
protected function gradientBackground_clickHandler(event:MouseEvent):void
{
    mainContainer.setStyle("skinClass", GradientBackgroundSkin );
}
```

## Flex Style with Skin Example

Let us follow the following steps to see skinning in action in a Flex application by creating a test application:

Step	Description
------	-------------

1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Create skin <i>GradientBackgroundSkin.mxml</i> under a package <i>com.tutorialspoint.skin</i> as explained above. Keep rest of the files unchanged.
3	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
4	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the *GradientBackgroundSkin.mxml* file *src/com/tutorialspoint/skin/GradientBackgroundSkin.mxml*.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx">
  <!-- host component -->
  <fx:Metadata>
    [HostComponent("spark.components.BorderContainer")]
  </fx:Metadata>

  <!-- states -->
  <s:states>
    <s:State name="disabled" />
    <s:State name="disabled" />
    <s:State name="normal" />
  </s:states>

  <!-- SkinParts
  name=contentGroup, type=spark.components.Group, required=false
  -->
  <!-- fill -->
  <s:Rect id="backgroundRect" left="0" right="0" height="100%" top="0">
    <s:fill>
      <s:LinearGradient rotation="90">
        <s:GradientEntry color="0x111111" ratio="0.2"/>
        <s:GradientEntry color="0x888888" ratio="1"/>
    </s:fill>
  </s:Rect>
</s:Skin>
```

```

        </s:LinearGradient>
    </s:fill>
</s:Rect>
<!-- must specify this for the host component --&gt;
&lt;s:Group id="contentGroup" left="0" right="0" top="0" bottom="0" /&gt;
&lt;/s:Skin&gt;
</pre>

```

Following is the content of the modified HelloWorld.mxml filesrc/com/tutorialspoint/client/HelloWorld.mxml.

```

<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    initialize="application_initializeHandler(event)">

    <fx:Style source="/com/tutorialspoint/client/Style.css"/>

    <fx:Script>
        <![CDATA[
            import com.tutorialspoint.skin.GradientBackgroundSkin;
            import mx.controls.Alert;
            import mx.events.FlexEvent;
            import spark.skins.spark.BorderContainerSkin;
            protected function btnClickMe_clickHandler(event:MouseEvent):void
            {
                Alert.show("Hello World!");
            }

            protected function application_initializeHandler(event:FlexEvent):void
            {
                lblHeader.text = "My Hello World Application";
            }

            protected function gradientBackground_clickHandler(event:MouseEvent):void
            {
                mainContainer.setStyle("skinClass", GradientBackgroundSkin );
            }
        ]]>
    </fx:Script>

```

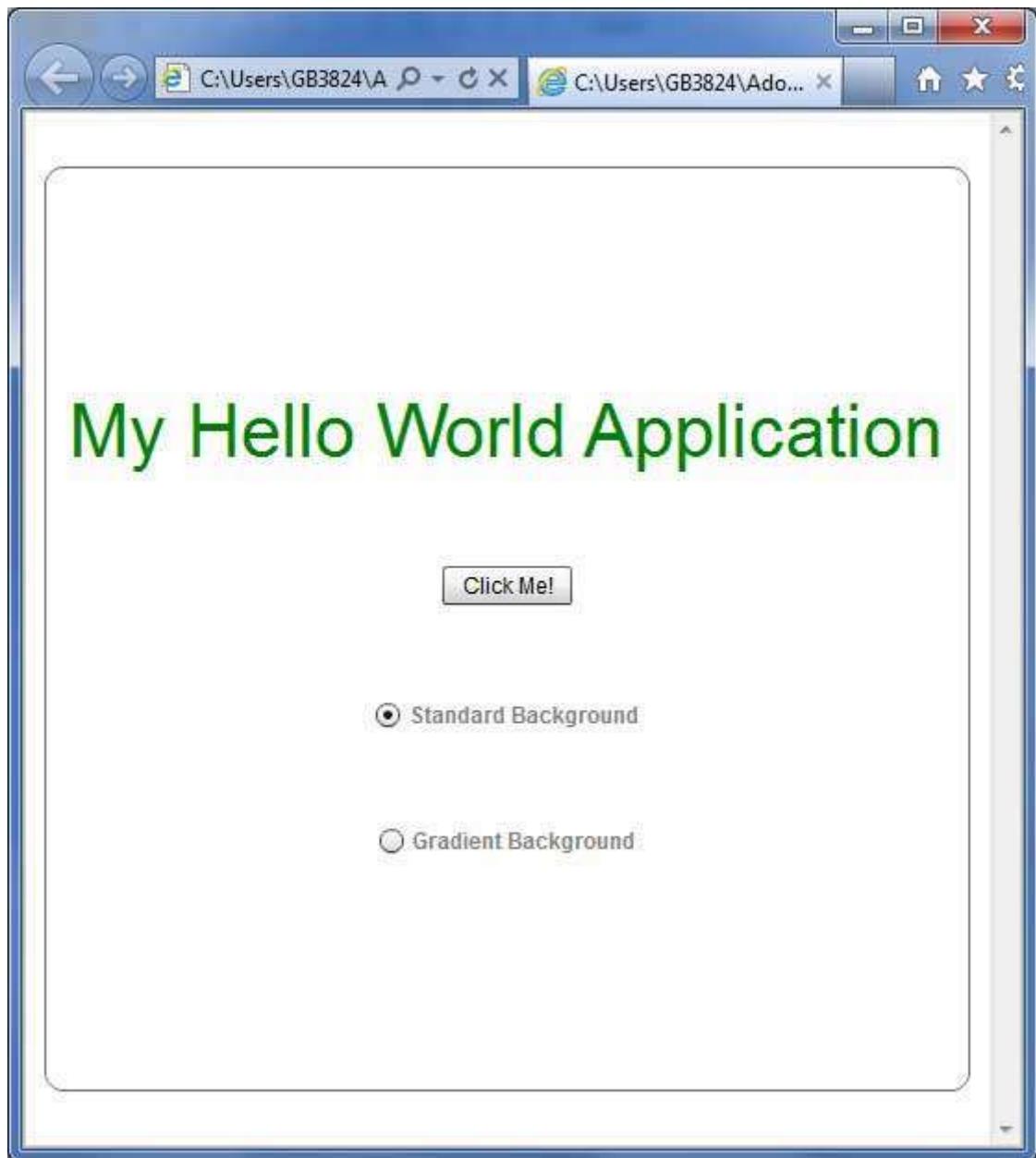
```

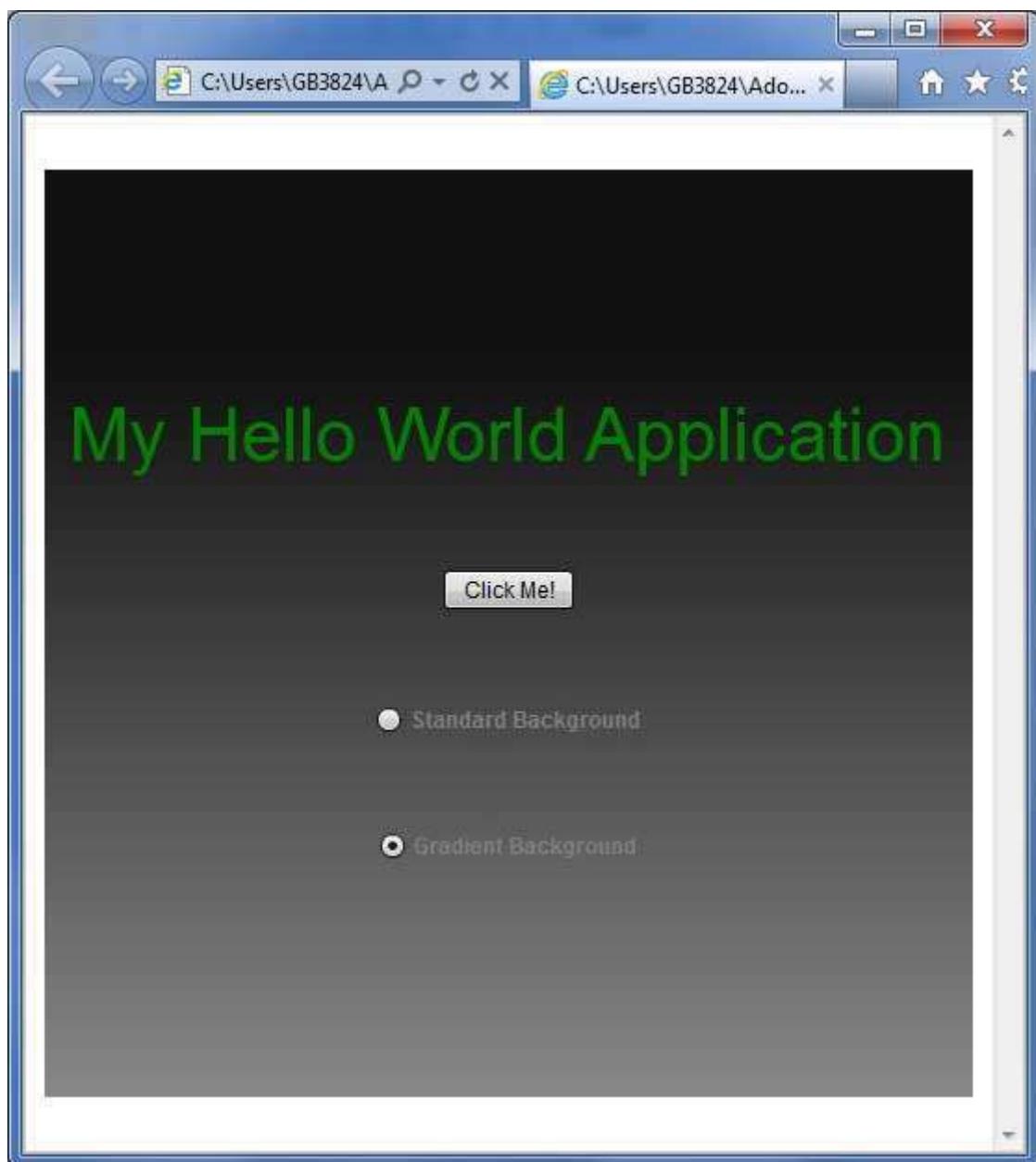
}

protected function standardBackground_clickHandler(event:MouseEvent):void
{
    mainContainer.setStyle("skinClass", BorderContainerSkin );
}
]]>
</fx:Script>
<fx:Declarations>
    <s:RadioButtonGroup id="selectorGroup" />
</fx:Declarations>
<s:BorderContainer width="500" height="500" id="mainContainer"
    skinClass="spark.skins.spark.BorderContainerSkin"
    horizontalCenter="0" verticalCenter="0" cornerRadius="10">      <s:VGroup
width="100%" height="100%" gap="50" horizontalAlign="center"
    verticalAlign="middle">
    <s:Label id="lblHeader" fontSize="40" color="green"
        styleName="heading"/>
    <s:Button label="Click Me!" id="btnClickMe"
        click="btnClickMe_clickHandler(event)"/>
    <s:RadioButton color="gray" fontWeight="bold"
        group="{selectorGroup}" label="Standard Background"
        click="standardBackground_clickHandler(event)" selected="true"/>
    <s:RadioButton color="gray" fontWeight="bold"
        group="{selectorGroup}" label="Gradient Background"
        click="gradientBackground_clickHandler(event)"/>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:





# 9. Flex – Data Binding

## What is Data Binding?

Data Binding is a process in which data of one object is tied to another object. It 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 as below

- Curly brace syntax in MXML Script ({} )
- <fx:binding> tag in MXML
- BindingUtils in ActionScript

### Data Binding – Using Curly Braces in MXML

The following example demonstrates how to use curly braces to specify data binding of a source to destination.

```
<s:TextInput id="txtInput1"/>
<s:TextInput id="txtInput2" text = "{txtInput1.text}"/>
```

### Data Binding – Using <fx:Binding> tag in MXML

The following example demonstrates how to use <fx:Binding> tag to specify data binding of a source to destination.

```
<fx:Binding source="txtInput1.text" destination="txtInput2.text" />
<s:TextInput id="txtInput1"/>
<s:TextInput id="txtInput2"/>
```

### Data Binding – Using BindingUtils in ActionScript

The following example demonstrates how to use BindingUtils to specify data binding of a source to destination.

```
<fx:Script>
<! [CDATA[
    import mx.binding.utils.BindingUtils;
    import mx.events.FlexEvent;

    protected function txtInput2_preinitializeHandler(event:FlexEvent):void
{
```

```

        BindingUtils.bindProperty(txtInput2,"text",txtInput1, "text");
    }
]]>
</fx:Script>
<s:TextInput id="txtInput1"/>
<s:TextInput id="txtInput2"
preinitialize="txtInput2_preinitializeHandler(event)"/>
```

## Flex Data Binding Example

Let us follow the steps given below to see skinning in action in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified *HelloWorld.mxml* file *src/com/tutorialspoint/client/HelloWorld.mxml*.

```

<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            import mx.binding.utils.BindingUtils;
            import mx.events.FlexEvent;

            protected function
            txtInput6_preinitializeHandler(event:FlexEvent):void
            {
                BindingUtils.bindProperty(txtInput6,"text",txtInput5, "text");
        ]]>
    </fx:Script>
</s:Application>
```

```

        }
    ]]>

</fx:Script>

<fx:Binding source="txtInput3.text" destination="txtInput4.text" />
<s:BorderContainer width="500" height="550" id="mainContainer"
styleName="container">

    <s:VGroup width="100%" height="100%" gap="50" horizontalAlign="center"
verticalAlign="middle">
        <s:Label id="lblHeader" text="Data Binding Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel title="Example #1 (Using Curly Braces,\{\})" width="400"
height="100" >
            <s:layout>
                <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
            </s:layout>
            <s:HGroup >
                <s:Label text = "Type here: " width="100" paddingTop="6"/>
                <s:TextInput id="txtInput1"/>
            </s:HGroup>
            <s:HGroup >
                <s:Label text = "Copied text: " width="100" paddingTop="6"/>
                <s:TextInput id="txtInput2" text = "{txtInput1.text}"/>
            </s:HGroup>
        </s:Panel>
        <s:Panel title="Example #2 (Using <fx:Binding>)" width="400"
height="100" >
            <s:layout>
                <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
            </s:layout>
            <s:HGroup >
                <s:Label text = "Type here: " width="100" paddingTop="6"/>
                <s:TextInput id="txtInput3"/>
            </s:HGroup>
            <s:HGroup >
                <s:Label text = "Copied text: " width="100" paddingTop="6"/>
                <s:Label id="txtInput4"/>
            </s:HGroup>
        </s:Panel>
    </s:VGroup>
</s:BorderContainer>

```

```
</s:Panel>

<s:Panel title="Example #3 (Using BindingUtils)" width="400"
height="100" > <s:layout>
    <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
</s:layout>
<s:HGroup >
    <s:Label text = "Type here: " width="100" paddingTop="6"/>
    <s:TextInput id="txtInput5"/>
</s:HGroup>
<s:HGroup >
    <s:Label text = "Copied text: " width="100" paddingTop="6"/>
    <s:TextInput enabled="false" id="txtInput6"
    preinitialize="txtInput6_preinitializeHandler(event)"/>
</s:HGroup>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:

The screenshot shows a Adobe Flex application window titled "Data Binding Demonstration". The window contains three examples of data binding:

- Example #1 (Using Curly Braces, {}):**

Type here:

Copied text:
- Example #2 (Using <fx:Binding>):**

Type here:

Copied text:
- Example #3 (Using BindingUtils):**

Type here:

Copied text:

# 10. Flex – Basic Controls

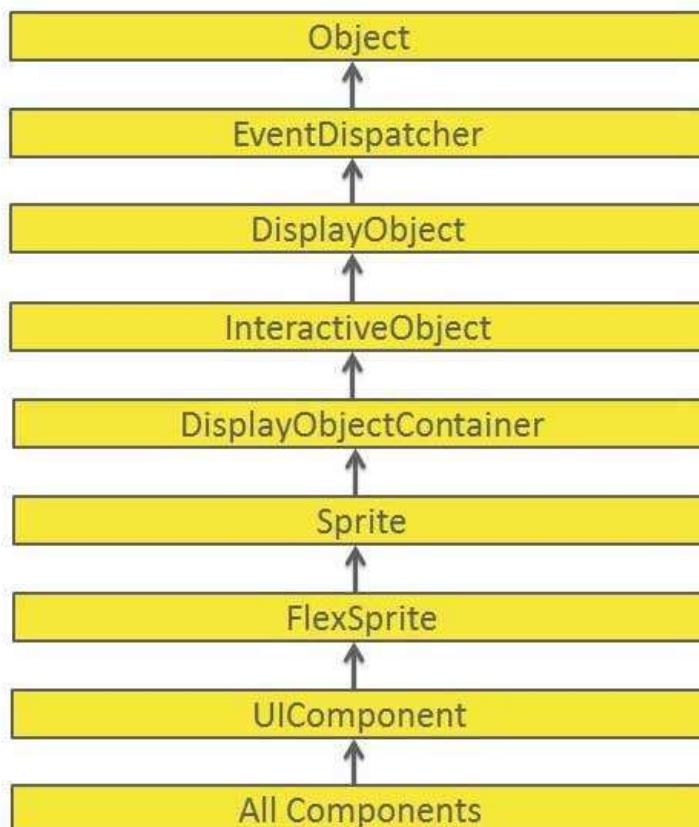
Every user interface considers the following three main aspects:

- **UI Elements:** These are the core visual elements the user eventually sees and interacts with. Flex provides a huge list of widely used and common elements varying from basic to complex which we will cover in this tutorial.
- **Layouts:** They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in Layout chapter.
- **Behavior:** These events occur when the user interacts with UI elements. This part will be covered in Event Handling chapter.

## Flex UI Elements

---

The Flex UI library provides classes in a well-defined class hierarchy to create complex web-based user interfaces. All classes in this component hierarchy have been derived from the **EventDispatcher** base class as shown below:



Every Basic UI control inherits properties from UI Component class which in turn inherits properties from EventDispatcher and other top level classes.

S.N.	Control & Description
1	<p>Flex EventDispatcher Class</p> <p>The EventDispatcher class is the base class for all classes that can dispatch events. The EventDispatcher class allows any object on the display list to be an event target and as such, to use the methods of the IEventDispatcher interface.</p>
2	<p>Flex UIComponent</p> <p>The UIComponent class is the base class for all visual components, both interactive and non-interactive.</p>

## Flex – Event Dispatcher Class

---

### Introduction

- The **EventDispatcher** class is the base class for all classes that dispatch events.
- The **EventDispatcher** class implements the **IEventDispatcher** interface.

The **EventDispatcher** class allows any object on the display list to be an event target and as such, to use the m

### What is an Event?

Event is a notification when a particular action is performed. For example, when a button is clicked then Click Event occurs.

### What is an Event Target

The **Event target** serves as the focal point for how events flow through the display list hierarchy.

When an event occurs, Flash Player dispatches an event object into the event flow from the root of the display list. The event object then makes its way through the display list until it reaches the event target, at which point it begins its return trip through the display list.

This round-trip journey to the event target is divided into three phases:

S.N.	Phase & Description
1	<p>capture</p> <p>This phase comprises the journey from the root to the last node before the event target's node</p>
2	<p>target</p> <p>This phase comprises only the event target node.</p>

3	bubbling
---	----------

This phase comprises any subsequent nodes encountered on the return trip to the root of the display list.

In general, any class which extends EventDispatcher gets the event dispatching capability.

## Class Declaration

Following is the declaration for **flash.events.EventDispatcher** class:

```
public class EventDispatcher
    extends java.lang.Object
    implements IEventDispatcher
```

## Public Methods

S.N.	Method & Description
1	<b>EventDispatcher(target:IEventDispatcher = null)</b>  Aggregates an instance of the EventDispatcher class.
2	<b>addEventListener(type:String, listener:Function, useCapture:Boolean = false, priority:int = 0, useWeakReference:Boolean = false):void</b>  Registers an event listener object with an EventDispatcher object so that the listener receives notification of an event.
3	<b>dispatchEvent(event:Event):Boolean</b>  Dispatches an event into the event flow.
4	<b>hasEventListener(type:String):Boolean</b>  Checks whether the EventDispatcher object has any listeners registered for a specific type of event.
5	<b>removeEventListener(type:String, listener:Function, useCapture:Boolean = false):void</b>  Removes a listener from the EventDispatcher object.
6	<b>willTrigger(type:String):Boolean</b>

	Checks whether an event listener is registered with this EventDispatcher object or any of its ancestors for the specified event type.
--	---

## Events

Following are the events for **flash.events.EventDispatcher** class:

S.N.	Event & Description
1	activate Dispatched when the Flash Player gains operating system focus and becomes active.
2	detivate Dispatched when the Flash Player loses operating system focus and becomes inactive.

## Methods Inherited

This class inherits methods from the following class:

- Object

## Flex – UIComponent Class

---

### Introduction

The **UIComponent** class is the base class for all visual components, both interactive and noninteractive.

### Class Declaration

Following is the declaration for **mx.core.UIComponent** class:

```
public class UIComponent
    extends FlexSprite
    implements IAutomationObject, IChildList, IConstraintClient,
               IDelayedInstantiationUIComponent, IFlexDisplayObject,
               IFlexModule, IIValidating, ILayoutManagerClient,
               IPropertyChangeNotifier, IRepeaterClient, IStateClient,
               IAdvancedStyleClient, IToolTipManagerClient,
               UIComponent, IValidatorListener, IVisualElement
```

## Public Properties

Following are the Public Properties for **mx.core.UIComponent** class:

S.N.	Name & Description
1	accessibilityDescription : String A convenience accessor for the description property in this UIComponent's accessibilityProperties object.
2	accessibilityEnabled : Boolean A convenience accessor for the silent property in this UIComponent's accessibilityProperties object.
3	accessibilityName : String A convenience accessor for the name property in this UIComponent's accessibilityProperties object.
4	accessibilityShortcut : String A convenience accessor for the shortcut property in this UIComponent's accessibilityProperties object.
5	activeEffects : Array [read-only] The list of effects that are currently playing on the component, as an Array of EffectInstance instances.
6	automationDelegate : Object The delegate object that handles the automation-related functionality.
7	automationEnabled : Boolean [read-only] True if this component is enabled for automation, false otherwise.
8	automationName : String Name that can be used as an identifier for this object.
9	automationOwner : DisplayObjectContainer [read-only] The owner of this component for automation purposes.
10	automationParent : DisplayObjectContainer [read-only] The parent of this component for automation purposes.

11	<b>automationTabularData</b> : Object  [read-only] An implementation of the IAutomationTabularData interface, which can be used to retrieve the data.
12	<b>automationValue</b> : Array  [read-only] This value generally corresponds to the rendered appearance of the object and should be usable for correlating the identifier with the object as it appears visually within the application.
13	<b>automationVisible</b> : Boolean  [read-only] True if this component is visible for automation, false otherwise.
14	<b>baseline</b> : Object  For components, this layout constraint property is a facade on top of the similarly-named style.
15	<b>baselinePosition</b> : Number  [read-only] The y-coordinate of the baseline of the first line of text of the component.
16	<b>bottom</b> : Object  For components, this layout constraint property is a facade on top of the similarly-named style.
17	<b>cacheHeuristic</b> : Boolean  [write-only] Used by Flex to suggest bitmap caching for the object.
18	<b>cachePolicy</b> : String  Specifies the bitmap caching policy for this object.
19	<b>className</b> : String  [read-only] The name of this instance's class, such as "Button".
20	<b>contentMouseX</b> : Number  [read-only] Returns the x position of the mouse, in the content coordinate system.
21	<b>contentMouseY</b> : Number

	[read-only] Returns the y position of the mouse, in the content coordinate system.
22	<p>currentState : String</p> <p>The current view state of the component.</p>
23	<p>cursorManager : ICursorManager</p> <p>[read-only] Gets the CursorManager that controls the cursor for this component and its peers.</p>
24	<p>depth : Number</p> <p>Determines the order in which items inside of containers are rendered.</p>
25	<p>descriptor : UIComponentDescriptor</p> <p>Reference to the UIComponentDescriptor, if any, that was used by the createComponentFromDescriptor() method to create this UIComponent instance.</p>
26	<p>designLayer : DesignLayer</p> <p>Specifies the optional DesignLayer instance associated with this visual element.</p>
27	<p>document : Object</p> <p>A reference to the document object associated with this UIComponent.</p>
28	<p>doubleClickEnabled : Boolean</p> <p>[override] Specifies whether the UIComponent object receives doubleClick events.</p>
29	<p>enabled : Boolean</p> <p>Whether the component can accept user interaction.</p>
30	<p>errorString : String</p> <p>The text that displayed by a component's error tip when a component is monitored by a Validator and validation fails.</p>
31	<p>explicitHeight : Number</p> <p>Number that specifies the explicit height of the component, in pixels, in the component's coordinates.</p>

32	<b>explicitMaxHeight : Number</b> The maximum recommended height of the component to be considered by the parent during layout.
33	<b>explicitMaxWidth : Number</b> The maximum recommended width of the component to be considered by the parent during layout.
34	<b>explicitMinHeight : Number</b> The minimum recommended height of the component to be considered by the parent during layout.
35	<b>explicitMinWidth : Number</b> The minimum recommended width of the component to be considered by the parent during layout.
36	<b>explicitWidth : Number</b> Number that specifies the explicit width of the component, in pixels, in the component's coordinates.
37	<b>flexContextMenu : IFlexContextMenu</b> The context menu for this UIComponent.
38	<b>focusEnabled : Boolean</b> Indicates whether the component can receive focus when tabbed to.
39	<b>focusManager : IFocusManager</b> Gets the FocusManager that controls focus for this component and its peers.
40	<b>focusPane : Sprite</b> The focus pane associated with this object.
41	<b>hasFocusableChildren : Boolean</b> A flag that indicates whether child objects can receive focus.
42	<b>hasLayoutMatrix3D : Boolean</b> [read-only] Contains true if the element has 3D Matrix.

43	<b>height : Number</b>  [override] Number that specifies the height of the component, in pixels, in the parent's coordinates.
44	<b>horizontalCenter : Object</b>  For components, this layout constraint property is a facade on top of the similarly-named style.
45	<b>id : String</b>  ID of the component.
46	<b>includeInLayout : Boolean</b>  Specifies whether this component is included in the layout of the parent container.
47	<b>inheritingStyles : Object</b>  The beginning of this component's chain of inheriting styles.
48	<b>initialized : Boolean</b>  A flag that determines if an object has been through all three phases of layout: commitment, measurement, and layout (provided that any were required).
49	<b>instanceIndex : int</b>  [read-only] The index of a repeated component.
50	<b>instanceIndices : Array</b>  An Array containing the indices required to reference this UIComponent object from its parent document.
51	<b>is3D : Boolean</b>  [read-only] Contains true when the element is in 3D.
52	<b>isDocument : Boolean</b>  [read-only] Contains true if this UIComponent instance is a document object.
53	<b>isPopUp : Boolean</b>  Set to true by the PopUpManager to indicate that component has been popped up.

54	<b>layoutMatrix3D</b> : Matrix3D [write-only] The transform matrix that is used to calculate a component's layout relative to its siblings.
55	<b>left</b> : Object For components, this layout constraint property is a facade on top of the similarly-named style.
56	<b>maintainProjectionCenter</b> : Boolean When true, the component keeps its projection matrix centered on the middle of its bounding box.
57	<b>maxHeight</b> : Number The maximum recommended height of the component to be considered by the parent during layout.
58	<b>maxWidth</b> : Number The maximum recommended width of the component to be considered by the parent during layout.
59	<b>measuredHeight</b> : Number The default height of the component, in pixels.
60	<b>measuredMinHeight</b> : Number The default minimum height of the component, in pixels.
61	<b>measuredMinWidth</b> : Number The default minimum width of the component, in pixels.
62	<b>measuredWidth</b> : Number The default width of the component, in pixels.
63	<b>minHeight</b> : Number The minimum recommended height of the component to be considered by the parent during layout.
64	<b>minWidth</b> : Number

	The minimum recommended width of the component to be considered by the parent during layout.
65	<b>moduleFactory : IFlexModuleFactory</b> A module factory is used as context for using embedded fonts and for finding the style manager that controls the styles for this component.
66	<b>mouseFocusEnabled : Boolean</b> Whether you can receive focus when clicked on.
67	<b>nestLevel : int</b> Depth of this object in the containment hierarchy.
68	<b>nonInheritingStyles : Object</b> The beginning of this component's chain of non-inheriting styles.
69	<b>numAutomationChildren : int</b> [read-only] The number of automation children this container has.
70	<b>owner : DisplayObjectContainer</b> The owner of this IVisualElement object.
71	<b>parent : DisplayObjectContainer</b> [override] [read-only] The parent container or component for this component.
72	<b>parentApplication : Object</b> [read-only] A reference to the Application object that contains this UIComponent instance.
73	<b>parentDocument : Object</b> [read-only] A reference to the parent document object for this UIComponent.
74	<b>percentHeight : Number</b> Specifies the height of a component as a percentage of its parent's size.
75	<b>percentWidth : Number</b> Specifies the width of a component as a percentage of its parent's size.

76	<b>postLayoutTransformOffsets</b> : mx.geom:TransformOffsets  Defines a set of adjustments that can be applied to the object's transform in a way that is invisible to its parent's layout.
77	<b>processedDescriptors</b> : Boolean  Set to true after immediate or deferred child creation, depending on which one happens.
78	<b>repeater</b> : IRepeater  [read-only] A reference to the Repeater object in the parent document that produced this UIComponent.
79	<b>repeaterIndex</b> : int  [read-only] The index of the item in the data provider of the Repeater that produced this UIComponent.
80	<b>repeaterIndices</b> : Array  An Array containing the indices of the items in the data provider of the Repeaters in the parent document that produced this UIComponent.
81	<b>repeaters</b> : Array  An Array containing references to the Repeater objects in the parent document that produced this UIComponent.
82	<b>right</b> : Object  For components, this layout constraint property is a facade on top of the similarly-named style.
83	<b>rotation</b> : Number  [override] Indicates the rotation of the DisplayObject instance, in degrees, from its original orientation.
84	<b>rotationX</b> : Number  [override] Indicates the x-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
85	<b>rotationY</b> : Number  [override] Indicates the y-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.

86	<b>rotationZ</b> : Number  [override] Indicates the z-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
87	<b>scaleX</b> : Number  [override] Number that specifies the horizontal scaling factor.
88	<b>scaleY</b> : Number  [override] Number that specifies the vertical scaling factor.
89	<b>scaleZ</b> : Number  [override] Number that specifies the scaling factor along the z axis.
90	<b>screen</b> : Rectangle  [read-only] Returns an object that contains the size and position of the base drawing surface for this object.
91	<b>showInAutomationHierarchy</b> : Boolean  A flag that determines if an automation object shows in the automation hierarchy.
92	<b>states</b> : Array  The view states that are defined for this component.
93	<b>styleDeclaration</b> : CSSStyleDeclaration  Storage for the inline inheriting styles on this object.
94	<b>styleManager</b> : IStyleManager2  [read-only] Returns the StyleManager instance used by this component.
95	<b>styleName</b> : Object  The class style used by this component.
96	<b>styleParent</b> : IAdvancedStyleClient  A component's parent is used to evaluate descendant selectors.
97	<b>systemManager</b> : ISysytemManager  Returns the SystemManager object used by this component.

98	<b>tabFocusEnabled : Boolean</b>  A flag that indicates whether this object can receive focus via the TAB key. This is similar to the tabEnabled property used by the Flash Player. This is usually true for components that handle keyboard input, but some components in controlbars have them set to false because they should not steal focus from another component like an editor.
99	<b>toolTip : String</b>  Text to display in the ToolTip.
100	<b>top : Object</b>  For components, this layout constraint property is a facade on top of the similarly-named style.
101	<b>transform : flash.geom:Transform</b>  [override] An object with properties pertaining to a display object's matrix, color transform, and pixel bounds.
102	<b>transformX : Number</b>  Sets the x coordinate for the transform center of the component.
103	<b>transformY : Number</b>  Sets the y coordinate for the transform center of the component.
104	<b>transformZ : Number</b>  Sets the z coordinate for the transform center of the component.
105	<b>transitions : Array</b>  An Array of Transition objects, where each Transition object defines a set of effects to play when a view state change occurs.
106	<b>tweeningProperties : Array</b>  Array of properties that are currently being tweened on this object.
107	<b>uid : String</b>  A unique identifier for the object.
108	<b>updateCompletePendingFlag : Boolean</b>

	A flag that determines if an object has been through all three phases of layout validation (provided that any were required).
109	<b>validationSubField</b> : String Used by a validator to associate a subfield with this component.
110	<b>verticalCenter</b> : Object For components, this layout constraint property is a facade on top of the similarly-named style.
111	<b>visible</b> : Boolean [override] Whether or not the display object is visible.
112	<b>width</b> : Number [override] Number that specifies the width of the component, in pixels, in the parent's coordinates.
113	<b>x</b> : Number [override] Number that specifies the component's horizontal position, in pixels, within its parent container.
114	<b>y</b> : Number [override] Number that specifies the component's vertical position, in pixels, within its parent container.
115	<b>z</b> : Number [override] Indicates the z coordinate position along the z-axis of the DisplayObject instance relative to the 3D parent container.

## Protected Properties

Following are the Protected Properties for **mx.core.UIComponent** class:

S.N.	Name & Description
1	<b>currentCSSState</b> : String [read-only] The state to be used when matching CSS pseudo-selectors.
2	<b>hasComplexLayoutMatrix</b> : Boolean

	[read-only] Returns true if the UIComponent has any non-translation (x,y) transform properties.
3	resourceManager : IResourceManager [read-only] A reference to the object which manages all of the application's localized resources.
4	unscaledHeight : Number [read-only] A convenience method for determining the unscaled height of the component.
5	unscaledWidth : Number [read-only] A convenience method for determining the unscaled width of the component All of a component's drawing and child layout should be done within a bounding rectangle of this width, which is also passed as an argument to updateDisplayList().

#### Event Active and Deactivate description

S.N.	Event & Description
1	activate Dispatched when the Flash Player gains operating system focus and becomes active.
2	detivate Dispatched when the Flash Player loses operating system focus and becomes inactive.

#### Public Methods

S.N.	Method & Description
1	UIComponent() Constructor.
2	addStyleClient(styleClient:IAdvancedStyleClient):void Adds a non-visual style client to this component instance.

3	<code>callLater(method:Function, args:Array = null):void</code> Queues a function to be called later.
4	<code>clearStyle(styleProp:String):void</code> Deletes a style property from this component instance.
5	<code>contentToGlobal(point:Point):Point</code> Converts a Point object from content coordinates to global coordinates.
6	<code>contentToLocal(point:Point):Point</code> Converts a Point object from content to local coordinates.
7	<code>createAutomationIDPart(child:IAutomationObject):Object</code> Returns a set of properties that identify the child within this container.
8	<code>createAutomationIDPartWithRequiredProperties(child:IAutomationObject, properties:Array):Object</code> Returns a set of properties that identify the child within this container.
9	<code>createReferenceOnParentDocument(parentDocument:IFlexDisplayObject):void</code> Creates an id reference to this UIComponent object on its parent document object.
10	<code>deleteReferenceOnParentDocument(parentDocument:IFlexDisplayObject):void</code> Deletes the id reference to this UIComponent object on its parent document object.
11	<code>determineTextFormatFromStyles():mx.core:UITextFormat</code> Returns a UITextFormat object corresponding to the text styles for this UIComponent.
12	<code>dispatchEvent(event:Event):Boolean</code> [override] Dispatches an event into the event flow.
13	<code>drawFocus(isFocused:Boolean):void</code> Shows or hides the focus indicator around this component.

14	<code>drawRoundRect(x:Number, y:Number, w:Number, h:Number, r:Object = null, c:Object = null, alpha:Object = null, rot:Object = null, gradient:String = null, ratios:Array = null, hole:Object = null):void</code> Programmatically draws a rectangle into this skin's Graphics object.
15	<code>effectFinished(effectInst:IEffectInstance):void</code> Called by the effect instance when it stops playing on the component.
16	<code>effectStarted(effectInst:IEffectInstance):void</code> Called by the effect instance when it starts playing on the component.
17	<code>endEffectsStarted():void</code> Ends all currently playing effects on the component.
18	<code>executeBindings(recurse:Boolean = false):void</code> Executes all the bindings for which the UIComponent object is the destination.
19	<code>finishPrint(obj:Object, target:IFlexDisplayObject):void</code> Called after printing is complete.
20	<code>getAutomationChildAt(index:int):IAutomationObject</code> Provides the automation object at the specified index.
21	<code>getAutomationChildren():Array</code> Provides the automation object list .
22	<code>getBoundsXAtSize(width:Number, height:Number, postLayoutTransform:Boolean = true):Number</code> Returns the x coordinate of the element's bounds at the specified element size.
23	<code>getBoundsYAtSize(width:Number, height:Number, postLayoutTransform:Boolean = true):Number</code> Returns the y coordinate of the element's bounds at the specified element size.
24	<code>getClassStyleDeclarations():Array</code> Finds the type selectors for this UIComponent instance.
25	<code>getConstraintValue(constraintName:String):*</code>

	Returns a layout constraint value, which is the same as getting the constraint style for this component.
26	<code>getExplicitOrMeasuredHeight():Number</code> A convenience method for determining whether to use the explicit or measured height
27	<code>getExplicitOrMeasuredWidth():Number</code> A convenience method for determining whether to use the explicit or measured width
28	<code>getFocus():InteractiveObject</code> Gets the object that currently has focus.
29	<code>getLayoutBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's layout height.
30	<code>getLayoutBoundsWidth(postLayoutTransform:Boolean = true):Number</code> Returns the element's layout width.
31	<code>getLayoutBoundsX(postLayoutTransform:Boolean = true):Number</code> Returns the x coordinate that the element uses to draw on screen.
32	<code>getLayoutBoundsY(postLayoutTransform:Boolean = true):Number</code> Returns the y coordinate that the element uses to draw on screen.
33	<code>getLayoutMatrix():Matrix</code> Returns the transform matrix that is used to calculate the component's layout relative to its siblings.
34	<code>getLayoutMatrix3D():Matrix3D</code> Returns the layout transform Matrix3D for this element.
35	<code>getMaxBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's maximum height.
36	<code>getMaxBoundsWidth(postLayoutTransform:Boolean = true):Number</code> Returns the element's maximum width.

37	<code>getMinBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's minimum height.
38	<code>getMinBoundsWidth(postLayoutTransform:Boolean = true):Number</code> Returns the element's minimum width.
39	<code>getPreferredBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's preferred height.
40	<code>getPreferredBoundsWidth(postLayoutTransform:Boolean = true):Number</code> Returns the element's preferred width.
41	<code>getRepeaterItem(whichRepeater:int = -1):Object</code> Returns the item in the dataProvider that was used by the specified Repeater to produce this Repeater, or null if this Repeater isn't repeated.
42	<code>getStyle(styleProp:String):*</code> Gets a style property that has been set anywhere in this component's style lookup chain.
43	<code>globalToContent(point:Point):Point</code> Converts a Point object from global to content coordinates.
45	<code>hasCSSState():Boolean</code> Returns true if currentCSSState is not null.
46	<code>hasState(stateName:String):Boolean</code> Determines whether the specified state has been defined on this UIComponent.
47	<code>horizontalGradientMatrix(x:Number, y:Number, width:Number, height:Number):Matrix</code> Returns a box Matrix which can be passed to the drawRoundRect() method as the rot parameter when drawing a horizontal gradient.
48	<code>initialize():void</code> Initializes the internal structure of this component.
49	<code>initializeRepeaterArrays(parent:IRepeaterClient):void</code>

	<p>Initializes various properties which keep track of repeated instances of this component.</p>
50	<p><code>invalidateDisplayList():void</code></p> <p>Marks a component so that its <code>updateDisplayList()</code> method gets called during a later screen update.</p>
51	<p><code>invalidateLayering():void</code></p> <p>Called by a component's items to indicate that their depth property has changed.</p>
52	<p><code>invalidateLayoutDirection():void</code></p> <p>An element must call this method when its <code>layoutDirection</code> changes or when its parent's <code>layoutDirection</code> changes.</p>
53	<p><code>invalidateProperties():void</code></p> <p>Marks a component so that its <code>commitProperties()</code> method gets called during a later screen update.</p>
54	<p><code>invalidateSize():void</code></p> <p>Marks a component so that its <code>measure()</code> method gets called during a later screen update.</p>
55	<p><code>localToContent(point:Point):Point</code></p> <p>Converts a <code>Point</code> object from local to content coordinates.</p>
56	<p><code>matchesCSSState(cssState:String):Boolean</code></p> <p>Returns true if <code>cssState</code> matches <code>currentCSSState</code>.</p>
57	<p><code>matchesCSSType(cssType:String):Boolean</code></p> <p>Determines whether this instance is the same as, or is a subclass of, the given type.</p>
58	<p><code>measureHTMLText(htmlText:String):flash.text:TextLineMetrics</code></p> <p>Measures the specified HTML text, which can contain HTML tags such as <code>&amp;lt;font&amp;ampgt</code> and <code>&amp;lt;b&amp;ampgt</code>, assuming that it is displayed in a single-line <code>UITextField</code> using a <code>UITextFormat</code> determined by the styles of this <code>UIComponent</code>.</p>
59	<p><code>measureText(text:String):flash.text:TextLineMetrics</code></p>

	Measures the specified text, assuming that it is displayed in a single-line UITextField (or UIFETextField) using a UITextFormat determined by the styles of this UIComponent.
60	<code>move(x:Number, y:Number):void</code> Moves the component to a specified position within its parent.
61	<code>notifyStyleChangeInChildren(styleProp:String, recursive:Boolean):void</code> Propagates style changes to the children.
62	<code>owns(child:DisplayObject):Boolean</code> Returns true if the chain of owner properties points from child to this UIComponent.
63	<code>parentChanged(p:DisplayObjectContainer):void</code> Called by Flex when a UIComponent object is added to or removed from a parent.
64	<code>prepareToPrint(target:IFlexDisplayObject):Object</code> Prepares an IFlexDisplayObject for printing.
65	<code>regenerateStyleCache(recursive:Boolean):void</code> Builds or rebuilds the CSS style cache for this component and, if the recursive parameter is true, for all descendants of this component as well.
66	<code>registerEffects(effects:Array):void</code> For each effect event, registers the EffectManager as one of the event listeners.
67	<code>removeStyleClient(styleClient:IAdvancedStyleClient):void</code> Removes a non-visual style client from this component instance.
68	<code>replayAutomatableEvent(event:Event):Boolean</code> Replays the specified event.
69	<code>resolveAutomationIDPart(criteria:Object):Array</code> Resolves a child by using the id provided.
70	<code>resumeBackgroundProcessing():void</code>

	[static] Resumes the background processing of methods queued by callLater(), after a call to suspendBackgroundProcessing().
71	<code>setActualSize(w:Number, h:Number):void</code> Sizes the object.
72	<code>setConstraintValue(constraintName:String, value:*):void</code> Sets a layout constraint value, which is the same as setting the constraint style for this component.
73	<code>setCurrentState(stateName:String, playTransition:Boolean = true):void</code> Set the current state.
74	<code>setFocus():void</code> Sets the focus to this component.
75	<code>setLayoutBoundsPosition(x:Number, y:Number, postLayoutTransform:Boolean = true):void</code> Sets the coordinates that the element uses to draw on screen.
76	<code>set Layout Bounds Size(width:Number, height:Number, post Layout Transform:Boolean = true):void</code> Sets the layout size of the element.
77	<code>setLayoutMatrix(value:Matrix, invalidateLayout:Boolean):void</code> Sets the transform Matrix that is used to calculate the component's layout size and position relative to its siblings.
78	<code>setLayoutMatrix3D(value:Matrix3D, invalidateLayout:Boolean):void</code> Sets the transform Matrix3D that is used to calculate the component's layout size and position relative to its siblings.
79	<code>setStyle(styleProp:String, newValue:*):void</code> Sets a style property on this component instance.
80	<code>setVisible(value:Boolean, noEvent:Boolean = false):void</code> Called when the visible property changes.

81	<code>styleChanged(styleProp:String):void</code> Detects changes to style properties.
82	<code>stylesInitialized():void</code> Flex calls the <code>stylesInitialized()</code> method when the styles for a component are first initialized.
83	<code>suspendBackgroundProcessing():void</code> [static] Blocks the background processing of methods queued by <code>callLater()</code> , until <code>resumeBackgroundProcessing()</code> is called.
84	<code>transformAround(transformCenter:Vector3D, scale:Vector3D = null, rotation:Vector3D = null, translation:Vector3D = null, postLayoutScale:Vector3D = null, postLayoutRotation:Vector3D = null, postLayoutTranslation:Vector3D = null, invalidateLayout:Boolean = true):void</code> A utility method to update the rotation, scale, and translation of the transform while keeping a particular point, specified in the component's own coordinate space, fixed in the parent's coordinate space.
85	<code>transform Point To Parent (localPosition:Vector3D, position:Vector3D, post Layout Position:Vector3D):void</code> A utility method to transform a point specified in the local coordinates of this object to its location in the object's parent's coordinates.
86	<code>validateDisplayList():void</code> Validates the position and size of children and draws other visuals.
87	<code>validateNow():void</code> Validate and update the properties and layout of this object and redraw it, if necessary.
88	<code>validateProperties():void</code> Used by layout logic to validate the properties of a component by calling the <code>commitProperties()</code> method.
89	<code>validateSize(recursive:Boolean = false):void</code> Validates the measured size of the component If the <code>LayoutManager.invalidateSize()</code> method is called with this <code>ILayoutManagerClient</code> , then the <code>validateSize()</code> method is called when it's time to do measurements.

90	<b>validationResultHandler(event:ValidationResultEvent):void</b> Handles both the valid and invalid events from a validator assigned to this component.
91	<b>verticalGradientMatrix(x:Number, y:Number, width:Number, height:Number):Matrix</b> Returns a box Matrix which can be passed to drawRoundRect() as the rot parameter when drawing a vertical gradient.

## Protected Methods

<b>S.N.</b>	<b>Method &amp; Description</b>
1	<b>adjustFocusRect(obj:DisplayObject = null):void</b> Adjust the focus rectangle.
2	<b>applyComputedMatrix():void</b> Commits the computed matrix built from the combination of the layout matrix and the transform offsets to the flash displayObject's transform.
3	<b>attachOverlay():void</b> This is an internal method used by the Flex framework to support the Dissolve effect.
4	<b>canSkipMeasurement():Boolean</b> Determines if the call to the measure() method can be skipped.
5	<b>childrenCreated():void</b> Performs any final processing after child objects are created.
6	<b>commitProperties():void</b> Processes the properties set on the component.
7	<b>createChildren():void</b> Create child objects of the component.
8	<b>createInFontContext(classObj:Class):Object</b> Creates a new object using a context based on the embedded font being used.

9	<code>createInModuleContext(moduleFactory:IFlexModuleFactory, className:String):Object</code> Creates the object using a given moduleFactory.
10	<code>dispatchPropertyChangeEvent(prop:String, oldValue:*, value:*):void</code> Helper method for dispatching a PropertyChangeEvent when a property is updated.
11	<code>focusInHandler(event:FocusEvent):void</code> The event handler called when a UIComponent object gets focus.
12	<code>focusOutHandler(event:FocusEvent):void</code> The event handler called when a UIComponent object loses focus.
13	<code>initAdvancedLayoutFeatures():void</code> Initializes the implementation and storage of some of the less frequently used advanced layout features of a component.
14	<code>initializationComplete():void</code> Finalizes the initialization of this component.
15	<code>initializeAccessibility():void</code> Initializes this component's accessibility code.
16	<code>invalidateParentSizeAndDisplayList():void</code> Helper method to invalidate parent size and display list if this object affects its layout (includeInLayout is true).
17	<code>isOurFocus(target:DisplayObject):Boolean</code> Typically overridden by components containing UITextField objects, where the UITextField object gets focus.
18	<code>keyDownHandler(event:KeyboardEvent):void</code> The event handler called for a keyDown event.
19	<code>keyUpHandler(event:KeyboardEvent):void</code> The event handler called for a keyUp event.

20	<b>measure():void</b> Calculates the default size, and optionally the default minimum size, of the component.
21	<b>resourcesChanged():void</b> This method is called when a UIComponent is constructed, and again whenever the ResourceManager dispatches a "change" Event to indicate that the localized resources have changed in some way.
22	<b>setStretchXY(stretchX:Number, stretchY:Number):void</b> Specifies a transform stretch factor in the horizontal and vertical direction.
23	<b>stateChanged(oldState:String, newState:String, recursive:Boolean):void</b> This method is called when a state changes to check whether state-specific styles apply to this component
24	<b>updateDisplayList(unscaledWidth:Number, unscaledHeight:Number):void</b> Draws the object and/or sizes and positions its children.

## Events

Following are the events for **mx.core.UIComponent** class:

S.N.	Event & Description
1	<b>add</b> when the component is added to a container as a content child by using the addChild(), addChildAt(), addElement(), or addElementAt() method.
2	<b>creationComplete</b> when the component has finished its construction, property processing, measuring, layout, and drawing.
3	<b>currentStateChange</b> after the view state has changed.
4	<b>currentStateChanging</b> after the currentState property changes, but before the view state changes.

5	<b>dragComplete</b> by the drag initiator (the component that is the source of the data being dragged) when the drag operation completes, either when you drop the dragged data onto a drop target or when you end the drag-and-drop operation without performing a drop.
6	<b>dragDrop</b> by the drop target when the user releases the mouse over it.
7	<b>dragEnter</b> by a component when the user moves the mouse over the component during a drag operation.
8	<b>dragExit</b> by the component when the user drags outside the component, but does not drop the data onto the target.
9	<b>dragOver</b> by a component when the user moves the mouse while over the component during a drag operation.
10	<b>dragStart</b> by the drag initiator when starting a drag operation.
11	<b>effectEnd</b> after an effect ends.
12	<b>effectStart</b> just before an effect starts.
13	<b>effectStop</b> after an effect is stopped, which happens only by a call to stop() on the effect.
14	<b>enterState</b> after the component has entered a view state.
15	<b>exitState</b> just before the component exits a view state.

16	hide when an object's state changes from visible to invisible.
17	initialize when the component has finished its construction and has all initialization properties set.
18	invalid when a component is monitored by a Validator and the validation failed.
19	mouseDownOutside from a component opened using the PopUpManager when the user clicks outside it.
20	mouseWheelOutside from a component opened using the PopUpManager when the user scrolls the mouse wheel outside it.
21	move when the object has moved.
22	preinitialize at the beginning of the component initialization sequence.
23	remove when the component is removed from a container as a content child by using the removeChild(), removeChildAt(), removeElement(), or removeElementAt() method.
24	resize when the component is resized.
25	show when an object's state changes from invisible to visible.
26	stateChangeComplete after the component has entered a new state and any state transition animation to that state has finished playing.

27	<b>stateChangeInterrupted</b> when a component interrupts a transition to its current state in order to switch to a new state.
28	<b>toolTipCreate</b> by the component when it is time to create a ToolTip.
29	<b>toolTipEnd</b> by the component when its ToolTip has been hidden and is to be discarded soon.
30	<b>toolTipHide</b> by the component when its ToolTip is about to be hidden.
31	<b>toolTipShow</b> by the component when its ToolTip is about to be shown.
32	<b>toolTipShown</b> by the component when its ToolTip has been shown.
33	<b>toolTipStart</b> by a component whose toolTip property is set, as soon as the user moves the mouse over it.
34	<b>touchInteractionEnd</b> A non-cancellable event, by a component when it is done responding to a touch interaction user gesture.
35	<b>touchInteractionStart</b> A non-cancellable event, by a component when it starts responding to a touch interaction user gesture.
36	<b>touchInteractionStarting</b> A cancellable event, by a component in an attempt to respond to a touch interaction user gesture.
37	<b>updateComplete</b> when an object has had its commitProperties(), measure(), and updateDisplayList() methods called (if needed).

38	<b>valid</b> when a component is monitored by a Validator and the validation succeeded.
39	<b>valueCommit</b> when values are changed programmatically or by user interaction.

## Methods Inherited

This class inherits methods from the following classes:

- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Basic Controls

---

Following are the few important *Basic Controls*:

S.N.	Controls & Description
1	<b>Label</b> Label is a low-level UIComponent that can render one or more lines of uniformly-formatted text.
2	<b>Text</b> The Text control lets you display HTML content as well as normal text in your application.
3	<b>Image</b> The Image control lets you import JPEG, PNG, GIF, and SWF files at runtime.
4	<b>LinkButton</b> The LinkButton control is a borderless Button control whose contents are highlighted when a user moves the mouse over it.

## Flex - Label Control

---

### Introduction

Label is a UIComponent that can render one or more lines of uniformly-formatted text. The text to be displayed is determined by the text property inherited from TextBase control.

## Class Declaration

Following is the declaration for **spark.components.Label** class:

```
public class Label
    extends TextBase
```

## Public Methods

S.N.	Method & Description
1	Label() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- mx.core.TextBase
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex Label Control Example

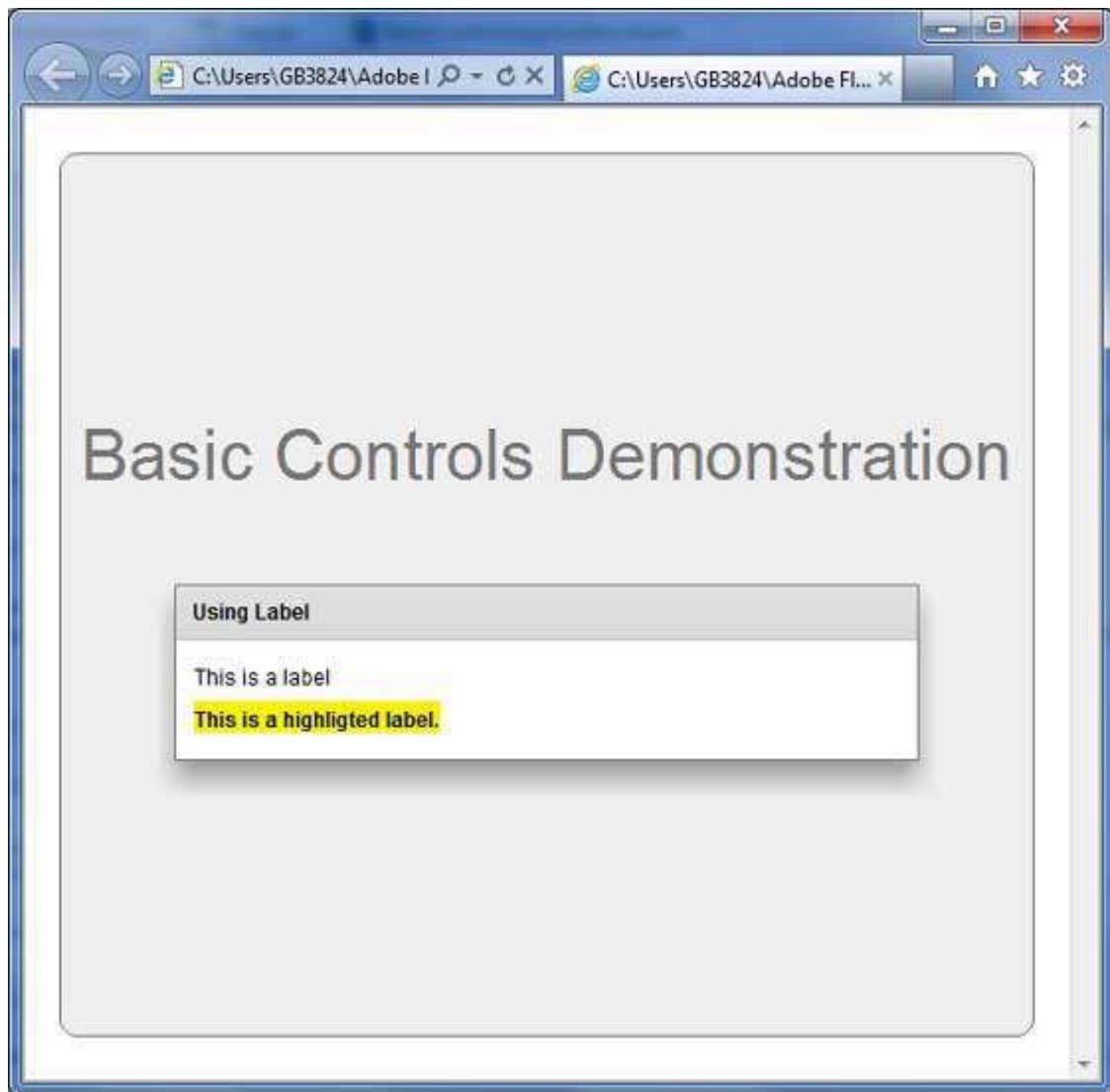
Let us follow the following steps to check usage of Label control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500" >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <s:BorderContainer width="550" height="500" id="mainContainer"
        styleName="container">
        <s:VGroup width="100%" height="100%" gap="50"
            horizontalAlign="center"
            verticalAlign="middle">
            <s:Label id="lblHeader" text="Basic Controls Demonstration"
                fontSize="40" color="0x777777" styleName="heading"/>
            <s:Panel title="Using Label" width="420" height="100" >
                <s:layout>
                    <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
                </s:layout>
                <s:Label text = "This is a label" paddingTop="6" />
                <s:Label text = "This is a highlighted label."
                    fontWeight="bold" backgroundColor="0xFFFF200"
                    paddingTop="6"/>
            </s:Panel>
        </s:VGroup>
    </s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – Text Control

### Introduction

Text is a UIComponent The Text control displays multiline, noneditable text.

### Class Declaration

Following is the declaration for **mx.controls.Text** class:

```
public class Text  
    extends Label
```

### Public Methods

S.N.	Method & Description
------	----------------------

1	Text()
	Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- mx.core.Label
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex Text Control Example

---

Let us follow the following steps to check usage of Text control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500" >
```

```
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<s:BorderContainer width="550" height="500" id="mainContainer"
styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center"
verticalAlign="middle">
        <s:Label id="lblHeader" text="Basic Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel title="Using Text" width="420" height="100" >
            <s:layout>
                <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
            </s:layout>
            <mx:Text htmlText="<i>We are using
< b>HTML</b> tags here.</i>" />
        </s:Panel>
    </s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – Image Control

---

### Introduction

The Image control lets you import JPEG, PNG, and GIF files at runtime. You can also embed any of these files at compile time by using @Embed(source='filename').

Embedded images load immediately, because they are compiled with of the Flex SWF file. However, they add to the size of the application and slow down the application initialization process. Embedded images also require you to recompile your applications whenever your image files change.

You can load an image from the local file system in which the SWF file runs, or you can access a remote image, typically though an HTTP request over a network. These images are independent of your Flex application, so you can change them without causing a recompile operation as long as the names of the modified images remain the same. The referenced images add no additional overhead to an application's initial loading time.

## Class Declaration

Following is the declaration for **spark.components.Image** class:

```
public class Image
    extends SkinnableComponent
```

## Public Properties

S.N.	Property & Description
1	bitmapData : BitmapData [read-only] Returns a copy of the BitmapData object representing the currently loaded image content (unscaled).
2	bytesLoaded : Number [read-only] The number of bytes of the image already loaded.
3	bytesTotal : Number [read-only] The total image data in bytes loaded or pending load.
4	clearOnLoad : Boolean Denotes whether or not to clear previous image content prior to loading new content.
5	contentLoader : IContentLoader Optional custom image loader
6	contentLoaderGrouping : String Optional content grouping identifier to pass to the an associated IContentLoader instance's load() method.
7	fillMode : String Determines how the bitmap fills in the dimensions.
8	horizontalAlign : String The horizontal alignment of the content when it does not have a one-to-one aspect ratio and scaleMode is set to mx.graphics.BitmapScaleMode.LETTERBOX.
9	preliminaryHeight : Number

	Provides an estimate to use for height when the "measured" bounds of the image is requested by layout, but the image data has yet to complete loading.
10	<p><code>preliminaryWidth : Number</code></p> <p>Provides an estimate to use for width when the "measured" bounds of the image is requested by layout, but the image data has yet to complete loading.</p>
11	<p><code>scaleMode : String</code></p> <p>Determines how the image is scaled when <code>fillMode</code> is set to <code>mx.graphics.BitmapFillMode.SCALE</code>.</p>
12	<p><code>smooth : Boolean</code></p> <p>Specifies whether to apply a smoothing algorithm to the bitmap image.</p>
13	<p><code>source : Object</code></p> <p>The source used for the bitmap fill.</p>
14	<p><code>sourceHeight : Number</code></p> <p>[read-only] Provides the unscaled height of the original image data.</p>
15	<p><code>sourceWidth : Number</code></p> <p>[read-only] Provides the unscaled width of the original image data.</p>
16	<p><code>trustedSource : Boolean</code></p> <p>[read-only] A read-only flag denoting whether the currently loaded content is considered loaded from a source whose security policy allows for cross domain image access.</p>
17	<p><code>verticalAlign : String</code></p> <p>The vertical alignment of the content when it does not have a one-to-one aspect ratio and <code>scaleMode</code> is set to <code>mx.graphics.BitmapScaleMode.LETTERBOX</code>.</p>

## Public Methods

S.N.	Method & Description
1	<p><code>Image()</code></p> <p>Constructor.</p>

## Events

S.N.	Events & Description
1	complete Dispatched when content loading is complete.
2	httpStatus Dispatched when a network request is made over HTTP and Flash Player can detect the HTTP status code.
3	ioError Dispatched when an input or output error occurs.
4	progress Dispatched when content is loading.
5	ready Dispatched when content loading is complete.
6	securityError Dispatched when a security error occurs.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex Image Control Example

---

Let us follow the following steps to check usage of Image control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Create a folder <i>assets</i> in <i>HelloWorld</i> application root folder <i>HelloWorld</i> .
3	Download a sample image <a href="#"><u>flex-mini.png</u></a> to a folder <i>assets</i> under <i>HelloWorld</i> folder.
4	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
5	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    applicationComplete="init(event)" >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.events.FlexEvent;

            [Bindable]
            [Embed(source="assets/flex-mini.jpg")]
            private var flexImage:Class;

            protected function init(event:FlexEvent):void
            {
                dynamicImage.source =
        
```

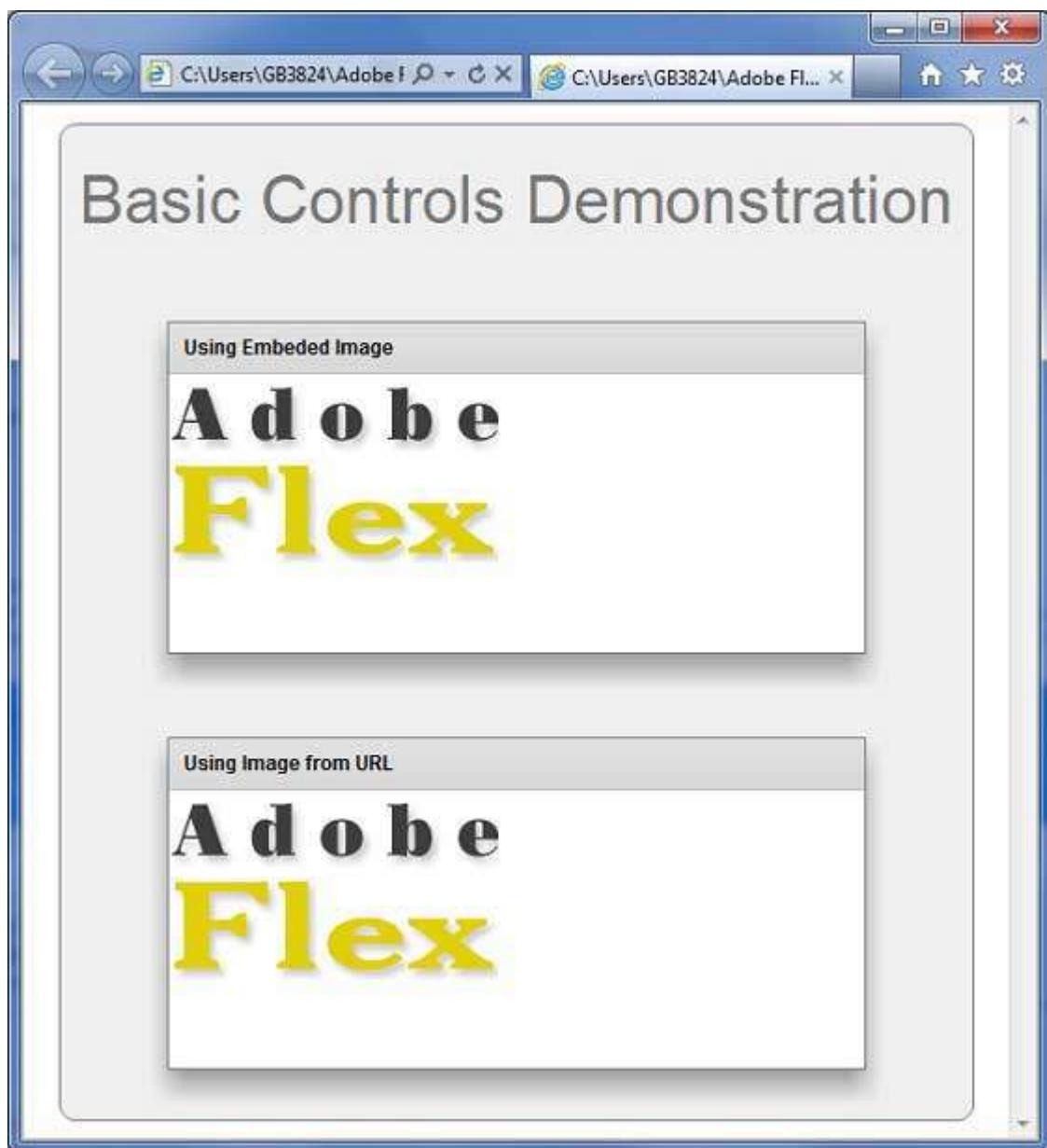
```

        "http://www.tutorialspoint.com/images/flex-mini.png";
    }

]]>
</fx:Script>
<s:BorderContainer width="550" height="600" id="mainContainer"
styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center"
verticalAlign="middle">
        <s:Label id="lblHeader" text="Basic Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel title="Using Embedded Image" width="420" height="200" >
            <s:Image source="{flexImage}" />
        </s:Panel>
        <s:Panel title="Using Image from URL" width="420" height="200" >
            <s:Image id="dynamicImage" />
        </s:Panel>
    </s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – LinkButton Control

The LinkButton control is a borderless Button control whose contents are highlighted when a user hover over it. You must specify a click event handler for the LinkButton control to perform some action similar to a button.

### Class Declaration

Following is the declaration for **mx.controls.LinkButton** class:

```
public class LinkButton  
    extends Button
```

## Public Methods

S.N.	Method & Description
1	LinkButton() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- mx.controls.Button
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex LinkButton Control Example

Let us follow the following steps to check usage of LinkButton control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file  
**src/com.tutorialspoint/HelloWorld.mxml**

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
```

```

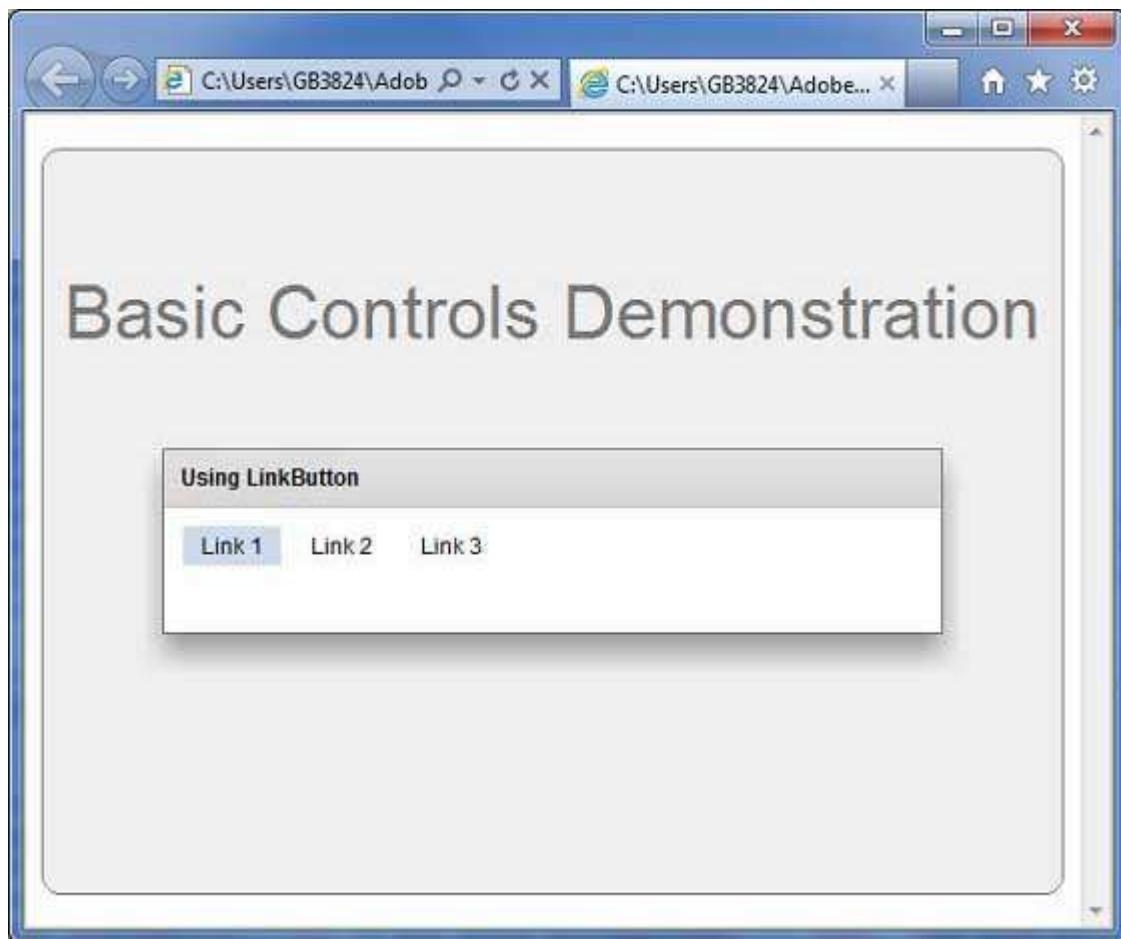
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx"
width="100%" height="100%" minWidth="500" minHeight="500">
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<fx:Script>
<![CDATA[
    import mx.controls.Alert;

    protected function clickMe_clickHandler(event:MouseEvent):void
    {
        var linkButton:LinkButton = event.target as LinkButton;
        Alert.show("LinkButton "+linkButton.id+" Clicked");
    }
]]>
</fx:Script>
<s:BorderContainer width="550" height="500" id="mainContainer"
styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center"
verticalAlign="middle">
        <s:Label id="lblHeader" text="Basic Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel title="Using LinkButton" width="420" height="100" >
            <s:layout>
                <s:HorizontalLayout paddingTop="10" paddingLeft="10"/>
            </s:layout>
            <mx:LinkButton label="Link 1" id="clickMe"
click="clickMe_clickHandler(event)"/>
            <mx:LinkButton label="Link 2" id="clickMe1"
click="clickMe_clickHandler(event)"/>
            <mx:LinkButton label="Link 3" id="clickMe2"
click="clickMe_clickHandler(event)"/>
        </s:Panel>
    </s:VGroup>
</s:BorderContainer>

```

```
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



# 11. Flex – Form Controls

Form controls allow the users to input data and provides them interaction capability with the application. Every Form UI controls and inherits properties from `UIComponent` class which in turn inherits properties from `EventDispatcher` and other top level classes.

S.No.	Control & Description
1	<p>Flex <code>EventDispatcher</code> Class</p> <p>The <code>EventDispatcher</code> class is the base class for all classes that can dispatch events. The <code>EventDispatcher</code> class allows any object on the display list to be an event target and as such, to use the methods of the <code>IEventDispatcher</code> interface.</p>
2	<p>Flex <code>UIComponent</code></p> <p>The <code>UIComponent</code> class is the base class for all visual components, both interactive and noninteractive.</p>

## Flex – Event Dispatcher Class

### Introduction

- The **EventDispatcher** class is the base class for all classes that dispatch events.
- The **EventDispatcher** class implements the **IEventDispatcher** interface.
- The **EventDispatcher** class allows any object on the display list to be an event target and as such, to use the methods of the `IEventDispatcher` interface.

In order to understand **EventDispatcher**, let us first look what are event and event targets.

### What is an Event?

**Event** is a notification when a particular action is performed. For example, when a button is clicked then Click Event occurs.

### What is an Event Target

The **Event target** serves as the focal point for how events flow through the display list hierarchy.

When an event occurs, Flash Player dispatches an event object into the event flow from the root of the display list. The event object then makes its way through the display list until it reaches the event target, at which point it begins its return trip through the display list.

This round-trip journey to the event target is divided into three phases:

S.N.	Phase & Description

1	capture  This phase comprises the journey from the root to the last node before the event target's node
2	target  This phase comprises only the event target node.
3	bubbling  This phase comprises any subsequent nodes encountered on the return trip to the root of the display list.

In general, any class which extends EventDispatcher gets the event dispatching capability.

## Class Declaration

Following is the declaration for **flash.events.EventDispatcher** class:

```
public class EventDispatcher
    extends java.lang.Object
    implements IEventDispatcher
```

## Public Methods

S.N.	Method & Description
1	EventDispatcher(target:IEventDispatcher = null)  Aggregates an instance of the EventDispatcher class.
2	addEventListener(type:String, listener:Function, useCapture:Boolean = false, priority:int = 0, useWeakReference:Boolean = false):void  Registers an event listener object with an EventDispatcher object so that the listener receives notification of an event.
3	dispatchEvent(event:Event):Boolean  Dispatches an event into the event flow.
4	hasEventListener(type:String):Boolean  Checks whether the EventDispatcher object has any listeners registered for a specific type of event.

5	<b>removeEventListener(type:String, listener:Function, useCapture:Boolean = false):void</b> Removes a listener from the EventDispatcher object.
6	<b>willTrigger(type:String):Boolean</b> Checks whether an event listener is registered with this EventDispatcher object or any of its ancestors for the specified event type.

## Events

Following are the events for **flash.events.EventDispatcher** class:

<b>S.N.</b>	<b>Event &amp; Description</b>
1	<b>activate</b> Dispatched when the Flash Player gains operating system focus and becomes active.
2	<b>detivate</b> Dispatched when the Flash Player loses operating system focus and becomes inactive.

## Methods Inherited

This class inherits methods from the following class:

- Object

## Flex - UIComponent Class

The **UIComponent** class is the base class for all visual components, both interactive and noninteractive.

### Class Declaration

Following is the declaration for **mx.core.UIComponent** class:

```
public class UIComponent
  extends FlexSprite
  implements IAutomationObject, IChildList, IConstraintClient,
  IDeferredInstantiationUIComponent, IFlexDisplayObject,
  IFlexModule, IInvalidate, ILayoutManagerClient,
```

**IPropertyChangeNotifier, IRepeaterClient, IStateClient,  
IAdvancedStyleClient, IToolTipManagerClient,  
UIComponent, IValidatorListener, IVisualElement**

## Public Properties

Following are the Public Properties for **mx.core.UIComponent** class:

S.N.	Name & Description
1	accessibilityDescription : String  A convenience accessor for the description property in this UIComponent's accessibilityProperties object.
2	accessibilityEnabled : Boolean  A convenience accessor for the silent property in this UIComponent's accessibilityProperties object.
3	accessibilityName : String  A convenience accessor for the name property in this UIComponent's accessibilityProperties object.
4	accessibilityShortcut : String  A convenience accessor for the shortcut property in this UIComponent's accessibilityProperties object.
5	activeEffects : Array  [read-only] The list of effects that are currently playing on the component, as an Array of EffectInstance instances.
6	automationDelegate : Object  The delegate object that handles the automation-related functionality.
7	automationEnabled : Boolean  [read-only] True if this component is enabled for automation, false otherwise.
8	automationName : String  Name that can be used as an identifier for this object.

9	automationOwner : DisplayObjectContainer [read-only] The owner of this component for automation purposes.
10	automationParent : DisplayObjectContainer [read-only] The parent of this component for automation purposes.
11	automationTabularData : Object [read-only] An implementation of the IAutomationTabularData interface, which can be used to retrieve the data.
12	automationValue : Array [read-only] This value generally corresponds to the rendered appearance of the object and should be usable for correlating the identifier with the object as it appears visually within the application.
13	automationVisible : Boolean [read-only] True if this component is visible for automation, false otherwise.
14	baseline : Object For components, this layout constraint property is a facade on top of the similarly-named style.
15	baselinePosition : Number [read-only] The y-coordinate of the baseline of the first line of text of the component.
16	bottom : Object For components, this layout constraint property is a facade on top of the similarly-named style.
17	cacheHeuristic : Boolean [write-only] Used by Flex to suggest bitmap caching for the object.
18	cachePolicy : String Specifies the bitmap caching policy for this object.
19	className : String [read-only] The name of this instance's class, such as "Button".

20	<b>contentMouseX</b> : Number  [read-only] Returns the x position of the mouse, in the content coordinate system.
21	<b>contentMouseY</b> : Number  [read-only] Returns the y position of the mouse, in the content coordinate system.
22	<b>currentState</b> : String  The current view state of the component.
23	<b>cursorManager</b> : ICursorManager  [read-only] Gets the CursorManager that controls the cursor for this component and its peers.
24	<b>depth</b> : Number  Determines the order in which items inside of containers are rendered.
25	<b>descriptor</b> : UIComponentDescriptor  Reference to the UIComponentDescriptor, if any, that was used by the <code>createComponentFromDescriptor()</code> method to create this UIComponent instance.
26	<b>designLayer</b> : DesignLayer  Specifies the optional DesignLayer instance associated with this visual element.
27	<b>document</b> : Object  A reference to the document object associated with this UIComponent.
28	<b>doubleClickEnabled</b> : Boolean  [override] Specifies whether the UIComponent object receives doubleClick events.
29	<b>enabled</b> : Boolean  Whether the component can accept user interaction.
30	<b>errorString</b> : String  The text that displayed by a component's error tip when a component is monitored by a Validator and validation fails.

31	<b>explicitHeight : Number</b> Number that specifies the explicit height of the component, in pixels, in the component's coordinates.
32	<b>explicitMaxHeight : Number</b> The maximum recommended height of the component to be considered by the parent during layout.
33	<b>explicitMaxWidth : Number</b> The maximum recommended width of the component to be considered by the parent during layout.
34	<b>explicitMinHeight : Number</b> The minimum recommended height of the component to be considered by the parent during layout.
35	<b>explicitMinWidth : Number</b> The minimum recommended width of the component to be considered by the parent during layout.
36	<b>explicitWidth : Number</b> Number that specifies the explicit width of the component, in pixels, in the component's coordinates.
37	<b>flexContextMenu : IFlexContextMenu</b> The context menu for this UIComponent.
38	<b>focusEnabled : Boolean</b> Indicates whether the component can receive focus when tabbed to.
39	<b>focusManager : IFocusManager</b> Gets the FocusManager that controls focus for this component and its peers.
40	<b>focusPane : Sprite</b> The focus pane associated with this object.
41	<b>hasFocusableChildren : Boolean</b> A flag that indicates whether child objects can receive focus.

42	<b>hasLayoutMatrix3D : Boolean</b> [read-only] Contains true if the element has 3D Matrix.
43	<b>height : Number</b> [override] Number that specifies the height of the component, in pixels, in the parent's coordinates.
44	<b>horizontalCenter : Object</b> For components, this layout constraint property is a facade on top of the similarly-named style.
45	<b>id : String</b> ID of the component.
46	<b>includeInLayout : Boolean</b> Specifies whether this component is included in the layout of the parent container.
47	<b>inheritingStyles : Object</b> The beginning of this component's chain of inheriting styles.
48	<b>initialized : Boolean</b> A flag that determines if an object has been through all three phases of layout: commitment, measurement, and layout (provided that any were required).
49	<b>instanceIndex : int</b> [read-only] The index of a repeated component.
50	<b>instanceIndices : Array</b> An Array containing the indices required to reference this UIComponent object from its parent document.
51	<b>is3D : Boolean</b> [read-only] Contains true when the element is in 3D.
52	<b>isDocument : Boolean</b> [read-only] Contains true if this UIComponent instance is a document object.
53	<b>isPopUp : Boolean</b>

	<p>Set to true by the PopUpManager to indicate that component has been popped up.</p>
54	<p><code>layoutMatrix3D : Matrix3D</code></p> <p>[write-only] The transform matrix that is used to calculate a component's layout relative to its siblings.</p>
55	<p><code>left : Object</code></p> <p>For components, this layout constraint property is a facade on top of the similarly-named style.</p>
56	<p><code>maintainProjectionCenter : Boolean</code></p> <p>When true, the component keeps its projection matrix centered on the middle of its bounding box.</p>
57	<p><code>maxHeight : Number</code></p> <p>The maximum recommended height of the component to be considered by the parent during layout.</p>
58	<p><code>maxWidth : Number</code></p> <p>The maximum recommended width of the component to be considered by the parent during layout.</p>
59	<p><code>measuredHeight : Number</code></p> <p>The default height of the component, in pixels.</p>
60	<p><code>measuredMinHeight : Number</code></p> <p>The default minimum height of the component, in pixels.</p>
61	<p><code>measuredMinWidth : Number</code></p> <p>The default minimum width of the component, in pixels.</p>
62	<p><code>measuredWidth : Number</code></p> <p>The default width of the component, in pixels.</p>
63	<p><code>minHeight : Number</code></p> <p>The minimum recommended height of the component to be considered by the parent during layout.</p>

64	<b>minWidth : Number</b> The minimum recommended width of the component to be considered by the parent during layout.
65	<b>moduleFactory : IFlexModuleFactory</b> A module factory is used as context for using embedded fonts and for finding the style manager that controls the styles for this component.
66	<b>mouseFocusEnabled : Boolean</b> Whether you can receive focus when clicked on.
67	<b>nestLevel : int</b> Depth of this object in the containment hierarchy.
68	<b>nonInheritingStyles : Object</b> The beginning of this component's chain of non-inheriting styles.
69	<b>numAutomationChildren : int</b> [read-only] The number of automation children this container has.
70	<b>owner : DisplayObjectContainer</b> The owner of this IVisualElement object.
71	<b>parent : DisplayObjectContainer</b> [override] [read-only] The parent container or component for this component.
72	<b>parentApplication : Object</b> [read-only] A reference to the Application object that contains this UIComponent instance.
73	<b>parentDocument : Object</b> [read-only] A reference to the parent document object for this UIComponent.
74	<b>percentHeight : Number</b> Specifies the height of a component as a percentage of its parent's size.
75	<b>percentWidth : Number</b>

	Specifies the width of a component as a percentage of its parent's size.
76	<p><code>postLayoutTransformOffsets : mx.geom:TransformOffsets</code></p> <p>Defines a set of adjustments that can be applied to the object's transform in a way that is invisible to its parent's layout.</p>
77	<p><code>processedDescriptors : Boolean</code></p> <p>Set to true after immediate or deferred child creation, depending on which one happens.</p>
78	<p><code>repeater : IRepeater</code></p> <p>[read-only] A reference to the Repeater object in the parent document that produced this UIComponent.</p>
79	<p><code>repeaterIndex : int</code></p> <p>[read-only] The index of the item in the data provider of the Repeater that produced this UIComponent.</p>
80	<p><code>repeaterIndices : Array</code></p> <p>An Array containing the indices of the items in the data provider of the Repeaters in the parent document that produced this UIComponent.</p>
81	<p><code>repeaters : Array</code></p> <p>An Array containing references to the Repeater objects in the parent document that produced this UIComponent.</p>
82	<p><code>right : Object</code></p> <p>For components, this layout constraint property is a facade on top of the similarly-named style.</p>
83	<p><code>rotation : Number</code></p> <p>[override] Indicates the rotation of the DisplayObject instance, in degrees, from its original orientation.</p>
84	<p><code>rotationX : Number</code></p> <p>[override] Indicates the x-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.</p>
85	<code>rotationY : Number</code>

	[override] Indicates the y-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
86	<b>rotationZ</b> : Number [override] Indicates the z-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
87	<b>scaleX</b> : Number [override] Number that specifies the horizontal scaling factor.
88	<b>scaleY</b> : Number [override] Number that specifies the vertical scaling factor.
89	<b>scaleZ</b> : Number [override] Number that specifies the scaling factor along the z axis.
90	<b>screen</b> : Rectangle [read-only] Returns an object that contains the size and position of the base drawing surface for this object.
91	<b>showInAutomationHierarchy</b> : Boolean A flag that determines if an automation object shows in the automation hierarchy.
92	<b>states</b> : Array The view states that are defined for this component.
93	<b>styleDeclaration</b> : CSSStyleDeclaration Storage for the inline inheriting styles on this object.
94	<b>styleManager</b> : IStyleManager2 [read-only] Returns the StyleManager instance used by this component.
95	<b>styleName</b> : Object The class style used by this component.
96	<b>styleParent</b> : IAdvancedStyleClient A component's parent is used to evaluate descendant selectors.

97	<b>systemManager : ISystemManager</b> Returns the SystemManager object used by this component.
98	<b>tabFocusEnabled : Boolean</b> A flag that indicates whether this object can receive focus via the TAB key. This is similar to the tabEnabled property used by the Flash Player. This is usually true for components that handle keyboard input, but some components in controlbars have them set to false because they should not steal focus from another component like an editor.
99	<b>toolTip : String</b> Text to display in the ToolTip.
100	<b>top : Object</b> For components, this layout constraint property is a facade on top of the similarly-named style.
101	<b>transform : flash.geom:Transform</b> [override] An object with properties pertaining to a display object's matrix, color transform, and pixel bounds.
102	<b>transformX : Number</b> Sets the x coordinate for the transform center of the component.
103	<b>transformY : Number</b> Sets the y coordinate for the transform center of the component.
104	<b>transformZ : Number</b> Sets the z coordinate for the transform center of the component.
105	<b>transitions : Array</b> An Array of Transition objects, where each Transition object defines a set of effects to play when a view state change occurs.
106	<b>tweeningProperties : Array</b> Array of properties that are currently being tweened on this object.
107	<b>uid : String</b>

	A unique identifier for the object.
108	<p>updateCompletePendingFlag : Boolean</p> <p>A flag that determines if an object has been through all three phases of layout validation (provided that any were required).</p>
109	<p>validationSubField : String</p> <p>Used by a validator to associate a subfield with this component.</p>
110	<p>verticalCenter : Object</p> <p>For components, this layout constraint property is a facade on top of the similarly-named style.</p>
111	<p>visible : Boolean</p> <p>[override] Whether or not the display object is visible.</p>
112	<p>width : Number</p> <p>[override] Number that specifies the width of the component, in pixels, in the parent's coordinates.</p>
113	<p>x : Number</p> <p>[override] Number that specifies the component's horizontal position, in pixels, within its parent container.</p>
114	<p>y : Number</p> <p>[override] Number that specifies the component's vertical position, in pixels, within its parent container.</p>
115	<p>z : Number</p> <p>[override] Indicates the z coordinate position along the z-axis of the DisplayObject instance relative to the 3D parent container.</p>

## Protected Properties

Following are the Protected Properties for **mx.core.UIComponent** class:

S.N.	Name & Description
1	currentCSSState : String

S.N.	Event & Description
1	activate Dispatched when the Flash Player gains operating system focus and becomes active.
2	detivate Dispatched when the Flash Player loses operating system focus and becomes inactive.
	[read-only] The state to be used when matching CSS pseudo-selectors.
2	hasComplexLayoutMatrix : Boolean [read-only] Returns true if the UIComponent has any non-translation (x,y) transform properties.
3	resourceManager : IResourceManager [read-only] A reference to the object which manages all of the application's localized resources.
4	unscaledHeight : Number [read-only] A convenience method for determining the unscaled height of the component.
5	unscaledWidth : Number [read-only] A convenience method for determining the unscaled width of the component All of a component's drawing and child layout should be done within a bounding rectangle of this width, which is also passed as an argument to updateDisplayList().

## Public Methods

S.N.	Method & Description
1	UIComponent() Constructor.

2	<code>addStyleClient(styleClient:IAdvancedStyleClient):void</code> Adds a non-visual style client to this component instance.
3	<code>callLater(method:Function, args:Array = null):void</code> Queues a function to be called later.
4	<code>clearStyle(styleProp:String):void</code> Deletes a style property from this component instance.
5	<code>contentToGlobal(point:Point):Point</code> Converts a Point object from content coordinates to global coordinates.
6	<code>contentToLocal(point:Point):Point</code> Converts a Point object from content to local coordinates.
7	<code>createAutomationIDPart(child:IAutomationObject):Object</code> Returns a set of properties that identify the child within this container.
8	<code>createAutomationIDPartWithRequiredProperties(child:IAutomationObject, properties:Array):Object</code> Returns a set of properties that identify the child within this container.
9	<code>createReferenceOnParentDocument(parentDocument:IFlexDisplayObject):void</code> Creates an id reference to this UIComponent object on its parent document object.
10	<code>deleteReferenceOnParentDocument(parentDocument:IFlexDisplayObject):void</code> Deletes the id reference to this UIComponent object on its parent document object.
11	<code>determineTextFormatFromStyles():mx.core:UITextFormat</code> Returns a UITextFormat object corresponding to the text styles for this UIComponent.
12	<code>dispatchEvent(event:Event):Boolean</code> [override] Dispatches an event into the event flow.
13	<code>drawFocus(isFocused:Boolean):void</code>

	Shows or hides the focus indicator around this component.
14	<b>drawRoundRect(x:Number, y:Number, w:Number, h:Number, r:Object = null, c:Object = null, alpha:Object = null, rot:Object = null, gradient:String = null, ratios:Array = null, hole:Object = null):void</b> Programmatically draws a rectangle into this skin's Graphics object.
15	<b>effectFinished(effectInst:IEffectInstance):void</b> Called by the effect instance when it stops playing on the component.
16	<b>effectStarted(effectInst:IEffectInstance):void</b> Called by the effect instance when it starts playing on the component.
17	<b>endEffectsStarted():void</b> Ends all currently playing effects on the component.
18	<b>executeBindings(recurse:Boolean = false):void</b> Executes all the bindings for which the UIComponent object is the destination.
19	<b>finishPrint(obj:Object, target:IFlexDisplayObject):void</b> Called after printing is complete.
20	<b>getAutomationChildAt(index:int):IAutomationObject</b> Provides the automation object at the specified index.
21	<b>getAutomationChildren():Array</b> Provides the automation object list .
22	<b>getBoundsXAtSize(width:Number, postLayoutTransform:Boolean = true):Number</b> height:Number, Returns the x coordinate of the element's bounds at the specified element size.
23	<b>getBoundsYAtSize(width:Number, postLayoutTransform:Boolean = true):Number</b> height:Number, Returns the y coordinate of the element's bounds at the specified element size.
24	<b>getClassStyleDeclarations():Array</b> Finds the type selectors for this UIComponent instance.

25	<code>getConstraintValue(constraintName:String):*</code> Returns a layout constraint value, which is the same as getting the constraint style for this component.
26	<code>getExplicitOrMeasuredHeight():Number</code> A convenience method for determining whether to use the explicit or measured height
27	<code>getExplicitOrMeasuredWidth():Number</code> A convenience method for determining whether to use the explicit or measured width
28	<code>getFocus():InteractiveObject</code> Gets the object that currently has focus.
29	<code>getLayoutBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's layout height.
30	<code>getLayoutBoundsWidth(postLayoutTransform:Boolean = true):Number</code> Returns the element's layout width.
31	<code>getLayoutBoundsX(postLayoutTransform:Boolean = true):Number</code> Returns the x coordinate that the element uses to draw on screen.
32	<code>getLayoutBoundsY(postLayoutTransform:Boolean = true):Number</code> Returns the y coordinate that the element uses to draw on screen.
33	<code>getLayoutMatrix():Matrix</code> Returns the transform matrix that is used to calculate the component's layout relative to its siblings.
34	<code>getLayoutMatrix3D():Matrix3D</code> Returns the layout transform Matrix3D for this element.
35	<code>getMaxBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's maximum height.
36	<code>getMaxBoundsWidth(postLayoutTransform:Boolean = true):Number</code>

	Returns the element's maximum width.
37	getMinBoundsHeight(postLayoutTransform:Boolean = true):Number Returns the element's minimum height.
38	getMinBoundsWidth(postLayoutTransform:Boolean = true):Number Returns the element's minimum width.
39	getPreferredBoundsHeight(postLayoutTransform:Boolean = true):Number Returns the element's preferred height.
40	getPreferredBoundsWidth(postLayoutTransform:Boolean = true):Number Returns the element's preferred width.
41	getRepeaterItem(whichRepeater:int = -1):Object Returns the item in the dataProvider that was used by the specified Repeater to produce this Repeater, or null if this Repeater isn't repeated.
42	getStyle(styleProp:String):* Gets a style property that has been set anywhere in this component's style lookup chain.
43	globalToContent(point:Point):Point Converts a Point object from global to content coordinates.
45	hasCSSState():Boolean Returns true if currentCSSState is not null.
46	hasState(stateName:String):Boolean Determines whether the specified state has been defined on this UIComponent.
47	horizontalGradientMatrix(x:Number, y:Number, width:Number, height:Number):Matrix Returns a box Matrix which can be passed to the drawRoundRect() method as the rot parameter when drawing a horizontal gradient.
48	initialize():void Initializes the internal structure of this component.

49	<code>initializeRepeaterArrays(parent:IRepeaterClient):void</code> Initializes various properties which keep track of repeated instances of this component.
50	<code>invalidateDisplayList():void</code> Marks a component so that its <code>updateDisplayList()</code> method gets called during a later screen update.
51	<code>invalidateLayering():void</code> Called by a component's items to indicate that their depth property has changed.
52	<code>invalidateLayoutDirection():void</code> An element must call this method when its <code>layoutDirection</code> changes or when its parent's <code>layoutDirection</code> changes.
53	<code>invalidateProperties():void</code> Marks a component so that its <code>commitProperties()</code> method gets called during a later screen update.
54	<code>invalidateSize():void</code> Marks a component so that its <code>measure()</code> method gets called during a later screen update.
55	<code>localToContent(point:Point):Point</code> Converts a <code>Point</code> object from local to content coordinates.
56	<code>matchesCSSState(cssState:String):Boolean</code> Returns true if <code>cssState</code> matches <code>currentCSSState</code> .
57	<code>matchesCSSType(cssType:String):Boolean</code> Determines whether this instance is the same as, or is a subclass of, the given type.
58	<code>measureHTMLText(htmlText:String):flash.text:TextLineMetrics</code> Measures the specified HTML text, which can contain HTML tags such as <code>&amp;lt;font&amp;ampgt</code> and <code>&amp;lt;b&amp;ampgt</code> , assuming that it is displayed in a single-line <code>UITextField</code> using a <code>UITextFormat</code> determined by the styles of this <code>UIComponent</code> .
59	<code>measureText(text:String):flash.text:TextLineMetrics</code>

	Measures the specified text, assuming that it is displayed in a single-line UITextField (or UIFETextField) using a UITextFormat determined by the styles of this UIComponent.
60	<code>move(x:Number, y:Number):void</code> Moves the component to a specified position within its parent.
61	<code>notifyStyleChangeInChildren(styleProp:String, recursive:Boolean):void</code> Propagates style changes to the children.
62	<code>owns(child:DisplayObject):Boolean</code> Returns true if the chain of owner properties points from child to this UIComponent.
63	<code>parentChanged(p:DisplayObjectContainer):void</code> Called by Flex when a UIComponent object is added to or removed from a parent.
64	<code>prepareToPrint(target:IFlexDisplayObject):Object</code> Prepares an IFlexDisplayObject for printing.
65	<code>regenerateStyleCache(recursive:Boolean):void</code> Builds or rebuilds the CSS style cache for this component and, if the recursive parameter is true, for all descendants of this component as well.
66	<code>registerEffects(effects:Array):void</code> For each effect event, registers the EffectManager as one of the event listeners.
67	<code>removeStyleClient(styleClient:IAdvancedStyleClient):void</code> Removes a non-visual style client from this component instance.
68	<code>replayAutomatableEvent(event:Event):Boolean</code> Replays the specified event.
69	<code>resolveAutomationIDPart(criteria:Object):Array</code> Resolves a child by using the id provided.
70	<code>resumeBackgroundProcessing():void</code>

	[static] Resumes the background processing of methods queued by callLater(), after a call to suspendBackgroundProcessing().
71	<code>setActualSize(w:Number, h:Number):void</code> Sizes the object.
72	<code>setConstraintValue(constraintName:String, value:*):void</code> Sets a layout constraint value, which is the same as setting the constraint style for this component.
73	<code>setCurrentState(stateName:String, playTransition:Boolean = true):void</code> Set the current state.
74	<code>setFocus():void</code> Sets the focus to this component.
75	<code>setLayoutBoundsPosition(x:Number, y:Number, postLayoutTransform:Boolean = true):void</code> Sets the coordinates that the element uses to draw on screen.
76	<code>set Layout Bounds Size(width:Number, height:Number, post Layout Transform:Boolean = true):void</code> Sets the layout size of the element.
77	<code>setLayoutMatrix(value:Matrix, invalidateLayout:Boolean):void</code> Sets the transform Matrix that is used to calculate the component's layout size and position relative to its siblings.
78	<code>setLayoutMatrix3D(value:Matrix3D, invalidateLayout:Boolean):void</code> Sets the transform Matrix3D that is used to calculate the component's layout size and position relative to its siblings.
79	<code>setStyle(styleProp:String, newValue:*):void</code> Sets a style property on this component instance.
80	<code>setVisible(value:Boolean, noEvent:Boolean = false):void</code> Called when the visible property changes.

81	<code>styleChanged(styleProp:String):void</code> Detects changes to style properties.
82	<code>stylesInitialized():void</code> Flex calls the <code>stylesInitialized()</code> method when the styles for a component are first initialized.
83	<code>suspendBackgroundProcessing():void</code> [static] Blocks the background processing of methods queued by <code>callLater()</code> , until <code>resumeBackgroundProcessing()</code> is called.
84	<code>transformAround(transformCenter:Vector3D, scale:Vector3D = null, rotation:Vector3D = null, translation:Vector3D = null, postLayoutScale:Vector3D = null, postLayoutRotation:Vector3D = null, postLayoutTranslation:Vector3D = null, invalidateLayout:Boolean = true):void</code> A utility method to update the rotation, scale, and translation of the transform while keeping a particular point, specified in the component's own coordinate space, fixed in the parent's coordinate space.
85	<code>transform Point To Parent (localPosition:Vector3D, position:Vector3D, post Layout Position:Vector3D):void</code> A utility method to transform a point specified in the local coordinates of this object to its location in the object's parent's coordinates.
86	<code>validateDisplayList():void</code> Validates the position and size of children and draws other visuals.
87	<code>validateNow():void</code> Validate and update the properties and layout of this object and redraw it, if necessary.
88	<code>validateProperties():void</code> Used by layout logic to validate the properties of a component by calling the <code>commitProperties()</code> method.
89	<code>validateSize(recursive:Boolean = false):void</code> Validates the measured size of the component If the <code>LayoutManager.invalidateSize()</code> method is called with this <code>ILayoutManagerClient</code> , then the <code>validateSize()</code> method is called when it's time to do measurements.

90	<b>validationResultHandler(event:ValidationResultEvent):void</b> Handles both the valid and invalid events from a validator assigned to this component.
91	<b>verticalGradientMatrix(x:Number, y:Number, width:Number, height:Number):Matrix</b> Returns a box Matrix which can be passed to drawRoundRect() as the rot parameter when drawing a vertical gradient.

## Protected Methods

<b>S.N.</b>	<b>Method &amp; Description</b>
1	<b>adjustFocusRect(obj:DisplayObject = null):void</b> Adjust the focus rectangle.
2	<b>applyComputedMatrix():void</b> Commits the computed matrix built from the combination of the layout matrix and the transform offsets to the flash displayObject's transform.
3	<b>attachOverlay():void</b> This is an internal method used by the Flex framework to support the Dissolve effect.
4	<b>canSkipMeasurement():Boolean</b> Determines if the call to the measure() method can be skipped.
5	<b>childrenCreated():void</b> Performs any final processing after child objects are created.
6	<b>commitProperties():void</b> Processes the properties set on the component.
7	<b>createChildren():void</b> Create child objects of the component.
8	<b>createInFontContext(classObj:Class):Object</b> Creates a new object using a context based on the embedded font being used.

9	<code>createInModuleContext(moduleFactory:IFlexModuleFactory, className:String):Object</code> Creates the object using a given moduleFactory.
10	<code>dispatchPropertyChangeEvent(prop:String, oldValue:*, value:*):void</code> Helper method for dispatching a PropertyChangeEvent when a property is updated.
11	<code>focusInHandler(event:FocusEvent):void</code> The event handler called when a UIComponent object gets focus.
12	<code>focusOutHandler(event:FocusEvent):void</code> The event handler called when a UIComponent object loses focus.
13	<code>initAdvancedLayoutFeatures():void</code> Initializes the implementation and storage of some of the less frequently used advanced layout features of a component.
14	<code>initializationComplete():void</code> Finalizes the initialization of this component.
15	<code>initializeAccessibility():void</code> Initializes this component's accessibility code.
16	<code>invalidateParentSizeAndDisplayList():void</code> Helper method to invalidate parent size and display list if this object affects its layout (includeInLayout is true).
17	<code>isOurFocus(target:DisplayObject):Boolean</code> Typically overridden by components containing UITextField objects, where the UITextField object gets focus.
18	<code>keyDownHandler(event:KeyboardEvent):void</code> The event handler called for a keyDown event.
19	<code>keyUpHandler(event:KeyboardEvent):void</code> The event handler called for a keyUp event.

20	<b>measure():void</b> Calculates the default size, and optionally the default minimum size, of the component.
21	<b>resourcesChanged():void</b> This method is called when a UIComponent is constructed, and again whenever the ResourceManager dispatches a "change" Event to indicate that the localized resources have changed in some way.
22	<b>setStretchXY(stretchX:Number, stretchY:Number):void</b> Specifies a transform stretch factor in the horizontal and vertical direction.
23	<b>stateChanged(oldState:String, newState:String, recursive:Boolean):void</b> This method is called when a state changes to check whether state-specific styles apply to this component
24	<b>updateDisplayList(unscaledWidth:Number, unscaledHeight:Number):void</b> Draws the object and/or sizes and positions its children.

## Events

Following are the events for **mx.core.UIComponent** class:

S.N.	Event & Description
1	<b>add</b> when the component is added to a container as a content child by using the addChild(), addChildAt(), addElement(), or addElementAt() method.
2	<b>creationComplete</b> when the component has finished its construction, property processing, measuring, layout, and drawing.
3	<b>currentStateChange</b> after the view state has changed.
4	<b>currentStateChanging</b> after the currentState property changes, but before the view state changes.

5	<b>dragComplete</b> by the drag initiator (the component that is the source of the data being dragged) when the drag operation completes, either when you drop the dragged data onto a drop target or when you end the drag-and-drop operation without performing a drop.
6	<b>dragDrop</b> by the drop target when the user releases the mouse over it.
7	<b>dragEnter</b> by a component when the user moves the mouse over the component during a drag operation.
8	<b>dragExit</b> by the component when the user drags outside the component, but does not drop the data onto the target.
9	<b>dragOver</b> by a component when the user moves the mouse while over the component during a drag operation.
10	<b>dragStart</b> by the drag initiator when starting a drag operation.
11	<b>effectEnd</b> after an effect ends.
12	<b>effectStart</b> just before an effect starts.
13	<b>effectStop</b> after an effect is stopped, which happens only by a call to stop() on the effect.
14	<b>enterState</b> after the component has entered a view state.
15	<b>exitState</b> just before the component exits a view state.

16	hide when an object's state changes from visible to invisible.
17	initialize when the component has finished its construction and has all initialization properties set.
18	invalid when a component is monitored by a Validator and the validation failed.
19	mouseDownOutside from a component opened using the PopUpManager when the user clicks outside it.
20	mouseWheelOutside from a component opened using the PopUpManager when the user scrolls the mouse wheel outside it.
21	move when the object has moved.
22	preinitialize at the beginning of the component initialization sequence.
23	remove when the component is removed from a container as a content child by using the removeChild(), removeChildAt(), removeElement(), or removeElementAt() method.
24	resize when the component is resized.
25	show when an object's state changes from invisible to visible.
26	stateChangeComplete after the component has entered a new state and any state transition animation to that state has finished playing.

27	<b>stateChangeInterrupted</b> when a component interrupts a transition to its current state in order to switch to a new state.
28	<b>toolTipCreate</b> by the component when it is time to create a ToolTip.
29	<b>toolTipEnd</b> by the component when its ToolTip has been hidden and is to be discarded soon.
30	<b>toolTipHide</b> by the component when its ToolTip is about to be hidden.
31	<b>toolTipShow</b> by the component when its ToolTip is about to be shown.
32	<b>toolTipShown</b> by the component when its ToolTip has been shown.
33	<b>toolTipStart</b> by a component whose toolTip property is set, as soon as the user moves the mouse over it.
34	<b>touchInteractionEnd</b> A non-cancellable event, by a component when it is done responding to a touch interaction user gesture.
35	<b>touchInteractionStart</b> A non-cancellable event, by a component when it starts responding to a touch interaction user gesture.
36	<b>touchInteractionStarting</b> A cancellable event, by a component in an attempt to respond to a touch interaction user gesture.
37	<b>updateComplete</b> when an object has had its commitProperties(), measure(), and updateDisplayList() methods called (if needed).

38	<b>valid</b> when a component is monitored by a Validator and the validation succeeded.
39	<b>valueCommit</b> when values are changed programmatically or by user interaction.

## Methods Inherited

This class inherits methods from the following classes:

- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Form Controls

---

Following are few important Form Controls:

S.N.	Control & Description
1	Button  The Button component is a commonly used rectangular button.
2	ToggleButton  The ToggleButton component defines a toggle button.
3	CheckBox  The CheckBox component consists of an optional label and a small box that can contain a check mark or not.
4	ColorPicker  The ColorPicker control provides a way for a user to choose a color from a swatch list.
5	ComboBox  The ComboBox control is a child class of the DropDownListBase control.
6	DateChooser  The DateChooser control displays the name of a month, the year, and a grid of the days of the month, with columns labeled for the day of the week.
7	RadioButton

	The RadioButton component allows the user make a single choice within a set of mutually exclusive choices.
8	<b>TextArea</b> TextArea is a text-entry control that lets users enter and edit multiple lines of formatted text.
9	<b>TextInput</b> TextInput is a text-entry control that lets users enter and edit a single line of uniformly-formatted text.
10	<b>DropDownList</b> The DropDownList control contains a drop-down list from which the user can select a single value.
11	<b>NumericStepper</b> The NumericStepper control lets you select a number from an ordered set.

## Flex – Button Control

The Button component is a commonly used rectangular button. Button typically uses event listeners to perform an action when the user selects the control. When a user clicks on a Button control, then it dispatches a click event and a buttonDown event.

### Class Declaration

Following is the declaration for **spark.components.Button** class:

```
public class Button
  extends ButtonBase
  implements IButton
```

### Public Properties

S.N.	Property & Description
1	<b>emphasized</b> : Boolean Reflects the default button as requested by the focus manager.

### Public Methods

S.N.	Method & Description

	1      Button()
	Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.ButtonBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex Button Control Example

Let us follow the following steps to check usage of Button control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
```

```

>

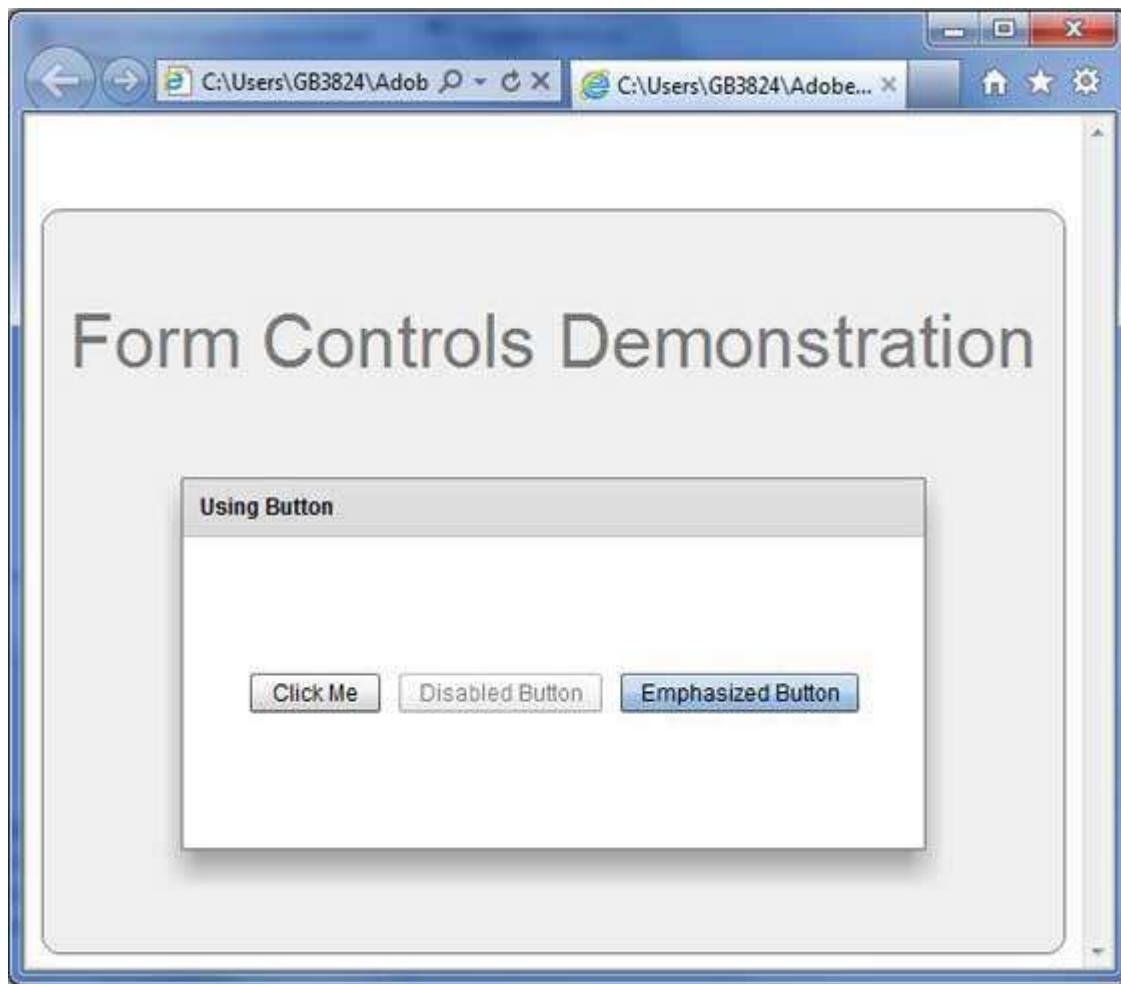
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<fx:Script>
<![CDATA[
    import mx.controls.Alert;
    import mx.events.FlexEvent;

    protected function button_clickHandler(event:MouseEvent):void
    {
        Alert.show("Hello World!");
    }
]]>
</fx:Script>
<s:BorderContainer width="550" height="400" id="mainContainer"
styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
    horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Form Controls Demonstration"
        fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="buttonPanel" title="Using Button" width="400"
        height="200" includeInLayout="true" visible="true">
            <s:layout>
                <s:HorizontalLayout gap="10" verticalAlign="middle"
                horizontalAlign="center"/>
            </s:layout>
            <s:Button id="clickMeButton" label="Click Me"
            click="button_clickHandler(event)" />
            <s:Button id="disabledButton" label="Disabled Button"
            enabled="false" />
            <s:Button id="emphasizedButton" label="Emphasized Button"
            emphasized="true" click="button_clickHandler(event)"/>
        </s:Panel>
    </s:VGroup>

```

```
</s:BorderContainer>  
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – ToggleButton Control

The **ToggleButton** component defines a toggle button. Clicking the button toggles the button between the up and an down states. When you click the button while it is in the up state, it toggles to the down state and so on.

### Class Declaration

Following is the declaration for **spark.components.ToggleButton** class:

```
public class ToggleButton  
    extends ToggleButtonBase
```

## Public Methods

S.N.	Method & Description
1	ToggleButton() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.ToggleButtonBase
- spark.components.supportClasses.ButtonBase
- spark.component.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex ToggleButton Control Example

Let us follow the following steps to check usage of ToggleButton control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
```

```

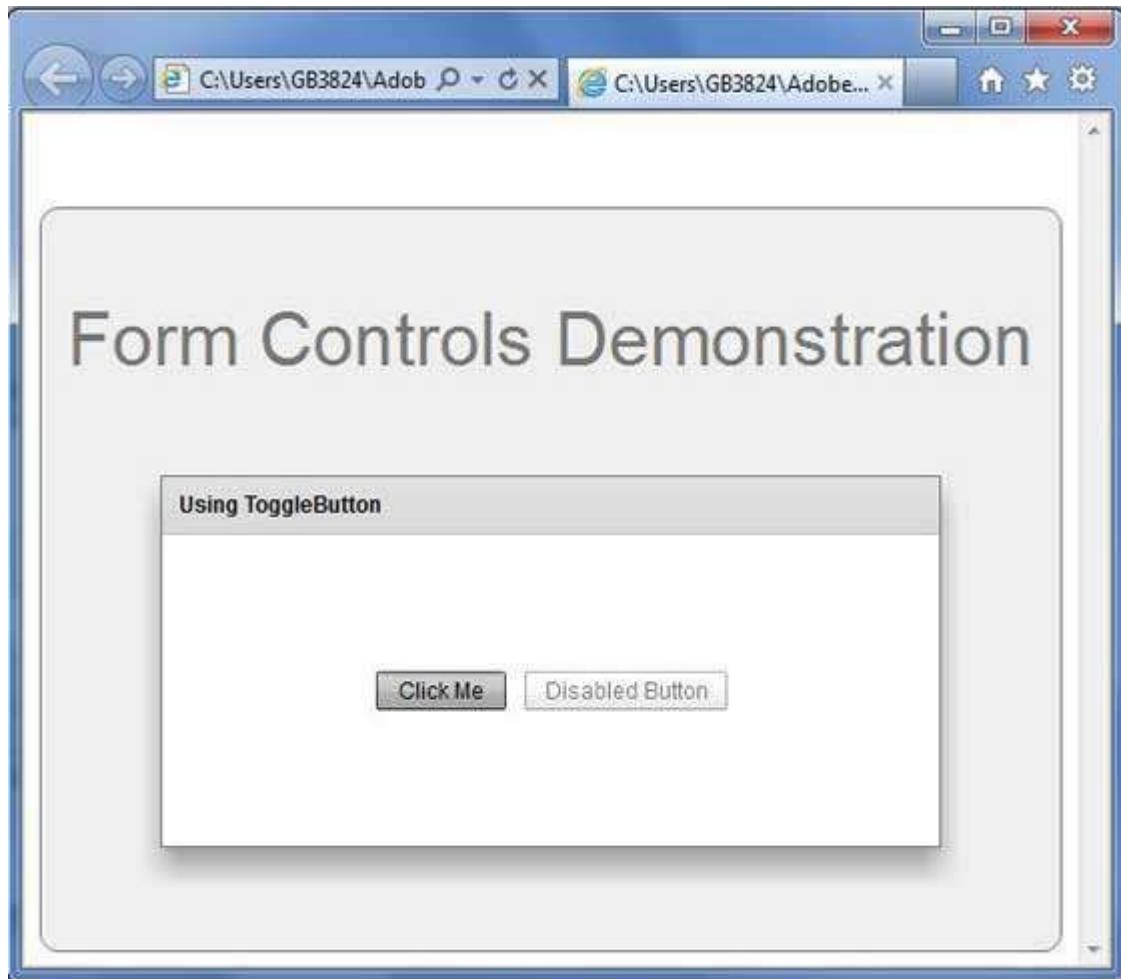
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx"
width="100%" height="100%" minWidth="500" minHeight="500"
>
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<fx:Script>
<![CDATA[
    import mx.controls.Alert;
    import mx.events.FlexEvent;

    protected function button_clickHandler(event:MouseEvent):void
    {
        Alert.show("Hello World!");
    }
]]>
</fx:Script>
<s:BorderContainer width="550" height="400" id="mainContainer"
styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
    horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Form Controls Demonstration"
        fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="toggleButtonPanel" title="Using ToggleButton"
        width="420" height="200">
            <s:layout>
                <s:HorizontalLayout gap="10" verticalAlign="middle"
                horizontalAlign="center"/>
            </s:layout>
            <s:ToggleButton id="clickMeToggleButton" label="Click Me"
            click="button_clickHandler(event)" />
            <s:ToggleButton id="disabledToggleButton" label="Disabled
Button"
            enabled="false" />
        </s:Panel>
    </s:VGroup>

```

```
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex - CheckBox Control

The CheckBox component represents a checkbox and an optional label. When a user clicks a CheckBox component or its text, then CheckBox component sets its selected property to true for checked, and to false for unchecked.

### Class Declaration

Following is the declaration for **spark.components.CheckBox** class:

```
public class CheckBox
    extends ToggleButtonBase
```

## Public Methods

S.N.	Method & Description
1	CheckBox() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.ToggleButtonBase
- spark.components.supportClasses.ButtonBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex CheckBox Control Example

Let us follow the following steps to check usage of CheckBox control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

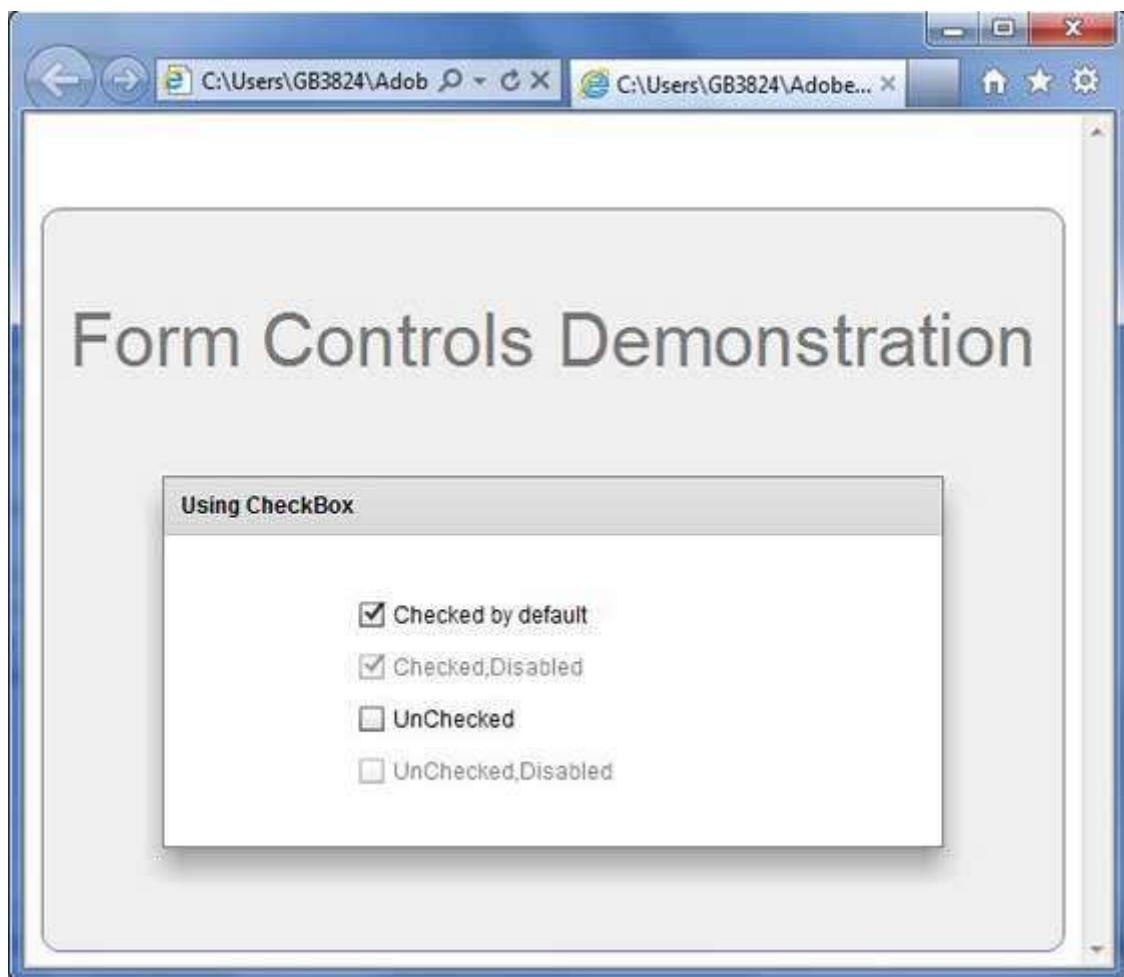
Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```

<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
>
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<s:BorderContainer width="550" height="400" id="mainContainer"
styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Form Controls Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="checkBoxPanel" title="Using CheckBox" width="420"
            height="200" >
            <s:layout>
                <s:VerticalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center"/>
            </s:layout>
            <s:CheckBox id="chkBox" label="Checked by default"
                selected="true" width="50%"/>
            <s:CheckBox id="chkBox1" label="Checked,Disabled"
                selected="true" enabled="false" width="50%"/>
            <s:CheckBox id="chkBox2" label="UnChecked" width="50%"/>
            <s:CheckBox id="chkBox3" label="UnChecked,Disabled"
                enabled="false" width="50%"/>
        </s:Panel >
    </s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – ColorPicker Control

---

### Introduction

The ColorPicker control provides user a way to choose a color from a swatch list. The default mode is to show a single swatch in a square button.

When the user clicks the swatch button, the swatch panel appears and displays the entire swatch list.

### Class Declaration

Following is the declaration for **mx.controls.ColorPicker** class:

```
public class ColorPicker
    extends ComboBox
```

### Public Properties

S.N.	Property & Description
------	------------------------

1	<b>colorField : String</b> Name of the field in the objects of the dataProvider Array that specifies the hexadecimal values of the colors that the swatch panel displays.
2	<b>labelField : String</b> Name of the field in the objects of the dataProvider Array that contain text to display as the label in the SwatchPanel object text box.
3	<b>selectedColor : uint</b> The value of the currently selected color in the SwatchPanel object.
4	<b>selectedIndex : int</b> [override] Index in the dataProvider of the selected item in the SwatchPanel object.
5	<b>showTextField : Boolean</b> Specifies whether to show the text box that displays the color label or hexadecimal color value.

## Protected Properties

S.N.	Property & Description
1	<b>swatchStyleFilters : Object</b> [read-only] Set of styles to pass from the ColorPicker through to the preview swatch.

## Public Methods

S.N.	Method & Description
1	<b>ColorPicker()</b> Constructor.
2	<b>close(trigger:Event = null):void</b> Hides the drop-down SwatchPanel object.
3	<b>open():void</b>

	Displays the drop-down SwatchPanel object that shows colors that users can select.
--	--

## Events

S.N.	Event & Description
1	change Dispatched when the selected color changes as a result of user interaction.
2	close Dispatched when the swatch panel closes.
3	enter Dispatched if the ColorPicker editable property is set to true and the user presses Enter after typing in a hexadecimal color value.
4	itemRollOut Dispatched when the user rolls the mouse out of a swatch in the SwatchPanel object.
5	itemRollOver Dispatched when the user rolls the mouse over a swatch in the SwatchPanel object.
6	open Dispatched when the color swatch panel opens.

## Methods Inherited

This class inherits methods from the following classes:

- mx.controls.comboBase
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex ColorPicker Control Example

Let us follow the following steps to check usage of ColorPicker control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

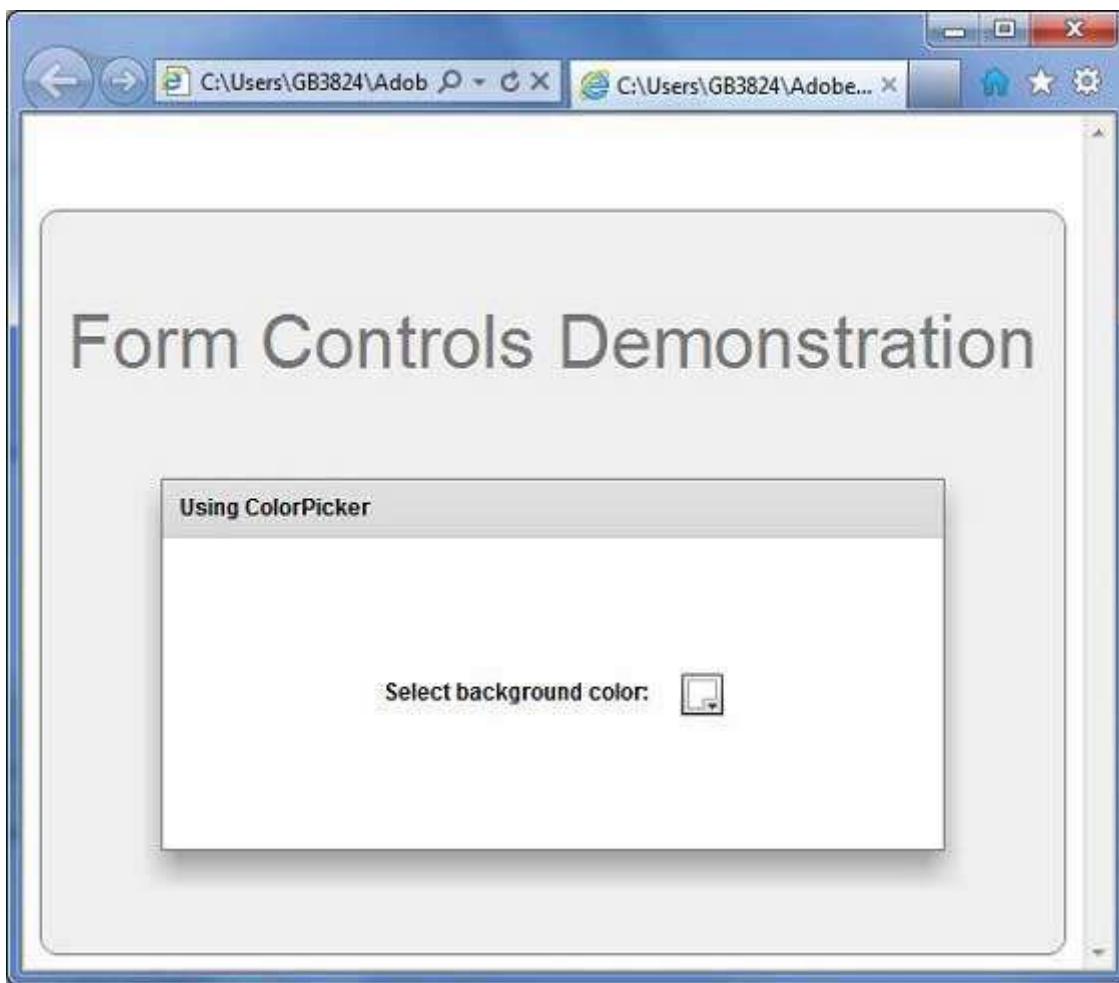
```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <s:BorderContainer width="550" height="400" id="mainContainer"
        styleName="container">
        <s:VGroup width="100%" height="100%" gap="50"
            horizontalAlign="center" verticalAlign="middle">
            <s:Label id="lblHeader" text="Form Controls Demonstration"
                fontSize="40" color="0x777777" styleName="heading"/>
            <s:Panel id="colorPickerPanel"
                backgroundColor="{colorPicker.selectedColor}"
                title="Using ColorPicker" width="420" height="200">
                <s:layout>
                    <s:HorizontalLayout gap="10" verticalAlign="middle"
                        horizontalAlign="center"/>
                </s:layout>
            
```

```

<s:Label width="150" color="black"
text="Select background color: " fontWeight="bold"/>
<mx:ColorPicker id="colorPicker"
showTextField="true" selectedColor="#FFFFFF"/>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – ComboBox Control

When the user selects an item from the drop-down list in the ComboBox control, the data item appears in the prompt area of the control.

One difference between the ComboBox and DropDownList controls is that the prompt area of the ComboBox control is implemented by using the TextInput control, whereas in case

of DropDownList control, it is Label control. Therefore, a user can edit the prompt area of the control to enter a value that is not one of the options available.

## Class Declaration

Following is the declaration for **spark.components.ComboBox** class:

```
public class ComboBox
    extends DropDownListBase
    implements IIIMESupport
```

## Public Properties

S.N.	Property & Description
1	enableIME : Boolean [read-only]
2	imeMode : String
3	itemMatchingFunction : Function = null  Specifies a callback function used to search the item list as the user enters characters into the prompt area.
4	labelToItemFunction : Function  Specifies a callback function to convert a new value entered into the prompt area to the same data type as the data items in the data provider.
5	maxChars : int  The maximum number of characters that the prompt area can contain, as entered by a user.
6	openOnInput : Boolean = true  If true, the drop-down list opens when the user edits the prompt area.
7	prompt : String  Text to be displayed if/when the input text is null.
8	restrict : String

	Specifies the set of characters that a user can enter into the prompt area.
--	---

## Public Methods

S.N.	Method & Description
1	ComboBox() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.DropDownListBase
- spark.components.List
- spark.components.supportClasses.ListBase
- spark.components.SkinnableDataContainer
- spark.components.supportClasses.SkinnableContainerBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex ComboBox Control Example

Let us follow the following steps to check usage of ComboBox control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.

- |   |   |
|---|---|
| 3 | Compile and run the application to make sure business logic is working as per the requirements. |
|---|---|

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >

    <fx:Style source="/com/tutorialspoint/client/Style.css"/>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            public var data:ArrayCollection = new ArrayCollection(
                [
                    {value:"France", code:"FR"},
                    {value:"Japan", code:"JP"},
                    {value:"India", code:"IN"},
                    {value:"Russia", code:"RS"},
                    {value:"United States", code:"US"}
                ]
            );
        ]>
    </fx:Script>

```

```
    private function mappingFunction(input:String):Object
    {
        switch (input){
            case "France":
                return {value:input, code:"FR"};
            case "Japan":
```

```

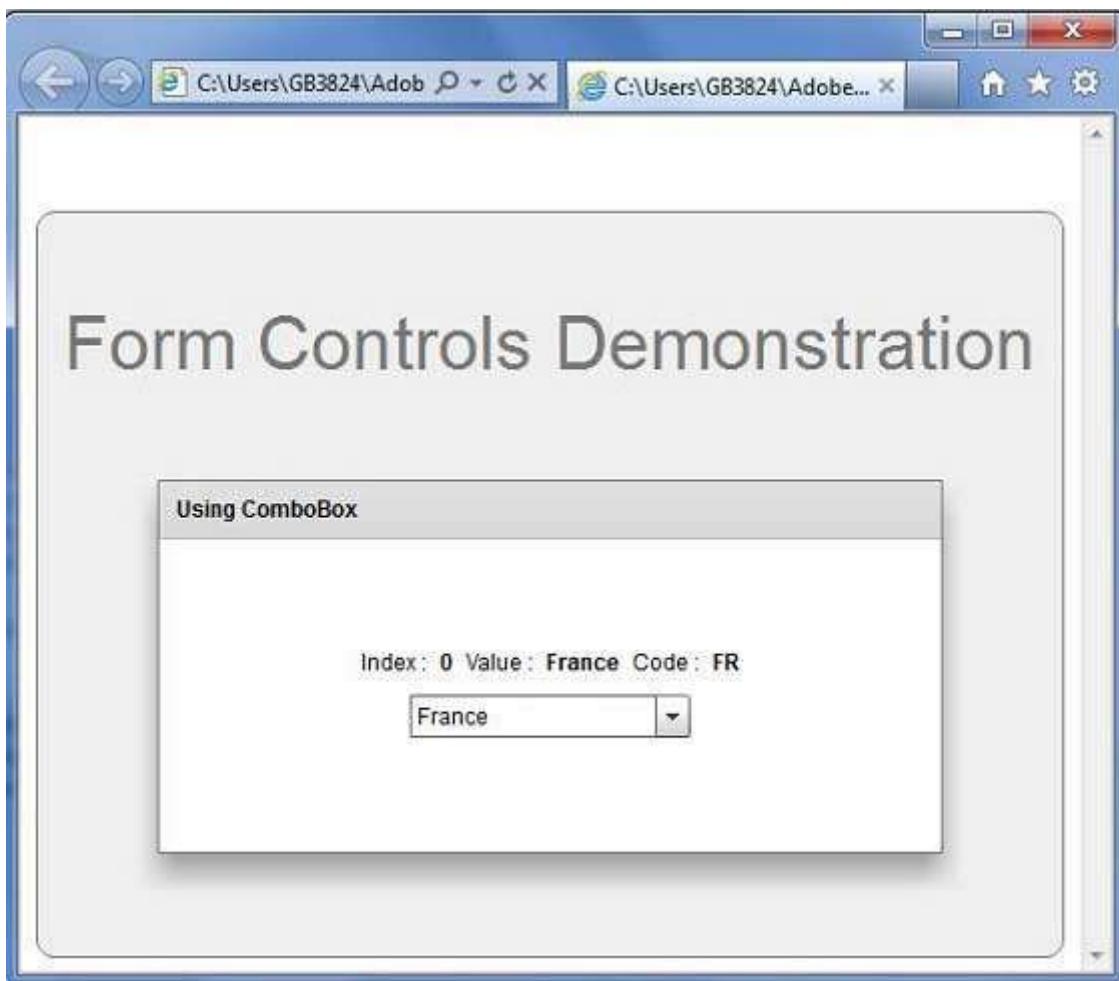
        return {value:input, code:"JP"};
    case "India":
        return {value:input, code:"IN"};
    case "Russia":
        return {value:input, code:"RS"};
    case "United States":
        return {value:input, code:"US"};
    }
    return null;
}

]]>
</fx:Script>
<fx:Declarations>
    <mx:DateFormatter id="dateFormatter" />
</fx:Declarations>
<s:BorderContainer width="550" height="400" id="mainContainer"
styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center"
verticalAlign="middle">
        <s:Label id="lblHeader" text="Form Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="comboBoxPanel" title="Using ComboBox" width="420"
height="200">
            <s:layout>
                <s:VerticalLayout gap="10" verticalAlign="middle"
horizontalAlign="center"/>
            </s:layout>
            <s:HGroup>
                <s:Label text="Index :"/> <s:Label
text="{comboBox.selectedIndex}" fontWeight="bold"/>
                <s:Label text="Value :"/> <s:Label
text="{comboBox.selectedItem.value}" fontWeight="bold"/>
                <s:Label text="Code :"/> <s:Label

```

```
        text="{comboBox.selectedItem.code}" fontWeight="bold"/>
    </s:HGroup>
    <s:ComboBox id="comboBox" dataProvider="{data}" width="150"
        labelToItemFunction="{mappingFunction}" selectedIndex="0"
        labelField="value"/>
    </s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – DateChooser Control

The DateChooser control is used to display the name of a month, the year, and a grid of the days of the month, with columns labeled for the day of the week.

The DateChooser control let the user to select a date, a range of dates, or multiple dates. The control contains forward and back arrow buttons for changing the month and year.

## Class Declaration

Following is the declaration for **mx.controls.DateChooser** class:

```
public class DateChooser
    extends UIComponent
    implements IFocusManagerComponent, IFontContextComponent
```

## Public Properties

S.N.	Property & Description
1	allowDisjointSelection : Boolean If true, specifies that non-contiguous(disjoint) selection is allowed in the DateChooser control.
2	allowMultipleSelection : Boolean If true, specifies that multiple selection is allowed in the DateChooser control.
3	dayNames : Array The weekday names for DateChooser control.
4	disabledDays : Array The days to disable in a week.
5	disabledRanges : Array Disables single and multiple days.
6	displayedMonth : int Used together with the displayedYear property, the displayedMonth property specifies the month displayed in the DateChooser control.
7	displayedYear : int Used together with the displayedMonth property, the displayedYear property specifies the year displayed in the DateChooser control.
8	firstDayOfWeek : Object

	Number representing the day of the week to display in the first column of the DateChooser control.
9	<b>maxYear</b> : int The last year selectable in the control.
10	<b>minYear</b> : int The first year selectable in the control.
11	<b>monthNames</b> : Array Names of the months displayed at the top of the DateChooser control.
12	<b>monthSymbol</b> : String This property is appended to the end of the value specified by the monthNames property to define the names of the months displayed at the top of the DateChooser control.
13	<b>selectableRange</b> : Object Range of dates between which dates are selectable.
14	<b>selectedDate</b> : Date Date selected in the DateChooser control.
15	<b>selectedRanges</b> : Array Selected Date ranges.
16	<b>showToday</b> : Boolean If true, specifies that today is highlighted in the DateChooser control.
17	<b>yearNavigationEnabled</b> : Boolean Enables year navigation.
18	<b>yearSymbol</b> : String This property is appended to the end of the year displayed at the top of the DateChooser control.

## Protected Properties

S.N.	Property & Description
1	calendarLayoutStyleFilters : Object [read-only] The set of styles to pass from the DateChooser to the calendar layout.
2	nextMonthStyleFilters : Object [read-only] The set of styles to pass from the DateChooser to the next month button.
3	nextYearStyleFilters : Object [read-only] The set of styles to pass from the DateChooser to the next year button.
4	prevMonthStyleFilters : Object [read-only] The set of styles to pass from the DateChooser to the previous month button.
5	prevYearStyleFilters : Object [read-only] The set of styles to pass from the DateChooser to the previous year button.

## Public Methods

S.N.	Method & Description
1	DateChooser() Constructor.

## Events

S.N.	Event & Description
1	<b>change</b> Dispatched when a date is selected or changed.
2	<b>scroll</b>

	Dispatched when the month changes due to user interaction.
--	--

## Methods Inherited

This class inherits methods from the following classes:

- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex DateChooser Control Example

Let us follow the following steps to check usage of DateChooser control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    applicationComplete="application_applicationCompleteHandler(event)">

    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
```

```

<fx:Script>
<![CDATA[
    import mx.events.CalendarLayoutChangeEvent;
    import mx.events.FlexEvent;

    [Bindable]
    private var selectedDate:String = "";

    protected function dateChooser_changeHandler
    (event:CalendarLayoutChangeEvent):void
    {
        var date:Date = DateChooser(event.target).selectedDate;
        selectedDate = dateFormatter.format(date);
    }

    protected function application_applicationCompleteHandler
    (event:FlexEvent):void
    {
        selectedDate = dateFormatter.format(new Date());
    }
]]>
</fx:Script>
<s:BorderContainer width="550" height="400" id="mainContainer"
styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
    horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Form Controls Demonstration"
        fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="dateChooserPanel" title="Using DateChooser"
width="400"
        height="300" includeInLayout="true" visible="true">
            <s:layout>
                <s:HorizontalLayout gap="10" verticalAlign="middle"
                horizontalAlign="center"/>
            </s:layout>
        
```

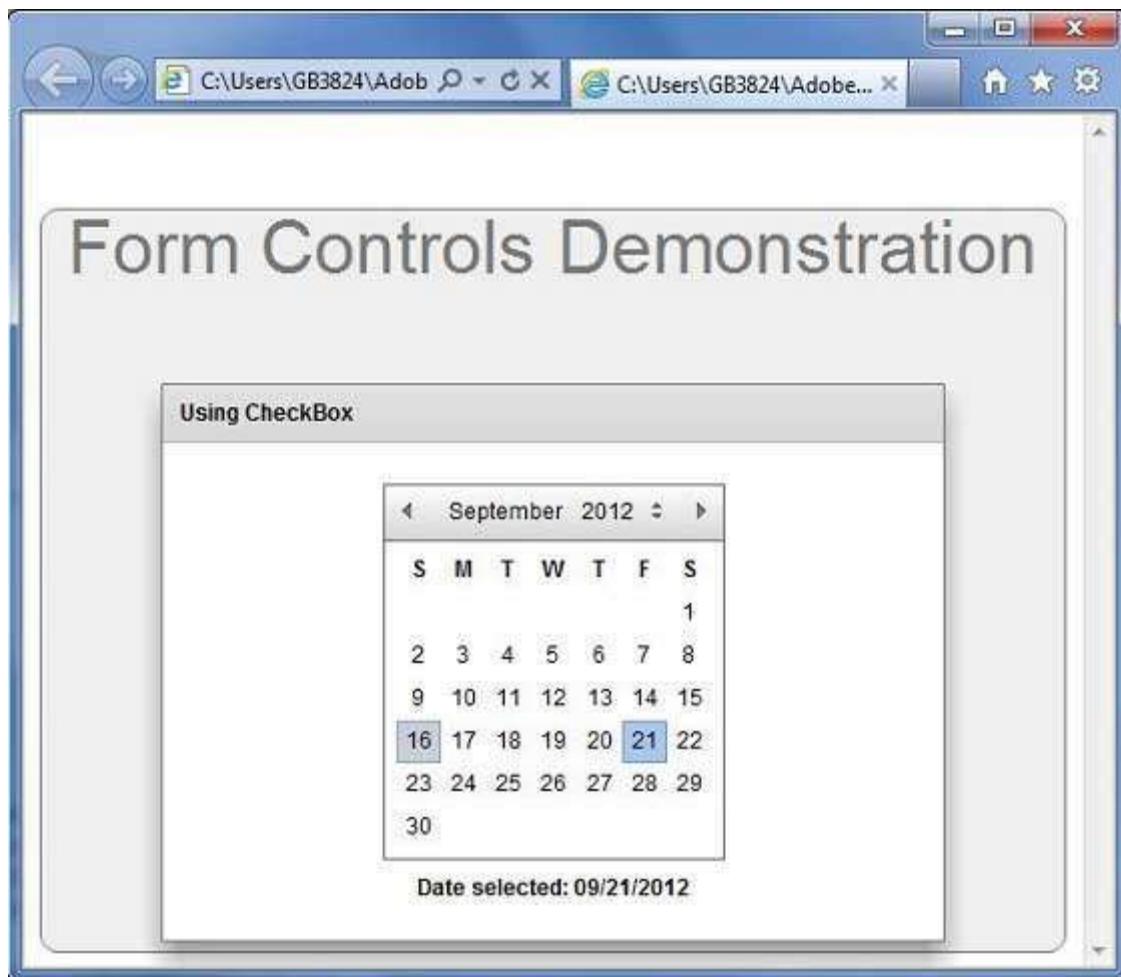
```

<mx:DateChooser id="dateChooser"
yearNavigationEnabled="true"

    change="dateChooser_changeHandler(event)"/>
    <s:Label id="selection" fontWeight="bold"
text="Date selected: {selectedDate}"/>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – RadioButton Control

The RadioButton control allows the user make a single choice within a set of mutually exclusive choices.

## Class Declaration

Following is the declaration for **spark.components.RadioButton** class:

```
public class RadioButton
    extends ToggleButtonBase
    implements IFocusManagerGroup
```

## Public Properties

S.N.	Properties & Description
1	<b>enabled</b> : Boolean [override] The RadioButton component is enabled if the RadioButtonGroup is enabled and the RadioButton itself is enabled.
2	<b>group</b> : RadioButtonGroup The RadioButtonGroup component to which this RadioButton belongs.
3	<b>groupName</b> : String Specifies the name of the group to which this RadioButton component belongs, or specifies the value of the id property of a RadioButtonGroup component if this RadioButton is part of a group defined by a RadioButtonGroup component.
4	<b>value</b> : Object Optional user-defined value that is associated with a RadioButton component.

## Public Methods

S.N.	Method & Description
1	<b>RadioButton()</b> Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.ToggleButtonBase
- spark.components.supportClasses.ButtonBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex RadioButton Control Example

Let us follow the following steps to check usage of RadioButton control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            [Bindable]
            private var selectedOption:String = "";
        ]]>
    </fx:Script>

```

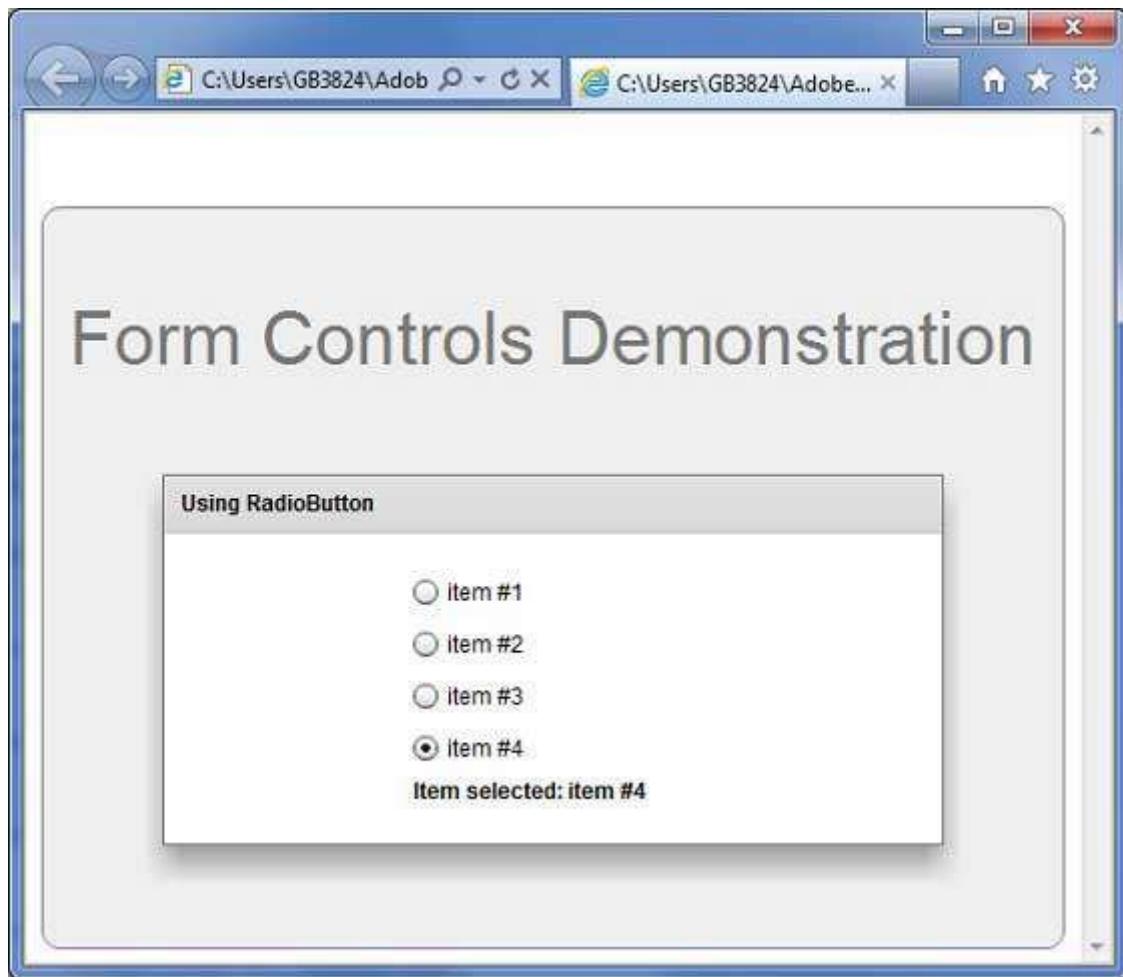
```

protected function option_clickHandler(event:MouseEvent):void
{
    var radioButton:RadioButton = event.target as RadioButton;
    switch (radioButton.id){
        case option1.id:
            selectedOption = option1.label;
            break;
        case option2.id:
            selectedOption = option2.label;
            break;
        case option3.id:
            selectedOption = option3.label;
            break;
        case option4.id:
            selectedOption = option4.label;
            break;
    }
}
]]>
</fx:Script>
<s:BorderContainer width="550" height="400" id="mainContainer"
styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
    horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Form Controls Demonstration"
        fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="radioButtonPanel" title="Using RadioButton"
        width="420" height="200" >
            <s:layout>
                <s:VerticalLayout gap="10" verticalAlign="middle"
                horizontalAlign="center"/>
            </s:layout>
            <s:RadioButton groupName="options" id="option1"
            label="item #1" width="150"

```

```
        click="option_clickHandler(event)"/>
<s:RadioButton groupName="options" id="option2"
    label="item #2" width="150"
    click="option_clickHandler(event)"/>
<s:RadioButton groupName="options" id="option3"
    label="item #3" width="150"
    click="option_clickHandler(event)"/>
<s:RadioButton groupName="options" id="option4"
    label="item #4" width="150"
    click="option_clickHandler(event)"/>
<s:Label id="selectedOptionLabel" fontWeight="bold"
    text="Item selected: {selectedOption}" width="150"/>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – TextArea Control

The TextArea control is a text-entry control that allows users enter and edit multiple lines of formatted text.

### Class Declaration

Following is the declaration for **spark.components.TextArea** class:

```
public class TextArea
    extends SkinnableTextBase
```

### Public Properties

S.N.	Properties & Description
1	<b>content : Object</b> This property is intended for use in MXML at compile time; to get or set rich text content at runtime, use the textFlow property instead.

2	<b>heightInLines : Number</b> The default height of the control, measured in lines.
3	<b>textFlow : flashx.textLayout.elements:TextFlow</b> The TextFlow representing the rich text displayed by this component.
4	<b>widthInChars : Number</b> The default width of the control, measured in em units.

## Public Methods

S.N.	Method & Description
1	<b>TextArea()</b> Constructor.
2	<b>getFormatOfRange(requestedFormats:Vector.&lt;String&gt; = null, anchorPosition:int = -1, activePosition:int = -1):flashx.textLayout.formats:TextLayoutFormat</b> Returns a TextLayoutFormat object specifying the computed formats for the specified range of characters.
3	<b>scrollToRange(anchorPosition:int = 0, activePosition:int):void</b> Scrolls so that the text range is visible in the container.
4	<b>setFormatOfRange(format:flashx.textLayout.formats:TextLayoutFormat, anchorPosition:int = -1, activePosition:int = -1):void</b> Applies the specified formats to each element in the specified range that correspond to the given format.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.SkinnableTextBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer

- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex TextArea Control Example

---

Let us follow the following steps to check usage of TextArea control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

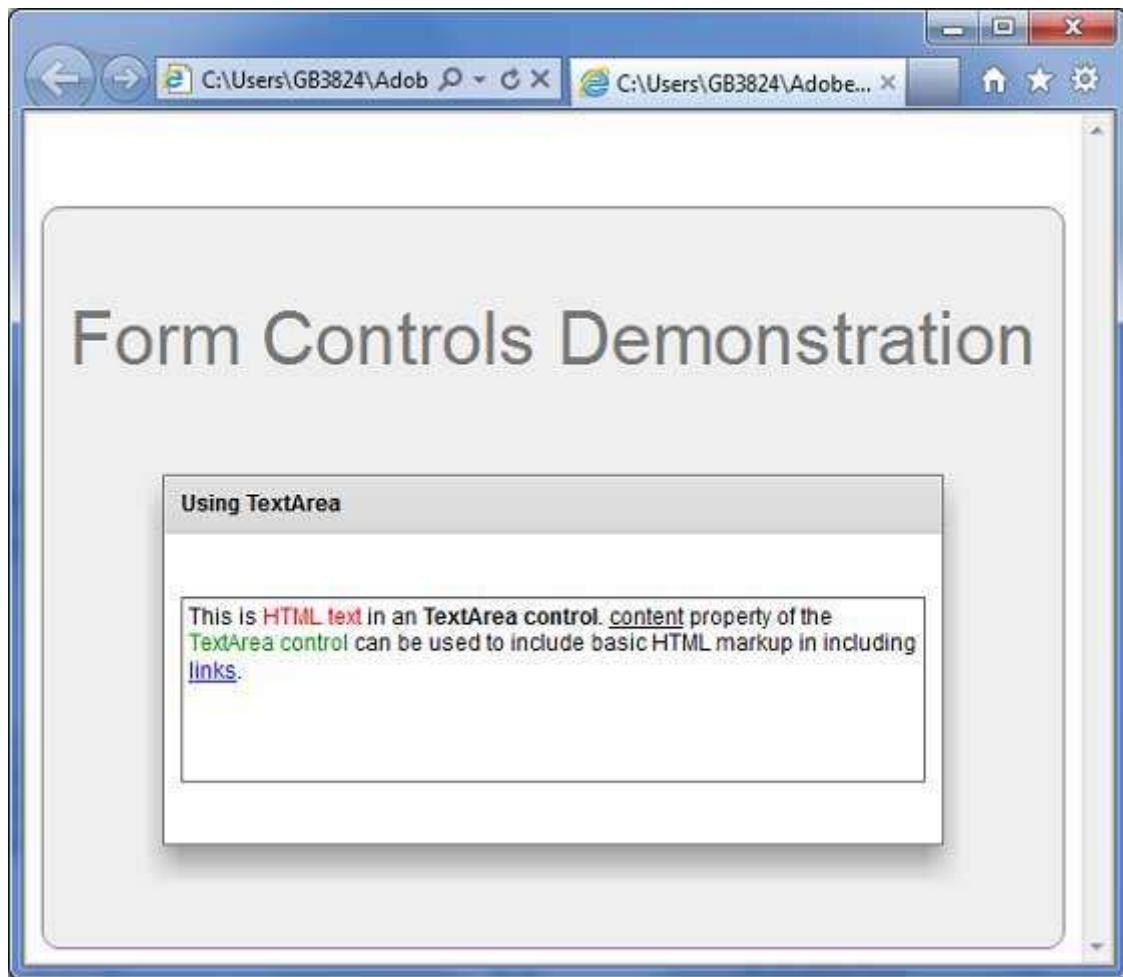
```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <s:BorderContainer width="550" height="400" id="mainContainer"
        styleName="container">
        <s:VGroup width="100%" height="100%" gap="50"
            horizontalAlign="center" verticalAlign="middle">
            <s:Label id="lblHeader" text="Form Controls Demonstration"
                fontSize="40" color="0x777777" styleName="heading"/>
            <s:Panel id="textAreaPanel" title="Using TextArea"
                width="420" height="200">
                <s:layout>
```

```

<s:VerticalLayout gap="10" verticalAlign="middle"
    horizontalAlign="center"/>
</s:layout>
<s:TextArea width="400" height="100">
    <s:content>
        This is <s:span color="#FF0000">HTML text</s:span>
        in an <s:span fontWeight="bold">TextArea
control</s:span>.
        <s:span textDecoration="underline">content</s:span>
property
        of the <s:span color="#008800">TextArea
control</s:span>
        can be used to include basic HTML markup in including
        <s:a href="http://www.tutorialspoint.com"
            target="_blank">links</s:a>.
    </s:content>
</s:TextArea>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – TextInput Control

The TextInput control is a text-entry control that lets users enter and edit a single line of uniformly-formatted text.

### Class Declaration

Following is the declaration for **spark.components.TextArea** class:

```
public class TextInput
    extends SkinnableTextBase
```

### Public Properties

S.N.	Properties & Description
1	<b>widthInChars</b> : Number The default width of the control, measured in em units.

## Public Methods

S.N.	Method & Description
1	TextInput() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.SkinnableTextBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex TextInput Control Example

Let us follow the following steps to check usage of TextInput control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
```

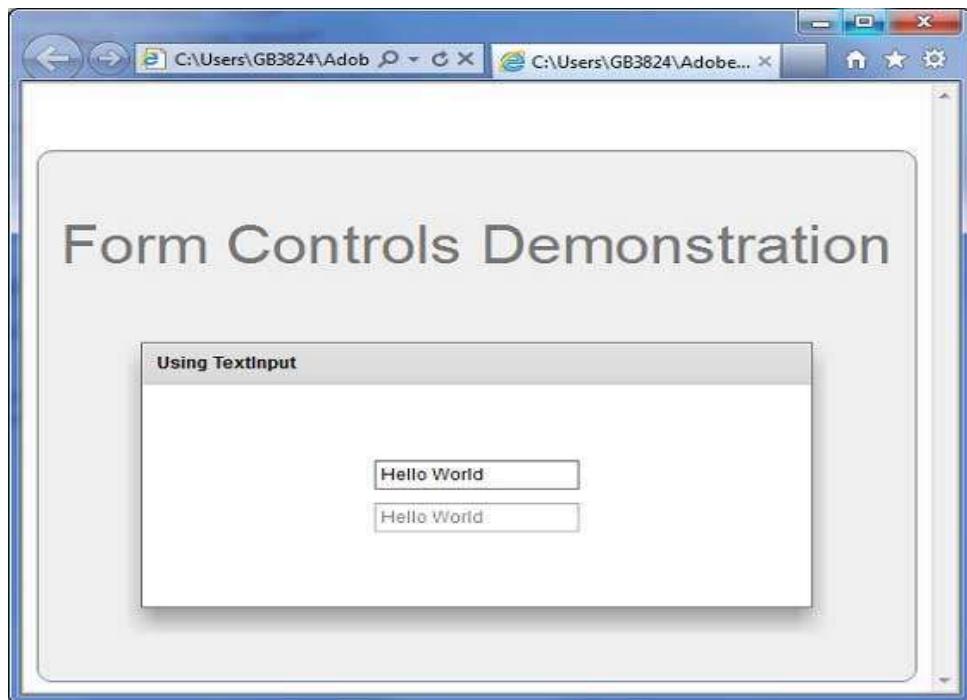
161

```

width="100%" height="100%" minWidth="500" minHeight="500"
>
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<s:BorderContainer width="550" height="400" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center" verticalAlign="middle">
<s:Label id="lblHeader" text="Form Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="textInputPanel" title="Using TextInput"
width="420" height="200" >
<s:layout>
<s:VerticalLayout gap="10"
verticalAlign="middle" horizontalAlign="center"/>
</s:layout>
<s:TextInput text="Hello World" />
<s:TextInput text="Hello World" enabled="false" />
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – DropDownList Control

The DropDownList control contains a drop-down list from which the user can select a single value similar to that of the SELECT form element in HTML.

### Class Declaration

Following is the declaration for **spark.components.DropDownList** class:

```
public class DropDownList
    extends DropDownListBase
```

### Public Properties

S.N.	Property & Description
1	<b>prompt : String</b> The prompt for the DropDownList control.
2	<b>typicalItem : Object</b> [override] Layouts use the preferred size of the typicalItem when fixed row or column sizes are required, but a specific rowHeight or columnWidth value is not set.

## Public Methods

S.N.	Method & Description
1	<b>DropDownList()</b> Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.DropDownListBase
- spark.components.List
- spark.components.supportClasses.ListBase
- spark.components.SkinnableDataContainer
- spark.components.supportClasses.SkinnableContainerBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex DropDownList Control Example

Let us follow the following steps to check usage of DropDownList control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
```

```

<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >

    <fx:Style source="/com/tutorialspoint/client/Style.css"/>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            import spark.events.IndexChangeEvent;

            [Bindable]
            public var data:ArrayCollection = new ArrayCollection(
            [
                {value:"France", code:"FR"},
                {value:"Japan", code:"JP"},
                {value:"India", code:"IN"},
                {value:"Russia", code:"RS"},
                {value:"United States", code:"US"}
            ]
        );
    
```

protected function dropDownList\_changeHandler  
 (event:IndexChangeEvent):void  
 {  
 ddlIndex.text=dropDownList.selectedIndex.toString();  
 ddlSelectedItem.text=dropDownList.selectedItem.value;  
 ddlCode.text=dropDownList.selectedItem.code;  
 }  
 ]]>

```

    </fx:Script>

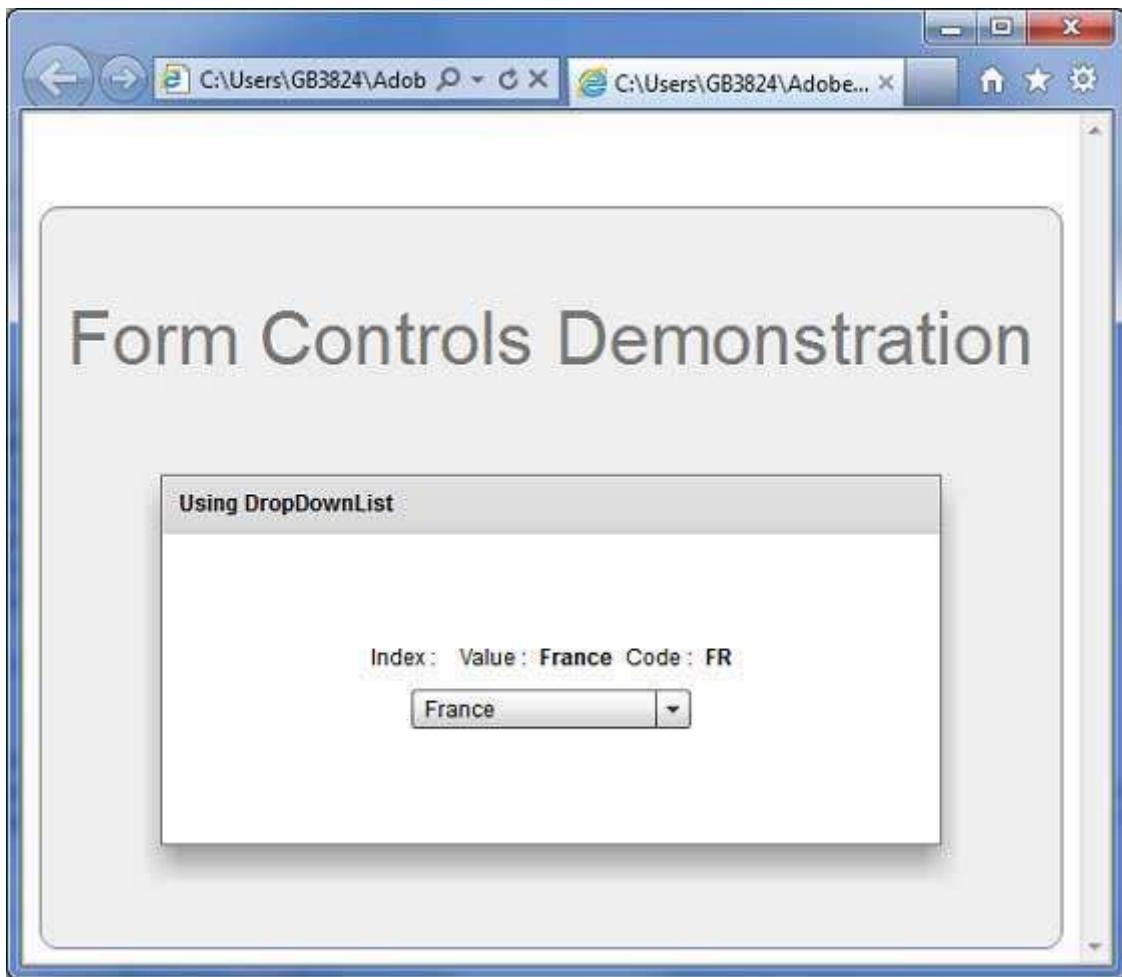
```

```

<s:BorderContainer width="550" height="400" id="mainContainer"
styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center"
    verticalAlign="middle">
        <s:Label id="lblHeader" text="Form Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="dropDownListPanel" title="Using DropDownList"
width="420" height="200">
            <s:layout>
                <s:VerticalLayout gap="10" verticalAlign="middle"
horizontalAlign="center"/>
            </s:layout>
            <s:HGroup>
                <s:Label text="Index :"/>
                <s:Label id="ddlIndex" fontWeight="bold"/>
                <s:Label text="Value :"/>
                <s:Label id="ddlSelectedItem"
text="{dropDownList.selectedItem.value}"
fontWeight="bold"/>
                <s:Label text="Code :"/>
                <s:Label id="ddlCode"
text="{dropDownList.selectedItem.code}"
fontWeight="bold"/>
            </s:HGroup>
            <s:DropDownList id="dropDownList" dataProvider="{data}"
width="150" change="dropDownList_changeHandler(event)"
selectedIndex="0" labelField="value"/>
        </s:Panel>
    </s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – NumericStepper Control

The NumericStepper control lets you select a number from an ordered set. The NumericStepper provides a TextInput control so that you can directly edit the value of the component.

The NumericStepper control also provides a pair of arrow buttons for stepping through the possible values. The Up Arrow and Down Arrow keys and the mouse wheel also cycle through the values.

### Class Declaration

Following is the declaration for **spark.components.NumericStepper** class:

```
public class NumericStepper
    extends Spinner
    implements IFocusManagerComponent, IIIMESupport
```

## Public Properties

S.N.	Property & Description
1	enableIME : Boolean [read-only] A flag that indicates whether the IME should be enabled when the component receives focus.
2	imeMode : String Specifies the IME (Input Method Editor) mode.
3	maxChars : int The maximum number of characters that can be entered in the field.
4	maximum : Number [override] Number which represents the maximum value possible for value.
5	valueFormatFunction : Function Callback function that formats the value displayed in the skin's textDisplay property.
6	valueParseFunction : Function Callback function that extracts the numeric value from the displayed value in the skin's textDisplay field.

## Public Methods

S.N.	Method & Description
1	<b>NumericStepper()</b> Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.Spinner
- spark.components.supportClasses.Range
- spark.components.supportClasses.SkinnableComponent

- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex DropDownList Control Example

---

Let us follow the following steps to check usage of DropDownList control in a Flex application by creating a test application:

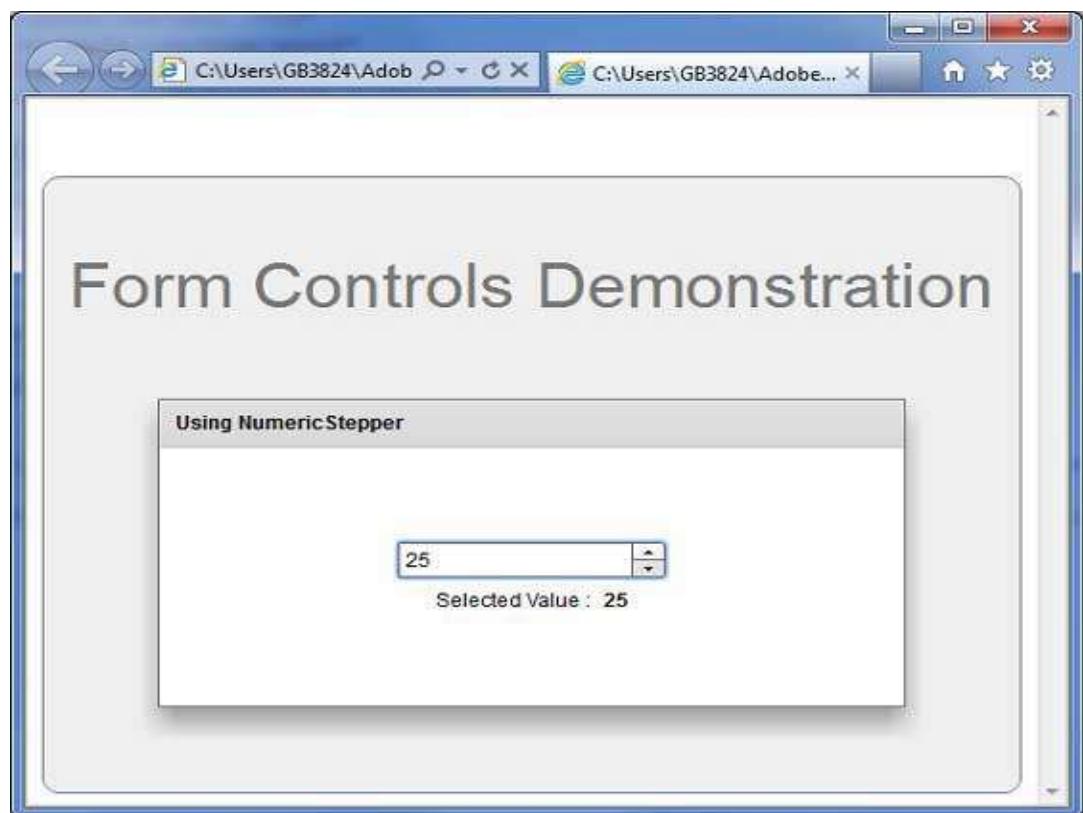
Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <s:BorderContainer width="550" height="400" id="mainContainer"
        styleName="container">
        <s:VGroup width="100%" height="100%" gap="50"
            horizontalAlign="center"
            verticalAlign="middle">
            <s:Label id="lblHeader" text="Form Controls Demonstration">
```

```
fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="numericStepperPanel" title="Using NumericStepper"
width="420" height="200">
<s:layout>
<s:VerticalLayout gap="10" verticalAlign="middle"
horizontalAlign="center"/>
</s:layout>
<s:NumericStepper id="numericStepper" width="150"
value="0" stepSize="5" minimum="0" maximum="50"/>
<s:HGroup>
<s:Label text="Selected Value :"/>
<s:Label text="{numericStepper.value}"
fontWeight="bold"/>
</s:HGroup>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



# 12. Flex – Complex Controls

Complex controls provide users with advanced capabilities to deal with large amount of data in an easier way and provides them interaction capability with the application. Every Complex UI control inherits properties from UIComponent class which in turn inherits properties from EventDispatcher and other top level classes.

S.N.	Control & Description
1	<p>Flex EventDispatcher Class</p> <p>The EventDispatcher class is the base class for all classes that can dispatch events. The EventDispatcher class allows any object on the display list to be an event target and as such, to use the methods of the IEventDispatcher interface.</p>
2	<p>Flex UIComponent</p> <p>The UIComponent class is the base class for all visual components, both interactive and non-interactive.</p>

## Flex – Event Dispatcher Class

---

### Introduction

- The **EventDispatcher** class is the base class for all classes that dispatch events.
- The **EventDispatcher** class implements the **IEventDispatcher** interface.
- The **EventDispatcher** class allows any object on the display list to be an event target and as such, to use the methods of the IEventDispatcher interface.

In order to understand **EventDispatcher**, let us first look what are event and event targets.

### What is an Event?

**Event** is a notification when a particular action is performed. For example, when a button is clicked then Click Event occurs.

### What is an Event Target

The **Event target** serves as the focal point for how events flow through the display list hierarchy.

When an event occurs, Flash Player dispatches an event object into the event flow from the root of the display list. The event object then makes its way through the display list until it reaches the event target, at which point it begins its return trip through the display list.

This round-trip journey to the event target is divided into three phases:

S.N.	Phase & Description
1	<p>capture</p> <p>This phase comprises the journey from the root to the last node before the event target's node</p>
2	<p>target</p> <p>This phase comprises only the event target node.</p>
3	<p>bubbling</p> <p>This phase comprises any subsequent nodes encountered on the return trip to the root of the display list.</p>

In general, any class which extends EventDispatcher gets the event dispatching capability.

## Class Declaration

Following is the declaration for **flash.events.EventDispatcher** class:

```
public class EventDispatcher
    extends java.lang.Object
    implements IEventDispatcher
```

## Public Methods

S.N.	Method & Description
1	<p>EventDispatcher(target:IEventDispatcher = null)</p> <p>Aggregates an instance of the EventDispatcher class.</p>
2	<p>addEventListener(type:String, listener:Function, useCapture:Boolean = false, priority:int = 0, useWeakReference:Boolean = false):void</p> <p>Registers an event listener object with an EventDispatcher object so that the listener receives notification of an event.</p>
3	<p>dispatchEvent(event:Event):Boolean</p> <p>Dispatches an event into the event flow.</p>

4	<b>hasEventListener(type:String):Boolean</b> Checks whether the EventDispatcher object has any listeners registered for a specific type of event.
5	<b>removeEventListener(type:String, listener:Function, useCapture:Boolean = false):void</b> Removes a listener from the EventDispatcher object.
6	<b>willTrigger(type:String):Boolean</b> Checks whether an event listener is registered with this EventDispatcher object or any of its ancestors for the specified event type.

## Events

Following are the events for **flash.events.EventDispatcher** class:

S.N.	<b>Event &amp; Description</b>
1	<b>activate</b> Dispatched when the Flash Player gains operating system focus and becomes active.
2	<b>detivate</b> Dispatched when the Flash Player loses operating system focus and becomes inactive.

## Methods Inherited

This class inherits methods from the following class:

- Object

## Flex - UIComponent Class

---

### Introduction

The **UIComponent** class is the base class for all visual components, both interactive and noninteractive.

### Class Declaration

Following is the declaration for **mx.core.UIComponent** class:

```

public class UIComponent
    extends FlexSprite
    implements IAutomationObject, IChildList, IConstraintClient,
    IDeferredInstantiationUIComponent, IFlexDisplayObject,
    IFlexModule, IIValidating, ILayoutManagerClient,
    IPropertyChangeNotifier, IRepeaterClient, IStateClient,
    IAdvancedStyleClient, IToolTipManagerClient,
    UIComponent, IValidatorListener, IVisualElement

```

## Public Properties

Following are the Public Properties for **mx.core.UIComponent** class:

S.N.	Name & Description
1	<b>accessibilityDescription</b> : String A convenience accessor for the description property in this UIComponent's accessibilityProperties object.
2	<b>accessibilityEnabled</b> : Boolean A convenience accessor for the silent property in this UIComponent's accessibilityProperties object.
3	<b>accessibilityName</b> : String A convenience accessor for the name property in this UIComponent's accessibilityProperties object.
4	<b>accessibilityShortcut</b> : String A convenience accessor for the shortcut property in this UIComponent's accessibilityProperties object.
5	<b>activeEffects</b> : Array [read-only] The list of effects that are currently playing on the component, as an Array of EffectInstance instances.
6	<b>automationDelegate</b> : Object The delegate object that handles the automation-related functionality.
7	<b>automationEnabled</b> : Boolean

	[read-only] True if this component is enabled for automation, false otherwise.
8	automationName : String  Name that can be used as an identifier for this object.
9	automationOwner : DisplayObjectContainer  [read-only] The owner of this component for automation purposes.
10	automationParent : DisplayObjectContainer  [read-only] The parent of this component for automation purposes.
11	automationTabularData : Object  [read-only] An implementation of the IAutomationTabularData interface, which can be used to retrieve the data.
12	automationValue : Array  [read-only] This value generally corresponds to the rendered appearance of the object and should be usable for correlating the identifier with the object as it appears visually within the application.
13	automationVisible : Boolean  [read-only] True if this component is visible for automation, false otherwise.
14	baseline : Object  For components, this layout constraint property is a facade on top of the similarly-named style.
15	baselinePosition : Number  [read-only] The y-coordinate of the baseline of the first line of text of the component.
16	bottom : Object  For components, this layout constraint property is a facade on top of the similarly-named style.
17	cacheHeuristic : Boolean  [write-only] Used by Flex to suggest bitmap caching for the object.

18	<b>cachePolicy</b> : String  Specifies the bitmap caching policy for this object.
19	<b>className</b> : String  [read-only] The name of this instance's class, such as "Button".
20	<b>contentMouseX</b> : Number  [read-only] Returns the x position of the mouse, in the content coordinate system.
21	<b>contentMouseY</b> : Number  [read-only] Returns the y position of the mouse, in the content coordinate system.
22	<b>currentState</b> : String  The current view state of the component.
23	<b>cursorManager</b> : ICursorManager  [read-only] Gets the CursorManager that controls the cursor for this component and its peers.
24	<b>depth</b> : Number  Determines the order in which items inside of containers are rendered.
25	<b>descriptor</b> : UIComponentDescriptor  Reference to the UIComponentDescriptor, if any, that was used by the createComponentFromDescriptor() method to create this UIComponent instance.
26	<b>designLayer</b> : DesignLayer  Specifies the optional DesignLayer instance associated with this visual element.
27	<b>document</b> : Object  A reference to the document object associated with this UIComponent.
28	<b>doubleClickEnabled</b> : Boolean  [override] Specifies whether the UIComponent object receives doubleClick events.

29	<b>enabled : Boolean</b> Whether the component can accept user interaction.
30	<b>errorString : String</b> The text that displayed by a component's error tip when a component is monitored by a Validator and validation fails.
31	<b>explicitHeight : Number</b> Number that specifies the explicit height of the component, in pixels, in the component's coordinates.
32	<b>explicitMaxHeight : Number</b> The maximum recommended height of the component to be considered by the parent during layout.
33	<b>explicitMaxWidth : Number</b> The maximum recommended width of the component to be considered by the parent during layout.
34	<b>explicitMinHeight : Number</b> The minimum recommended height of the component to be considered by the parent during layout.
35	<b>explicitMinWidth : Number</b> The minimum recommended width of the component to be considered by the parent during layout.
36	<b>explicitWidth : Number</b> Number that specifies the explicit width of the component, in pixels, in the component's coordinates.
37	<b>flexContextMenu : IFlexContextMenu</b> The context menu for this UIComponent.
38	<b>focusEnabled : Boolean</b> Indicates whether the component can receive focus when tabbed to.
39	<b>focusManager : IFocusManager</b>

	Gets the FocusManager that controls focus for this component and its peers.
40	<b>focusPane : Sprite</b> The focus pane associated with this object.
41	<b>hasFocusableChildren : Boolean</b> A flag that indicates whether child objects can receive focus.
42	<b>hasLayoutMatrix3D : Boolean</b> [read-only] Contains true if the element has 3D Matrix.
43	<b>height : Number</b> [override] Number that specifies the height of the component, in pixels, in the parent's coordinates.
44	<b>horizontalCenter : Object</b> For components, this layout constraint property is a facade on top of the similarly-named style.
45	<b>id : String</b> ID of the component.
46	<b>includeInLayout : Boolean</b> Specifies whether this component is included in the layout of the parent container.
47	<b>inheritingStyles : Object</b> The beginning of this component's chain of inheriting styles.
48	<b>initialized : Boolean</b> A flag that determines if an object has been through all three phases of layout: commitment, measurement, and layout (provided that any were required).
49	<b>instanceIndex : int</b> [read-only] The index of a repeated component.
50	<b>instanceIndices : Array</b> An Array containing the indices required to reference this UIComponent object from its parent document.

51	<b>is3D : Boolean</b> [read-only] Contains true when the element is in 3D.
52	<b>isDocument : Boolean</b> [read-only] Contains true if this UIComponent instance is a document object.
53	<b>isPopUp : Boolean</b> Set to true by the PopUpManager to indicate that component has been popped up.
54	<b>layoutMatrix3D : Matrix3D</b> [write-only] The transform matrix that is used to calculate a component's layout relative to its siblings.
55	<b>left : Object</b> For components, this layout constraint property is a facade on top of the similarly-named style.
56	<b>maintainProjectionCenter : Boolean</b> When true, the component keeps its projection matrix centered on the middle of its bounding box.
57	<b>maxHeight : Number</b> The maximum recommended height of the component to be considered by the parent during layout.
58	<b>maxWidth : Number</b> The maximum recommended width of the component to be considered by the parent during layout.
59	<b>measuredHeight : Number</b> The default height of the component, in pixels.
60	<b>measuredMinHeight : Number</b> The default minimum height of the component, in pixels.
61	<b>measuredMinWidth : Number</b> The default minimum width of the component, in pixels.

62	<b>measuredWidth : Number</b> The default width of the component, in pixels.
63	<b>minHeight : Number</b> The minimum recommended height of the component to be considered by the parent during layout.
64	<b>minWidth : Number</b> The minimum recommended width of the component to be considered by the parent during layout.
65	<b>moduleFactory : IFlexModuleFactory</b> A module factory is used as context for using embedded fonts and for finding the style manager that controls the styles for this component.
66	<b>mouseFocusEnabled : Boolean</b> Whether you can receive focus when clicked on.
67	<b>nestLevel : int</b> Depth of this object in the containment hierarchy.
68	<b>nonInheritingStyles : Object</b> The beginning of this component's chain of non-inheriting styles.
69	<b>numAutomationChildren : int</b> [read-only] The number of automation children this container has.
70	<b>owner : DisplayObjectContainer</b> The owner of this IVisualElement object.
71	<b>parent : DisplayObjectContainer</b> [override] [read-only] The parent container or component for this component.
72	<b>parentApplication : Object</b> [read-only] A reference to the Application object that contains this UIComponent instance.
73	<b>parentDocument : Object</b>

	[read-only] A reference to the parent document object for this UIComponent.
74	<p>percentHeight : Number</p> <p>Specifies the height of a component as a percentage of its parent's size.</p>
75	<p>percentWidth : Number</p> <p>Specifies the width of a component as a percentage of its parent's size.</p>
76	<p>postLayoutTransformOffsets : mx.geom:TransformOffsets</p> <p>Defines a set of adjustments that can be applied to the object's transform in a way that is invisible to its parent's layout.</p>
77	<p>processedDescriptors : Boolean</p> <p>Set to true after immediate or deferred child creation, depending on which one happens.</p>
78	<p>repeater : IRepeater</p> <p>[read-only] A reference to the Repeater object in the parent document that produced this UIComponent.</p>
79	<p>repeaterIndex : int</p> <p>[read-only] The index of the item in the data provider of the Repeater that produced this UIComponent.</p>
80	<p>repeaterIndices : Array</p> <p>An Array containing the indices of the items in the data provider of the Repeaters in the parent document that produced this UIComponent.</p>
81	<p>repeaters : Array</p> <p>An Array containing references to the Repeater objects in the parent document that produced this UIComponent.</p>
82	<p>right : Object</p> <p>For components, this layout constraint property is a facade on top of the similarly-named style.</p>
83	<p>rotation : Number</p> <p>[override] Indicates the rotation of the DisplayObject instance, in degrees, from its original orientation.</p>

84	<b>rotationX : Number</b>  [override] Indicates the x-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
85	<b>rotationY : Number</b>  [override] Indicates the y-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
86	<b>rotationZ : Number</b>  [override] Indicates the z-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
87	<b>scaleX : Number</b>  [override] Number that specifies the horizontal scaling factor.
88	<b>scaleY : Number</b>  [override] Number that specifies the vertical scaling factor.
89	<b>scaleZ : Number</b>  [override] Number that specifies the scaling factor along the z axis.
90	<b>screen : Rectangle</b>  [read-only] Returns an object that contains the size and position of the base drawing surface for this object.
91	<b>showInAutomationHierarchy : Boolean</b>  A flag that determines if an automation object shows in the automation hierarchy.
92	<b>states : Array</b>  The view states that are defined for this component.
93	<b>styleDeclaration : CSSStyleDeclaration</b>  Storage for the inline inheriting styles on this object.
94	<b>styleManager : IStyleManager2</b>  [read-only] Returns the StyleManager instance used by this component.
95	<b>styleName : Object</b>

	The class style used by this component.
96	<b>styleParent : IAdvancedStyleClient</b> A component's parent is used to evaluate descendant selectors.
97	<b>systemManager : ISytemManager</b> Returns the SystemManager object used by this component.
98	<b>tabFocusEnabled : Boolean</b> A flag that indicates whether this object can receive focus via the TAB key. This is similar to the tabEnabled property used by the Flash Player. This is usually true for components that handle keyboard input, but some components in controlbars have them set to false because they should not steal focus from another component like an editor.
99	<b>toolTip : String</b> Text to display in the ToolTip.
100	<b>top : Object</b> For components, this layout constraint property is a facade on top of the similarly-named style.
101	<b>transform : flash.geom:Transform</b> [override] An object with properties pertaining to a display object's matrix, color transform, and pixel bounds.
102	<b>transformX : Number</b> Sets the x coordinate for the transform center of the component.
103	<b>transformY : Number</b> Sets the y coordinate for the transform center of the component.
104	<b>transformZ : Number</b> Sets the z coordinate for the transform center of the component.
105	<b>transitions : Array</b> An Array of Transition objects, where each Transition object defines a set of effects to play when a view state change occurs.

106	<b>tweeningProperties : Array</b> Array of properties that are currently being tweened on this object.
107	<b>uid : String</b> A unique identifier for the object.
108	<b>updateCompletePendingFlag : Boolean</b> A flag that determines if an object has been through all three phases of layout validation (provided that any were required).
109	<b>validationSubField : String</b> Used by a validator to associate a subfield with this component.
110	<b>verticalCenter : Object</b> For components, this layout constraint property is a facade on top of the similarly-named style.
111	<b>visible : Boolean</b> [override] Whether or not the display object is visible.
112	<b>width : Number</b> [override] Number that specifies the width of the component, in pixels, in the parent's coordinates.
113	<b>x : Number</b> [override] Number that specifies the component's horizontal position, in pixels, within its parent container.
114	<b>y : Number</b> [override] Number that specifies the component's vertical position, in pixels, within its parent container.
115	<b>z : Number</b> [override] Indicates the z coordinate position along the z-axis of the DisplayObject instance relative to the 3D parent container.

## Protected Properties

Following are the Protected Properties for **mx.core.UIComponent** class:

S.N.	Name & Description
1	currentCSSState : String [read-only] The state to be used when matching CSS pseudo-selectors.
2	hasComplexLayoutMatrix : Boolean [read-only] Returns true if the UIComponent has any non-translation (x,y) transform properties.
3	resourceManager : IResourceManager [read-only] A reference to the object which manages all of the application's localized resources.
4	unscaledHeight : Number [read-only] A convenience method for determining the unscaled height of the component.
5	unscaledWidth : Number [read-only] A convenience method for determining the unscaled width of the component All of a component's drawing and child layout should be done within a bounding rectangle of this width, which is also passed as an argument to updateDisplayList().

S.N.	Event & Description
1	<b>activate</b> Dispatched when the Flash Player gains operating system focus and becomes active.
2	<b>detivate</b>

	Dispatched when the Flash Player loses operating system focus and becomes inactive.
--	---

## Public Methods

S.N.	Method & Description
1	UIComponent() Constructor.
2	addStyleClient(styleClient:IAdvancedStyleClient):void Adds a non-visual style client to this component instance.
3	callLater(method:Function, args:Array = null):void Queues a function to be called later.
4	clearStyle(styleProp:String):void Deletes a style property from this component instance.
5	contentToGlobal(point:Point):Point Converts a Point object from content coordinates to global coordinates.
6	contentToLocal(point:Point):Point Converts a Point object from content to local coordinates.
7	createAutomationIDPart(child:IAutomationObject):Object Returns a set of properties that identify the child within this container.
8	createAutomationIDPartWithRequiredProperties(child:IAutomationObject, properties:Array):Object Returns a set of properties that identify the child within this container.
9	createReferenceOnParentDocument(parentDocument:IFlexDisplayObject):void Creates an id reference to this IUIComponent object on its parent document object.

10	<code>deleteReferenceOnParentDocument(parentDocument:IFlexDisplayObject):void</code> Deletes the id reference to this UIComponent object on its parent document object.
11	<code>determineTextFormatFromStyles():mx.core:UITextFormat</code> Returns a UITextFormat object corresponding to the text styles for this UIComponent.
12	<code>dispatchEvent(event:Event):Boolean</code> [override] Dispatches an event into the event flow.
13	<code>drawFocus(isFocused:Boolean):void</code> Shows or hides the focus indicator around this component.
14	<code>drawRoundRect(x:Number, y:Number, w:Number, h:Number, r:Object = null, c:Object = null, alpha:Object = null, rot:Object = null, gradient:String = null, ratios:Array = null, hole:Object = null):void</code> Programmatically draws a rectangle into this skin's Graphics object.
15	<code>effectFinished(effectInst:IEffectInstance):void</code> Called by the effect instance when it stops playing on the component.
16	<code>effectStarted(effectInst:IEffectInstance):void</code> Called by the effect instance when it starts playing on the component.
17	<code>endEffectsStarted():void</code> Ends all currently playing effects on the component.
18	<code>executeBindings(recurse:Boolean = false):void</code> Executes all the bindings for which the UIComponent object is the destination.
19	<code>finishPrint(obj:Object, target:IFlexDisplayObject):void</code> Called after printing is complete.
20	<code>getAutomationChildAt(index:int):IAutomationObject</code> Provides the automation object at the specified index.
21	<code>getAutomationChildren():Array</code>

	Provides the automation object list .
22	getBoundsXAtSize(width:Number, postLayoutTransform:Boolean = true):Number  Returns the x coordinate of the element's bounds at the specified element size.
23	getBoundsYAtSize(width:Number, postLayoutTransform:Boolean = true):Number  Returns the y coordinate of the element's bounds at the specified element size.
24	getClassStyleDeclarations():Array  Finds the type selectors for this UIComponent instance.
25	getConstraintValue(constraintName:String):*  Returns a layout constraint value, which is the same as getting the constraint style for this component.
26	getExplicitOrMeasuredHeight():Number  A convenience method for determining whether to use the explicit or measured height
27	getExplicitOrMeasuredWidth():Number  A convenience method for determining whether to use the explicit or measured width
28	getFocus():InteractiveObject  Gets the object that currently has focus.
29	getLayoutBoundsHeight(postLayoutTransform:Boolean = true):Number  Returns the element's layout height.
30	getLayoutBoundsWidth(postLayoutTransform:Boolean = true):Number  Returns the element's layout width.
31	getLayoutBoundsX(postLayoutTransform:Boolean = true):Number  Returns the x coordinate that the element uses to draw on screen.
32	getLayoutBoundsY(postLayoutTransform:Boolean = true):Number

	Returns the y coordinate that the element uses to draw on screen.
33	<code>getLayoutMatrix():Matrix</code> Returns the transform matrix that is used to calculate the component's layout relative to its siblings.
34	<code>getLayoutMatrix3D():Matrix3D</code> Returns the layout transform Matrix3D for this element.
35	<code>getMaxBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's maximum height.
36	<code>getMaxBoundsWidth(postLayoutTransform:Boolean = true):Number</code> Returns the element's maximum width.
37	<code>getMinBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's minimum height.
38	<code>getMinBoundsWidth(postLayoutTransform:Boolean = true):Number</code> Returns the element's minimum width.
39	<code>getPreferredBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's preferred height.
40	<code>getPreferredBoundsWidth(postLayoutTransform:Boolean = true):Number</code> Returns the element's preferred width.
41	<code>getRepeaterItem(whichRepeater:int = -1):Object</code> Returns the item in the dataProvider that was used by the specified Repeater to produce this Repeater, or null if this Repeater isn't repeated.
42	<code>getStyle(styleProp:String):*</code> Gets a style property that has been set anywhere in this component's style lookup chain.
43	<code>globalToContent(point:Point):Point</code> Converts a Point object from global to content coordinates.

45	<code>hasCSSState():Boolean</code> Returns true if currentCSSState is not null.
46	<code>hasState(stateName:String):Boolean</code> Determines whether the specified state has been defined on this UIComponent.
47	<code>horizontalGradientMatrix(x:Number, y:Number, width:Number, height:Number):Matrix</code> Returns a box Matrix which can be passed to the drawRoundRect() method as the rot parameter when drawing a horizontal gradient.
48	<code>initialize():void</code> Initializes the internal structure of this component.
49	<code>initializeRepeaterArrays(parent:IRepeaterClient):void</code> Initializes various properties which keep track of repeated instances of this component.
50	<code>invalidateDisplayList():void</code> Marks a component so that its updateDisplayList() method gets called during a later screen update.
51	<code>invalidateLayering():void</code> Called by a component's items to indicate that their depth property has changed.
52	<code>invalidateLayoutDirection():void</code> An element must call this method when its layoutDirection changes or when its parent's layoutDirection changes.
53	<code>invalidateProperties():void</code> Marks a component so that its commitProperties() method gets called during a later screen update.
54	<code>invalidateSize():void</code> Marks a component so that its measure() method gets called during a later screen update.
55	<code>localToContent(point:Point):Point</code>

	Converts a Point object from local to content coordinates.
56	<p><code>matchesCSSState(cssState:String):Boolean</code></p> <p>Returns true if cssState matches currentCSSState.</p>
57	<p><code>matchesCSSType(cssType:String):Boolean</code></p> <p>Determines whether this instance is the same as, or is a subclass of, the given type.</p>
58	<p><code>measureHTMLText(htmlText:String):flash.text:TextLineMetrics</code></p> <p>Measures the specified HTML text, which can contain HTML tags such as &amp;lt;font&amp;ampgt and &amp;&lt;b&amp;ampgt, assuming that it is displayed in a single-line UITextField using a UITextFormat determined by the styles of this UIComponent.</p>
59	<p><code>measureText(text:String):flash.text:TextLineMetrics</code></p> <p>Measures the specified text, assuming that it is displayed in a single-line UITextField (or UIFETextField) using a UITextFormat determined by the styles of this UIComponent.</p>
60	<p><code>move(x:Number, y:Number):void</code></p> <p>Moves the component to a specified position within its parent.</p>
61	<p><code>notifyStyleChangeInChildren(styleProp:String, recursive:Boolean):void</code></p> <p>Propagates style changes to the children.</p>
62	<p><code>owns(child:DisplayObject):Boolean</code></p> <p>Returns true if the chain of owner properties points from child to this UIComponent.</p>
63	<p><code>parentChanged(p:DisplayObjectContainer):void</code></p> <p>Called by Flex when a UIComponent object is added to or removed from a parent.</p>
64	<p><code>prepareToPrint(target:IFlexDisplayObject):Object</code></p> <p>Prepares an IFlexDisplayObject for printing.</p>
65	<p><code>regenerateStyleCache(recursive:Boolean):void</code></p> <p>Builds or rebuilds the CSS style cache for this component and, if the recursive parameter is true, for all descendants of this component as well.</p>

66	<code>registerEffects(effects:Array):void</code> For each effect event, registers the EffectManager as one of the event listeners.
67	<code>removeStyleClient(styleClient:IAdvancedStyleClient):void</code> Removes a non-visual style client from this component instance.
68	<code>replayAutomatableEvent(event:Event):Boolean</code> Replays the specified event.
69	<code>resolveAutomationIDPart(criteria:Object):Array</code> Resolves a child by using the id provided.
70	<code>resumeBackgroundProcessing():void</code> [static] Resumes the background processing of methods queued by callLater(), after a call to suspendBackgroundProcessing().
71	<code>setActualSize(w:Number, h:Number):void</code> Sizes the object.
72	<code>setConstraintValue(constraintName:String, value:*):void</code> Sets a layout constraint value, which is the same as setting the constraint style for this component.
73	<code>setCurrentState(stateName:String, playTransition:Boolean = true):void</code> Set the current state.
74	<code>setFocus():void</code> Sets the focus to this component.
75	<code>setLayoutBoundsPosition(x:Number, y:Number, postLayoutTransform:Boolean = true):void</code> Sets the coordinates that the element uses to draw on screen.
76	<code>setLayoutBoundsSize(width:Number, height:Number, postLayoutTransform:Boolean = true):void</code> Sets the layout size of the element.
77	<code>setLayoutMatrix(value:Matrix, invalidateLayout:Boolean):void</code>

	Sets the transform Matrix that is used to calculate the component's layout size and position relative to its siblings.
78	<b>setLayoutMatrix3D(value:Matrix3D, invalidateLayout:Boolean):void</b> Sets the transform Matrix3D that is used to calculate the component's layout size and position relative to its siblings.
79	<b>setStyle(styleProp:String, newValue:*):void</b> Sets a style property on this component instance.
80	<b>setVisible(value:Boolean, noEvent:Boolean = false):void</b> Called when the visible property changes.
81	<b>styleChanged(styleProp:String):void</b> Detects changes to style properties.
82	<b>stylesInitialized():void</b> Flex calls the stylesInitialized() method when the styles for a component are first initialized.
83	<b>suspendBackgroundProcessing():void</b> [static] Blocks the background processing of methods queued by callLater(), until resumeBackgroundProcessing() is called.
84	<b>transformAround(transformCenter:Vector3D, scale:Vector3D = null, rotation:Vector3D = null, translation:Vector3D = null, postLayoutScale:Vector3D = null, postLayoutRotation:Vector3D = null, postLayoutTranslation:Vector3D = null, invalidateLayout:Boolean = true):void</b> A utility method to update the rotation, scale, and translation of the transform while keeping a particular point, specified in the component's own coordinate space, fixed in the parent's coordinate space.
85	<b>transform Point To Parent (localPosition:Vector3D, position:Vector3D, post Layout Position:Vector3D):void</b> A utility method to transform a point specified in the local coordinates of this object to its location in the object's parent's coordinates.
86	<b>validateDisplayList():void</b> Validates the position and size of children and draws other visuals.

87	<b>validateNow():void</b> Validate and update the properties and layout of this object and redraw it, if necessary.
88	<b>validateProperties():void</b> Used by layout logic to validate the properties of a component by calling the commitProperties() method.
89	<b>validateSize(recursive:Boolean = false):void</b> Validates the measured size of the component. If the LayoutManager.invalidateSize() method is called with this ILayoutManagerClient, then the validateSize() method is called when it's time to do measurements.
90	<b>validationResultHandler(event:ValidationResultEvent):void</b> Handles both the valid and invalid events from a validator assigned to this component.
91	<b>verticalGradientMatrix(x:Number, y:Number, width:Number, height:Number):Matrix</b> Returns a box Matrix which can be passed to drawRoundRect() as the rot parameter when drawing a vertical gradient.

## Protected Methods

S.N.	<b>Method &amp; Description</b>
1	<b>adjustFocusRect(obj:DisplayObject = null):void</b> Adjust the focus rectangle.
2	<b>applyComputedMatrix():void</b> Commits the computed matrix built from the combination of the layout matrix and the transform offsets to the flash displayObject's transform.
3	<b>attachOverlay():void</b> This is an internal method used by the Flex framework to support the Dissolve effect.
4	<b>canSkipMeasurement():Boolean</b>

	Determines if the call to the measure() method can be skipped.
5	childrenCreated():void Performs any final processing after child objects are created.
6	commitProperties():void Processes the properties set on the component.
7	createChildren():void Create child objects of the component.
8	createInFontContext(classObj:Class):Object Creates a new object using a context based on the embedded font being used.
9	createInModuleContext(moduleFactory:IFlexModuleFactory, className:String):Object Creates the object using a given moduleFactory.
10	dispatchPropertyChangeEvent(prop:String, oldValue:*, value:*):void Helper method for dispatching a PropertyChangeEvent when a property is updated.
11	focusInHandler(event:FocusEvent):void The event handler called when a UIComponent object gets focus.
12	focusOutHandler(event:FocusEvent):void The event handler called when a UIComponent object loses focus.
13	initAdvancedLayoutFeatures():void Initializes the implementation and storage of some of the less frequently used advanced layout features of a component.
14	initializationComplete():void Finalizes the initialization of this component.
15	initializeAccessibility():void Initializes this component's accessibility code.

16	<b>invalidateParentSizeAndDisplayList():void</b> Helper method to invalidate parent size and display list if this object affects its layout (includeInLayout is true).
17	<b>isOurFocus(target:DisplayObject):Boolean</b> Typically overridden by components containing UITextField objects, where the UITextField object gets focus.
18	<b>keyDownHandler(event:KeyboardEvent):void</b> The event handler called for a keyDown event.
19	<b>keyUpHandler(event:KeyboardEvent):void</b> The event handler called for a keyUp event.
20	<b>measure():void</b> Calculates the default size, and optionally the default minimum size, of the component.
21	<b>resourcesChanged():void</b> This method is called when a UIComponent is constructed, and again whenever the ResourceManager dispatches a "change" Event to indicate that the localized resources have changed in some way.
22	<b>setStretchXY(stretchX:Number, stretchY:Number):void</b> Specifies a transform stretch factor in the horizontal and vertical direction.
23	<b>stateChanged(oldState:String, newState:String, recursive:Boolean):void</b> This method is called when a state changes to check whether state-specific styles apply to this component
24	<b>updateDisplayList(unscaledWidth:Number, unscaledHeight:Number):void</b> Draws the object and/or sizes and positions its children.

## Events

Following are the events for **mx.core.UIComponent** class:

S.N.	<b>Event &amp; Description</b>

1	<b>add</b> when the component is added to a container as a content child by using the addChild(), addChildAt(), addElement(), or addElementAt() method.
2	<b>creationComplete</b> when the component has finished its construction, property processing, measuring, layout, and drawing.
3	<b>currentStateChange</b> after the view state has changed.
4	<b>currentStateChanging</b> after the currentState property changes, but before the view state changes.
5	<b>dragComplete</b> by the drag initiator (the component that is the source of the data being dragged) when the drag operation completes, either when you drop the dragged data onto a drop target or when you end the drag-and-drop operation without performing a drop.
6	<b>dragDrop</b> by the drop target when the user releases the mouse over it.
7	<b>dragEnter</b> by a component when the user moves the mouse over the component during a drag operation.
8	<b>dragExit</b> by the component when the user drags outside the component, but does not drop the data onto the target.
9	<b>dragOver</b> by a component when the user moves the mouse while over the component during a drag operation.
10	<b>dragStart</b> by the drag initiator when starting a drag operation.
11	<b>effectEnd</b>

	after an effect ends.
12	effectStart just before an effect starts.
13	effectStop after an effect is stopped, which happens only by a call to stop() on the effect.
14	enterState after the component has entered a view state.
15	exitState just before the component exits a view state.
16	hide when an object's state changes from visible to invisible.
17	initialize when the component has finished its construction and has all initialization properties set.
18	invalid when a component is monitored by a Validator and the validation failed.
19	mouseDownOutside from a component opened using the PopUpManager when the user clicks outside it.
20	mouseWheelOutside from a component opened using the PopUpManager when the user scrolls the mouse wheel outside it.
21	move when the object has moved.
22	preinitialize at the beginning of the component initialization sequence.

23	<b>remove</b> when the component is removed from a container as a content child by using the <code>removeChild()</code> , <code>removeChildAt()</code> , <code>removeElement()</code> , or <code>removeElementAt()</code> method.
24	<b>resize</b> when the component is resized.
25	<b>show</b> when an object's state changes from invisible to visible.
26	<b>stateChangeComplete</b> after the component has entered a new state and any state transition animation to that state has finished playing.
27	<b>stateChangeInterrupted</b> when a component interrupts a transition to its current state in order to switch to a new state.
28	<b>toolTipCreate</b> by the component when it is time to create a ToolTip.
29	<b>toolTipEnd</b> by the component when its ToolTip has been hidden and is to be discarded soon.
30	<b>toolTipHide</b> by the component when its ToolTip is about to be hidden.
31	<b>toolTipShow</b> by the component when its ToolTip is about to be shown.
32	<b>toolTipShown</b> by the component when its ToolTip has been shown.
33	<b>toolTipStart</b> by a component whose <code>toolTip</code> property is set, as soon as the user moves the mouse over it.

34	touchInteractionEnd  A non-cancellable event, by a component when it is done responding to a touch interaction user gesture.
35	touchInteractionStart  A non-cancellable event, by a component when it starts responding to a touch interaction user gesture.
36	touchInteractionStarting  A cancellable event, by a component in an attempt to respond to a touch interaction user gesture.
37	updateComplete  when an object has had its commitProperties(), measure(), and updateDisplayList() methods called (if needed).
38	valid  when a component is monitored by a Validator and the validation succeeded.
39	valueCommit  when values are changed programmatically or by user interaction.

## Methods Inherited

This class inherits methods from the following classes:

- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Complex Controls

---

Following are the few important Complex Controls:

S.N.	Control & Description
1	DataGrid  The DataGrid control displays a row of column headings above a scrollable grid.
2	AdvancedDataGrid

	The AdvancedDataGrid adds few additional functionality to the standard DataGrid control to add data visualization features.
3	Menu The Menu control creates a pop-up menu of individually selectable choices.
4	ProgressBar The ProgressBar control provides a visual representation of the progress of a task over time.
5	RichTextEditor The RichTextEditor control lets users enter and format text.
6	TileList The TileList control displays a number of items laid out in tiles.
7	Tree The Tree control lets a user view hierarchical data arranged as an expandable tree.
8	VideoPlayer The VideoPlayer control is a skinnable video player that supports progressive download, multi-bitrate streaming, and streaming video.
9	Accordion An Accordion control has a collection of child MX containers or Spark NavigatorContent containers, but only one of them at a time is visible.
10	TabNavigator The TabNavigator control includes a TabBar container for navigating between its child containers.
11	ToggleButtonBar The ToggleButtonBar control defines a horizontal or vertical group of buttons that maintain their selected or deselected state.

## Flex - DataGrid Control

---

The DataGrid control displays a row of column headings above a scrollable grid.

### Class Declaration

Following is the declaration for **spark.components.DataGrid** class:

```
public class DataGrid
    extends SkinnableContainerBase
    implements IFocusManagerComponent, IIIMESupport
```

## Public Properties

S.N.	Property & Description
1	columnsLength : int [read-only] Returns the value of columns.length if the columns IList was specified, otherwise 0.
2	dataProvider : IList A list of data items that correspond to the rows in the grid.
3	dataProviderLength : int [read-only] Returns the value of dataProvider.length if the dataProvider IList was specified, otherwise 0.
4	dataTipField : String The name of the field in the data provider to display as the datatip.
5	dataTipFunction : Function Specifies a callback function to run on each item of the data provider to determine its data tip.
6	editable : Boolean The default value for the GridColumn editable property, which indicates if a corresponding cell's data provider item can be edited.
7	editorColumnIndex : int [read-only] The zero-based column index of the cell that is being edited.
8	editorRowIndex : int [read-only] The zero-based row index of the cell that is being edited.
9	enableIME : Boolean [read-only] A flag that indicates whether the IME should be enabled when the component receives focus.
10	imeMode : String The default value for the GridColumn imeMode property, which specifies the IME (Input Method Editor) mode.

11	<b>itemEditor : IFactory</b>  The default value for the GridColumn itemEditor property, which specifies the IGridItemEditor class used to create item editor instances.
12	<b>itemEditorInstance : IGridItemEditor</b>  [read-only] A reference to the currently active instance of the item editor, if it exists.
13	<b>itemRenderer : IFactory</b>  The item renderer that's used for columns that do not specify one.
14	<b>preserveSelection : Boolean</b>  If true, the selection is preserved when the data provider refreshes its collection.
15	<b>requestedColumnCount : int</b>  The measured width of this grid is large enough to display the first requestedColumnCount columns.
16	<b>requestedMaxRowCount : int</b>  The measured height of the grid is large enough to display no more than requestedMaxRowCount rows.
17	<b>requestedMinColumnCount : int</b>  The measured width of this grid is large enough to display at least requestedMinColumnCount columns.
18	<b>requestedMinRowCount : int</b>  The measured height of this grid is large enough to display at least requestedMinRowCount rows.
19	<b>requestedRowCount : int</b>  The measured height of this grid is large enough to display the first requestedRowCount rows.
20	<b>requireSelection : Boolean</b>  If true and the selectionMode property is not GridSelectionMode.NONE, an item must always be selected in the grid.
21	<b>resizableColumns : Boolean</b>

	Indicates whether the user can change the size of the columns.
22	<p><code>rowHeight : Number</code></p> <p>If <code>variableRowHeight</code> is <code>false</code>, then this property specifies the actual height of each row, in pixels.</p>
23	<p><code>selectedCell : CellPosition</code></p> <p>If <code>selectionMode</code> is <code>GridSelectionMode.SINGLE_CELL</code> or <code>GridSelectionMode.MULTIPLE_CELLS</code>, returns the first selected cell starting at row 0 column 0 and progressing through each column in a row before moving to the next row.</p>
24	<p><code>selectedCells : Vector.&lt;CellPosition&gt;</code></p> <p>If <code>selectionMode</code> is <code>GridSelectionMode.SINGLE_CELL</code> or <code>GridSelectionMode.MULTIPLE_CELLS</code>, returns a Vector of <code>CellPosition</code> Objects representing the positions of the selected cells in the grid.</p>
25	<p><code>selectedIndex : int</code></p> <p>If <code>selectionMode</code> is <code>GridSelectionMode.SINGLE_ROW</code> or <code>GridSelectionMode.MULTIPLE_ROWS</code>, returns the <code>rowIndex</code> of the first selected row.</p>
26	<p><code>selectedIndices : Vector.&lt;int&gt;</code></p> <p>If <code>selectionMode</code> is <code>GridSelectionMode.SINGLE_ROW</code> or <code>GridSelectionMode.MULTIPLE_ROWS</code>, returns a Vector of the selected rows indices.</p>
27	<p><code>selectedItem : Object</code></p> <p>If <code>selectionMode</code> is <code>GridSelectionMode.SINGLE_ROW</code> or <code>GridSelectionMode.MULTIPLE_ROWS</code>, returns the item in the the data provider that is currently selected or undefined if no rows are selected.</p>
28	<p><code>selectedItems : Vector.&lt;Object&gt;</code></p> <p>If <code>selectionMode</code> is <code>GridSelectionMode.SINGLE_ROW</code> or <code>GridSelectionMode.MULTIPLE_ROWS</code>, returns a Vector of the dataProvider items that are currently selected.</p>
29	<p><code>selectionLength : int</code></p> <p>[read-only] If <code>selectionMode</code> is <code>GridSelectionMode.SINGLE_ROW</code> or <code>GridSelectionMode.MULTIPLE_ROWS</code>, returns the number of selected rows.</p>

30	<b>selectionMode : String</b> The selection mode of the control.
31	<b>showDataTips : Boolean</b> If true then a dataTip is displayed for all visible cells.
32	<b>sortableColumns : Boolean</b> Specifies whether the user can interactively sort columns.
33	<b>typicalItem : Object</b> The grid's layout ensures that columns whose width is not specified is wide enough to display an item renderer for this default data provider item.
34	<b>variableRowHeight : Boolean</b> If true, each row's height is the maximum of preferred heights of the cells displayed so far.
35	<b>columns : IList</b> The list of GridColumn Objects displayed by this grid.

## Public Methods

S.N.	Method & Description
1	<b>DataGrid()</b> Constructor.
2	<b>addSelectedCell(rowIndex:int, columnIndex:int):Boolean</b> If selectionMode is GridSelectionMode.SINGLE_CELL or GridSelectionMode.MULTIPLE_CELLS, adds the cell to the selection and sets the caret position to the cell.
3	<b>addSelectedIndex(rowIndex:int):Boolean</b> If selectionMode is GridSelectionMode.MULTIPLE_ROWS, adds this row to the selection and sets the caret position to this row.
4	<b>clearSelection():Boolean</b>

	Removes all of the selected rows and cells, if selectionMode is not GridSelectionMode.NONE.
5	<p><code>endItemEditorSession(cancel:Boolean = false):Boolean</code></p> <p>Closes the currently active editor and optionally saves the editor's value by calling the item editor's <code>save()</code> method.</p>
6	<p><code>ensureCellIsVisible(rowIndex:int, columnIndex:int = -1):void</code></p> <p>If necessary, set the <code>verticalScrollPosition</code> and <code>horizontalScrollPosition</code> properties so that the specified cell is completely visible.</p>
7	<p><code>invalidateCell(rowIndex:int, columnIndex:int):void</code></p> <p>If the specified cell is visible, it is redisplayed.</p>
8	<code>invalidateTypicalItem():void</code>
9	<p><code>removeSelectedCell(rowIndex:int, columnIndex:int):Boolean</code></p> <p>If selectionMode is <code>GridSelectionMode.SINGLE_CELL</code> or <code>GridSelectionMode.MULTIPLE_CELLS</code>, removes the cell from the selection and sets the caret position to the cell.</p>
10	<p><code>removeSelectedIndex(rowIndex:int):Boolean</code></p> <p>If selectionMode is <code>GridSelectionMode.SINGLE_ROW</code> or <code>GridSelectionMode.MULTIPLE_ROWS</code>, removes this row from the selection and sets the caret position to this row.</p>
11	<p><code>selectAll():Boolean</code></p> <p>If selectionMode is <code>GridSelectionMode.MULTIPLE_ROWS</code>, selects all rows and removes the caret or if selectionMode is <code>GridSelectionMode.MULTIPLE_CELLS</code> selects all cells and removes the caret.</p>
12	<p><code>selectCellRegion(rowIndex:int, columnIndex:int, rowCount:uint, columnCount:uint):Boolean</code></p> <p>If selectionMode is <code>GridSelectionMode.MULTIPLE_CELLS</code>, sets the selection to all the cells in the cell region and the caret position to the last cell in the cell region.</p>
13	<p><code>selectIndices(rowIndex:int, rowCount:int):Boolean</code></p> <p>If selectionMode is <code>GridSelectionMode.MULTIPLE_ROWS</code>, sets the selection to the specified rows and the caret position to <code>endRowIndex</code>.</p>

14	<code>selectionContainsCell(rowIndex:int, columnIndex:int):Boolean</code> If selectionMode is GridSelectionMode.SINGLE_CELL or GridSelectionMode.MULTIPLE_CELLS, returns true if the cell is in the current selection.
15	<code>selectionContainsCellRegion(rowIndex:int, columnIndex:int, rowCount:int, columnCount:int):Boolean</code> If selectionMode is GridSelectionMode.MULTIPLE_CELLS, returns true if the cells in the cell region are in the current selection.
16	<code>selectionContainsIndex(rowIndex:int):Boolean</code> If selectionMode is GridSelectionMode.SINGLE_ROW or GridSelectionMode.MULTIPLE_ROWS, returns true if the row at index is in the current selection.
17	<code>selectionContainsIndices(rowIndices:Vector.&lt;int&gt;):Boolean</code> If selectionMode is GridSelectionMode.MULTIPLE_ROWS, returns true if the rows in indices are in the current selection.
18	<code>setSelectedCell(rowIndex:int, columnIndex:int):Boolean</code> If selectionMode is GridSelectionMode.SINGLE_CELL or GridSelectionMode.MULTIPLE_CELLS, sets the selection and the caret position to this cell.
19	<code>setSelectedIndex(rowIndex:int):Boolean</code> If selectionMode is GridSelectionMode.SINGLE_ROW or GridSelectionMode.MULTIPLE_ROWS, sets the selection and the caret position to this row.
20	<code>sortByColumns(columnIndices:Vector.&lt;int&gt;, isInteractive:Boolean = false):Boolean</code> Sort the DataGrid by one or more columns, and refresh the display.
21	<code>startItemEditorSession(rowIndex:int, columnIndex:int):Boolean</code> Starts an editor session on a selected cell in the grid.

## Protected Methods

S.N.	Method & Description

1	commitCaretPosition(newCaretRowIndex:int, newCaretColumnIndex:int):void Updates the grid's caret position.
2	commitInteractiveSelection(selectionEventKind:String, rowIndex:int, columnIndex:int, rowCount:int = 1, columnCount:int = 1):Boolean In response to user input (mouse or keyboard) which changes the selection, this method dispatches the selectionChanging event.

## Events

S.N.	Event & Description
1	caretChange Dispatched by the grid skin part when the caret position, size, or visibility has changed due to user interaction or being programmatically set.
2	gridClick Dispatched by the grid skin part when the mouse is clicked over a cell.
3	gridDoubleClick Dispatched by the grid skin part when the mouse is double-clicked over a cell.
4	gridItemEditorSessionCancel Dispatched after the item editor has been closed without saving its data.
5	gridItemEditorSessionSave Dispatched after the data in item editor has been saved into the data provider and the editor has been closed.
6	gridItemEditorSessionStart Dispatched immediately after an item editor has been opened.
7	gridItemEditorSessionStarting Dispatched when a new item editor session has been requested.
8	gridMouseDown Dispatched by the grid skin part when the mouse button is pressed over a grid cell.

9	gridMouseDrag Dispatched by the grid skin part after a gridMouseDown event if the mouse moves before the button is released.
10	gridMouseUp Dispatched by the grid skin part after a gridMouseDown event when the mouse button is released, even if the mouse is no longer within the grid.
11	gridRollOut Dispatched by the grid skin part when the mouse leaves a grid cell.
12	gridRollOver Dispatched by the grid skin part when the mouse enters a grid cell.
13	selectionChange Dispatched when the selection has changed.
14	selectionChanging Dispatched when the selection is going to change.
15	sortChange Dispatched after the sort has been applied to the data provider's collection.
16	sortChanging Dispatched before the sort has been applied to the data provider's collection.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.SkinnableContainerBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex DataGrid Control Example

Let us follow the following steps to check usage of DataGrid control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            [Bindable]
            public var data:ArrayCollection = new ArrayCollection(
                [
                    {value:"France", code:"FR"},
                    {value:"Japan", code:"JP"},
                    {value:"India", code:"IN"},
                    {value:"Russia", code:"RS"},
                    {value:"United States", code:"US"}
                ]
            );
        ]]>
    </fx:Script>
</s:Application>
```

```

]]>

</fx:Script>

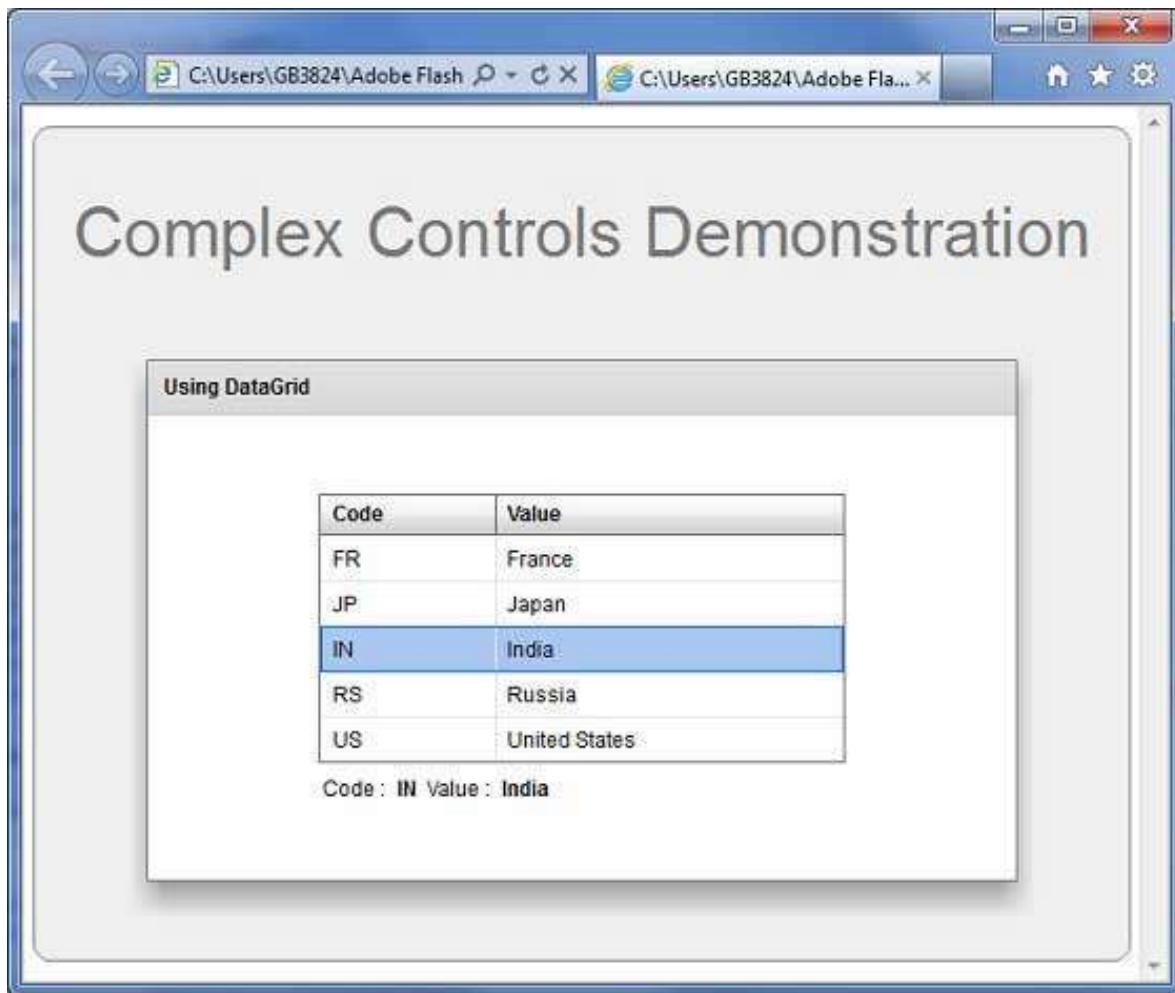
<s:BorderContainer width="630" height="480" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center" verticalAlign="middle">
<s:Label id="lblHeader" text="Complex Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="dataGridPanel" title="Using DataGrid"
width="500" height="300">
<s:layout>
<s:VerticalLayout gap="10" verticalAlign="middle"
horizontalAlign="center"/>

</s:layout>
<s:DataGrid dataProvider="{data}" id="dataGrid">
<s:columns>
<s:ArrayList>
<s:GridColumn dataField="code" width="100"
headerText="Code" />
<s:GridColumn dataField="value" width="200"
headerText="Value" />
</s:ArrayList>
</s:columns>
</s:DataGrid>
<s:HGroup width="60%">
<s:Label text="Code :"/>
<s:Label text="{dataGrid.selectedItem.code}"
fontWeight="bold"/>
<s:Label text="Value :"/>
<s:Label text="{dataGrid.selectedItem.value}"
fontWeight="bold"/>
</s:HGroup>
</s:Panel>
</s:VGroup>

```

```
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – AdvancedDataGrid Control

The AdvancedDataGrid control added multiple functionalities to the standard DataGrid control to add data visualization features to Flex application. These features provide greater control of data display, data aggregation, and data formatting.

### Class Declaration

Following is the declaration for **mx.controls.AdvancedDataGrid** class:

```
public class AdvancedDataGrid
    extends AdvancedDataGridBaseEx
```

## Public Properties

S.N.	Property & Description
1	displayDisclosureIcon : Boolean  Controls the creation and visibility of disclosure icons in the navigation tree.
2	displayItemsExpanded : Boolean  If true, expand the navigation tree to show all items.
3	firstVisibleItem : Object  The data provider element that corresponds to the item that is currently displayed in the top row of the AdvancedDataGrid control.
4	groupedColumns : Array  An Array that defines the hierarchy of AdvancedGridColumn instances when performing column grouping.
5	groupIconFunction : Function  A user-supplied callback function to run on each group item to determine its branch icon in the navigation tree.
6	groupItemRenderer : IFactory  Specifies the item renderer used to display the branch nodes in the navigation tree that correspond to groups.
7	groupLabelFunction : Function  A callback function to run on each item to determine its label in the navigation tree.
8	groupRowHeight : Number  The height of the grouped row, in pixels.
9	hierarchicalCollectionView : IHierarchicalCollectionView  The IHierarchicalCollectionView instance used by the control.
10	itemIcons : Object  An object that specifies the icons for the items.

11	<code>lockedColumnCount : int</code> [override] The index of the first column in the control that scrolls.
12	<code>lockedRowCount : int</code> [override] The index of the first row in the control that scrolls.
13	<code>rendererProviders : Array</code> Array of AdvancedDataGridRendererProvider instances.
14	<code>selectedCells : Array</code> Contains an Array of cell locations as row and column indices.
15	<code>treeColumn : AdvancedGridColumn</code> The column in which the tree is displayed.

## Protected Properties

S.N.	Property & Description
1	<code>anchorColumnIndex : int = -1</code> The column index of the current anchor.
2	<code>caretColumnIndex : int = -1</code> The column name of the item under the caret.
3	<code>cellSelectionTweens : Object</code> A hash table of selection tweens.
4	<code>highlightColumnIndex : int = -1</code> The column index of the item that is currently rolled over or under the cursor.
5	<code>selectedColumnIndex : int = -1</code> The column of the selected cell.
6	<code>treeColumnIndex : int</code>

	[read-only] The tree column number.
7	<b>tween : Object</b> The tween object that animates rows Users can add event listeners to this Object to get notified when the tween starts, updates and ends.
8	<b>visibleCellRenderers : Object</b> A hash table of data provider item renderers currently in view.

## Public Methods

S.N.	Method & Description
1	<b>AdvancedDataGrid()</b> Constructor.
2	<b>collapseAll():void</b> Collapses all the nodes of the navigation tree.
3	<b>expandAll():void</b> Expands all the nodes of the navigation tree in the control.
4	<b>expandChildrenOf(item:Object, open:Boolean):void</b> Opens or closes all the nodes of the navigation tree below the specified item.
5	<b>expandItem(item:Object, open:Boolean, animate:Boolean = false, dispatchEvent:Boolean = false, cause:Event = null):void</b> Opens or closes a branch node of the navigation tree.
6	<b>getParentItem(item:Object):*</b> Returns the parent of a child item.
7	<b>isItemOpen(item:Object):Boolean</b> Returns true if the specified branch node is open.
8	<b>setItemIcon(item:Object, iconID:Class, iconID2:Class):void</b>

	Sets the associated icon in the navigation tree for the item.
--	---

## Protected Methods

S.N.	Method & Description
1	<pre>addCellSelectionData(uid:String, columnIndex:int, selectionData:AdvancedDataGridBaseSelectionData):void</pre> <p>Adds cell selection information to the control, as if you used the mouse to select the cell.</p>
2	<pre>applyCellSelectionEffect(indicator:Sprite, uid:String, columnIndex:int, itemRenderer:IListItemRenderer):void</pre> <p>Sets up the effect for applying the selection indicator.</p>
3	<pre>applyUserStylesForItemRenderer(givenItemRenderer:IListItemRenderer):void</pre> <p>Applies styles from the AdvancedDataGrid control to an item renderer.</p>
4	<pre>atLeastOneProperty(o:Object):Boolean</pre> <p>Returns true if the Object has at least one property, which means that the dictionary has at least one key.</p>
5	<pre>clearCellSelectionData():void</pre> <p>Clears information on cell selection.</p>
6	<pre>clearIndicators():void</pre> <p>[override] Removes all selection and highlight and caret indicators.</p>
7	<pre>clearSelectedCells(transition:Boolean = false):void</pre> <p>Clears the selectedCells property.</p>
8	<pre>dragCompleteHandler(event:DragEvent):void</pre> <p>[override] Handler for the DragEvent.DRAG_COMPLETE event.</p>
9	<pre>dragDropHandler(event:DragEvent):void</pre>

	[override] Handler for the DragEvent.DRAG_DROP event.
10	drawVerticalLine(s:Sprite, colIndex:int, color:uint, x:Number):void [override] Draws a vertical line between columns.
11	finishKeySelection():void [override] Sets selected items based on the caretIndex and anchorIndex properties.
12	initListData(item:Object, adgListData:AdvancedDataGridListData):void Initializes an AdvancedDataGridListData object that is used by the AdvancedDataGrid item renderer.
13	moveIndicators(uid:String, offset:int, absolute:Boolean):void [override] Moves the cell and row selection indicators up or down by the given offset as the control scrolls its display.
14	removeCellSelectionData(uid:String, columnIndex:int):void Removes cell selection information from the control.
15	selectCellItem(item:IListItemRenderer, shiftKey:Boolean, ctrlKey:Boolean, transition:Boolean = true):Boolean Updates the list of selected cells, assuming that the specified item renderer was clicked by the mouse, and the keyboard modifiers are in the specified state.
16	selectItem(item:IListItemRenderer, shiftKey:Boolean, ctrlKey:Boolean, transition:Boolean = true):Boolean [override] Updates the set of selected items given that the item renderer provided was clicked by the mouse and the keyboard modifiers are in the given state.
17	treeNavigationHandler(event:KeyboardEvent):Boolean Handler for keyboard navigation for the navigation tree.

## Events

S.N.	Event & Description

1	headerDragOutside Dispatched when the user drags a column outside of its column group.
2	headerDropOutside Dispatched when the user drops a column outside of its column group.
3	itemClose Dispatched when a branch of the navigation tree is closed or collapsed.
4	itemOpen Dispatched when a branch of the navigation tree is opened or expanded.
5	itemOpening Dispatched when a tree branch open or close operation is initiated.

## Methods Inherited

This class inherits methods from the following classes:

- mx.controls.AdvancedDataGridBaseEx
- mx.controls.AdvancedDataGridBase
- mx.controls.listClasses.AdvancedDataGridBase
- mx.core.ScrollControlBase
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex AdvancedDataGrid Control Example

Let us follow the following steps to check usage of AdvancedDataGrid control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.

2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
            [Bindable]
            public var data:ArrayCollection = new ArrayCollection(
            [
                {value:"France", code:"FR"},
                {value:"Japan", code:"JP"},
                {value:"India", code:"IN"},
                {value:"Russia", code:"RS"},
                {value:"United States", code:"US"}
            ]
            );
        ]]>
    </fx:Script>
    <s:BorderContainer width="630" height="480" id="mainContainer"
        styleName="container">
        <s:VGroup width="100%" height="100%" gap="50"
            horizontalAlign="center" verticalAlign="middle">
            <s:Label id="lblHeader" text="Complex Controls Demonstration">
        </s:VGroup>
    </s:BorderContainer>
</s:Application>
```

```

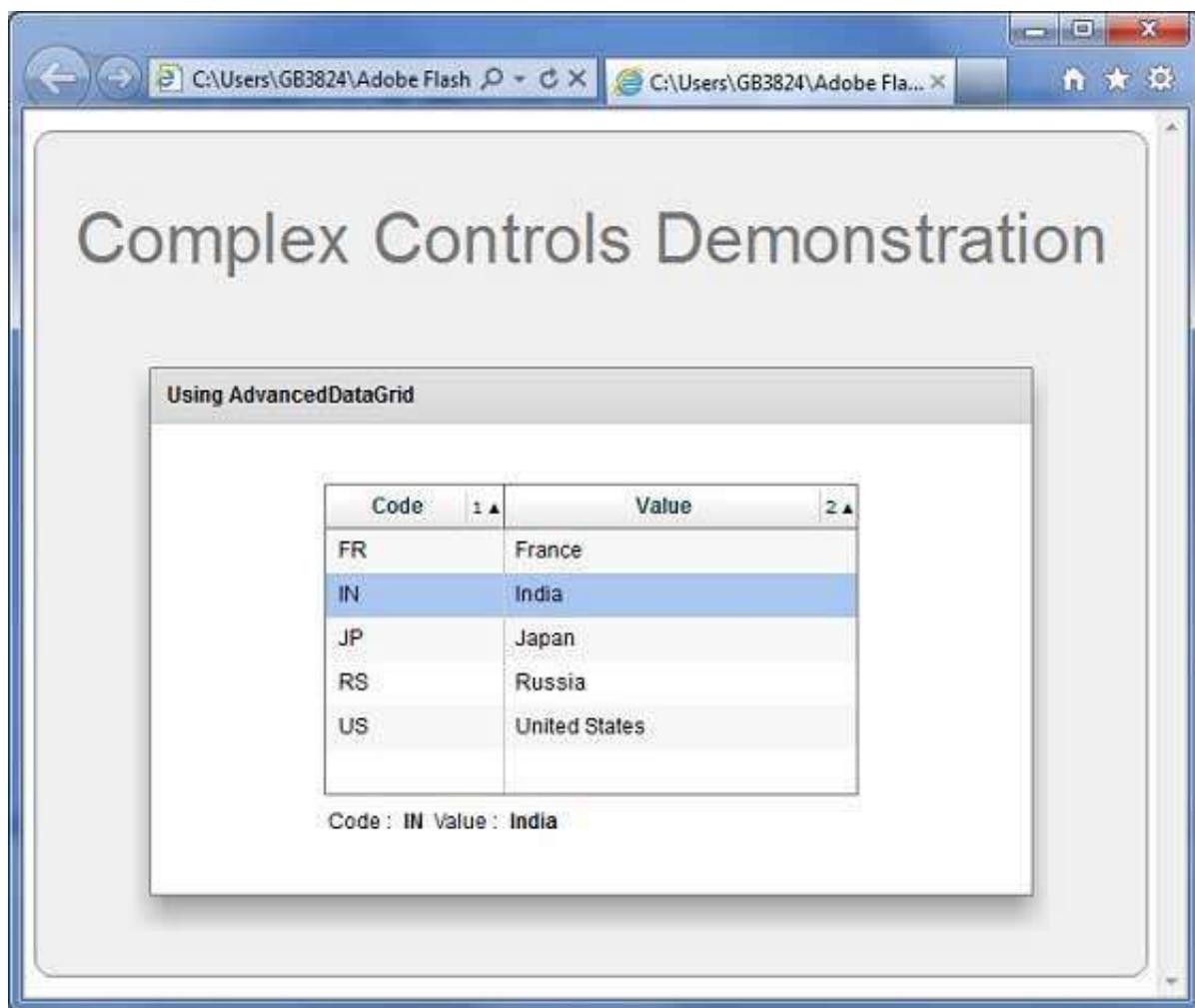
        fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="dataGridPanel" title="Using DataGrid"
width="500" height="300">
<s:layout>
<s:VerticalLayout gap="10" verticalAlign="middle"
horizontalAlign="center"/>

</s:layout>
<mx:AdvancedDataGrid dataProvider="{data}"
id="advancedDataGrid" >
<mx:columns>
<mx:AdvancedDataGridColumn dataField="code"
width="100"
headerText="Code" />
<mx:AdvancedDataGridColumn dataField="value"
width="200"
headerText="Value"/>

</mx:columns>
</mx:AdvancedDataGrid>
<s:HGroup width="60%">
<s:Label text="Code :"/>
<s:Label text="{advancedDataGrid.selectedItem.code}"
fontWeight="bold"/>
<s:Label text="Value :"/>
<s:Label text="{advancedDataGrid.selectedItem.value}"
fontWeight="bold"/>
</s:HGroup>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – Menu Control

The Menu control creates a pop-up menu of singly selectable choices or menu-items. The popped up menu can have as many levels of submenus as needed.

### Class Declaration

Following is the declaration for **mx.controls.Menu** class:

```
public class Menu
    extends List
    implements IFocusManagerContainer
```

### Public Properties

S.N.	Property & Description
1	dataDescriptor : IMenuDataDescriptor

	The object that accesses and manipulates data in the data provider.
2	<p><code>hasRoot : Boolean</code></p> <p>[read-only] A flag that indicates that the current data provider has a root node; for example, a single top node in a hierarchical structure.</p>
3	<p><code>parentMenu : Menu</code></p> <p>The parent menu in a hierarchical chain of menus, where the current menu is a submenu of the parent.</p>
4	<p><code>showRoot : Boolean</code></p> <p>A Boolean flag that specifies whether to display the data provider's root node.</p>

## Public Methods

S.N.	Method & Description
1	<p><code>Menu()</code></p> <p>Constructor.</p>
2	<p><code>createMenu(parent:DisplayObjectContainer, mdp:Object, showRoot:Boolean = true):Menu</code></p> <p>[static] Creates and returns an instance of the Menu class.</p>
3	<p><code>hide():void</code></p> <p>Hides the Menu control and any of its submenus if the Menu control is visible.</p>
4	<p><code>popUpMenu(menu:Menu, parent:DisplayObjectContainer, mdp:Object):void</code></p> <p>[static] Sets thedataProvider of an existing Menu control and places the Menu control in the specified parent container.</p>
5	<p><code>show(xShow:Object = null, yShow:Object = null):void</code></p> <p>Shows the Menu control.</p>

## Protected Methods

S.N.	Method & Description

1	<code>makeListData(data:Object, uid:String, rowNum:int):BaseListData</code> [override] Creates a new MenuListData instance and populates the fields based on the input data provider item.
2	<code>measure():void</code> [override] Calculates the preferred width and height of the Menu based on the widths and heights of its menu items.
3	<code>setMenuItemToggled(item:Object, toggle:Boolean):void</code> Toggles the menu item.

## Events

S.N.	Event & Description
1	<code>change</code> Dispatched when selection changes as a result of user interaction.
2	<code>itemClick</code> Dispatched when a menu item is selected.
3	<code>itemRollOut</code> Dispatched when a user rolls the mouse out of a menu item.
4	<code>itemRollOver</code> Dispatched when a user rolls the mouse over a menu item.
5	<code>menuHide</code> Dispatched when a menu or submenu is dismissed.
6	<code>menuShow</code> Dispatched when a menu or submenu opens.

## Methods Inherited

This class inherits methods from the following classes:

- `mx.controls.List`
- `mx.controls.listClasses.ListBase`

- mx.core.ScrollControlBase
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex Menu Control Example

Let us follow the following steps to check usage of Menu control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            import mx.controls.Menu;
            import mx.events.MenuEvent;
            protected var menu:Menu;
        ]]>
    </fx:Script>

```

```

protected function showMenu(event:MouseEvent):void
{
    menu = Menu.createMenu(null, menuData, false);
    menu.labelXField="@label";
    menu.show(mainContainer.x+menuPanel.x+ 2,
    mainContainer.y +menuPanel.y+32);
    menu.addEventListener(MenuEvent.CHANGE, onMenuChange);
}

protected function hideMenu(event:MouseEvent):void
{
    menu.hide();
}

protected function onMenuChange(event:MenuEvent):void
{
    lblSelected.text = event.label;
}
]]>

</fx:Script>
<fx:Declarations>

<fx:XML format="e4x" id="menuData">
    <root>
        <menuitem label="Menu Item A" >
            <menuitem label="SubMenu Item A 1" enabled="false"/>
            <menuitem label="SubMenu Item A 2"/>
        </menuitem>
        <menuitem label="Menu Item B" type="check" toggled="true"/>
        <menuitem label="Menu Item C" type="check" toggled="false"/>
        <menuitem type="separator"/>
        <menuitem label="Menu Item D" >
            <menuitem label="SubMenu Item D 1" type="radio"
                groupName="one"/>
            <menuitem label="SubMenu Item D 2" type="radio"
                groupName="one" toggled="true"/>
        </menuitem>
    </root>
</fx:XML>

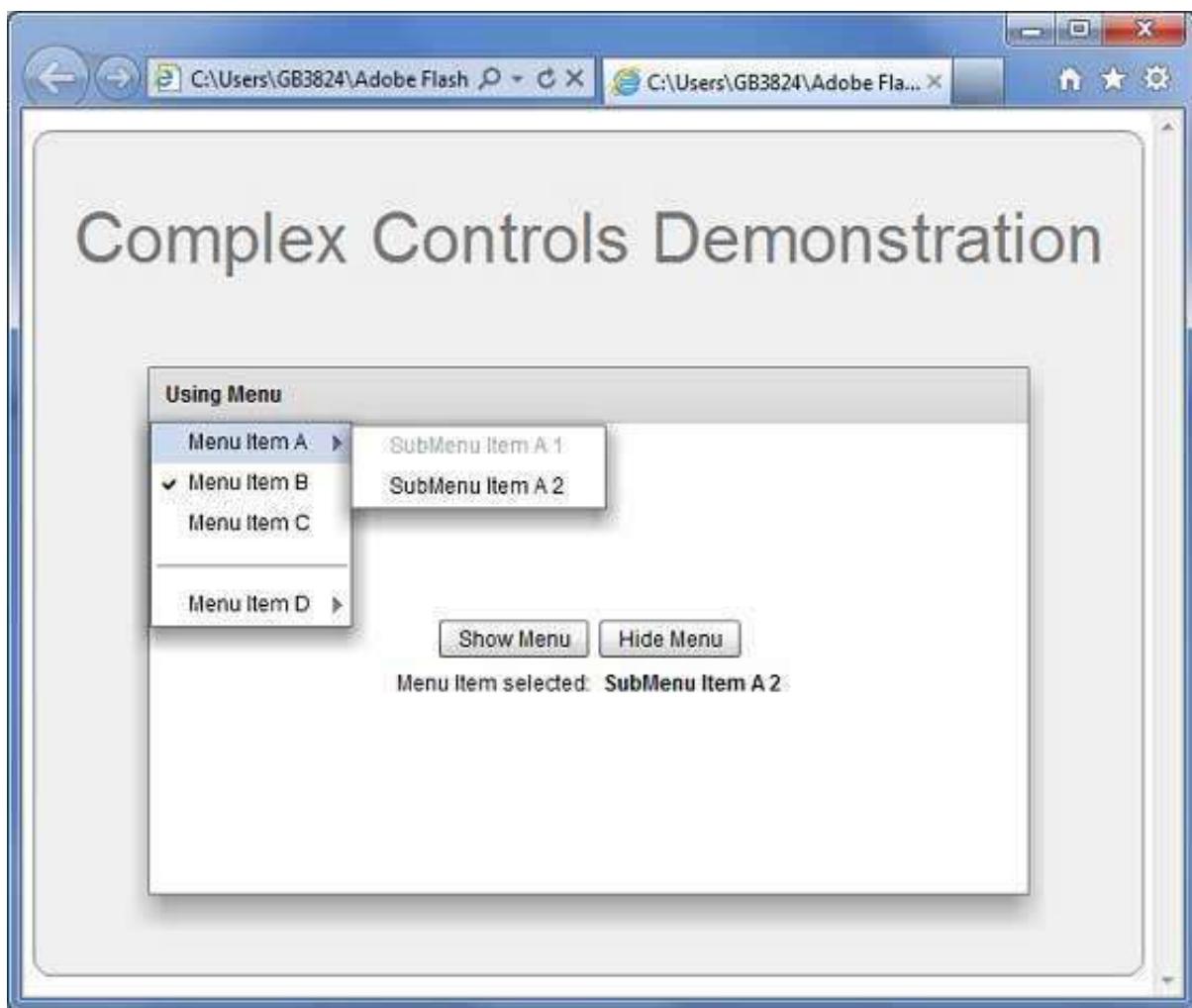
```

```

<menuitem label="SubMenu Item D 3" type="radio"
          groupName="one"/>
</menuitem>
</root>
</fx:XML>
</fx:Declarations>
<s:BorderContainer width="630" height="480" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center" verticalAlign="middle">
<s:Label id="lblHeader" text="Complex Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="menuPanel" title="Using Menu" width="500"
height="300">
<s:layout>
<s:VerticalLayout gap="10" verticalAlign="middle"
horizontalAlign="center"/>
</s:layout>
<s:HGroup>
<s:Button label="Show Menu" click="showMenu(event)" />
<s:Button label="Hide Menu" click="hideMenu(event)" />
</s:HGroup>
<s:HGroup>
<s:Label text="Menu Item selected:" />
<s:Label id="lblSelected" fontWeight="bold"/>
</s:HGroup>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – ProgressBar Control

The ProgressBar control provides users a visual representation of the progress of a task over time.

### Class Declaration

Following is the declaration for **mx.controls.ProgressBar** class:

```
public class ProgressBar
    extends UIComponent
    implements IFontContextComponent
```

### Public Properties

S.N.	Property & Description

1	<b>alignToolTip : String = "Align"</b> The ToolTip that appears when the user hovers over the text alignment buttons.
2	<b>conversion : Number</b> Number used to convert incoming current bytes loaded value and the total bytes loaded values.
3	<b>direction : String</b> Direction in which the fill of the ProgressBar expands toward completion.
4	<b>indeterminate : Boolean</b> Whether the ProgressBar control has a determinate or indeterminate appearance.
5	<b>label : String</b> Text that accompanies the progress bar.
6	<b>labelPlacement : String</b>
7	<b>maximum : Number</b> Largest progress value for the ProgressBar.
8	<b>minimum : Number</b> Smallest progress value for the ProgressBar.
9	<b>mode : String</b> Specifies the method used to update the bar.
10	<b>percentComplete : Number</b> [read-only] Percentage of process that is completed.The range is 0 to 100.
11	<b>source : Object</b> Refers to the control that the ProgressBar is measuring the progress of.
12	<b>value : Number</b> [read-only] Read-only property that contains the amount of progress that has been made - between the minimum and maximum values.

## Public Methods

S.N.	Method & Description
1	ProgressBar() Constructor.
2	setProgress(value:Number, total:Number):void Sets the state of the bar to reflect the amount of progress made when using manual mode.

## Events

S.N.	Event & Description
1	complete Dispatched when the load completes.
2	hide Dispatched when an object's state changes from visible to invisible.
3	progress Dispatched as content loads in event or polled mode.
4	show Dispatched when the component becomes visible.

## Methods Inherited

This class inherits methods from the following classes:

- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex ProgressBar Control Example

Let us follow the following steps to check usage of ProgressBar control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            private var increment:uint=10;

            private function runProgressBar():void
            {
                if(increment <= 100)
                {
                    progressBar.setProgress(increment,100);
                    progressBar.label= "Current Progress" + " " + increment + "%";
                    increment+=10;
                }
                if(increment > 100)
        
```

```

    {
        increment=0;
    }
}

]]>

</fx:Script>

<s:BorderContainer width="630" height="480" id="mainContainer"
styleName="container">

<s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center" verticalAlign="middle">

<s:Label id="lblHeader" text="Complex Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>

<s:Panel id="progressBarPanel" title="Using ProgressBar"
width="500" height="300">

<s:layout>
    <s:VerticalLayout gap="10" verticalAlign="middle"
        horizontalAlign="center"/>
</s:layout>

<s:Label color="0x323232"
text="Click the button to start the progress bar." />

<s:Button id="btnStart" label="Start"
click="runProgressBar();"/>

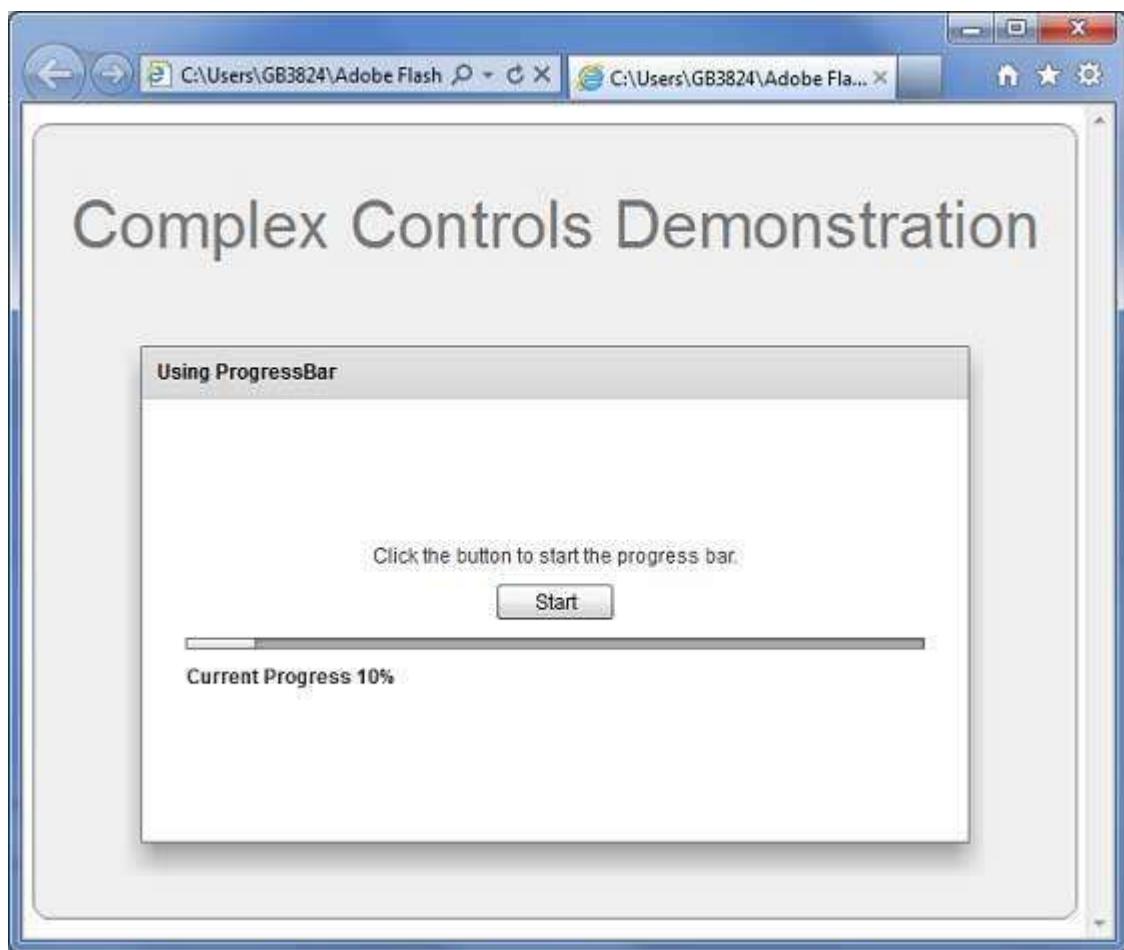
<mx:ProgressBar id="progressBar"
labelPlacement="bottom" minimum="0"
visible="true" maximum="100"
color="0x323232" label="CurrentProgress 0%"
direction="right" mode="manual" width="90%"/>

</s:Panel>

</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – RichTextEditor Control

The RichTextEditor control provide user option to enter and format text. User can change font family, color, size, and style, and other properties such as text alignment, bullets and URL links. The control consists of a Panel control with two controls

- A Text Area control where users can enter text.
- A Container with format controls that let a user specify the text characteristics. The format controls affect text being typed or selected text.

### Class Declaration

Following is the declaration for **mx.controls.RichTextEditor** class:

```
public class RichTextEditor
    extends Panel
```

### Public Properties

S.N.	Property & Description

1	<code>alignToolTip : String = "Align"</code> The ToolTip that appears when the user hovers over the text alignment buttons.
2	<code>boldToolTip : String = "Bold"</code> The ToolTip that appears when the user hovers over the text bold button.
3	<code>bulletToolTip : String = "Bullet"</code> The ToolTip that appears when the user hovers over the bulleted list button.
4	<code>colorPickerToolTip : String = "Color"</code> The ToolTip that appears when the user hovers over the ColorPicker control.
5	<code>defaultLinkProtocol : String</code> The default protocol string to use at the start of link text.
6	<code>fontFamilyToolTip : String = "Font Family"</code> The ToolTip that appears when the user hovers over the font drop-down list.
7	<code>fontSizeToolTip : String = "Font Size"</code> The ToolTip that appears when the user hovers over the font size drop-down list.
8	<code>htmlText : String</code> Text containing HTML markup that displays in the RichTextEditor control's TextArea subcontrol.
9	<code>italicToolTip : String = "Italic"</code> The ToolTip that appears when the user hovers over the text italic button.
10	<code>linkToolTip : String = "Link"</code> The ToolTip that appears when the user hovers over the link text input field.
11	<code>selection : mx.controls.textClasses:TextRange</code> [read-only] A TextRange object containing the selected text in the TextArea subcontrol.
12	<code>showControlBar : Boolean</code>

	Specifies whether to display the control bar that contains the text formatting controls.
13	<b>showToolTips : Boolean</b> Specifies whether to display tooltips for the text formatting controls.
14	<b>text : String</b> Plain text without markup that displays in the RichTextEditor control's TextArea subcontrol.
15	<b>underlineToolTip : String = "Underline"</b> The ToolTip that appears when the user hovers over the text underline button.

## Public Methods

S.N.	Method & Description
1	<b>RichTextEditor()</b> Constructor.

## Events

S.N.	Event & Description
1	<b>change</b> Dispatched when the user changes the contents or format of the text in the TextArea control.

## Methods Inherited

This class inherits methods from the following classes:

- mx.containers.Panel
- mx.core.Container
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex RichTextEditor Control Example

Let us follow the following steps to check usage of RichTextEditor control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            private var richTextString:String = "You can enter text into
            the"
            +" <b>RichTextEditor control</b>.Click <b>"
            +"<font color='#BB50AA'>buttons</font></b> to display"
            +" entered text as <i>plain text</i>, "
            +"or as <i>HTML-formatted</i> text.";
        ]]>
    </fx:Script>
    <s:BorderContainer width="630" height="480" id="mainContainer"
        styleName="container">
        <s:VGroup width="100%" height="100%" gap="50">
```

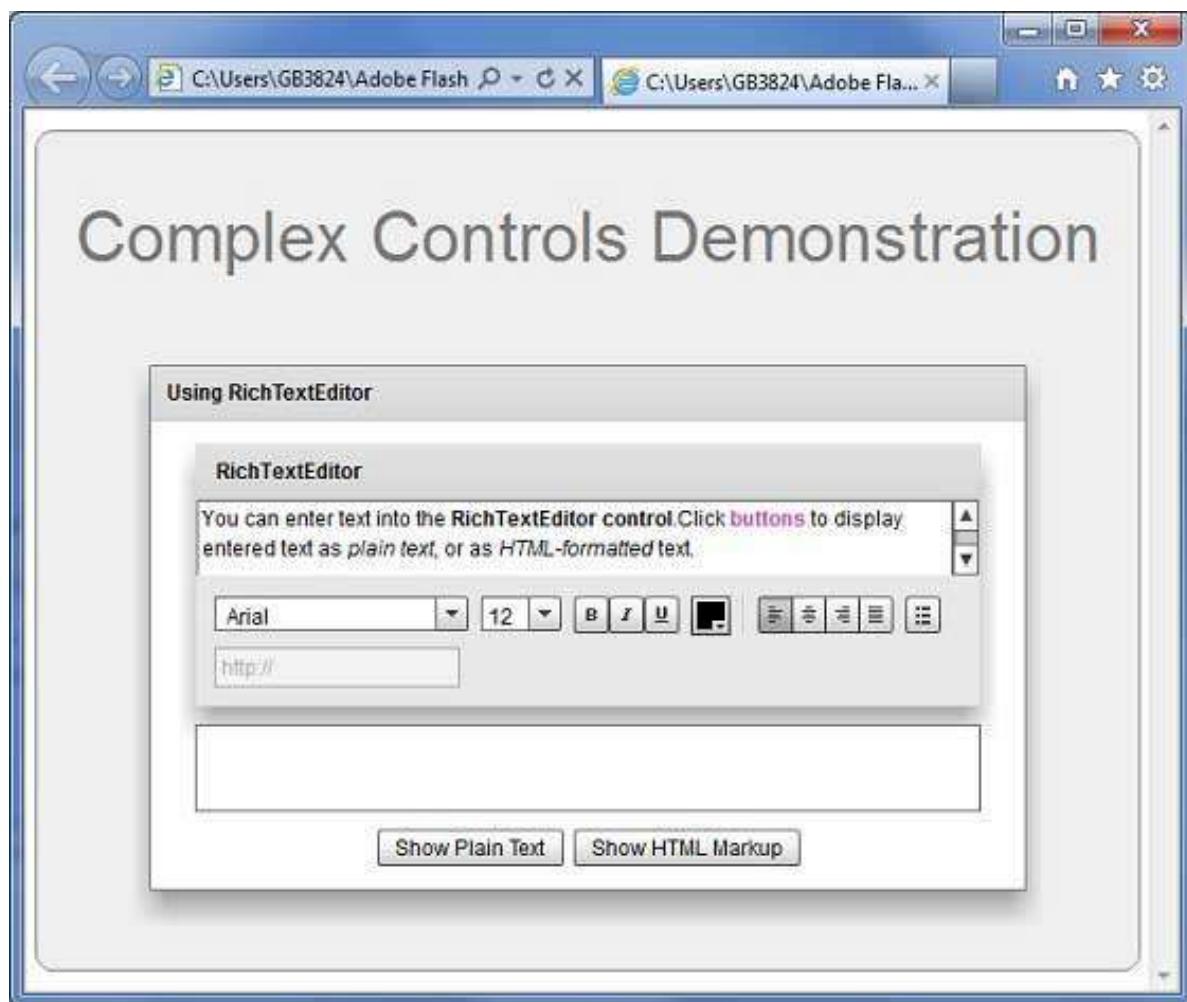
```

horizontalAlign="center" verticalAlign="middle">
<s:Label id="lblHeader" text="Complex Controls Demonstration"
    fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="richTextEditorPanel" title="Using RichTextEditor"
    width="500" height="300">
    <s:layout>
        <s:VerticalLayout gap="10" verticalAlign="middle"
            horizontalAlign="center"/>
    </s:layout>
    <mx:RichTextEditor id="richTextEditor"
        title="RichTextEditor"
        width="90%" height="150"
        borderAlpha="0.15"

creationComplete="richTextEditor.htmlText=richTextString;" />
    <s:TextArea id="richText" width="90%" height="50" />
    <s:HGroup>
        <s:Button label="Show Plain Text"
            click="richText.text=richTextEditor.text;"/>
        <s:Button label="Show HTML Markup"
            click="richText.text=richTextEditor.htmlText;"/>
    </s:HGroup>
    </s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – TileList Control

The TileList control displays a number of items laid out in tiles.

### Class Declaration

Following is the declaration for **mx.controls.TileList** class:

```
public class TileList
    extends TileBase
```

### Public Methods

S.N.	Method & Description
1	TileList() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- mx.controls.listClasess.TileBase
- mx.controls.listClasess.ListBase
- mx.core.ScrollControlBase
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex TileList Control Example

Let us follow the following steps to check usage of TileList control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Add a folder <i>images</i> to <i>src</i> folder. And add sample images to it.
3	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
4	Compile and run the application to make sure business logic is working as per the requirements.

Following is the way to embed an image in a flex application.

```
<fx:Script>
<![CDATA[
    //images folder should be in src folder
    //and maven-mini-logo.png should reside inside images folder.
    [Bindable]
]]>
```

```
[Embed(source="images/maven-mini-logo.png")]
public var image1:Class;
]]>
</fx:Script>
```

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            [Bindable]
            [Embed(source="images/maven-mini-logo.png")]
            public var image1:Class;

            [Bindable]
            [Embed(source="images/flex-mini.png")]
            public var image2:Class;

            [Bindable]
            [Embed(source="images/gwt-mini.png")]
            public var image3:Class;

            [Bindable]
            [Embed(source="images/junit-mini-logo.png")]
            public var image4:Class;

            [Bindable]
            [Embed(source="images/cpp-mini-lib-logo.png")]
            public var image5:Class;
        ]]>
    </fx:Script>
</s:Application>
```

```

]]>

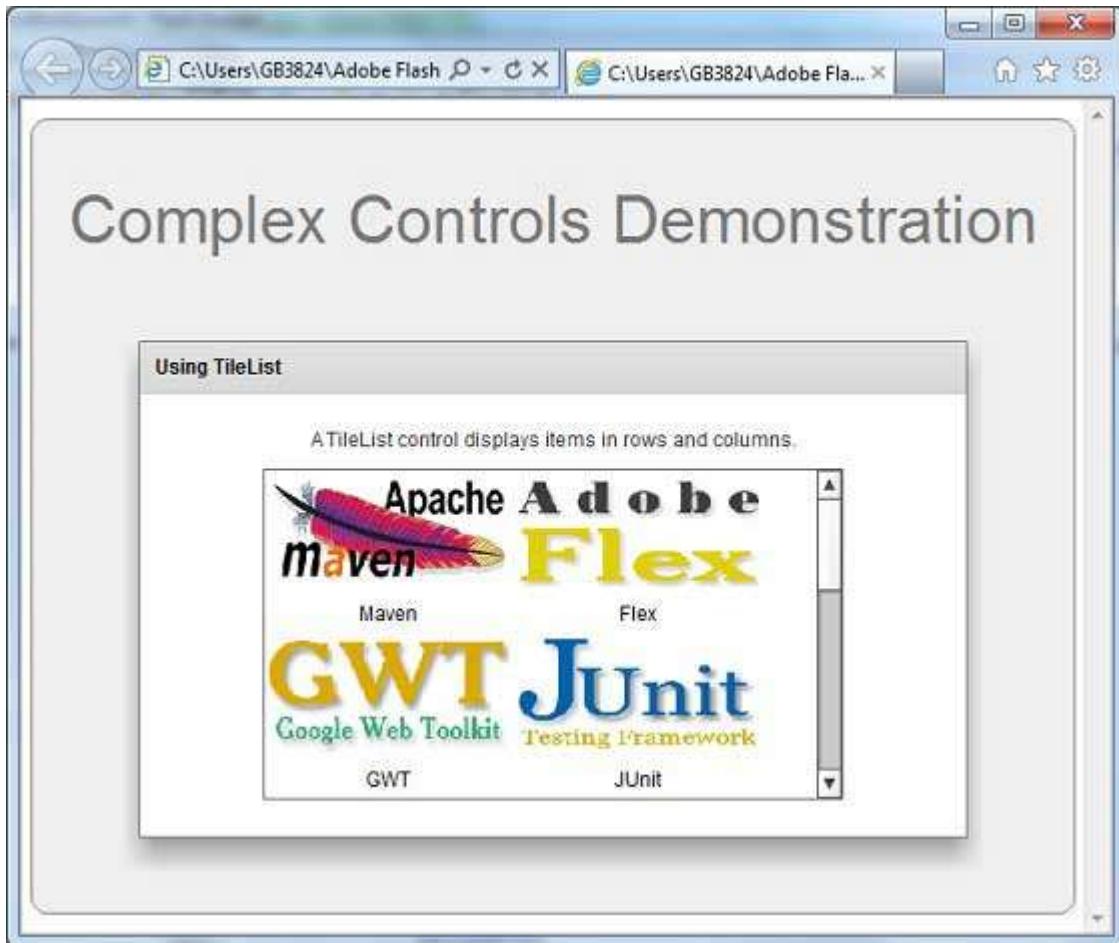
</fx:Script>

<s:BorderContainer width="630" height="480" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center" verticalAlign="middle">
<s:Label id="lblHeader" text="Complex Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="tileListPanel" title="Using TileList"
width="500" height="300">
<s:layout>
<s:VerticalLayout gap="10" verticalAlign="middle"
horizontalAlign="center"/>

</s:layout>
<s:Label width="100%" textAlign="center" color="0x323232"
text="A TileList control displays items in rows and
columns."/>
<mx:TileList id="tileList" height="200" width="350"
maxColumns="2" rowHeight="100" columnWidth="150">
<mx:dataProvider>
<fx:Array>
<fx:Object label="Maven" icon="{image1}"/>
<fx:Object label="Flex" icon="{image2}"/>
<fx:Object label="GWT" icon="{image3}"/>
<fx:Object label="JUnit" icon="{image4}"/>
<fx:Object label="C++" icon="{image5}"/>
</fx:Array>
</mx:dataProvider>
</mx:TileList>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – Tree Control

The Tree control displays hierarchical data arranged as an expandable tree.

### Class Declaration

Following is the declaration for **mx.controls.Tree** class:

```
public class Tree
    extends List
    implements IIIMESupport
```

### Public Properties

S.N.	Property & Description

1	<b>dataDescriptor</b> : mx.controls.treeClasses:ITreeDataDescriptor  Returns the current ITreeDataDescriptor.
2	<b>dataProvider</b> : Object  [override] An object that contains the data to be displayed.
3	<b>dragMoveEnabled</b> : Boolean  [override] Indicates that items can be moved instead of just copied from the Tree control as part of a drag-and-drop operation.
4	<b>firstVisibleItem</b> : Object  The item that is currently displayed in the top row of the tree.
5	<b>hasRoot</b> : Boolean  [read-only] Indicates that the current dataProvider has a root item; for example, a single top node in a hierarchical structure.
6	<b>itemIcons</b> : Object  An object that specifies the icons for the items.
7	<b>maxHorizontalScrollPosition</b> : Number  [override] The maximum value for the maxHorizontalScrollPosition property for the Tree control.
8	<b>openItems</b> : Object  The items that have been opened or set opened.
9	<b>showRoot</b> : Boolean  Sets the visibility of the root item.

## Public Methods

S.N.	<b>Method &amp; Description</b>
1	<b>Tree()</b> Constructor.

2	<code>expandChildrenOf(item:Object, open:Boolean):void</code> Opens or closes all the tree items below the specified item.
3	<code>expandItem(item:Object, open:Boolean, animate:Boolean = false, dispatchEvent:Boolean = false, cause:Event = null):void</code> Opens or closes a branch item.
4	<code>getParentItem(item:Object):*</code> Returns the known parent of a child item.
5	<code>isItemOpen(item:Object):Boolean</code> Returns true if the specified item branch is open (is showing its children).
6	<code>setItemIcon(item:Object, iconID:Class, iconID2:Class):void</code> Sets the associated icon for the item.

## Protected Methods

S.N.	Method & Description
1	<code>dragCompleteHandler(event:DragEvent):void</code> [override] Handles DragEvent.DRAG_COMPLETE events.
2	<code>dragDropHandler(event:DragEvent):void</code> [override] Handles DragEvent.DRAG_DROP events.
3	<code>initListData(item:Object, treeListData:mx.controls.treeClasses:TreeListData):void</code> Initializes a TreeListData object that is used by the tree item renderer.
4	<code>makeListData(data:Object, uid:String, rowNum:int):BaseListData</code> [override] Creates a new TreeListData instance and populates the fields based on the input data provider item.

## Events

S.N.	Event & Description

1	itemClose Dispatched when a branch is closed or collapsed.
2	itemOpen Dispatched when a branch is opened or expanded.
3	itemOpening Dispatched when a branch open or close is initiated.

## Methods Inherited

This class inherits methods from the following classes:

- mx.controls.List
- mx.controls.listClasess.ListBase
- mx.core.ScrollControlBase
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex Tree Control Example

Let us follow the following steps to check usage of Tree control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            [Bindable]
            public var selectedNode:XML;

            // Event handler for the Tree control change event.
            public function treeChanged(event:Event):void {
                selectedNode=Tree(event.target).selectedItem as XML;
            }
        ]]>
    </fx:Script>
    <fx:Declarations>
        <fx:XMLList id="treeData">
            <node label="E-Mail Box">
                <node label="Inbox">
                    <node label="Client"/>
                    <node label="Product"/>
                    <node label="Personal"/>
                </node>
                <node label="Sent Items">
                    <node label="Professional"/>
                    <node label="Personal"/>
                </node>
                <node label="Deleted Items"/>
                <node label="Spam"/>
            </node>
        </fx:XMLList>
    </fx:Declarations>

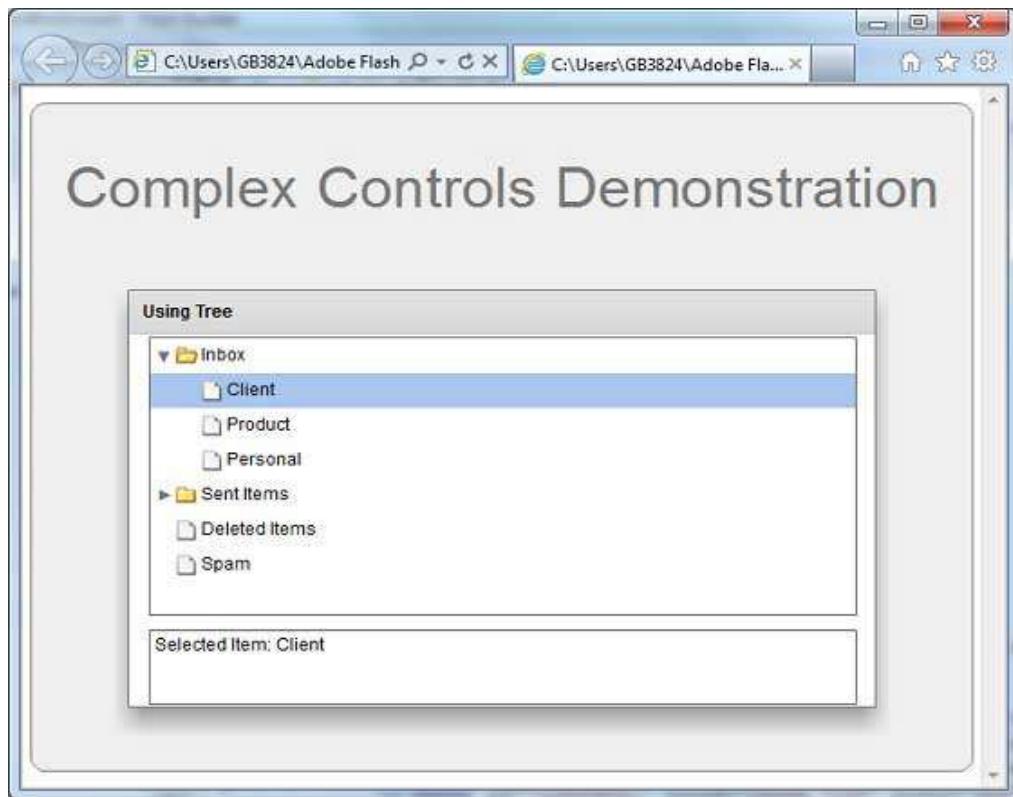
```

```

</fx:XMLList>
</fx:Declarations>
<s:BorderContainer width="630" height="480" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center" verticalAlign="middle">
<s:Label id="lblHeader" text="Complex Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="treePanel" title="Using Tree"
width="500" height="300">
<s:layout>
<s:VerticalLayout gap="10" verticalAlign="middle"
horizontalAlign="center"/>
</s:layout>
<mx:Tree id="tree" width="95%" height="75%"
labelField="@label" showRoot="false"
dataProvider="{treeData}"
change="treeChanged(event)"/>
<s:TextArea height="20%" width="95%"
text="Selected Item: {selectedNode.@label}"/>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – VideoPlayer Control

The Video control supports playback of FLV and F4v files. This control contains a full-featured UI for controlling video playback.

### Class Declaration

Following is the declaration for **spark.components.VideoPlayer** class:

```
public class VideoPlayer
    extends SkinnableComponent
```

### Public Properties

S.N.	Property & Description
1	<b>autoDisplayFirstFrame : Boolean</b>  If autoPlay = false, then autoDisplayFirstFrame controls whether the video is loaded when the source is set.
2	<b>autoPlay : Boolean</b>

	Specifies whether the video starts playing immediately when the source property is set.
3	<p><b>autoRewind : Boolean</b></p> <p>Specifies whether the FLV file should rewind to the first frame when play stops, either by calling the stop() method or by reaching the end of the stream.</p>
4	<p><b>bytesLoaded : Number</b></p> <p>[read-only] The number of bytes of data that have been downloaded into the application.</p>
5	<p><b>bytesTotal : Number</b></p> <p>[read-only] The total size in bytes of the data being downloaded into the application.</p>
6	<p><b>currentTime : Number</b></p> <p>[read-only] Current time of the playhead, measured in seconds, since the video starting playing.</p>
7	<p><b>duration : Number</b></p> <p>[read-only] Duration of the video's playback, in seconds.</p>
8	<p><b>loop : Boolean</b></p> <p>Indicates whether the media should play again after playback has completed.</p>
8	<p><b>mediaPlayerState : String</b></p> <p>[read-only] The current state of the video.</p>

10	<b>muted : Boolean</b>  Set to true to mute the video, false to unmute the video.
11	<b>pauseWhenHidden : Boolean</b>  Controls whether the video continues to play when it is "hidden".
12	<b>playing : Boolean</b>  [read-only] Contains true if the video is playing or is attempting to play.
13	<b>scaleMode : String</b>  The scaleMode property describes different ways of sizing the video content.
14	<b>source : Object</b>  The video source.
15	<b>videoObject : Video</b>  [read-only] The underlying flash player flash.media.Video object.
16	<b>volume : Number</b>  The volume level, specified as a value between 0 and 1.

## Public Methods

S.N.	Method & Description
1	VideoPlayer() Constructor.
2	pause():void

	Pauses playback without moving the playhead.
3	<code>play():void</code> Causes the video to play.
4	<code>seek(time:Number):void</code> Seeks to given time in the video.
5	<code>stop():void</code> Stops video playback.

## Protected Methods

S.N.	Method & Description
1	<code>formatTimeValue(value:Number):String</code> Formats a time value, specified in seconds, into a String that gets used for currentTime and the duration.

## Events

S.N.	Method & Description
1	<code>bytesLoadedChange</code> Dispatched when the data is received as a download operation progresses.
2	<code>complete</code> Dispatched when the playhead reaches the duration for playable media.
3	<code>currentTimeChange</code> Dispatched when the currentTime property of the MediaPlayer has changed.
4	<code>durationChange</code> Dispatched when the duration property of the media has changed.
5	<code>mediaPlayerStateChange</code> Dispatched when the MediaPlayer's state has changed.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex VideoPlayer Control Example

Let us follow the following steps to check usage of VideoPlayer control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Add a folder <i>video</i> to <i>src</i> folder. And add sample video to it.
3	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
4	Compile and run the application to make sure business logic is working as per the requirements.

Following is the way to embed an video in a flex application.

```
<s:VideoPlayer source="video/just for laugh magic trick.flv"
width="350" height="250" loop="true"/>
```

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
```

```

>
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<s:BorderContainer width="630" height="480" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center" verticalAlign="middle">
<s:Label id="lblHeader" text="Complex Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="videoPlayerPanel" title="Using VideoPlayer"
width="500" height="300" >
<s:layout>
<s:HorizontalLayout gap="10" verticalAlign="middle"
horizontalAlign="center"/>
</s:layout>
<s:VideoPlayer
source="video/just for laugh magic trick.flv"
width="350" height="250"
loop="true"/>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – Accordion Control

The Accordion control is a navigator container which has a collection of child containers, but only one of them at a time is visible.

### Class Declaration

Following is the declaration for **mx.containers.Accordion** class:

```
public class Accordion
    extends Container
    implements IHistoryManagerClient, IFocusManagerComponent
```

### Public Properties

S.N.	Property & Description
1	<b>headerRenderer : IFactory</b> A factory used to create the navigation buttons for each child.

2	historyManagementEnabled : Boolean  If set to true, this property enables history management within this Accordion container.
3	resizeToContent : Boolean  If set to true, this Accordion automatically resizes to the size of its current child.
4	selectedChild : INavigatorContent  A reference to the currently visible child container.
5	selectedIndex : int  The zero-based index of the currently visible child container.

## Protected Properties

S.N.	Property & Description
1	contentHeight : Number  [read-only] The height of the area, in pixels, in which content is displayed.
2	contentWidth : Number  [read-only] The width of the area, in pixels, in which content is displayed.

## Public Methods

S.N.	Method & Description
1	Accordion()  Constructor.
2	getHeaderAt(index:int):Button  Returns a reference to the navigator button for a child container.
3	loadState(state:Object):void  Loads the state of this object.
4	saveState():Object

	Saves the state of this object.
--	---------------------------------

## Events

S.N.	Event & Description
1	<p>change</p> <p>Dispatched when the selected child container changes.</p>

## Methods Inherited

This class inherits methods from the following classes:

- mx.core.Container
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex Accordion Control Example

Let us follow the following steps to check usage of Accordion control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
```

256

```

xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx"
width="100%" height="100%" minWidth="500" minHeight="500"
>
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<s:BorderContainer width="630" height="480" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="50"
horizontalAlign="center" verticalAlign="middle">
<s:Label id="lblHeader" text="Complex Controls Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="accordionPanel" title="Using Accordian"
width="500" height="300" >
<s:layout>
<s:HorizontalLayout gap="10" verticalAlign="middle"
horizontalAlign="center"/>
</s:layout>
<mx:Accordion id="accordion" width="95%" height="90%">
<s:NavigatorContent label="Section 1" width="100%"
height="100%">
<s:VGroup verticalAlign="middle"
horizontalAlign="center"
width="100%" height="100%">
<s:Label text="Contents for Section 1"/>
</s:VGroup>
</s:NavigatorContent>
<s:NavigatorContent label="Section 2" width="100%"
height="100%">
<s:VGroup verticalAlign="middle"
horizontalAlign="center"
width="100%" height="100%">
<s:Label text="Contents for Section 2"/>
</s:VGroup>
</s:NavigatorContent>
<s:NavigatorContent label="Section 3" width="100%">

```

```
        height="100%"

        <s:VGroup verticalAlign="middle"
horizontalAlign="center"

            width="100%" height="100%"

            <s:Label text="Contents for Section 3"/>

        </s:VGroup>
    </s:NavigatorContent>

    <s:NavigatorContent label="Section 4" width="100%"

        height="100%"

        <s:VGroup verticalAlign="middle"
horizontalAlign="center"

            width="100%" height="100%"

            <s:Label text="Contents for Section 4"/>

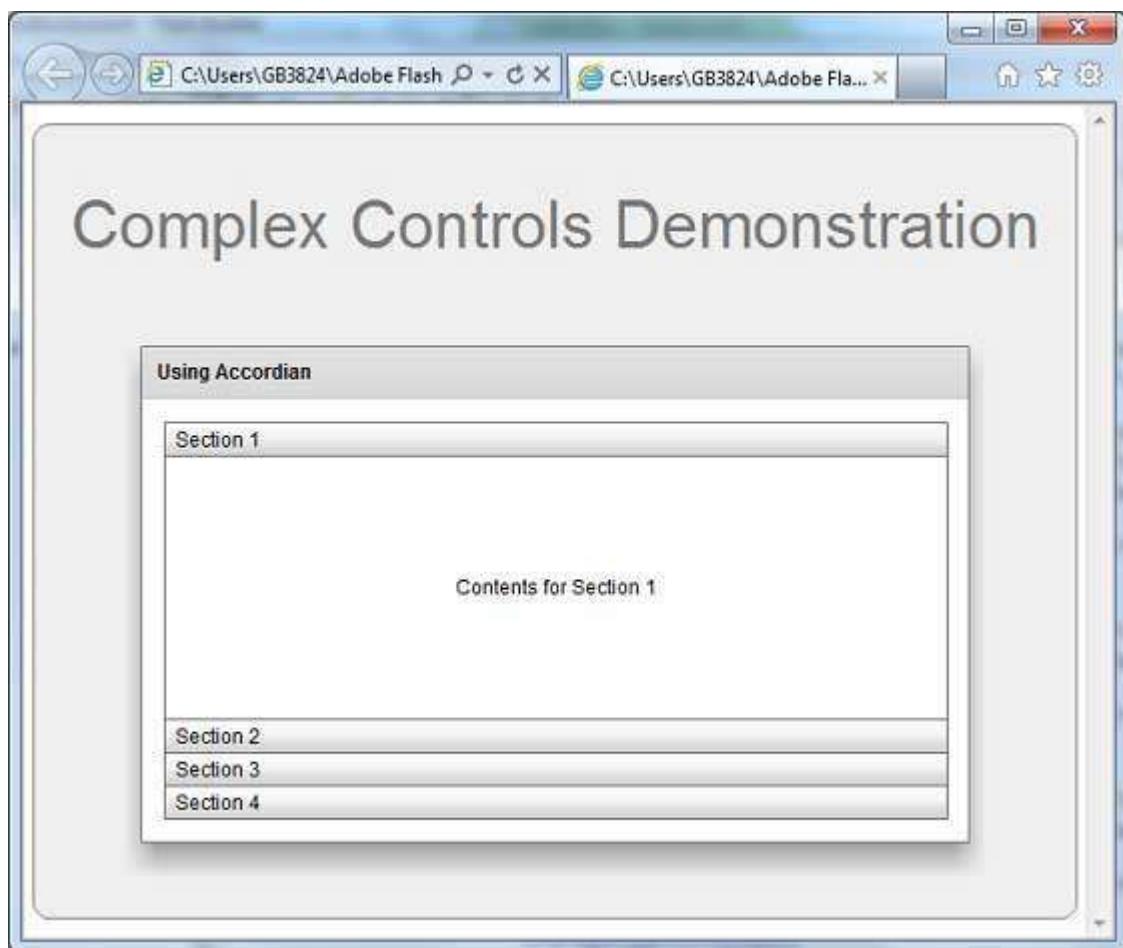
        </s:VGroup>
    </s:NavigatorContent>

</mx:Accordion>

</s:Panel>

</s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – TabNavigator Control

The TabNavigator control is a navigator container which has a collection of child containers, but only one of them at a time is visible.

### Class Declaration

Following is the declaration for **mx.containers.TabNavigator** class:

```
public class TabNavigator
    extends ViewStack
    implements IFocusManagerComponent
```

### Protected Properties

S.N.	Property & Description
1	<b>tabBar : mx.controls:TabBar</b> A reference to the TabBar inside this TabNavigator.

2	<b>tabBarStyleFilters : Object</b>
---	------------------------------------

[read-only] The set of styles to pass from the TabNavigator to the tabBar.

## Public Methods

S.N.	Method & Description
1	<b>TabNavigator()</b> Constructor.
2	<b>getTabAt(index:int):Button</b> Returns the tab of the navigator's TabBar control at the specified index.

## Protected Methods

S.N.	Method & Description
1	<b>measure():void</b> [override] Calculates the default sizes and mininum and maximum values of this TabNavigator container.
2	<b>updateDisplayList(unscaledWidth:Number, unscaledHeight:Number):void</b> [override] Responds to size changes by setting the positions and sizes of this container's tabs and children.

## Methods Inherited

This class inherits methods from the following classes:

- mx.containers.ViewStack
- mx.core.Container
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher

- Object

## Flex TabNavigator Control Example

Let us follow the following steps to check usage of TabNavigator control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name HelloWorld under a package com.tutorialspoint.client as explained in the Flex - Create Application chapter.
2	Modify HelloWorld.mxml as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml files src/com.tutorialspoint/HelloWorld.mxml.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <s:BorderContainer width="630" height="480" id="mainContainer"
        styleName="container">
        <s:VGroup width="100%" height="100%" gap="50"
            horizontalAlign="center" verticalAlign="middle">
            <s:Label id="lblHeader" text="Complex Controls Demonstration"
                fontSize="40" color="0x777777" styleName="heading"/>
            <s:Panel id="tabNavigatorPanel" title="Using TabNavigator"
                width="500" height="300" >
                <s:layout>
                    <s:HorizontalLayout gap="10" verticalAlign="middle"
                        horizontalAlign="center"/>
                
```

```

        </s:layout>

<mx:TabNavigator id="tabNavigator" width="95%" height="90%>
    <s:NavigatorContent label="Section 1" width="100%" height="100%">
        <s:VGroup verticalAlign="middle" horizontalAlign="center" width="100%" height="100%">
            <s:Label text="Contents for Section 1"/>
        </s:VGroup>
    </s:NavigatorContent>
    <s:NavigatorContent label="Section 2" width="100%" height="100%">
        <s:VGroup verticalAlign="middle" horizontalAlign="center" width="100%" height="100%">
            <s:Label text="Contents for Section 2"/>
        </s:VGroup>
    </s:NavigatorContent>
    <s:NavigatorContent label="Section 3" width="100%" height="100%">
        <s:VGroup verticalAlign="middle" horizontalAlign="center" width="100%" height="100%">
            <s:Label text="Contents for Section 3"/>
        </s:VGroup>
    </s:NavigatorContent>
    <s:NavigatorContent label="Section 4" width="100%" height="100%">
        <s:VGroup verticalAlign="middle" horizontalAlign="center" width="100%" height="100%">
            <s:Label text="Contents for Section 4"/>
        </s:VGroup>
    </s:NavigatorContent>
</mx:TabNavigator>

```

```

    </s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – ToggleButtonBar Control

The ToggleButtonBar control provides a horizontal or vertical group of buttons that maintain their selected or deselected state.

### Class Declaration

Following is the declaration for **mx.controls.ToggleButtonBar** class:

```

public class ToggleButtonBar
    extends ButtonBar

```

## Public Properties

S.N.	Property & Description
1	selectedIndex : int [override] Index of the selected button.
2	toggleOnClick : Boolean Specifies whether the currently selected button can be deselected by the user.

## Public Methods

S.N.	Method & Description
1	ToggleButtonBar() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- mx.controls.ButtonBar
- mx.controls.NavBar
- mx.containers.Box
- mx.core.Container
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex ToggleButtonBar Control Example

Let us follow the following steps to check usage of ToggleButtonBar control in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.

2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            import mx.events.ItemClickEvent;
            private function sectionClickHandler(event:ItemClickEvent):void
            {
                sectionContents.text = "Contents for " + event.label;
            }
        ]]>
    </fx:Script>
    <fx:Declarations>
        <fx:Array id="sectionList">
            <fx:String>Section 1</fx:String>
            <fx:String>Section 2</fx:String>
            <fx:String>Section 3</fx:String>
            <fx:String>Section 4</fx:String>
            <fx:String>Section 5</fx:String>
        </fx:Array>
    </fx:Declarations>
    <s:BorderContainer width="630" height="480" id="mainContainer"
        styleName="container">
        <s:VGroup width="100%" height="100%" gap="50">
```

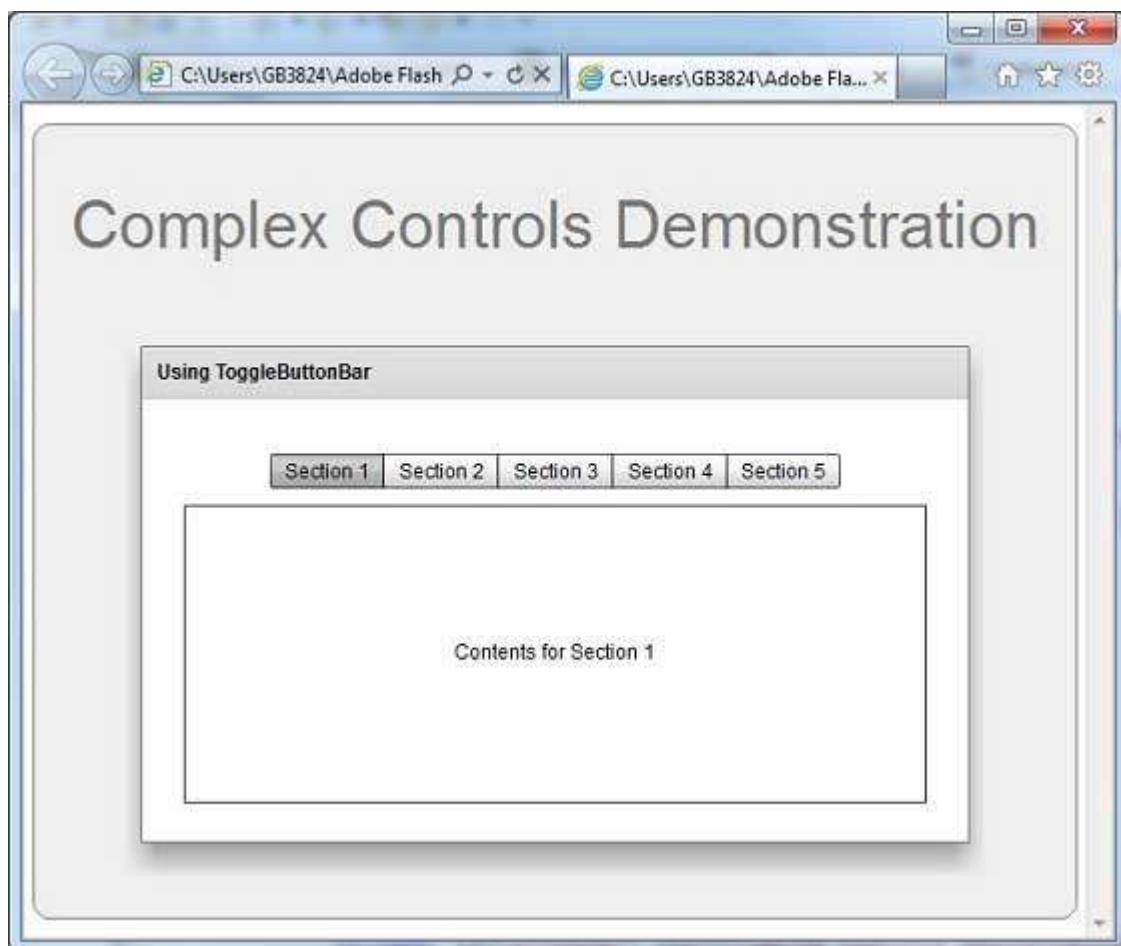
```

        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Complex Controls Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="toggleButtonBarPanel" title="Using ToggleButtonBar"
            width="500" height="300" >
            <s:layout>
                <s:VerticalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center" paddingTop="10"/>
            </s:layout>
            <mx:ToggleButtonBar
                itemClick="sectionClickHandler(event);"
               dataProvider="{sectionList}" />
            <s:BorderContainer width="90%" height="70%"
                borderColor="0x323232">
                <s:VGroup verticalAlign="middle"
                    horizontalAlign="center" width="100%" height="100%">
                    <s:Label id="sectionContents"
                        text="Contents for Section 1"/>
                </s:VGroup>
            </s:BorderContainer>

        </s:Panel>
    </s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



# 13. Flex – Layout Panels

Layout panel controls provides users to organize UI controls on the page. Every Layout control inherits properties from UIComponent class which in turn inherits properties from EventDispatcher and other top level classes.

S.N.	Control & Description
1	<p>Flex EventDispatcher Class</p> <p>The EventDispatcher class is the base class for all classes that can dispatch events. The EventDispatcher class allows any object on the display list to be an event target and as such, to use the methods of the IEventDispatcher interface.</p>
2	<p>Flex UIComponent</p> <p>The UIComponent class is the base class for all visual components, both interactive and non-interactive.</p>

## Flex – Event Dispatcher Class

---

### Introduction

- The **EventDispatcher** class is the base class for all classes that dispatch events.
- The **EventDispatcher** class implements the **IEventDispatcher** interface.
- The **EventDispatcher** class allows any object on the display list to be an event target and as such, to use the methods of the IEventDispatcher interface.

In order to understand **EventDispatcher**, let us first look what are event and event targets.

### What is an Event?

**Event** is a notification when a particular action is performed. For example, when a button is clicked then Click Event occurs.

### What is an Event Target

The **Event target** serves as the focal point for how events flow through the display list hierarchy.

When an event occurs, Flash Player dispatches an event object into the event flow from the root of the display list. The event object then makes its way through the display list until it reaches the event target, at which point it begins its return trip through the display list.

This round-trip journey to the event target is divided into three phases:

S.N.	Phase & Description
1	<p>capture</p> <p>This phase comprises the journey from the root to the last node before the event target's node</p>
2	<p>target</p> <p>This phase comprises only the event target node.</p>
3	<p>bubbling</p> <p>This phase comprises any subsequent nodes encountered on the return trip to the root of the display list.</p>

In general, any class which extends EventDispatcher gets the event dispatching capability.

## Class Declaration

Following is the declaration for **flash.events.EventDispatcher** class:

```
public class EventDispatcher
    extends java.lang.Object
    implements IEventDispatcher
```

## Public Methods

S.N.	Method & Description
1	<p><code>EventDispatcher(target:IEventDispatcher = null)</code></p> <p>Aggregates an instance of the EventDispatcher class.</p>
2	<p><code>addEventListener(type:String, listener:Function, useCapture:Boolean = false, priority:int = 0, useWeakReference:Boolean = false):void</code></p> <p>Registers an event listener object with an EventDispatcher object so that the listener receives notification of an event.</p>
3	<p><code>dispatchEvent(event:Event):Boolean</code></p> <p>Dispatches an event into the event flow.</p>

4	<code>hasEventListener(type:String):Boolean</code> Checks whether the EventDispatcher object has any listeners registered for a specific type of event.
5	<code>removeEventListener(type:String, listener:Function, useCapture:Boolean = false):void</code> Removes a listener from the EventDispatcher object.
6	<code>willTrigger(type:String):Boolean</code> Checks whether an event listener is registered with this EventDispatcher object or any of its ancestors for the specified event type.

## Events

Following are the events for **flash.events.EventDispatcher** class:

S.N.	Event & Description
1	<code>activate</code> Dispatched when the Flash Player gains operating system focus and becomes active.
2	<code>detivate</code> Dispatched when the Flash Player loses operating system focus and becomes inactive.

## Methods Inherited

This class inherits methods from the following class:

- Object

## Flex - UIComponent Class

The **UIComponent** class is the base class for all visual components, both interactive and noninteractive.

### Class Declaration

Following is the declaration for **mx.core.UIComponent** class:

```
public class UIComponent
```

```

extends FlexSprite

implements IAutomationObject, IChildList, IConstraintClient,
IDeferredInstantiationUIComponent, IFlexDisplayObject,
IFlexModule, IInvalidating, ILayoutManagerClient,
IPropertyChangeNotifier, IRepeaterClient, IStateClient,
IAdvancedStyleClient, IToolTipManagerClient,
IUIComponent, IValidatorListener, IVisualElement

```

## Public Properties

Following are the Public Properties for **mx.core.UIComponent** class:

S.N.	Name & Description
1	<b>accessibilityDescription</b> : String A convenience accessor for the description property in this UIComponent's accessibilityProperties object.
2	<b>accessibilityEnabled</b> : Boolean A convenience accessor for the silent property in this UIComponent's accessibilityProperties object.
3	<b>accessibilityName</b> : String A convenience accessor for the name property in this UIComponent's accessibilityProperties object.
4	<b>accessibilityShortcut</b> : String A convenience accessor for the shortcut property in this UIComponent's accessibilityProperties object.
5	<b>activeEffects</b> : Array [read-only] The list of effects that are currently playing on the component, as an Array of EffectInstance instances.
6	<b>automationDelegate</b> : Object The delegate object that handles the automation-related functionality.
7	<b>automationEnabled</b> : Boolean [read-only] True if this component is enabled for automation, false otherwise.

8	automationName : String  Name that can be used as an identifier for this object.
9	automationOwner : DisplayObjectContainer  [read-only] The owner of this component for automation purposes.
10	automationParent : DisplayObjectContainer  [read-only] The parent of this component for automation purposes.
11	automationTabularData : Object  [read-only] An implementation of the IAutomationTabularData interface, which can be used to retrieve the data.
12	automationValue : Array  [read-only] This value generally corresponds to the rendered appearance of the object and should be usable for correlating the identifier with the object as it appears visually within the application.
13	automationVisible : Boolean  [read-only] True if this component is visible for automation, false otherwise.
14	baseline : Object  For components, this layout constraint property is a facade on top of the similarly-named style.
15	baselinePosition : Number  [read-only] The y-coordinate of the baseline of the first line of text of the component.
16	bottom : Object  For components, this layout constraint property is a facade on top of the similarly-named style.
17	cacheHeuristic : Boolean  [write-only] Used by Flex to suggest bitmap caching for the object.
18	cachePolicy : String  Specifies the bitmap caching policy for this object.

19	<b>className</b> : String [read-only] The name of this instance's class, such as "Button".
20	<b>contentMouseX</b> : Number [read-only] Returns the x position of the mouse, in the content coordinate system.
21	<b>contentMouseY</b> : Number [read-only] Returns the y position of the mouse, in the content coordinate system.
22	<b>currentState</b> : String The current view state of the component.
23	<b>cursorManager</b> : ICursorManager [read-only] Gets the CursorManager that controls the cursor for this component and its peers.
24	<b>depth</b> : Number Determines the order in which items inside of containers are rendered.
25	<b>descriptor</b> : UIComponentDescriptor Reference to the UIComponentDescriptor, if any, that was used by the createComponentFromDescriptor() method to create this UIComponent instance.
26	<b>designLayer</b> : DesignLayer Specifies the optional DesignLayer instance associated with this visual element.
27	<b>document</b> : Object A reference to the document object associated with this UIComponent.
28	<b>doubleClickEnabled</b> : Boolean [override] Specifies whether the UIComponent object receives doubleClick events.
29	<b>enabled</b> : Boolean Whether the component can accept user interaction.

30	<b>errorString : String</b>  The text that displayed by a component's error tip when a component is monitored by a Validator and validation fails.
31	<b>explicitHeight : Number</b>  Number that specifies the explicit height of the component, in pixels, in the component's coordinates.
32	<b>explicitMaxHeight : Number</b>  The maximum recommended height of the component to be considered by the parent during layout.
33	<b>explicitMaxWidth : Number</b>  The maximum recommended width of the component to be considered by the parent during layout.
34	<b>explicitMinHeight : Number</b>  The minimum recommended height of the component to be considered by the parent during layout.
35	<b>explicitMinWidth : Number</b>  The minimum recommended width of the component to be considered by the parent during layout.
36	<b>explicitWidth : Number</b>  Number that specifies the explicit width of the component, in pixels, in the component's coordinates.
37	<b>flexContextMenu : IFlexContextMenu</b>  The context menu for this UIComponent.
38	<b>focusEnabled : Boolean</b>  Indicates whether the component can receive focus when tabbed to.
39	<b>focusManager : IFocusManager</b>  Gets the FocusManager that controls focus for this component and its peers.
40	<b>focusPane : Sprite</b>

	The focus pane associated with this object.
41	<b>hasFocusableChildren</b> : Boolean A flag that indicates whether child objects can receive focus.
42	<b>hasLayoutMatrix3D</b> : Boolean [read-only] Contains true if the element has 3D Matrix.
43	<b>height</b> : Number [override] Number that specifies the height of the component, in pixels, in the parent's coordinates.
44	<b>horizontalCenter</b> : Object For components, this layout constraint property is a facade on top of the similarly-named style.
45	<b>id</b> : String ID of the component.
46	<b>includeInLayout</b> : Boolean Specifies whether this component is included in the layout of the parent container.
47	<b>inheritingStyles</b> : Object The beginning of this component's chain of inheriting styles.
48	<b>initialized</b> : Boolean A flag that determines if an object has been through all three phases of layout: commitment, measurement, and layout (provided that any were required).
49	<b>instanceIndex</b> : int [read-only] The index of a repeated component.
50	<b>instanceIndices</b> : Array An Array containing the indices required to reference this UIComponent object from its parent document.
51	<b>is3D</b> : Boolean [read-only] Contains true when the element is in 3D.

52	<b>isDocument : Boolean</b> [read-only] Contains true if this UIComponent instance is a document object.
53	<b>isPopUp : Boolean</b> Set to true by the PopUpManager to indicate that component has been popped up.
54	<b>layoutMatrix3D : Matrix3D</b> [write-only] The transform matrix that is used to calculate a component's layout relative to its siblings.
55	<b>left : Object</b> For components, this layout constraint property is a facade on top of the similarly-named style.
56	<b>maintainProjectionCenter : Boolean</b> When true, the component keeps its projection matrix centered on the middle of its bounding box.
57	<b>maxHeight : Number</b> The maximum recommended height of the component to be considered by the parent during layout.
58	<b>maxWidth : Number</b> The maximum recommended width of the component to be considered by the parent during layout.
59	<b>measuredHeight : Number</b> The default height of the component, in pixels.
60	<b>measuredMinHeight : Number</b> The default minimum height of the component, in pixels.
61	<b>measuredMinWidth : Number</b> The default minimum width of the component, in pixels.
62	<b>measuredWidth : Number</b> The default width of the component, in pixels.

63	<b>minHeight : Number</b> The minimum recommended height of the component to be considered by the parent during layout.
64	<b>minWidth : Number</b> The minimum recommended width of the component to be considered by the parent during layout.
65	<b>moduleFactory : IFlexModuleFactory</b> A module factory is used as context for using embedded fonts and for finding the style manager that controls the styles for this component.
66	<b>mouseFocusEnabled : Boolean</b> Whether you can receive focus when clicked on.
67	<b>nestLevel : int</b> Depth of this object in the containment hierarchy.
68	<b>nonInheritingStyles : Object</b> The beginning of this component's chain of non-inheriting styles.
69	<b>numAutomationChildren : int</b> [read-only] The number of automation children this container has.
70	<b>owner : DisplayObjectContainer</b> The owner of this IVisualElement object.
71	<b>parent : DisplayObjectContainer</b> [override] [read-only] The parent container or component for this component.
72	<b>parentApplication : Object</b> [read-only] A reference to the Application object that contains this UIComponent instance.
73	<b>parentDocument : Object</b> [read-only] A reference to the parent document object for this UIComponent.
74	<b>percentHeight : Number</b>

	<p>Specifies the height of a component as a percentage of its parent's size.</p>
75	<p><code>percentWidth : Number</code> Specifies the width of a component as a percentage of its parent's size.</p>
76	<p><code>postLayoutTransformOffsets : mx.geom:TransformOffsets</code> Defines a set of adjustments that can be applied to the object's transform in a way that is invisible to its parent's layout.</p>
77	<p><code>processedDescriptors : Boolean</code> Set to true after immediate or deferred child creation, depending on which one happens.</p>
78	<p><code>repeater : IRepeater</code> [read-only] A reference to the Repeater object in the parent document that produced this UIComponent.</p>
79	<p><code>repeaterIndex : int</code> [read-only] The index of the item in the data provider of the Repeater that produced this UIComponent.</p>
80	<p><code>repeaterIndices : Array</code> An Array containing the indices of the items in the data provider of the Repeaters in the parent document that produced this UIComponent.</p>
81	<p><code>repeaters : Array</code> An Array containing references to the Repeater objects in the parent document that produced this UIComponent.</p>
82	<p><code>right : Object</code> For components, this layout constraint property is a facade on top of the similarly-named style.</p>
83	<p><code>rotation : Number</code> [override] Indicates the rotation of the DisplayObject instance, in degrees, from its original orientation.</p>
84	<p><code>rotationX : Number</code></p>

	[override] Indicates the x-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
85	<b>rotationY</b> : Number [override] Indicates the y-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
86	<b>rotationZ</b> : Number [override] Indicates the z-axis rotation of the DisplayObject instance, in degrees, from its original orientation relative to the 3D parent container.
87	<b>scaleX</b> : Number [override] Number that specifies the horizontal scaling factor.
88	<b>scaleY</b> : Number [override] Number that specifies the vertical scaling factor.
89	<b>scaleZ</b> : Number [override] Number that specifies the scaling factor along the z axis.
90	<b>screen</b> : Rectangle [read-only] Returns an object that contains the size and position of the base drawing surface for this object.
91	<b>showInAutomationHierarchy</b> : Boolean A flag that determines if an automation object shows in the automation hierarchy.
92	<b>states</b> : Array The view states that are defined for this component.
93	<b>styleDeclaration</b> : CSSStyleDeclaration Storage for the inline inheriting styles on this object.
94	<b>styleManager</b> : IStyleManager2 [read-only] Returns the StyleManager instance used by this component.
95	<b>styleName</b> : Object The class style used by this component.

96	<b>styleParent : IAdvancedStyleClient</b> A component's parent is used to evaluate descendant selectors.
97	<b>systemManager : ISystemManager</b> Returns the SystemManager object used by this component.
98	<b>tabFocusEnabled : Boolean</b> A flag that indicates whether this object can receive focus via the TAB key. This is similar to the tabEnabled property used by the Flash Player. This is usually true for components that handle keyboard input, but some components in controlbars have them set to false because they should not steal focus from another component like an editor.
99	<b>toolTip : String</b> Text to display in the ToolTip.
100	<b>top : Object</b> For components, this layout constraint property is a facade on top of the similarly-named style.
101	<b>transform : flash.geom:Transform</b> [override] An object with properties pertaining to a display object's matrix, color transform, and pixel bounds.
102	<b>transformX : Number</b> Sets the x coordinate for the transform center of the component.
103	<b>transformY : Number</b> Sets the y coordinate for the transform center of the component.
104	<b>transformZ : Number</b> Sets the z coordinate for the transform center of the component.
105	<b>transitions : Array</b> An Array of Transition objects, where each Transition object defines a set of effects to play when a view state change occurs.
106	<b>tweeningProperties : Array</b>

	Array of properties that are currently being tweened on this object.
107	<p>uid : String</p> <p>A unique identifier for the object.</p>
108	<p>updateCompletePendingFlag : Boolean</p> <p>A flag that determines if an object has been through all three phases of layout validation (provided that any were required).</p>
109	<p>validationSubField : String</p> <p>Used by a validator to associate a subfield with this component.</p>
110	<p>verticalCenter : Object</p> <p>For components, this layout constraint property is a facade on top of the similarly-named style.</p>
111	<p>visible : Boolean</p> <p>[override] Whether or not the display object is visible.</p>
112	<p>width : Number</p> <p>[override] Number that specifies the width of the component, in pixels, in the parent's coordinates.</p>
113	<p>x : Number</p> <p>[override] Number that specifies the component's horizontal position, in pixels, within its parent container.</p>
114	<p>y : Number</p> <p>[override] Number that specifies the component's vertical position, in pixels, within its parent container.</p>
115	<p>z : Number</p> <p>[override] Indicates the z coordinate position along the z-axis of the DisplayObject instance relative to the 3D parent container.</p>

## Protected Properties

Following are the Protected Properties for **mx.core.UIComponent** class:

S.N.	Name & Description
1	currentCSSState : String [read-only] The state to be used when matching CSS pseudo-selectors.
2	hasComplexLayoutMatrix : Boolean [read-only] Returns true if the UIComponent has any non-translation (x,y) transform properties.
3	resourceManager : IResourceManager [read-only] A reference to the object which manages all of the application's localized resources.
4	unscaledHeight : Number [read-only] A convenience method for determining the unscaled height of the component.
5	unscaledWidth : Number [read-only] A convenience method for determining the unscaled width of the component All of a component's drawing and child layout should be done within a bounding rectangle of this width, which is also passed as an argument to updateDisplayList().

S.N.	Event & Description
1	<b>activate</b> Dispatched when the Flash Player gains operating system focus and becomes active.
2	<b>detivate</b> Dispatched when the Flash Player loses operating system focus and becomes inactive.

## Public Methods

S.N.	Method & Description
1	UIComponent() Constructor.
2	addStyleClient(styleClient:IAdvancedStyleClient):void Adds a non-visual style client to this component instance.
3	callLater(method:Function, args:Array = null):void Queues a function to be called later.
4	clearStyle(styleProp:String):void Deletes a style property from this component instance.
5	contentToGlobal(point:Point):Point Converts a Point object from content coordinates to global coordinates.
6	contentToLocal(point:Point):Point Converts a Point object from content to local coordinates.
7	createAutomationIDPart(child:IAutomationObject):Object Returns a set of properties that identify the child within this container.
8	createAutomationIDPartWithRequiredProperties(child:IAutomationObject, properties:Array):Object Returns a set of properties that identify the child within this container.
9	createReferenceOnParentDocument(parentDocument:IFlexDisplayObject):void Creates an id reference to this IUIComponent object on its parent document object.
10	deleteReferenceOnParentDocument(parentDocument:IFlexDisplayObject):void Deletes the id reference to this IUIComponent object on its parent document object.
11	determineTextFormatFromStyles():mx.core:UITextFormat

	Returns a <code>UITextFormat</code> object corresponding to the text styles for this <code>UIComponent</code> .
12	<code>dispatchEvent(event:Event):Boolean</code> [override] Dispatches an event into the event flow.
13	<code>drawFocus(isFocused:Boolean):void</code> Shows or hides the focus indicator around this component.
14	<code>drawRoundRect(x:Number, y:Number, w:Number, h:Number, r:Object = null, c:Object = null, alpha:Object = null, rot:Object = null, gradient:String = null, ratios:Array = null, hole:Object = null):void</code> Programmatically draws a rectangle into this skin's <code>Graphics</code> object.
15	<code>effectFinished(effectInst:IEffectInstance):void</code> Called by the effect instance when it stops playing on the component.
16	<code>effectStarted(effectInst:IEffectInstance):void</code> Called by the effect instance when it starts playing on the component.
17	<code>endEffectsStarted():void</code> Ends all currently playing effects on the component.
18	<code>executeBindings(recurse:Boolean = false):void</code> Executes all the bindings for which the <code>UIComponent</code> object is the destination.
19	<code>finishPrint(obj:Object, target:IFlexDisplayObject):void</code> Called after printing is complete.
20	<code>getAutomationChildAt(index:int):IAutomationObject</code> Provides the automation object at the specified index.
21	<code>getAutomationChildren():Array</code> Provides the automation object list .
22	<code>getBoundsXAtSize(width:Number, postLayoutTransform:Boolean = true):Number</code> height:Number, Returns the x coordinate of the element's bounds at the specified element size.

23	<code>getBoundsYAtSize(width:Number, postLayoutTransform:Boolean = true):Number</code>	height:Number,
	Returns the y coordinate of the element's bounds at the specified element size.	
24	<code>getClassStyleDeclarations():Array</code>	Finds the type selectors for this UIComponent instance.
25	<code>getConstraintValue(constraintName:String):*</code>	Returns a layout constraint value, which is the same as getting the constraint style for this component.
26	<code>getExplicitOrMeasuredHeight():Number</code>	A convenience method for determining whether to use the explicit or measured height
27	<code>getExplicitOrMeasuredWidth():Number</code>	A convenience method for determining whether to use the explicit or measured width
28	<code>getFocus():InteractiveObject</code>	Gets the object that currently has focus.
29	<code>getLayoutBoundsHeight(postLayoutTransform:Boolean = true):Number</code>	Returns the element's layout height.
30	<code>getLayoutBoundsWidth(postLayoutTransform:Boolean = true):Number</code>	Returns the element's layout width.
31	<code>getLayoutBoundsX(postLayoutTransform:Boolean = true):Number</code>	Returns the x coordinate that the element uses to draw on screen.
32	<code>getLayoutBoundsY(postLayoutTransform:Boolean = true):Number</code>	Returns the y coordinate that the element uses to draw on screen.
33	<code>getLayoutMatrix():Matrix</code>	Returns the transform matrix that is used to calculate the component's layout relative to its siblings.

34	<code>getLayoutMatrix3D():Matrix3D</code> Returns the layout transform Matrix3D for this element.
35	<code>getMaxBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's maximum height.
36	<code>getMaxBoundsWidth(postLayoutTransform:Boolean = true):Number</code> Returns the element's maximum width.
37	<code>getMinBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's minimum height.
38	<code>getMinBoundsWidth(postLayoutTransform:Boolean = true):Number</code> Returns the element's minimum width.
39	<code>getPreferredBoundsHeight(postLayoutTransform:Boolean = true):Number</code> Returns the element's preferred height.
40	<code>getPreferredBoundsWidth(postLayoutTransform:Boolean = true):Number</code> Returns the element's preferred width.
41	<code>getRepeaterItem(whichRepeater:int = -1):Object</code> Returns the item in the dataProvider that was used by the specified Repeater to produce this Repeater, or null if this Repeater isn't repeated.
42	<code>getStyle(styleProp:String):*</code> Gets a style property that has been set anywhere in this component's style lookup chain.
43	<code>globalToContent(point:Point):Point</code> Converts a Point object from global to content coordinates.
45	<code>hasCSSState():Boolean</code> Returns true if currentCSSState is not null.
46	<code>hasState(stateName:String):Boolean</code> Determines whether the specified state has been defined on this UIComponent.

47	<code>horizontalGradientMatrix(x:Number, y:Number, width:Number, height:Number):Matrix</code> Returns a box Matrix which can be passed to the drawRoundRect() method as the rot parameter when drawing a horizontal gradient.
48	<code>initialize():void</code> Initializes the internal structure of this component.
49	<code>initializeRepeaterArrays(parent:IRepeaterClient):void</code> Initializes various properties which keep track of repeated instances of this component.
50	<code>invalidateDisplayList():void</code> Marks a component so that its updateDisplayList() method gets called during a later screen update.
51	<code>invalidateLayering():void</code> Called by a component's items to indicate that their depth property has changed.
52	<code>invalidateLayoutDirection():void</code> An element must call this method when its layoutDirection changes or when its parent's layoutDirection changes.
53	<code>invalidateProperties():void</code> Marks a component so that its commitProperties() method gets called during a later screen update.
54	<code>invalidateSize():void</code> Marks a component so that its measure() method gets called during a later screen update.
55	<code>localToContent(point:Point):Point</code> Converts a Point object from local to content coordinates.
56	<code>matchesCSSState(cssState:String):Boolean</code> Returns true if cssState matches currentCSSState.
57	<code>matchesCSSType(cssType:String):Boolean</code>

	Determines whether this instance is the same as, or is a subclass of, the given type.
58	<p><code>measureHTMLText(htmlText:String):flash.text:TextLineMetrics</code></p> <p>Measures the specified HTML text, which can contain HTML tags such as &amp;lt;font&amp;ampgt and &amp;&lt;b&amp;ampgt, assuming that it is displayed in a single-line UITextField using a UITextFormat determined by the styles of this UIComponent.</p>
59	<p><code>measureText(text:String):flash.text:TextLineMetrics</code></p> <p>Measures the specified text, assuming that it is displayed in a single-line UITextField (or UITETextField) using a UITextFormat determined by the styles of this UIComponent.</p>
60	<p><code>move(x:Number, y:Number):void</code></p> <p>Moves the component to a specified position within its parent.</p>
61	<p><code>notifyStyleChangeInChildren(styleProp:String, recursive:Boolean):void</code></p> <p>Propagates style changes to the children.</p>
62	<p><code>owns(child:DisplayObject):Boolean</code></p> <p>Returns true if the chain of owner properties points from child to this UIComponent.</p>
63	<p><code>parentChanged(p:DisplayObjectContainer):void</code></p> <p>Called by Flex when a UIComponent object is added to or removed from a parent.</p>
64	<p><code>prepareToPrint(target:IFlexDisplayObject):Object</code></p> <p>Prepares an IFlexDisplayObject for printing.</p>
65	<p><code>regenerateStyleCache(recursive:Boolean):void</code></p> <p>Builds or rebuilds the CSS style cache for this component and, if the recursive parameter is true, for all descendants of this component as well.</p>
66	<p><code>registerEffects(effects:Array):void</code></p> <p>For each effect event, registers the EffectManager as one of the event listeners.</p>
67	<p><code>removeStyleClient(styleClient:IAdvancedStyleClient):void</code></p> <p>Removes a non-visual style client from this component instance.</p>

68	<code>replayAutomatableEvent(event:Event):Boolean</code> Replays the specified event.
69	<code>resolveAutomationIDPart(criteria:Object):Array</code> Resolves a child by using the id provided.
70	<code>resumeBackgroundProcessing():void</code> [static] Resumes the background processing of methods queued by <code>callLater()</code> , after a call to <code>suspendBackgroundProcessing()</code> .
71	<code>setActualSize(w:Number, h:Number):void</code> Sizes the object.
72	<code>setConstraintValue(constraintName:String, value:*):void</code> Sets a layout constraint value, which is the same as setting the constraint style for this component.
73	<code>setCurrentState(stateName:String, playTransition:Boolean = true):void</code> Set the current state.
74	<code>setFocus():void</code> Sets the focus to this component.
75	<code>setLayoutBoundsPosition(x:Number, y:Number, postLayoutTransform:Boolean = true):void</code> Sets the coordinates that the element uses to draw on screen.
76	<code>setLayoutBoundsSize(width:Number, height:Number, postLayoutTransform:Boolean = true):void</code> Sets the layout size of the element.
77	<code>setLayoutMatrix(value:Matrix, invalidateLayout:Boolean):void</code> Sets the transform Matrix that is used to calculate the component's layout size and position relative to its siblings.
78	<code>setLayoutMatrix3D(value:Matrix3D, invalidateLayout:Boolean):void</code> Sets the transform Matrix3D that is used to calculate the component's layout size and position relative to its siblings.

79	<code>setStyle(styleProp:String, newValue:*):void</code> Sets a style property on this component instance.
80	<code>setVisible(value:Boolean, noEvent:Boolean = false):void</code> Called when the visible property changes.
81	<code>styleChanged(styleProp:String):void</code> Detects changes to style properties.
82	<code>stylesInitialized():void</code> Flex calls the stylesInitialized() method when the styles for a component are first initialized.
83	<code>suspendBackgroundProcessing():void</code> [static] Blocks the background processing of methods queued by callLater(), until resumeBackgroundProcessing() is called.
84	<code>transformAround(transformCenter:Vector3D, scale:Vector3D = null, rotation:Vector3D = null, translation:Vector3D = null, postLayoutScale:Vector3D = null, postLayoutRotation:Vector3D = null, postLayoutTranslation:Vector3D = null, invalidateLayout:Boolean = true):void</code> A utility method to update the rotation, scale, and translation of the transform while keeping a particular point, specified in the component's own coordinate space, fixed in the parent's coordinate space.
85	<code>transform Point To Parent (localPosition:Vector3D, position:Vector3D, post Layout Position:Vector3D):void</code> A utility method to transform a point specified in the local coordinates of this object to its location in the object's parent's coordinates.
86	<code>validateDisplayList():void</code> Validates the position and size of children and draws other visuals.
87	<code>validateNow():void</code> Validate and update the properties and layout of this object and redraw it, if necessary.
88	<code>validateProperties():void</code>

	Used by layout logic to validate the properties of a component by calling the commitProperties() method.
89	<b>validateSize(recursive:Boolean = false):void</b> Validates the measured size of the component If the LayoutManager.invalidateSize() method is called with this ILayoutManagerClient, then the validateSize() method is called when it's time to do measurements.
90	<b>validationResultHandler(event:ValidationResultEvent):void</b> Handles both the valid and invalid events from a validator assigned to this component.
91	<b>verticalGradientMatrix(x:Number, y:Number, width:Number, height:Number):Matrix</b> Returns a box Matrix which can be passed to drawRoundRect() as the rot parameter when drawing a vertical gradient.

## Protected Methods

S.N.	Method & Description
1	<b>adjustFocusRect(obj:DisplayObject = null):void</b> Adjust the focus rectangle.
2	<b>applyComputedMatrix():void</b> Commits the computed matrix built from the combination of the layout matrix and the transform offsets to the flash displayObject's transform.
3	<b>attachOverlay():void</b> This is an internal method used by the Flex framework to support the Dissolve effect.
4	<b>canSkipMeasurement():Boolean</b> Determines if the call to the measure() method can be skipped.
5	<b>childrenCreated():void</b> Performs any final processing after child objects are created.
6	<b>commitProperties():void</b>

	Processes the properties set on the component.
7	<b>createChildren():void</b> Create child objects of the component.
8	<b>createInFontContext(classObj:Class):Object</b> Creates a new object using a context based on the embedded font being used.
9	<b>createInModuleContext(moduleFactory:IFlexModuleFactory, className:String):Object</b> Creates the object using a given moduleFactory.
10	<b>dispatchPropertyChangeEvent(prop:String, oldValue:*, value:*):void</b> Helper method for dispatching a PropertyChangeEvent when a property is updated.
11	<b>focusInHandler(event:FocusEvent):void</b> The event handler called when a UIComponent object gets focus.
12	<b>focusOutHandler(event:FocusEvent):void</b> The event handler called when a UIComponent object loses focus.
13	<b>initAdvancedLayoutFeatures():void</b> Initializes the implementation and storage of some of the less frequently used advanced layout features of a component.
14	<b>initializationComplete():void</b> Finalizes the initialization of this component.
15	<b>initializeAccessibility():void</b> Initializes this component's accessibility code.
16	<b>invalidateParentSizeAndDisplayList():void</b> Helper method to invalidate parent size and display list if this object affects its layout (includeInLayout is true).
17	<b>isOurFocus(target:DisplayObject):Boolean</b>

	Typically overridden by components containing UITextField objects, where the UITextField object gets focus.
18	<code>keyDownHandler(event:KeyboardEvent):void</code> The event handler called for a keyDown event.
19	<code>keyUpHandler(event:KeyboardEvent):void</code> The event handler called for a keyUp event.
20	<code>measure():void</code> Calculates the default size, and optionally the default minimum size, of the component.
21	<code>resourcesChanged():void</code> This method is called when a UIComponent is constructed, and again whenever the ResourceManager dispatches a "change" Event to indicate that the localized resources have changed in some way.
22	<code>setStretchXY(stretchX:Number, stretchY:Number):void</code> Specifies a transform stretch factor in the horizontal and vertical direction.
23	<code>stateChanged(oldState:String, newState:String, recursive:Boolean):void</code> This method is called when a state changes to check whether state-specific styles apply to this component
24	<code>updateDisplayList(unscaledWidth:Number, unscaledHeight:Number):void</code> Draws the object and/or sizes and positions its children.

## Events

Following are the events for **mx.core.UIComponent** class:

S.N.	Event & Description
1	<code>add</code> when the component is added to a container as a content child by using the <code>addChild()</code> , <code>addChildAt()</code> , <code>addElement()</code> , or <code>addElementAt()</code> method.

2	<b>creationComplete</b> when the component has finished its construction, property processing, measuring, layout, and drawing.
3	<b>currentStateChange</b> after the view state has changed.
4	<b>currentStateChanging</b> after the currentState property changes, but before the view state changes.
5	<b>dragComplete</b> by the drag initiator (the component that is the source of the data being dragged) when the drag operation completes, either when you drop the dragged data onto a drop target or when you end the drag-and-drop operation without performing a drop.
6	<b>dragDrop</b> by the drop target when the user releases the mouse over it.
7	<b>dragEnter</b> by a component when the user moves the mouse over the component during a drag operation.
8	<b>dragExit</b> by the component when the user drags outside the component, but does not drop the data onto the target.
9	<b>dragOver</b> by a component when the user moves the mouse while over the component during a drag operation.
10	<b>dragStart</b> by the drag initiator when starting a drag operation.
11	<b>effectEnd</b> after an effect ends.
12	<b>effectStart</b>

	just before an effect starts.
13	<b>effectStop</b> after an effect is stopped, which happens only by a call to stop() on the effect.
14	<b>enterState</b> after the component has entered a view state.
15	<b>exitState</b> just before the component exits a view state.
16	<b>hide</b> when an object's state changes from visible to invisible.
17	<b>initialize</b> when the component has finished its construction and has all initialization properties set.
18	<b>invalid</b> when a component is monitored by a Validator and the validation failed.
19	<b>mouseDownOutside</b> from a component opened using the PopUpManager when the user clicks outside it.
20	<b>mouseWheelOutside</b> from a component opened using the PopUpManager when the user scrolls the mouse wheel outside it.
21	<b>move</b> when the object has moved.
22	<b>preinitialize</b> at the beginning of the component initialization sequence.
23	<b>remove</b>

	when the component is removed from a container as a content child by using the removeChild(), removeChildAt(), removeElement(), or removeElementAt() method.
24	resize when the component is resized.
25	show when an object's state changes from invisible to visible.
26	stateChangeComplete after the component has entered a new state and any state transition animation to that state has finished playing.
27	stateChangeInterrupted when a component interrupts a transition to its current state in order to switch to a new state.
28	toolTipCreate by the component when it is time to create a ToolTip.
29	toolTipEnd by the component when its ToolTip has been hidden and is to be discarded soon.
30	toolTipHide by the component when its ToolTip is about to be hidden.
31	toolTipShow by the component when its ToolTip is about to be shown.
32	toolTipShown by the component when its ToolTip has been shown.
33	toolTipStart by a component whose toolTip property is set, as soon as the user moves the mouse over it.
34	touchInteractionEnd

	A non-cancellable event, by a component when it is done responding to a touch interaction user gesture.
35	<b>touchInteractionStart</b> A non-cancellable event, by a component when it starts responding to a touch interaction user gesture.
36	<b>touchInteractionStarting</b> A cancellable event, by a component in an attempt to respond to a touch interaction user gesture.
37	<b>updateComplete</b> when an object has had its commitProperties(), measure(), and updateDisplayList() methods called (if needed).
38	<b>valid</b> when a component is monitored by a Validator and the validation succeeded.
39	<b>valueCommit</b> when values are changed programmatically or by user interaction.

## Methods Inherited

This class inherits methods from the following classes:

- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Layout Panels

---

Following are few important Layout Panels:

S.N.	Panel & Description
1	<b>BorderContainer</b> The BorderContainer class provides a set of CSS styles that control the appearance of the border and background fill of the container.
2	<b>Form</b>

	The Form container provides control over the layout of a form, mark form fields as required or optional, handle error messages, and bind form data to the Flex data model to perform data checking and validation.
3	VGroup The VGroup container is a Group container that uses the VerticalLayout class.
4	HGroup The HGroup container is a Group container that uses the HorizontalLayout class.
5	Panel The Panel class is a container that includes a title bar, a caption, a border, and a content area for its children.
6	SkinnableContainer The SkinnableContainer class is the base class for skinnable containers that provide visual content.
7	TabBar The TabBar displays a set of identical tabs.
8	TitleWindow The TitleWindow extends Panel to include a close button and move area.

## Flex – BorderContainer

The BorderContainer class defines a set of CSS styles that control the appearance of the border and background fill of the container.

### Class Declaration

Following is the declaration for **spark.components.BorderContainer** class:

```
public class BorderContainer
    extends SkinnableContainer
```

### Public Properties

S.N.	Property & Description
1	backgroundFill : IFill Defines the background of the BorderContainer.
2	borderStroke : IStroke

	Defines the stroke of the BorderContainer container.
--	--

## Public Methods

S.N.	Method & Description
1	BorderContainer() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.SkinnableContainer
- spark.components.supportClasses.SkinnableContainerBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex BorderContainer Example

Let us follow the following steps to check usage of BorderContainer in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```

<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            private function applyBorderContainerProperties():void
            {

borderContainer.setStyle("borderWeight",weightStepper.value);

borderContainer.setStyle("borderColor",colorPicker.selectedColor);

borderContainer.setStyle("dropShadowVisible",chkShadow.selected);
        }
    ]]>
</fx:Script>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Layout Panels Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="borderContainerPanel"
            title="Using BorderContainer" width="500"
            height="300" includeInLayout="true" visible="true">
            <s:layout>
                <s:VerticalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center"/>

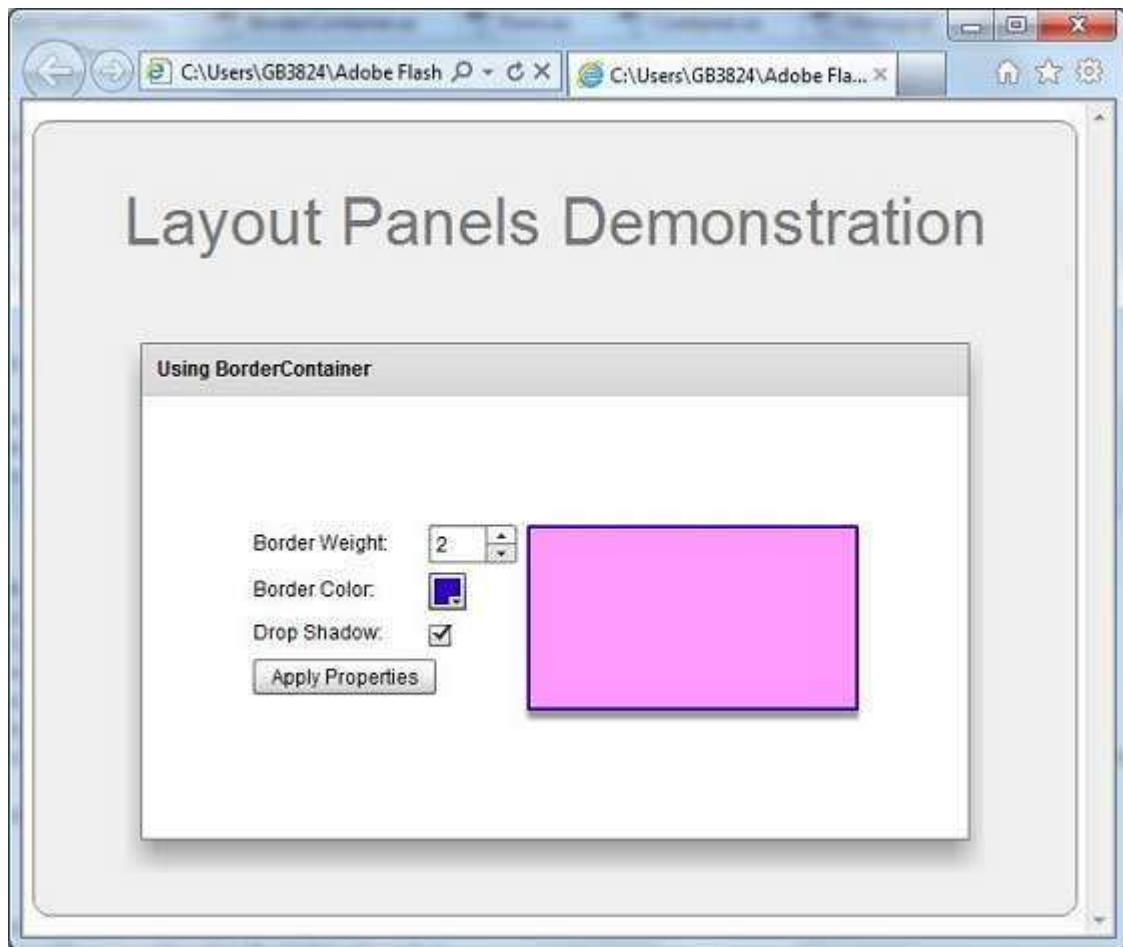
            </s:layout>
        
```

```

<s:HGroup>
<s:VGroup>
    <s:HGroup verticalAlign="middle" >
        <s:Label text="Border Weight:" width="100"/>
        <s:NumericStepper id="weightStepper" value="1"/>
    </s:HGroup>
    <s:HGroup verticalAlign="middle">
        <s:Label text="Border Color:" width="100"/>
        <mx:ColorPicker id="colorPicker" color="0x323232"/>
    </s:HGroup>
    <s:HGroup verticalAlign="middle">
        <s:Label text="Drop Shadow:" width="100"/>
        <s:CheckBox id="chkShadow" selected="false"/>
    </s:HGroup>
    <s:Button label="Apply Properties"
        click="applyBorderContainerProperties()"/>
</s:VGroup>
<s:BorderContainer id="borderContainer" width="200"
    backgroundColor="0xff99ff">
</s:BorderContainer>
</s:HGroup>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – Form

The Form container lets you control the layout of a form, mark form fields as required or optional, handle error messages, and bind your form data to the Flex data model to perform data checking and validation. It also lets you use style sheets to configure the appearance of your forms.

### Class Declaration

Following is the declaration for **spark.components.Form** class:

```
public class Form
    extends Container
```

### Public Properties

S.N.	Property & Description
------	------------------------

1	invalidElements : Array [read-only] A sorted Array of descendant elements that are in an invalid state.
---	--

## Public Methods

S.N.	Method & Description
1	Form() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- mx.core.Component
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex Form Example

Let us follow the following steps to check usage of Form in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

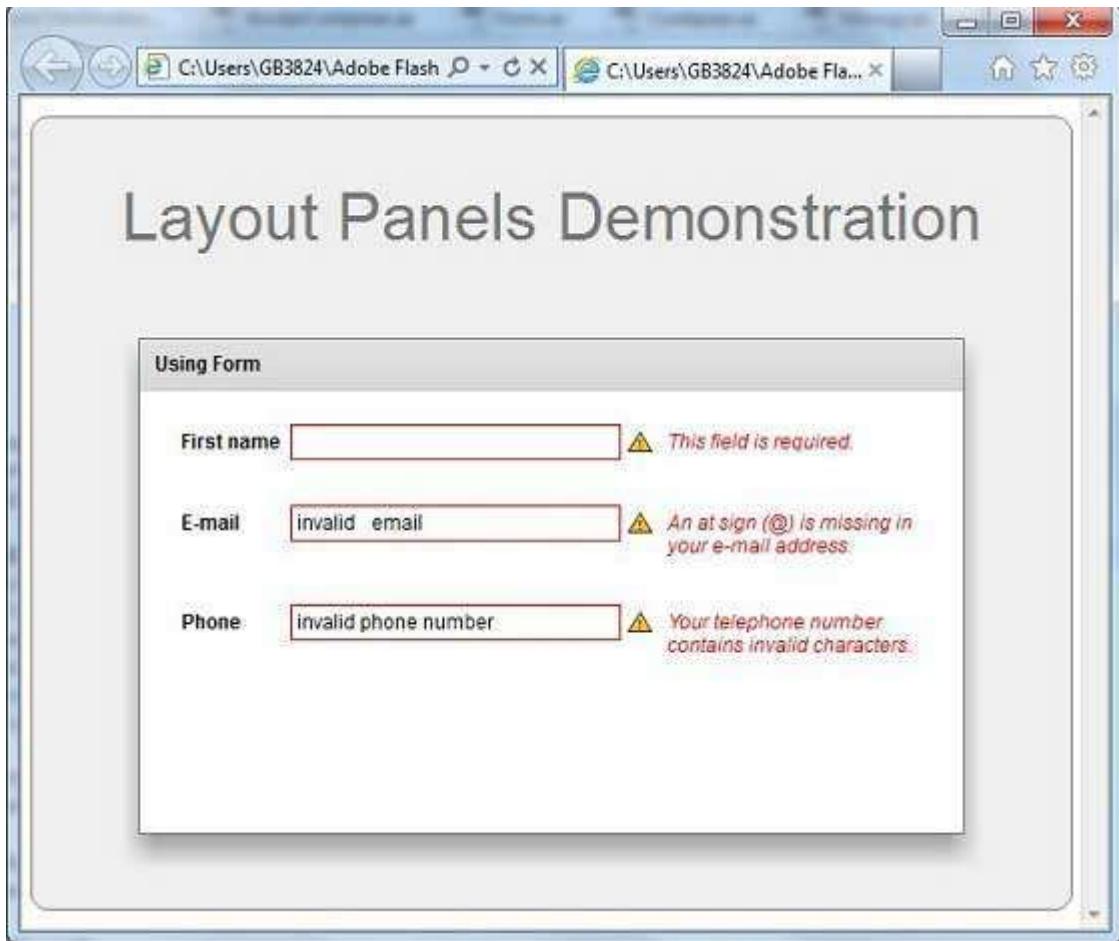
```

<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
>
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<fx:Declarations>
    <mx:StringValidator source="{fname}" property="text"
        minLength="4" maxLength="12"/>
    <mx:PhoneNumberValidator source="{phone}" property="text"/>
    <mx:EmailValidator source="{email}" property="text"/>
</fx:Declarations>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Layout Panels Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="formPanel" title="Using Form"
            width="500" height="300"
            includeInLayout="true" visible="true">
            <s:Form >
                <s:FormItem label="First name">
                    <s:TextInput id="fname" width="200"/>
                </s:FormItem>
                <s:FormItem label="E-mail">
                    <s:TextInput id="email" width="200"/>
                </s:FormItem>
                <s:FormItem label="Phone">
                    <s:TextInput id="phone" width="200"/>
                </s:FormItem>
            </s:Form>
        </s:Panel>
    </s:VGroup>
</s:BorderContainer>

```

```
</s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – VGroup

---

### Introduction

The VGroup container is a Group container that uses the VerticalLayout class. Use the properties of the VGroup class to modify the characteristics of the VerticalLayout class.

### Class Declaration

Following is the declaration for **spark.components.VGroup** class:

```
public class VGroup
```

extends Group
---------------

## Public Properties

S.N.	Property & Description
1	<b>firstIndexInView : int</b> [read-only] The index of the first layout element that is part of the layout and within the layout target's scroll rectangle, or -1 if nothing has been displayed yet.
2	<b>gap : int</b> The vertical space between layout elements, in pixels.
3	<b>horizontalAlign : String</b> The horizontal alignment of layout elements.
4	<b>lastIndexInView : int</b> [read-only] The index of the last row that's part of the layout and within the container's scroll rectangle, or -1 if nothing has been displayed yet.
5	<b>paddingBottom : Number</b> Number of pixels between the container's bottom edge and the bottom edge of the last layout element.
6	<b>paddingLeft : Number</b> The minimum number of pixels between the container's left edge and the left edge of the layout element.
7	<b>paddingRight : Number</b> The minimum number of pixels between the container's right edge and the right edge of the layout element.
8	<b>paddingTop : Number</b> Number of pixels between the container's top edge and the top edge of the first layout element.
9	<b>requestedMaxRowCount : int</b>

	The measured height of this layout is large enough to display at most requestedMaxRowCount layout elements.
10	<b>requestedMinRowCount : int</b> The measured height of this layout is large enough to display at least requestedMinRowCount layout elements.
11	<b>requestedRowCount : int</b> The measured size of this layout is tall enough to display the first requestedRowCount layout elements.
12	<b>rowCount : int</b> [read-only] The current number of visible elements.
13	<b>rowHeight : Number</b> If variableRowHeight is false, then this property specifies the actual height of each child, in pixels.
14	<b>variableRowHeight : Boolean</b> Specifies whether layout elements are allocated their preferred height.
15	<b>verticalAlign : String</b> The vertical alignment of the content relative to the container's height.

## Public Methods

S.N.	Method & Description
1	<b>VGroup()</b> Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.Group
- spark.components.supportClasses.GroupBase
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer

- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex VGroup Example

---

Let us follow the following steps to check usage of VGroup in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            private function applyVGroupProperties():void
            {
                mainVGroup.paddingTop = padTopV.value;
                mainVGroup.paddingLeft = paddingLeftV.value;
                mainVGroup.paddingRight = paddingRightV.value;
                mainVGroup.paddingBottom = paddingBottomV.value;
                mainVGroup.gap = gapV.value;
            }
        ]]>
    </fx:Script>

```

```

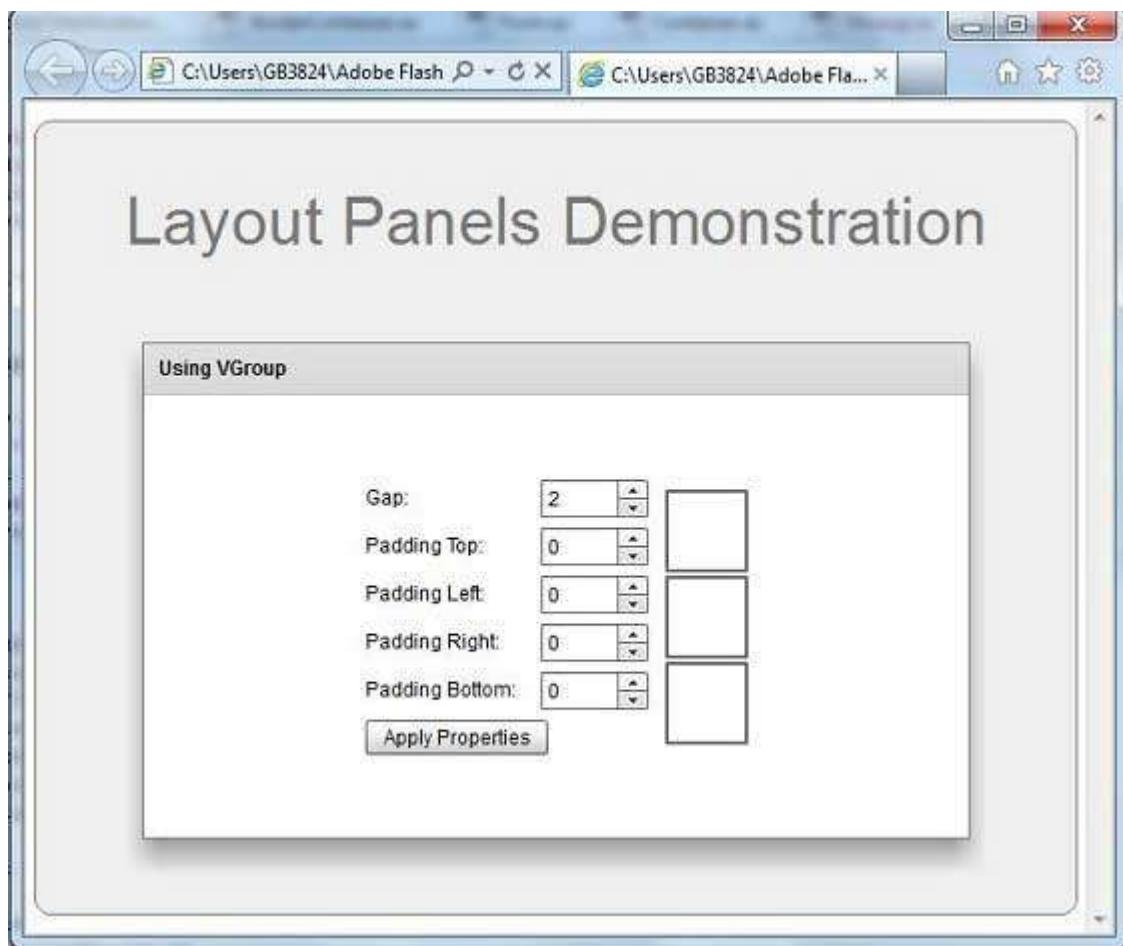
        }
    ]]>
</fx:Script>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Layout Panels Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="vGroupPanel" title="Using VGroup"
            width="500" height="300" includeInLayout="true"
            visible="true">
            <s:layout>
                <s:HorizontalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center"/>
            </s:layout>
            <s:VGroup top="10" left="15">
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Gap:" width="100"/>
                    <s:NumericStepper id="gapV" maximum="400"/>
                </s:HGroup>
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Padding Top:" width="100"/>
                    <s:NumericStepper id="padTopV" maximum="400"/>
                </s:HGroup>
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Padding Left:" width="100"/>
                    <s:NumericStepper id="padLeftV" maximum="400"/>
                </s:HGroup>
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Padding Right:" width="100"/>
                    <s:NumericStepper id="padRightV" maximum="400"/>
                </s:HGroup>
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Padding Bottom:" width="100"/>
                </s:HGroup>
            </s:VGroup>
        </s:Panel>
    </s:VGroup>
</s:BorderContainer>

```

```
<s:NumericStepper id="padBottomV" maximum="400"/>
</s:HGroup>
<s:Button label="Apply Properties"
click="applyVGroupProperties()"/>

</s:VGroup>
<s:VGroup left="300" top="25" id="mainVGroup">
<s:BorderContainer width="50" height="50"
borderWeight="2" color="0x323232" />
<s:BorderContainer width="50" height="50"
borderWeight="2" color="0x323232" />
<s:BorderContainer width="50" height="50"
borderWeight="2" color="0x323232" />
</s:VGroup>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – HGroup

The HGroup container is a Group container that uses the `HorizontalLayout` class. Use the properties of the `HGroup` class to modify the characteristics of the `HorizontalLayout` class.

### Class Declaration

Following is the declaration for **`spark.components.HGroup`** class:

```
public class HGroup
    extends Group
```

### Public Properties

S.N.	Property & Description
1	<b>firstIndexInView : int</b> [read-only] The index of the first layout element that is part of the layout and within the layout target's scroll rectangle, or -1 if nothing has been displayed yet.

2	<b>gap : int</b> The vertical space between layout elements, in pixels.
3	<b>horizontalAlign : String</b> The horizontal alignment of layout elements.
4	<b>lastIndexInView : int</b> [read-only] The index of the last row that's part of the layout and within the container's scroll rectangle, or -1 if nothing has been displayed yet.
5	<b>paddingBottom : Number</b> Number of pixels between the container's bottom edge and the bottom edge of the last layout element.
6	<b>paddingLeft : Number</b> The minimum number of pixels between the container's left edge and the left edge of the layout element.
7	<b>paddingRight : Number</b> The minimum number of pixels between the container's right edge and the right edge of the layout element.
8	<b>paddingTop : Number</b> Number of pixels between the container's top edge and the top edge of the first layout element.
9	<b>requestedMaxRowCount : int</b> The measured height of this layout is large enough to display at most requestedMaxRowCount layout elements.
10	<b>requestedMinRowCount : int</b> The measured height of this layout is large enough to display at least requestedMinRowCount layout elements.
11	<b>requestedRowCount : int</b> The measured size of this layout is tall enough to display the first requestedRowCount layout elements.
12	<b>rowCount : int</b>

	[read-only] The current number of visible elements.
13	rowHeight : Number  If variableRowHeight is false, then this property specifies the actual height of each child, in pixels.
14	variableRowHeight : Boolean  Specifies whether layout elements are allocated their preferred height.
15	verticalAlign : String  The vertical alignment of the content relative to the container's height.

## Public Methods

S.N.	Method & Description
1	<b>HGroup()</b>  Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.Group
- spark.components.supportClasses.GroupBase
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex HGroup Example

Let us follow the following steps to check usage of HGroup in a Flex application by creating a test application:

Step	Description

1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            private function applyVGroupProperties():void
            {
                mainHGroup.paddingTop = padTopH.value;
                mainHGroup.paddingLeft = paddingLeftH.value;
                mainHGroup.paddingRight = paddingRightH.value;
                mainHGroup.paddingBottom = paddingBottomH.value;
                mainHGroup.gap = gapH.value;
            }
        ]]>
    </fx:Script>
    <s:BorderContainer width="630" height="480" id="mainContainer"
        styleName="container">
        <s:VGroup width="100%" height="100%" gap="50"
            horizontalAlign="center" verticalAlign="middle">
            <s:Label id="lblHeader" text="Layout Panels Demonstration">
        </s:VGroup>
    </s:BorderContainer>
</s:Application>
```

```

        fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="hGroupPanel" title="Using HGroup"
width="500" height="300" includeInLayout="true"
visible="true">
    <s:layout>
        <s:HorizontalLayout gap="10" verticalAlign="middle"
            horizontalAlign="center"/>
    </s:layout>
    <s:VGroup top="10" left="15">
        <s:HGroup verticalAlign="middle">
            <s:Label text="Gap:" width="100"/>
            <s:NumericStepper id="gapH" maximum="400"/>
        </s:HGroup>
        <s:HGroup verticalAlign="middle">
            <s:Label text="Padding Top:" width="100"/>
            <s:NumericStepper id="padTopH" maximum="400"/>
        </s:HGroup>
        <s:HGroup verticalAlign="middle">
            <s:Label text="Padding Left:" width="100"/>
            <s:NumericStepper id="padLeftH" maximum="400"/>
        </s:HGroup>
        <s:HGroup verticalAlign="middle">
            <s:Label text="Padding Right:" width="100"/>
            <s:NumericStepper id="padRightH" maximum="400"/>
        </s:HGroup>
        <s:HGroup verticalAlign="middle">
            <s:Label text="Padding Bottom:" width="100"/>
            <s:NumericStepper id="padBottomH" maximum="400"/>
        </s:HGroup>
        <s:Button label="Apply Properties"
            click="applyVGroupProperties()"/>
    </s:VGroup>
    <s:HGroup left="300" top="25" id="mainHGroup">
        <s:BorderContainer width="50" height="50"

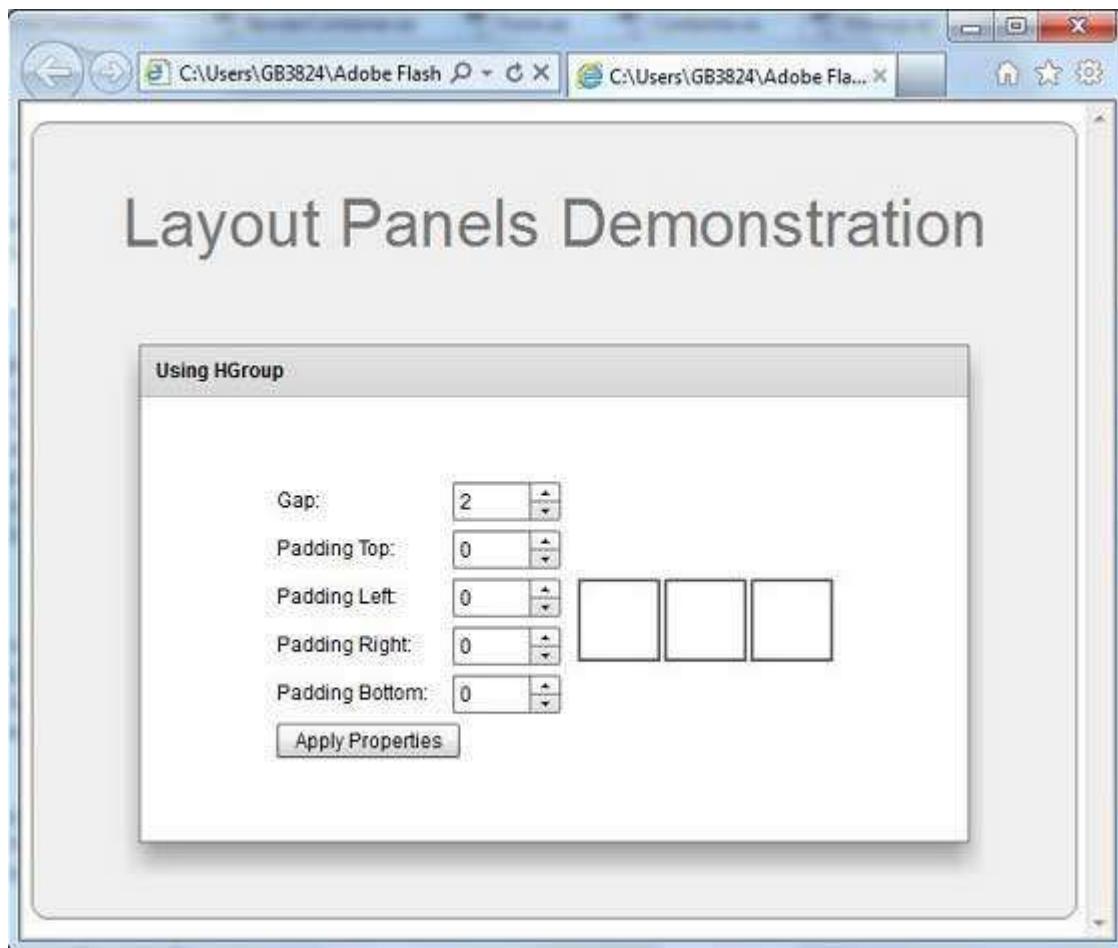
```

```

        borderWeight="2" color="0x323232" />
<s:BorderContainer width="50" height="50"
        borderWeight="2" color="0x323232" />
<s:BorderContainer width="50" height="50"
        borderWeight="2" color="0x323232" />
    </s:HGroup>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – Panel

The Panel class is a container that includes a title bar, a caption, a border, and a content area for its children.

## Class Declaration

Following is the declaration for **spark.components.Panel** class:

```
public class Panel
    extends SkinnableContainer
```

## Public Properties

S.N.	Property & Description
1	controlBarContent : Array The set of components to include in the control bar area of the Panel container.
2	controlBarLayout : LayoutBase Defines the layout of the control bar area of the container.
3	controlBarVisible : Boolean If true, the control bar is visible.
4	title : String Title or caption displayed in the title bar.

## Public Methods

S.N.	Method & Description
1	Panel() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.SkinnableContainer
- spark.components.supportClasses.SkinnableContainerBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject

- flash.events.EventDispatcher
- Object

## Flex Panel Example

Let us follow the following steps to check usage of Panel in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <s:BorderContainer width="630" height="480" id="mainContainer"
        styleName="container">
        <s:VGroup width="100%" height="100%" gap="50"
            horizontalAlign="center" verticalAlign="middle">
            <s:Label id="lblHeader" text="Layout Panels Demonstration"
                fontSize="40" color="0x777777" styleName="heading"/>
            <s:Panel id="panelPanel" title="Using Panel" width="500"
                height="300">
                <s:layout>
                    <s:HorizontalLayout gap="10"
                        verticalAlign="middle" horizontalAlign="center"/>
                
```

```
</s:layout>

<s:Panel title="Basic Layout Panel" top="0"
    left="5" height="200" width="150" >

    <s:BorderContainer width="50" height="50"
        borderWeight="2" color="0x323232" top="0" />
    <s:BorderContainer width="50" height="50"
        borderWeight="2" color="0x323232" top="55" />
    <s:BorderContainer width="50" height="50"
        borderWeight="2" color="0x323232" top="110" />
</s:Panel>

<s:Panel title="Horizontal Layout Panel" top="0"
    left="5" height="200" width="170" >
    <s:layout>
        <s:HorizontalLayout/>
    </s:layout>
    <s:BorderContainer width="50" height="50"
        borderWeight="2" color="0x323232" top="0" />
    <s:BorderContainer width="50" height="50"
        borderWeight="2" color="0x323232" top="55" />
    <s:BorderContainer width="50" height="50"
        borderWeight="2" color="0x323232" top="110" />
</s:Panel>

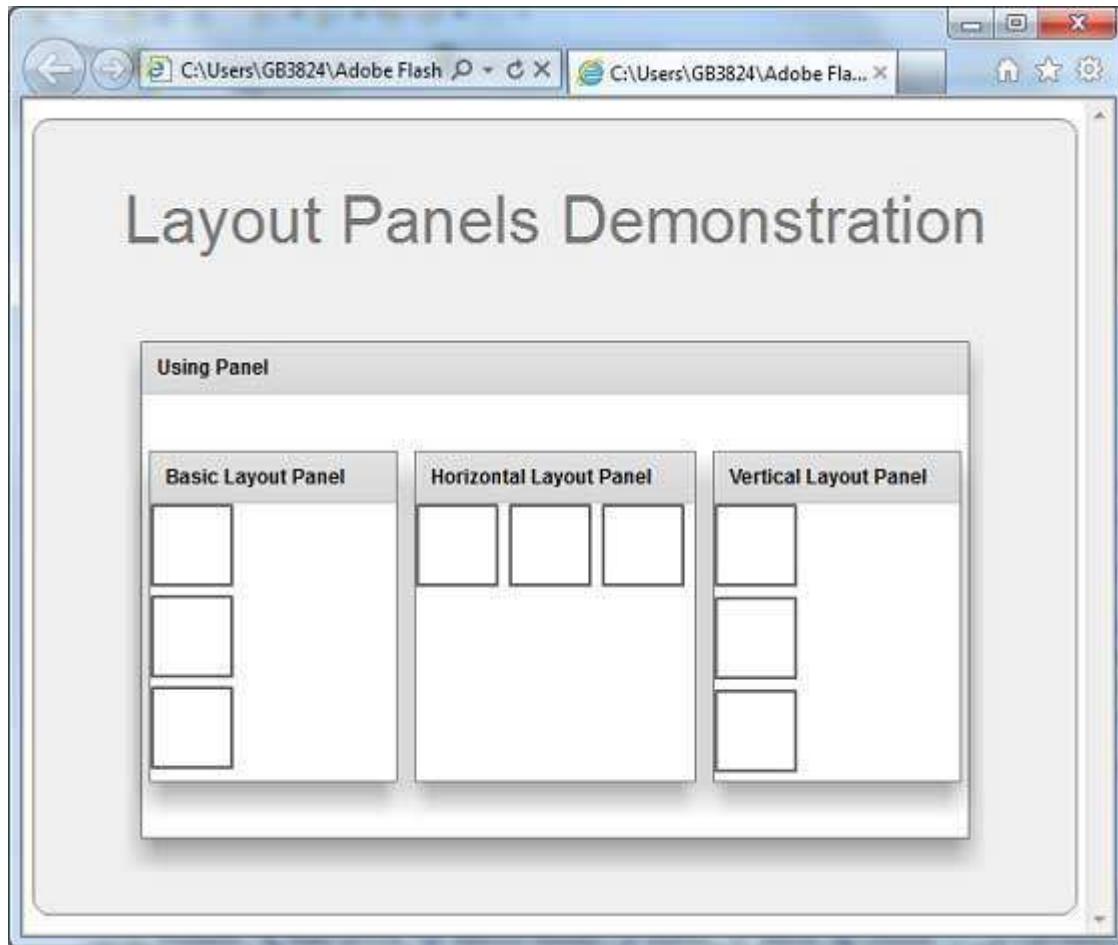
<s:Panel title="Vertical Layout Panel" top="0"
    left="5" height="200" width="150" >
    <s:layout>
        <s:VerticalLayout />
    </s:layout>
    <s:BorderContainer width="50" height="50"
        borderWeight="2" color="0x323232" top="0" />
    <s:BorderContainer width="50" height="50"
        borderWeight="2" color="0x323232" top="55" />
    <s:BorderContainer width="50" height="50"
        borderWeight="2" color="0x323232" top="110" />
</s:Panel>
```

```

</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – SkinnableContainer

The SkinnableContainer class is the base class for all the skinnable containers that have visual content.

### Class Declaration

Following is the declaration for **spark.components.SkinnableContainer** class:

```
public class SkinnableContainer
```

```
extends SkinnableContainerBase
implements IDeferredContentOwner, IVisualElementContainer
```

## Public Properties

S.N.	Property & Description
1	autoLayout : Boolean If true, measurement and layout are done when the position or size of a child is changed.
2	creationPolicy : String Content creation policy for this component.
3	deferredContentCreated : Boolean [read-only] Contains true if deferred content has been created.
4	layout : LayoutBase The layout object for this container.
5	mxmlContent : Array [write-only] The visual content children for this Group.
6	mxmlContentFactory : IDeferredInstance [write-only] A factory object that creates the initial value for the content property.
7	numElements : int [read-only] The number of visual elements in this container.

## Public Methods

S.N.	Method & Description
1	SkinnableContainer() Constructor.
2	addElement(element:IVisualElement):IVisualElement

	Adds a visual element to this container.
3	<b>addElementAt(element:IVisualElement, index:int):IVisualElement</b> Adds a visual element to this container.
4	<b>createDeferredContent():void</b> Create the content for this component.
5	<b>getElementAt(index:int):IVisualElement</b> Returns the visual element at the specified index.
6	<b>getElementIndex(element:IVisualElement):int</b> Returns the index position of a visual element.
7	<b>removeAllElements():void</b> Removes all visual elements from the container.
8	<b>removeElement(element:IVisualElement):IVisualElement</b> Removes the specified visual element from the child list of this container.
9	<b>removeElementAt(index:int):IVisualElement</b> Removes a visual element from the specified index position in the container.
10	<b>setElementIndex(element:IVisualElement, index:int):void</b> Changes the position of an existing visual element in the visual container.
11	<b>swapElements(element1:IVisualElement, element2:IVisualElement):void</b> Swaps the index of the two specified visual elements.
12	<b>swapElementsAt(index1:int, index2:int):void</b> Swaps the visual elements at the two specified index positions in the container.

## Protected Methods

S.N.	Method & Description
1	<b>createChildren():void</b>

	[override] Create content children, if the creationPolicy property is not equal to none.
2	partAdded(partName:String, instance:Object):void [override] Called when a skin part is added.
3	partRemoved(partName:String, instance:Object):void [override] Called when an instance of a skin part is being removed.

## Events

S.N.	Event & Description
1	<b>contentCreationComplete</b>  Dispatched after the content for this component has been created.
2	<b>elementAdd</b>  Dispatched when a visual element is added to the content holder.
3	<b>elementRemove</b>  Dispatched when a visual element is removed from the content holder.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.SkinnableContainerBase
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex SkinnableContainer Example

---

Let us follow the following steps to check usage of SkinnableContainer in a Flex application by creating a test application:

<b>Step</b>	<b>Description</b>
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Create a skin <i>SkinnableContainerSkin</i> for a host Component <i>SkinnableContainer</i> under a package <i>com.tutorialspoint.skin</i> as explained in <i>Flex - Style with skin</i> chapter. Keep rest of the files unchanged.
3	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
4	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file *src/com.tutorialspoint/skin/SkinnableContainerSkin.mxml*.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
         xmlns:s="library://ns.adobe.com/flex/spark"
         xmlns:fb="http://ns.adobe.com/flashbuilder/2009"
         alpha.disabled="0.5">

    <fx:Metadata>
        <![CDATA[
            /**
             * @copy spark.skins.spark.ApplicationSkin#hostComponent
             */
            [HostComponent("spark.components.SkinnableContainer")]
        ]]>
    </fx:Metadata>
    <s:states>
        <s:State name="normal" />
    </s:states>
```

```

<s:State name="disabled" />
</s:states>
<s:Rect left="0" right="0" top="0"
bottom="0" radiusX="5" radiusY="5">
<s:stroke>
<s:LinearGradientStroke weight="2"/>
</s:stroke>
<s:fill>
<s:LinearGradient rotation="90">
<s:entries>
<s:GradientEntry color="0xdddddd"/>
<s:GradientEntry color="0x020202" alpha=".5" />
</s:entries>
</s:LinearGradient>
</s:fill>
</s:Rect>
<s:Group id="contentGroup" left="0" right="0"
top="0" bottom="0" minWidth="0" minHeight="0">
<s:layout>
<s:BasicLayout/>
</s:layout>
</s:Group>
</s:Skin>

```

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```

<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx"
width="100%" height="100%" minWidth="500" minHeight="500"
>
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<s:BorderContainer width="630" height="480" id="mainContainer">

```

```

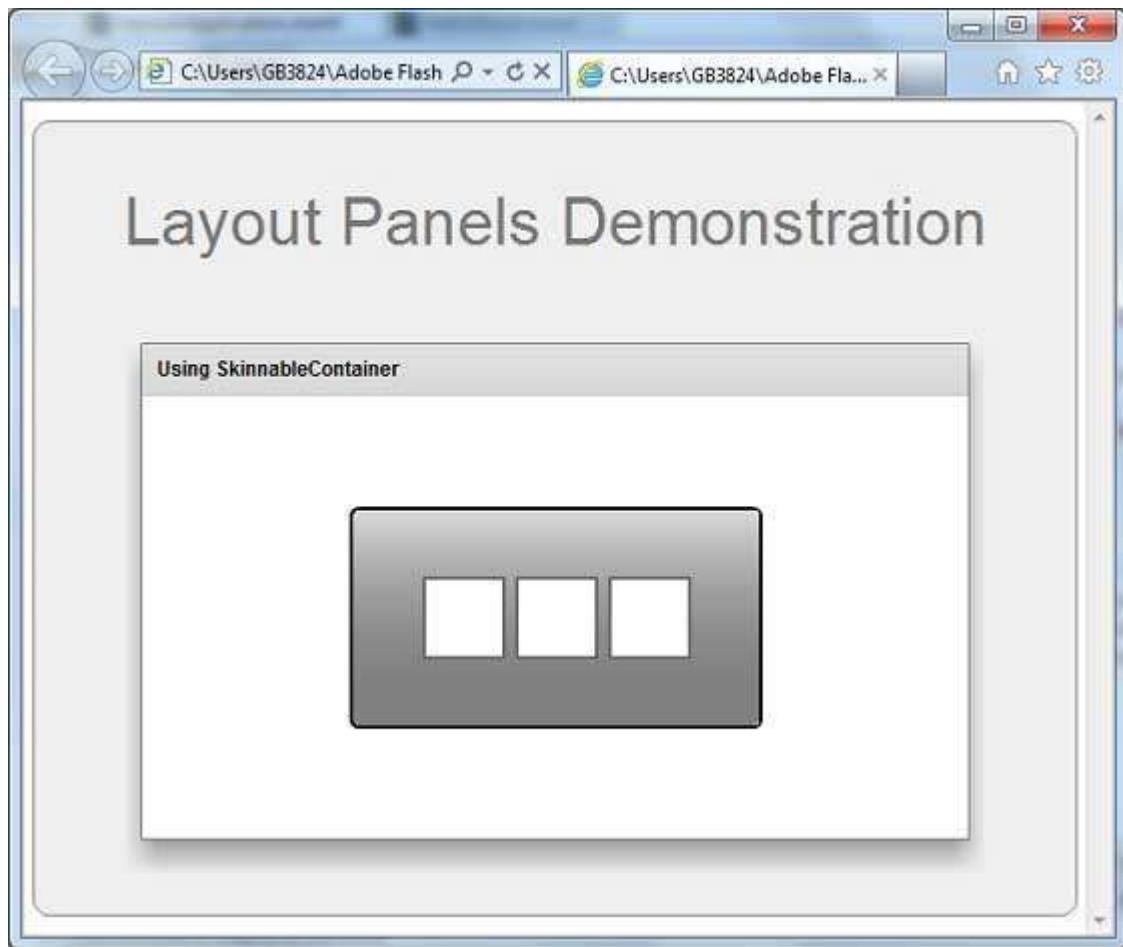
styleName="container">

<s:VGroup width="100%" height="100%" gap="50"
    horizontalAlign="center" verticalAlign="middle">
    <s:Label id="lblHeader" text="Layout Panels Demonstration"
        fontSize="40" color="0x777777" styleName="heading"/>
    <s:Panel id="skinnableContainerPanel" title="Using
SkinnableContainer"
        width="500" height="300" >
        <s:layout>
            <s:VerticalLayout gap="10" verticalAlign="middle"
                horizontalAlign="center"/>
        </s:layout>
        <s:SkinnableContainer

skinClass="com.tutorialspoint.skin.SkinnableContainerSkin"
    width="50%" height="50%" horizontalCenter="0"
    verticalCenter="0">
    <s:HGroup horizontalCenter="0" verticalCenter="0">
        <s:BorderContainer width="50" height="50"
            borderWeight="2" color="0x323232" />
        <s:BorderContainer width="50" height="50"
            borderWeight="2" color="0x323232" />
        <s:BorderContainer width="50" height="50"
            borderWeight="2" color="0x323232" />
    </s:HGroup>
    </s:SkinnableContainer>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – TabBar

The TabBar class represents a set of identical tabs.

### Class Declaration

Following is the declaration for **spark.components.TabBar** class:

```
public class TabBar
    extends ButtonBarBase
    implements IFocusManagerComponent
```

### Public Methods

S.N.	Method & Description
1	TabBar() Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.supportClasses.ButtonBarBase
- spark.components.ListBase
- spark.components.SkinnableDataContainer
- spark.components.supportClasses.SkinnableContainerBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex TabBar Example

Let us follow the following steps to check usage of TabBar in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <s:BorderContainer width="630" height="480" id="mainContainer">
```

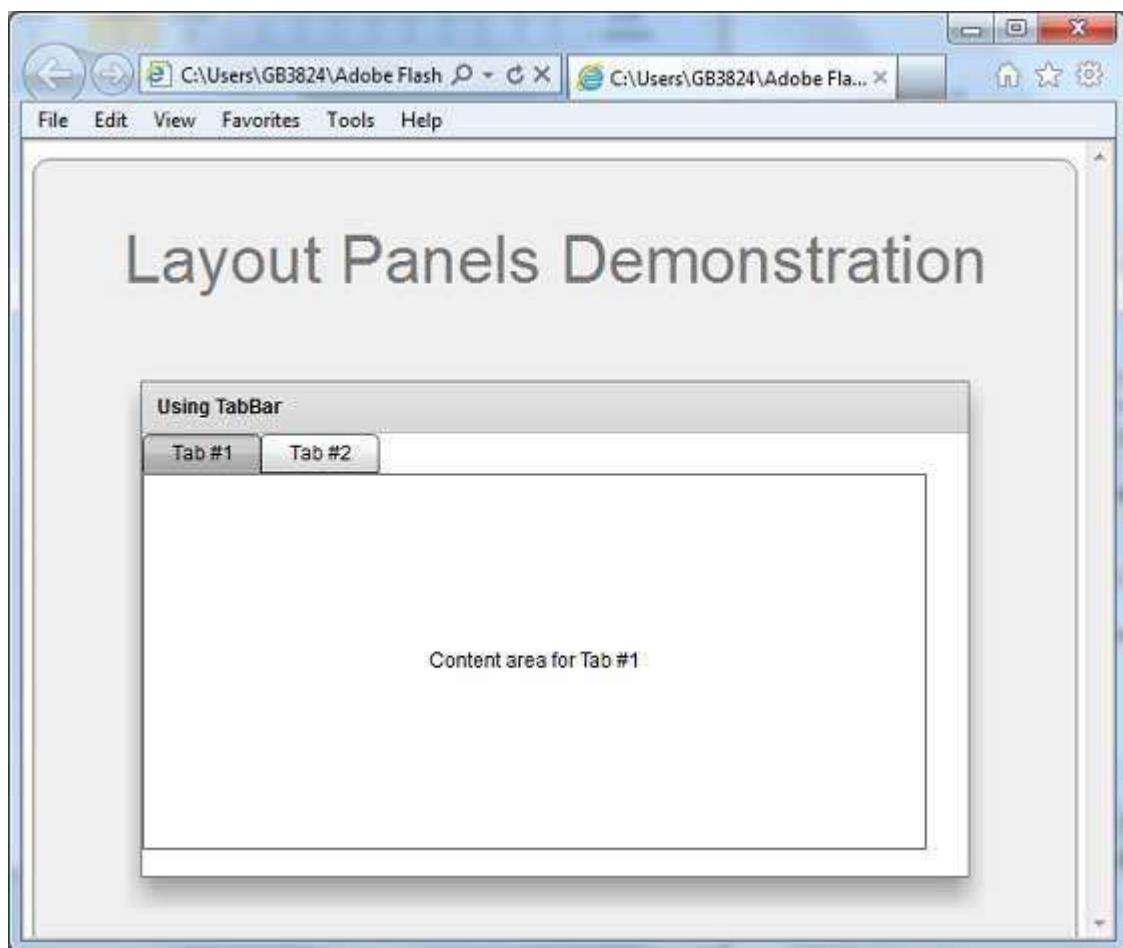
```

styleName="container">

<s:VGroup width="100%" height="100%" gap="50"
    horizontalAlign="center" verticalAlign="middle">
    <s:Label id="lblHeader" text="Layout Panels Demonstration"
        fontSize="40" color="0x777777" styleName="heading"/>
    <s:Panel id="tabBarPanel" title="Using TabBar"
        width="500" height="300" >
        <s:layout>
            <s:VerticalLayout gap="0" />
        </s:layout>
        <s:TabBar id="tabs" dataProvider="{viewStack}" left="5"
        top="5"/>
        <mx:ViewStack id="viewStack" width="95%" height="85%"
        left="5">
            <s:NavigatorContent label="Tab #1" width="100%"
            height="100%">
                <s:BorderContainer width="100%" height="100%" >
                    <s:Label text="Content area for Tab #1"
                        verticalCenter="0" horizontalCenter="0"/>
                </s:BorderContainer>
            </s:NavigatorContent>
            <s:NavigatorContent label="Tab #2" width="100%"
            height="100%">
                <s:BorderContainer width="100%" height="100%" >
                    <s:Label text="Content area for Tab #2"
                        verticalCenter="0" horizontalCenter="0"/>
                </s:BorderContainer>
            </s:NavigatorContent>
        </mx:ViewStack>
    </s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – TitleWindow

---

The TitleWindow class extends Panel class to include a close button and move area.

### Class Declaration

Following is the declaration for **spark.components.TitleWindow** class:

```
public class TitleWindow
    extends Panel
```

### Public Methods

S.N.	Method & Description
1	TitleWindow() Constructor.

## Events

S.N.	Event & Description
1	<b>close</b>  Dispatched when the user selects the close button.
2	<b>windowMove</b>  Dispatched after the user dragged the window successfully.
3	<b>windowMoveEnd</b>  Dispatched when the user releases the mouse button after dragging.
4	<b>windowMoveStart</b>  Dispatched when the user presses and hold the mouse button in the move area and begins to drag the window.
5	<b>windowMoving</b>  Dispatched when the user drags the window.

## Methods Inherited

This class inherits methods from the following classes:

- spark.components.Panel
- spark.components.SkinnableContainer
- spark.components.supportClasses.SkinnableContainerBase
- spark.components.supportClasses.SkinnableComponent
- mx.core.UIComponent
- mx.core.FlexSprite
- flash.display.Sprite
- flash.display.DisplayObjectContainer
- flash.display.InteractiveObject
- flash.display.DisplayObject
- flash.events.EventDispatcher
- Object

## Flex TitleWindow Example

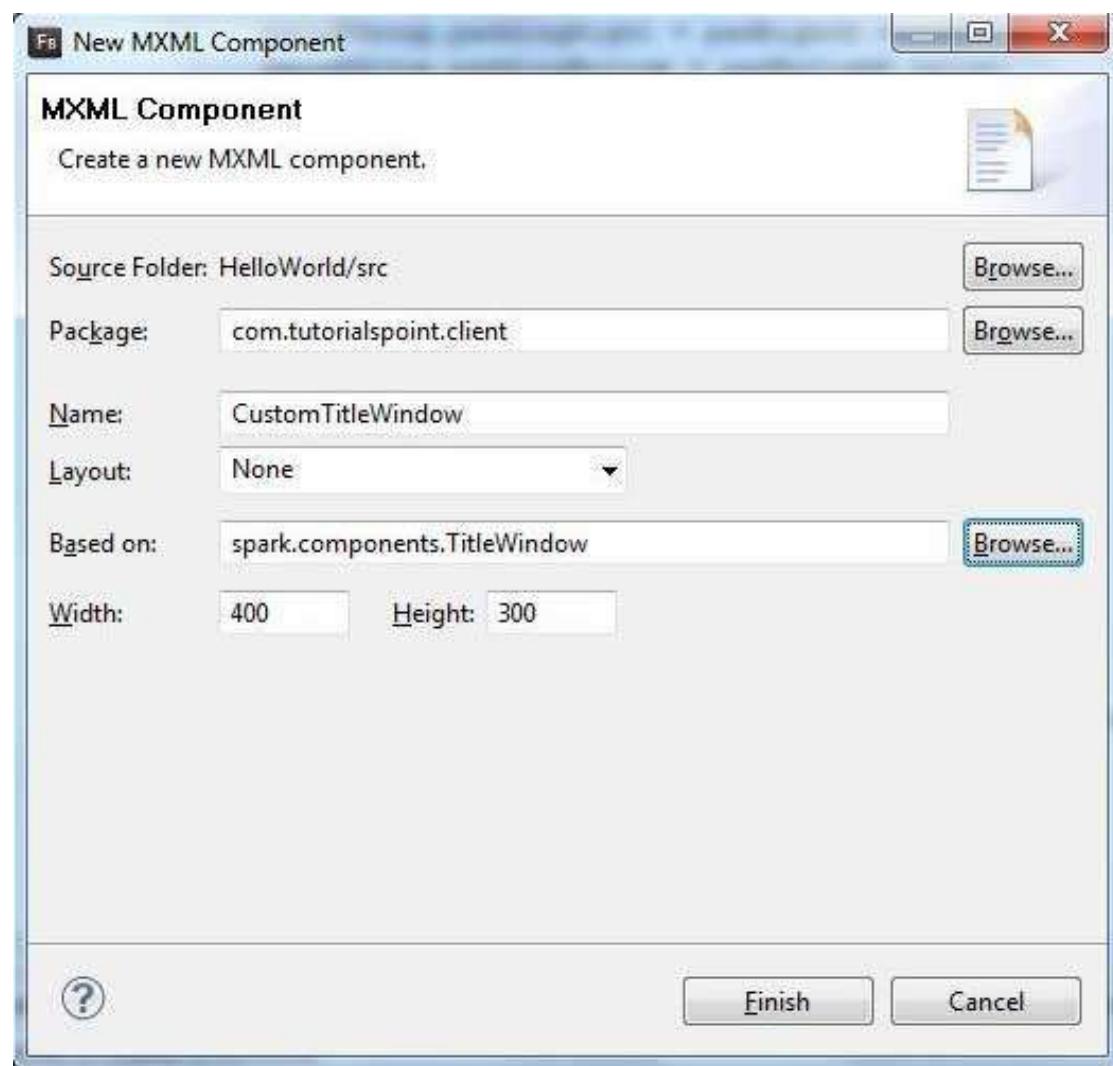
Let us follow the following steps to check usage of TitleWindow in a Flex application by creating a test application:

### Step 1 – Create a Project

Create a project with a name *HelloWorld* under a package *com.tutorialspoint.client* as explained in the *Flex - Create Application* chapter.

### Step 2 – Create a custom Title Window component

Launch Create MXML Component wizard using the option **File > New > MXML Component**



Enter Package as **com.tutorialspoint.client**, name as **CustomTitleWindow** and choose based on as existing flex TitleWindow control **spark.component.TitleWindow**. Modified it as mentioned below.

Following is the content of the modified mxmxml file

**src/com.tutorialspoint.client/CustomTitleWindow.mxml.**

```

<?xml version="1.0" encoding="utf-8"?>
<s:TitleWindow xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx" width="400" height="300"
    title="Title Window Container"
    close="PopUpManager.removePopUp(this);">
    <s:layout>
        <s:VerticalLayout verticalAlign="middle" horizontalAlign="center"
    />
    </s:layout>
    <fx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;
        ]]>
    </fx:Script>
    <s:Label text="Content area of title window"/>
</s:TitleWindow>

```

### Step 3 – Modify HelloWorld.mxml

Modify *HelloWorld.mxml* as explained below. Keep rest of the files unchanged.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```

<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            import mx.core.IFlexDisplayObject;
            import mx.managers.PopUpManager;
            private function showWindow():void{
                var popUp:IFlexDisplayObject =

```

```

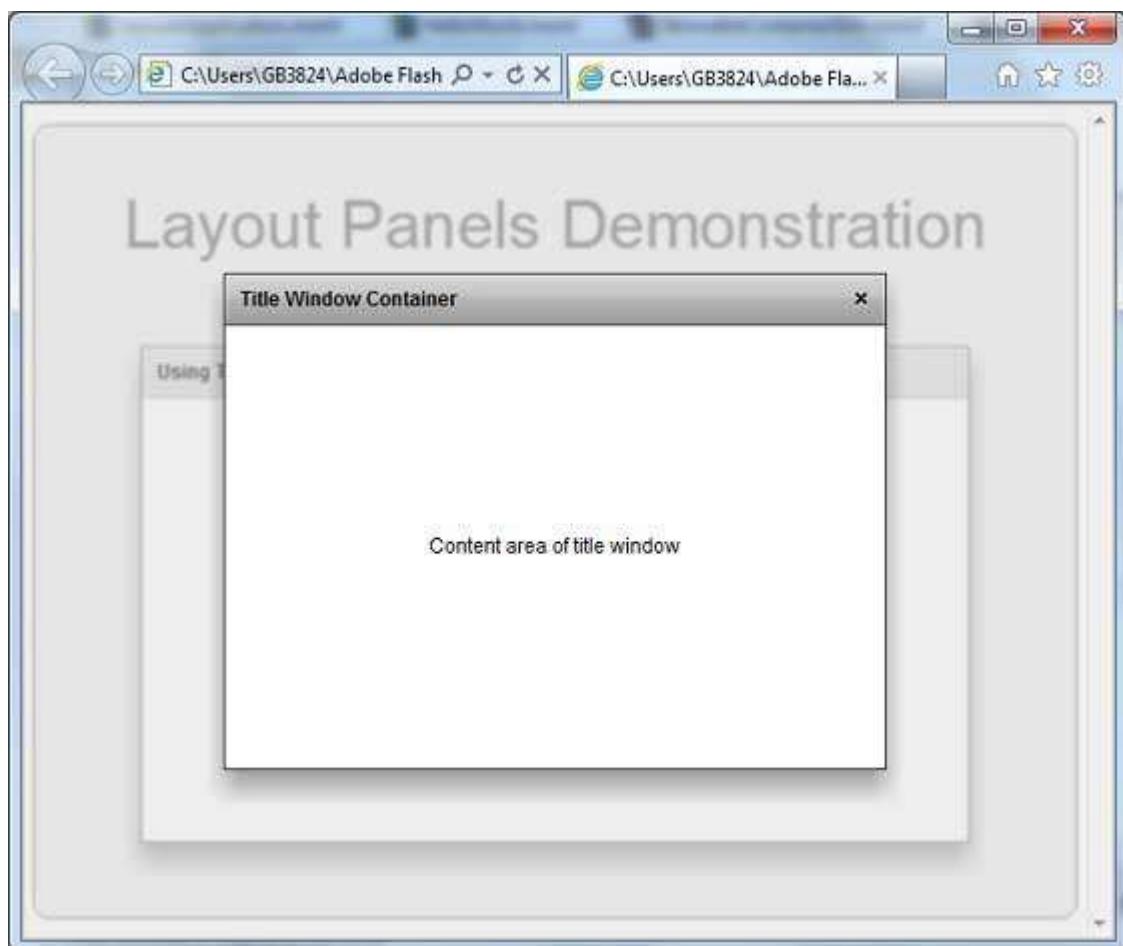
        PopUpManager.createPopUp(this,CustomTitleWindow,true);
        PopUpManager.centerPopUp(popUp);
    }
]]>
</fx:Script>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Layout Panels Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="titleWindowPanel" title="Using TitleWindow"
            width="500" height="300">
            <s:layout>
                <s:VerticalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center"/>
            </s:layout>
            <s:Button id="showButton"
                label="Click to show the TitleWindow container"
                click="showWindow();"/>
        </s:Panel>
    </s:VGroup>
</s:BorderContainer>
</s:Application>

```

## Step 4 – Complie and Run application

Compile and run the application to make sure business logic is working as per the requirements.

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



# 14. Flex – Visual Effects

We can add behavior to flex application using the concept of Effects. For example, when a text box gets focus, we can make its text become bolder and make its size slight bigger.

Every effect inherits properties from Effect class which in turn inherits properties from EventDispatcher and other top level classes.

S.N.	Effect & Description
1	Flex Effect Class  The Effect class is an abstract base class that defines the basic functionality of all Flex effects. This class defines the base factory class for all effects.

## Flex - Effect

The Effect class is an abstract base class that defines the basic functionality of all Flex effects. The Effect class defines the base factory class for all effects. The EffectInstance class defines the base class for all effect instance subclasses.

### Class Declaration

Following is the declaration for **mx.effects.Effect** class:

```
public class Effect
    extends EventDispatcher
    implements IEff
```

### Public Properties

S.N.	Property & Description
1	className : String  [read-only] The name of the effect class, such as "Fade".
2	customFilter : mx.effects:EffectTargetFilter  Specifies a custom filter object, of type EffectTargetFilter, used by the effect to determine the targets on which to play the effect.
3	duration : Number  Duration of the effect in milliseconds.

4	<b>effectTargetHost : mx.effects:IEffectTargetHost</b> A property that lets you access the target list-based control of a data effect.
5	<b>filter : String</b> Specifies an algorithm for filtering targets for an effect.
6	<b>hideFocusRing : Boolean</b> Determines whether the effect should hide the focus ring when starting the effect.
7	<b>instanceClass : Class</b> An object of type Class that specifies the effect instance class class for this effect class.
8	<b>isPlaying : Boolean</b> [read-only] A read-only flag which is true if any instances of the effect are currently playing, and false if none are.
9	<b>perElementOffset : Number</b> Additional delay, in milliseconds, for effect targets after the first target of the effect.
10	<b>playheadTime : Number</b> Current time position of the effect.
11	<b>relevantProperties : Array</b> An Array of property names to use when performing filtering.
12	<b>relevantStyles : Array</b> An Array of style names to use when performing filtering.
13	<b>repeatCount : int = 1</b> Number of times to repeat the effect.
14	<b>repeatDelay : int = 0</b> Amount of time, in milliseconds, to wait before repeating the effect.
15	<b>startDelay : int = 0</b>

	Amount of time, in milliseconds, to wait before starting the effect.
16	suspendBackgroundProcessing : Boolean = false If true, blocks all background processing while the effect is playing.
17	target : Object The object to which this effect is applied.
18	targets : Array An Array of objects that are targets for the effect.
19	triggerEvent : Event The Event object passed to this Effect by the EffectManager when an effect is triggered, or null if the effect is not being played by the EffectManager.

## Protected Properties

S.N.	Property & Description
1	applyTransitionEndProperties : Boolean This flag controls whether the effect, when run in a transition, automatically applies the property values according to the end state, as opposed to leaving values as set by the effect itself.
2	endValuesCaptured : Boolean = false A flag containing true if the end values of an effect have already been determined, or false if they should be acquired from the current properties of the effect targets when the effect runs.

## Public Methods

S.N.	Method & Description
1	Effect(target:Object = null) Constructor.
2	captureEndValues():void

	Captures the current values of the relevant properties on the effect's targets and saves them as end values.
3	<p><code>captureMoreStartValues(targets:Array):void</code></p> <p>Captures the current values of the relevant properties of an additional set of targets Flex uses this function when a data change effect is run.</p>
4	<p><code>captureStartValues():void</code></p> <p>Captures the current values of the relevant properties on the effect's targets.</p>
5	<p><code>createInstance(target:Object = null):IEffectInstance</code></p> <p>Creates a single effect instance and initializes it.</p>
6	<p><code>createInstances(targets:Array = null):Array</code></p> <p>Takes an Array of target objects and invokes the <code>createInstance()</code> method on each target.</p>
7	<p><code>deleteInstance(instance:IEffectInstance):void</code></p> <p>Removes event listeners from an instance and removes it from the list of instances.</p>
8	<p><code>end(effectInstance:IEffectInstance = null):void</code></p> <p>Interrupts an effect that is currently playing, and jumps immediately to the end of the effect.</p>
9	<p><code>getAffectedProperties():Array</code></p> <p>Returns an Array of Strings, where each String is the name of a property changed by this effect.</p>
10	<p><code>pause():void</code></p> <p>Pauses the effect until you call the <code>resume()</code> method.</p>
11	<p><code>play(targets:Array = null, playReversedFromEnd:Boolean = false):Array</code></p> <p>Begins playing the effect.</p>
12	<p><code>resume():void</code></p> <p>Resumes the effect after it has been paused by a call to the <code>pause()</code> method.</p>

13	<code>reverse():void</code> Plays the effect in reverse, if the effect is currently playing, starting from the current position of the effect.
14	<code>stop():void</code> Stops the effect, leaving the effect targets in their current state.

## Protected Methods

S.N.	Method & Description
1	<code>applyValueToTarget(target:Object, property:String, value:*, props:Object):void</code> Used internally by the Effect infrastructure.
2	<code>effectEndHandler(event:EffectEvent):void</code> Called when an effect instance has finished playing.
3	<code>effectStartHandler(event:EffectEvent):void</code> This method is called when the effect instance starts playing.
4	<code>effectStopHandler(event:EffectEvent):void</code> Called when an effect instance has stopped by a call to the stop() method.
5	<code>filterInstance(propChanges:Array, target:Object):Boolean</code> Determines the logic for filtering out an effect instance.
6	<code>getValueFromTarget(target:Object, property:String):*</code> Called by the captureStartValues() method to get the value of a property from the target.
7	<code>initInstance(instance:IEffectInstance):void</code> Copies properties of the effect to the effect instance.

## Events

S.N.	Event & Description

1	<b>effectEnd</b>
	Dispatched when one of the effect's instances finishes playing, either when the instance finishes playing or when the effect is interrupted by a call to the end() method.
2	<b>effectStart</b>
	Dispatched when the effect starts playing.
3	<b>effectStop</b>
	Dispatched when the effect has been stopped, which only occurs when the effect is interrupted by a call to the stop() method.

## Methods Inherited

This class inherits methods from the following classes:

- flash.events.EventDispatcher
- Object

## Basic Effects

---

Following are the few important Basic Visual Effects:

S.N.	Effect & Description
1	Fade  The Fade effect animates the alpha property of a component.
2	WipeLeft  The WipeLeft class defines a wipe left effect.
3	WipeRight  The WipeRight class defines a wipe right effect.
4	Move3D  The Move3D class moves a target object in the x, y, and z dimensions.
5	Scale3D  The Scale3D class scales a target object in three dimensions around the transform center.

6	Rotate3D  The Rotate3D class rotates a target object in three dimensions around the x, y, or z axes.
7	Animate  This Animate effect animates an arbitrary set of properties between values. Specify the properties and values to animate by setting the motionPaths property.

## Flex – Fade Effect

The Fade effect animates the alpha property of a component. If played on an object having visible property as false, and set to animate alpha from zero to a nonzero value, it will set object visible to true as a side-effect of fading it in.

### Class Declaration

Following is the declaration for **spark.effects.Fade** class:

```
public class Fade
    extends Animate
```

### Public Properties

S.N.	Property & Description
1	alphaFrom : Number  Initial value of the alpha property, between 0.0 and 1.0, where 0.0 means transparent and 1.0 means fully opaque.
2	alphaTo : Number  Final value of the alpha property, between 0.0 and 1.0, where 0.0 means transparent and 1.0 means fully opaque.

### Public Methods

S.N.	Method & Description
1	Fade(target:Object = null)  Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.effects.Animate
- mx.effects.Effect
- flash.events.EventDispatcher
- Object

## Flex Fade Effect Example

Let us follow the following steps to check usage of Fade Effect in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            protected function btnFadeIn_clickHandler
                (event:MouseEvent):void {
                    fadeIn.play();
                }
        ]]>
    </fx:Script>

```

```

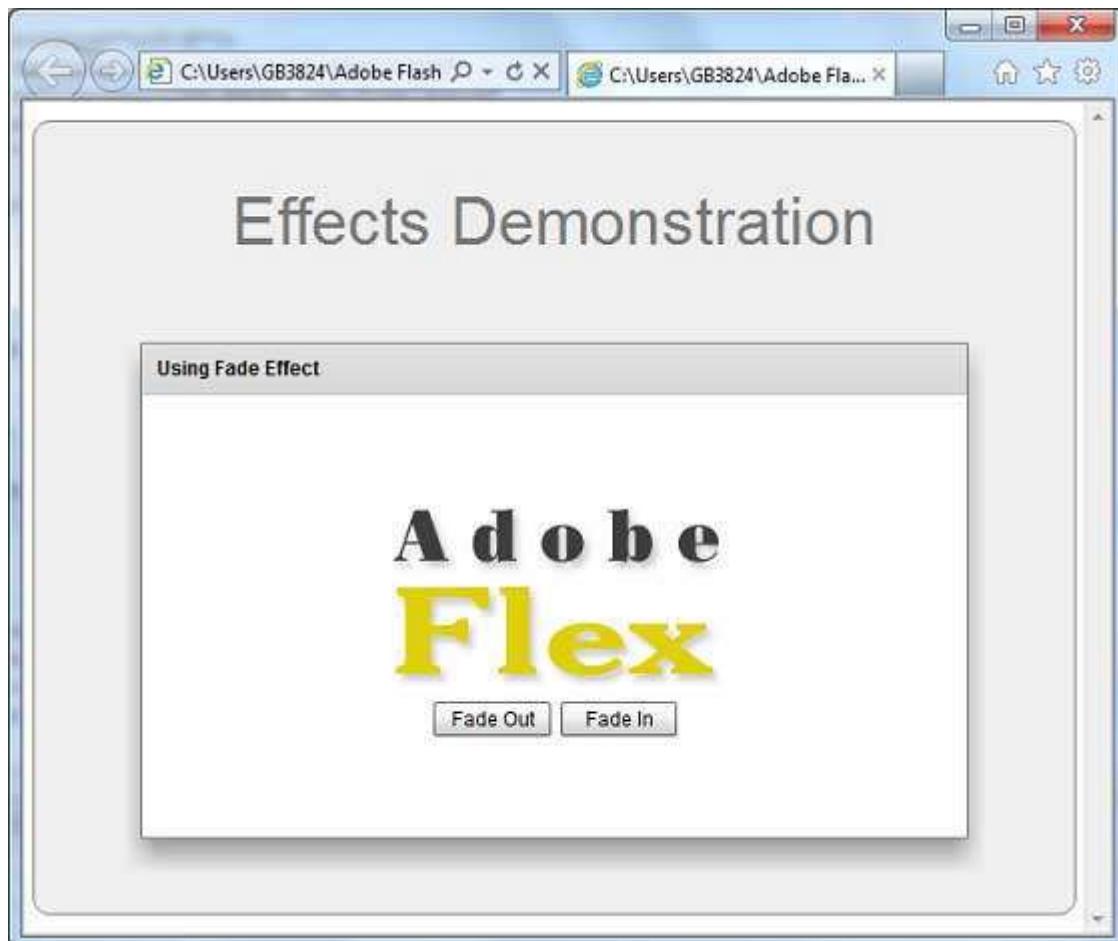
protected function btnFadeOut_clickHandler
(event:MouseEvent):void {
    fadeOut.play();
}
]]>
</fx:Script>
<fx:Declarations>
    <s:Fade id="fadeIn" duration="2000" target="{imageFlex}"
        alphaFrom="0" alphaTo="1"/>
    <s:Fade id="fadeOut" duration="2000" target="{imageFlex}"
        alphaFrom="1" alphaTo="0"/>
</fx:Declarations>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Effects Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="fadePanel" title="Using Fade Effect"
            width="500" height="300" includeInLayout="true"
visible="true">
            <s:layout>
                <s:VerticalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center"/>
            </s:layout>
            <s:Image id="imageFlex"
                source="http://www.tutorialspoint.com/images/flex-
mini.png" />
            <s:HGroup>
                <s:Button id="btnFadeOut" label="Fade Out"
                    click="btnFadeOut_clickHandler(event)"/>
                <s:Button id="btnFadeIn" label="Fade In"
                    click="btnFadeIn_clickHandler(event)"/>
            </s:HGroup>
    
```

```

</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – WipeLeft Effect

The WipeLeft class defines a wipe left effect. The before or after state of the component must be invisible.

### Class Declaration

Following is the declaration for **mx.effects.WipeLeft** class:

```

public class WipeLeft
    extends MaskEffect

```

## Public Methods

S.N.	Method & Description
1	WipeLeft(target:Object = null) Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- mx.effects.MaskEffect
- mx.effects.Effect
- flash.events.EventDispatcher
- Object

## Flex WipeLeft Effect Example

Let us follow the following steps to check usage of WipeLeft Effect in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
```

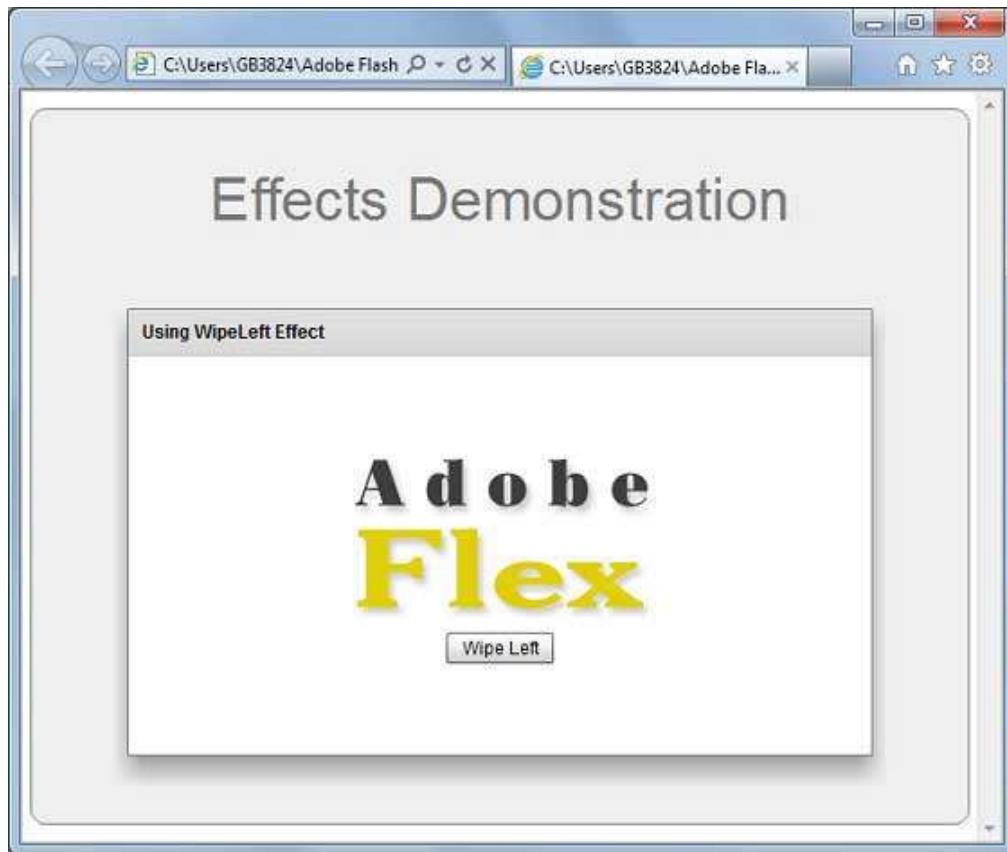
```

<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<fx:Script>
<![CDATA[
    protected function btnWipeLeft_clickHandler
    (event:MouseEvent):void {
        imageFlex.visible = true;
    }
]]>
</fx:Script>
<fx:Declarations>
    <mx:WipeLeft id="wipeLeft" duration="2000"/>
</fx:Declarations>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Effects Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="wipeLeftPanel" title="Using WipeLeft Effect"
            width="500" height="300" includeInLayout="true"
            visible="true">
            <s:layout>
                <s:VerticalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center"/>
            </s:layout>
            <mx:Image id="imageFlex" visible="{false}"
                source="http://www.tutorialspoint.com/images/flex-
mini.png"
                showEffect="{wipeLeft}" />
            <s:Button id="btnWipeLeft" label="Wipe Left"
                click="btnWipeLeft_clickHandler(event)"/>
        </s:Panel>
    </s:VGroup>
</s:BorderContainer>

```

```
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – WipeRight Effect

The WipeRight class defines a wipe right effect. The before or after state of the component must be invisible.

### Class Declaration

Following is the declaration for **mx.effects.WipeRight** class:

```
public class WipeRight
    extends MaskEffects
```

## Public Methods

S.N.	Method & Description
1	WipeRight(target:Object = null) Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- mx.effects.MaskEffect
- mx.effects.Effect
- flash.events.EventDispatcher
- Object

## Flex WipeRight Effect Example

---

Let us follow the following steps to check usage of WipeRight Effect in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
```

```

<fx:Script>
<![CDATA[
    protected function btnWipeRight_clickHandler
        (event:MouseEvent):void {
        imageFlex.visible = true;
    }
]]>
</fx:Script>
<fx:Declarations>
    <mx:WipeRight id="wipeRight" duration="2000"/>
</fx:Declarations>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Effects Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="wipeRightPanel" title="Using WipeRight Effect"
            width="500" height="300" includeInLayout="true"
            visible="true">
            <s:layout>
                <s:VerticalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center"/>

            </s:layout>
            <mx:Image id="imageFlex" visible="{false}"
                source="http://www.tutorialspoint.com/images/flex-
mini.png"
                showEffect="{wipeRight}" />
            <s:Button id="btnWipeRight" label="Wipe Right"
                click="btnWipeRight_clickHandler(event)"/>
        </s:Panel>
    </s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – Move3D Effect

The Move3D class moves a target object in the x, y, and z dimensions. The x, y, and z property specifications of the Move3D effect specify the change in x, y, and z that should occur to the transform center around which the overall transform effect occurs.

### Class Declaration

Following is the declaration for **spark.effects.Move3D** class:

```
public class Move3D
    extends AnimateTransform3D
```

### Public Properties

S.N.	Property & Description
------	------------------------

1	xBy : Number Number of pixels by which to modify the x position of the target.
2	xFrom : Number Initial x position of the target, in pixels.
3	xTo : Number Final x, in pixels.
4	yBy : Number Number of pixels by which to modify the y position of the target.
5	yFrom : Number Initial y position of the target, in pixels.
6	yTo : Number Final y position of the target, in pixels.
7	zBy : Number Number of pixels by which to modify the z position of the target.
8	zFrom : Number Initial z position of the target.
9	zTo : Number Final z position of the target.

## Public Methods

S.N.	Method & Description
1	Move3D(target:Object = null) Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.effects.AnimateTransform3D
- spark.effects.AnimateTransform
- spark.effects.Animate
- mx.effects.Effect
- flash.events.EventDispatcher
- Object

## Flex Move3D Effect Example

---

Let us follow the following steps to check usage of Move3D Effect in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            private function applyMoveProperties():void {
                moveEffect.play();
            }
        ]]>
    </fx:Script>
    <fx:Declarations>
```

```

<s:Move3D id="moveEffect" target="{imageFlex}"
    xFrom="{imageFlex.x}" xBy="{Number(moveX.text)}"
    yFrom="{imageFlex.y}" yBy="{Number(moveY.text)}"
    zFrom="{imageFlex.z}" zBy="{Number(moveZ.text)}" />
</fx:Declarations>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Effects Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="move3DPanel" title="Using Move3D Effect"
            width="500" height="300">
            <s:layout>
                <s:HorizontalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center"/>
            </s:layout>
            <s:VGroup top="10" left="15">
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Move By X:" width="70"/>
                    <s:TextInput id="moveX" text="50" width="50"/>
                </s:HGroup>
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Move By Y:" width="70"/>
                    <s:TextInput id="moveY" text="50" width="50"/>
                </s:HGroup>
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Move By Z:" width="70"/>
                    <s:TextInput id="moveZ" text="50" width="50"/>
                </s:HGroup>
                <s:Button label="Apply Properties"
                    click="applyMoveProperties()"/>
            </s:VGroup>
            <s:Image id="imageFlex"

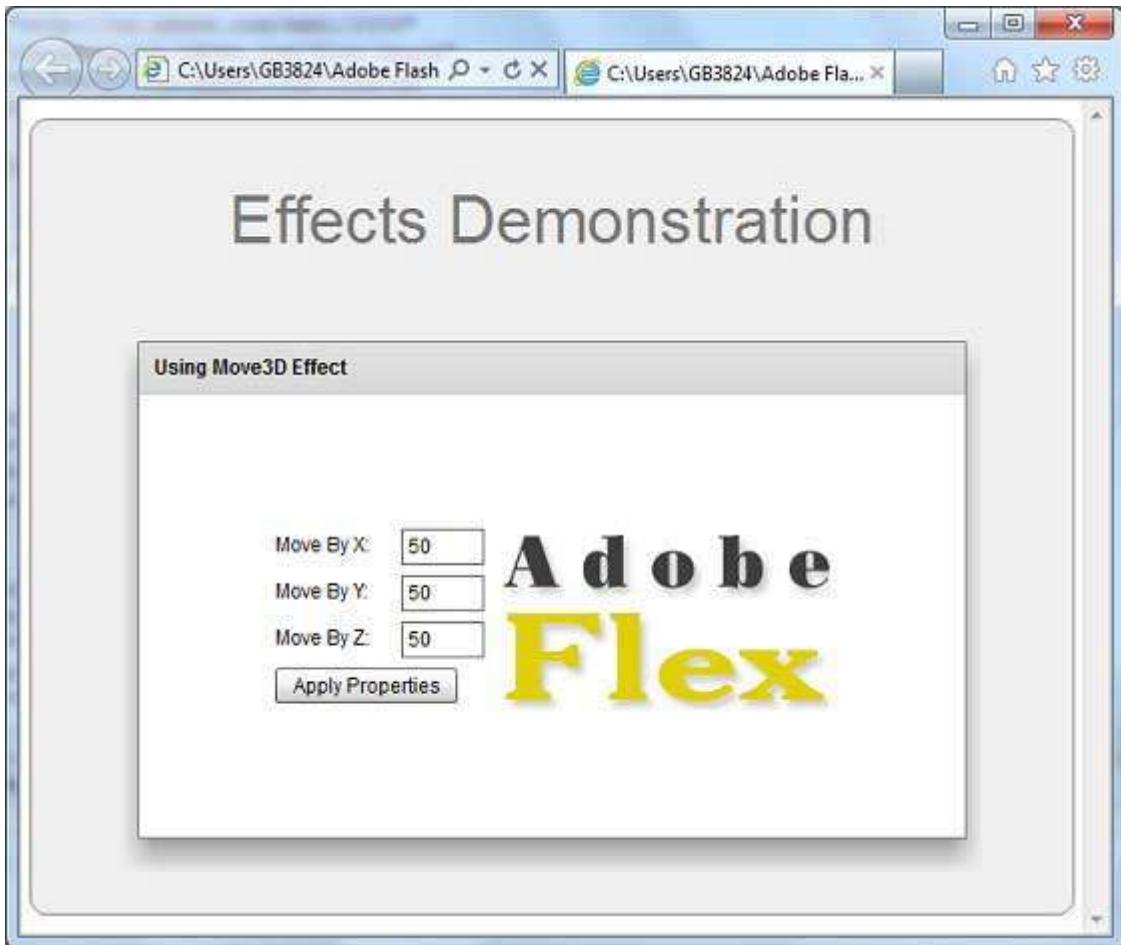
```

```

        source="http://www.tutorialspoint.com/images/flex-
mini.png"/>
    </s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



## Flex – Scale3D Effect

The Scale3D class scales a target object in three dimensions around the transform center. A scale of 2.0 means the object is magnified by a factor of 2, and a scale of 0.5 means the object is reduced by a factor of 2.

### Class Declaration

Following is the declaration for **spark.effects.Scale3D** class:

```
public class Scale3D
```

extends AnimateTransform3D
----------------------------

## Public Properties

S.N.	Property & Description
1	<b>scaleXBy</b> : Number The factor by which to scale the object in the x direction.
2	<b>scaleXFrom</b> : Number The starting scale factor in the x direction.
3	<b>scaleXTo</b> : Number The ending scale factor in the x direction.
4	<b>scaleYBy</b> : Number The factor by which to scale the object in the y direction.
5	<b>scaleYFrom</b> : Number The starting scale factor in the y direction.
6	<b>scaleYTo</b> : Number The ending scale factor in the y direction.
7	<b>scaleZBy</b> : Number The factor by which to scale the object in the z direction.
8	<b>scaleZFrom</b> : Number The starting scale factor in the z direction.
9	<b>scaleZTo</b> : Number The ending scale factor in the z direction.

## Public Methods

S.N.	Method & Description
------	----------------------

1	Scale3D(target:Object = null)
	Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.effects.AnimateTransform3D
- spark.effects.AnimateTransform
- spark.effects.Animate
- mx.effects.Effect
- flash.events.EventDispatcher
- Object

## Flex Scale3D Effect Example

Let us follow the following steps to check usage of Scale3D Effect in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
```

```

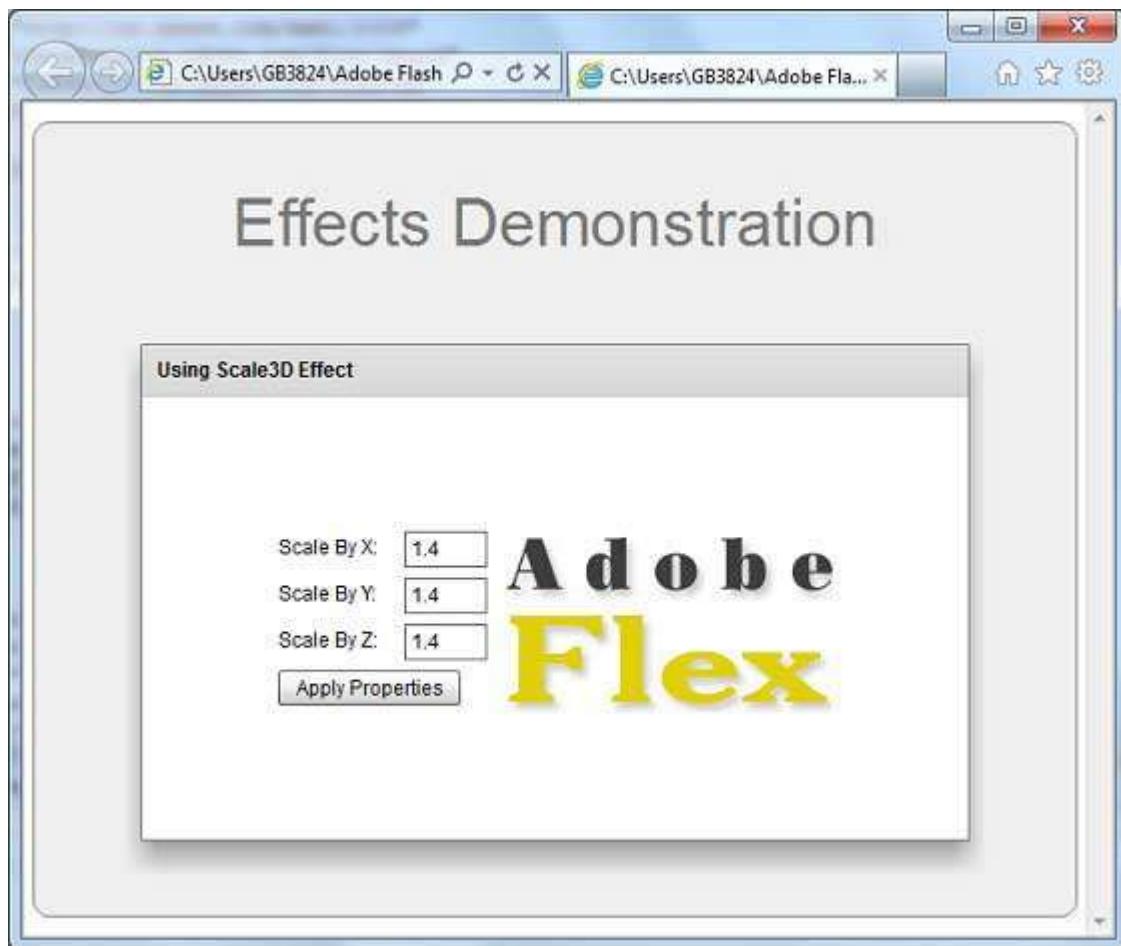
private function applyScaleProperties():void {
    scaleEffect.play();
}
]]>
</fx:Script>
<fx:Declarations>
    <s:Scale3D id="scaleEffect" target="{imageFlex}"
        scaleXFrom="1.0" scaleXTo="{Number(scalingX.text)}"
        scaleYFrom="1.0" scaleYTo="{Number(scalingY.text)}"
        scaleZFrom="1.0" scaleZTo="{Number(scalingZ.text)}"
    />
</fx:Declarations>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Effects Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="scale3DPanel" title="Using Scale3D Effect"
            width="500" height="300">
            <s:layout>
                <s:HorizontalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center"/>
            </s:layout>
            <s:VGroup top="10" left="15">
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Scale By X:" width="70"/>
                    <s:TextInput id="scalingX" text="1.4" width="50"/>
                </s:HGroup>
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Scale By Y:" width="70"/>
                    <s:TextInput id="scalingY" text="1.4" width="50"/>
                </s:HGroup>
                <s:HGroup verticalAlign="middle">

```

```
<s:Label text="Scale By Z:" width="70"/>
<s:TextInput id="scalingZ" text="1.4" width="50"/>
</s:HGroup>
<s:Button label="Apply Properties"
click="applyScaleProperties()"/>

</s:VGroup>
<s:Image id="imageFlex"
source="http://www.tutorialspoint.com/images/flex-
mini.png"/>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – Rotate3D Effect

The Rotate3D class rotates a target object in three dimensions around the x, y, or z axes. The rotation occurs around the transform center of the target.

### Class Declaration

Following is the declaration for **spark.effects.Rotate3D** class:

```
public class Rotate3D
    extends AnimateTransform3D
```

### Public Properties

S.N.	Property & Description
1	<b>angleXFrom</b> : Number The starting angle of rotation of the target object around the x axis, expressed in degrees.
2	<b>angleXTo</b> : Number The ending angle of rotation of the target object around the x axis, expressed in degrees.
3	<b>angleYFrom</b> : Number The starting angle of rotation of the target object around the y axis, expressed in degrees.
4	<b>angleYTo</b> : Number The ending angle of rotation of the target object around the y axis, expressed in degrees.
5	<b>angleZFrom</b> : Number The starting angle of rotation of the target object around the z axis, expressed in degrees.
6	<b>angleZTo</b> : Number The ending angle of rotation of the target object around the z axis, expressed in degrees.

## Public Methods

S.N.	Method & Description
1	Rotate3D(target:Object = null) Constructor.

## Methods Inherited

This class inherits methods from the following classes:

- spark.effects.AnimateTransform3D
- spark.effects.AnimateTransform
- spark.effects.Animate
- mx.effects.Effect
- flash.events.EventDispatcher
- Object

## Flex Rotate3D Effect Example

Let us follow the following steps to check usage of Rotate3D Effect in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
>
```

```

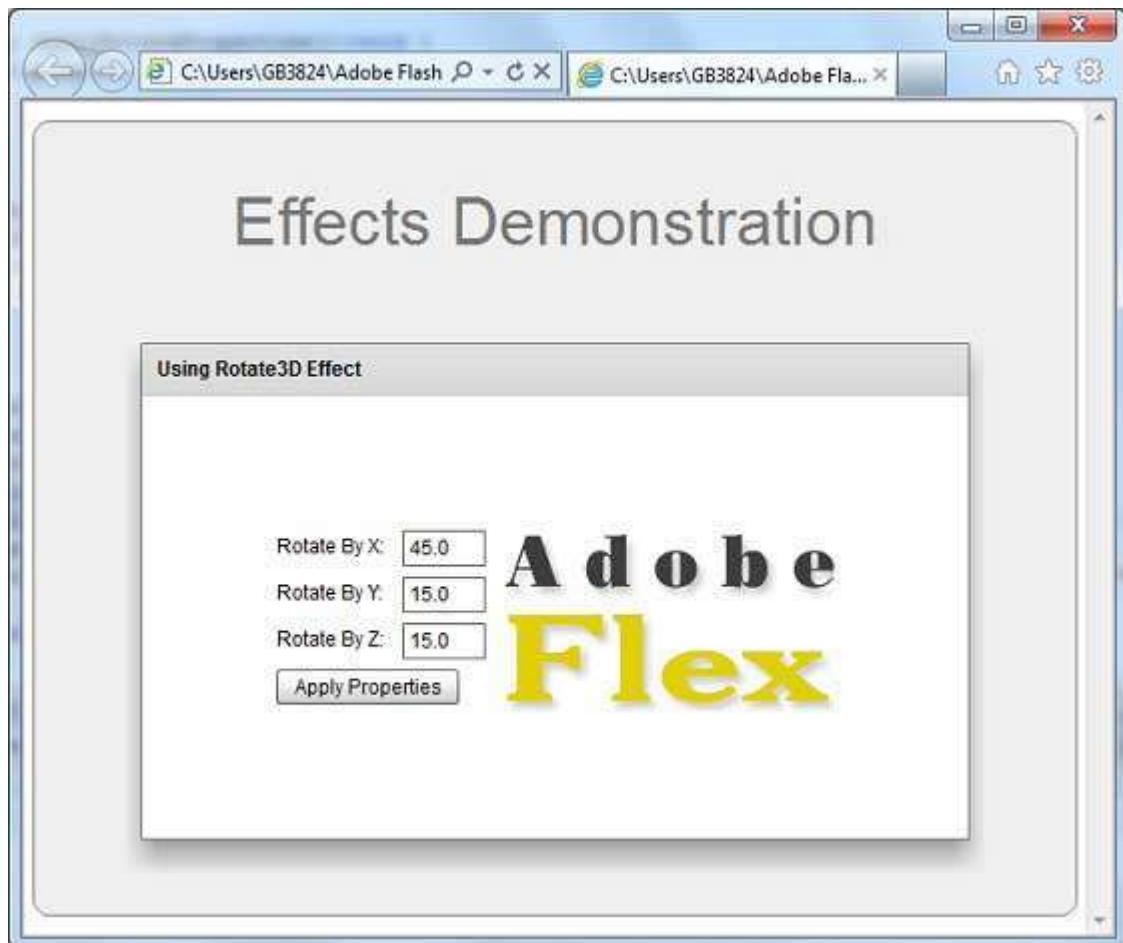
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<fx:Script>
<![CDATA[
    private function applyRotateProperties():void {
        rotateEffect.play();
    }
]]>
</fx:Script>
<fx:Declarations>
<s:Rotate3D id="rotateEffect" target="{imageFlex}"
    angleXFrom="0.0" angleXTo="{Number(rotateX.text)}"
    angleYFrom="0.0" angleYTo="{Number(rotateY.text)}"
    angleZFrom="0.0" angleZTo="{Number(rotateZ.text)}"
/>
</fx:Declarations>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Effects Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="rotate3DPanel" title="Using Rotate3D Effect"
            width="500" height="300">
            <s:layout>
                <s:HorizontalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center"/>
            </s:layout>
            <s:VGroup top="10" left="15">
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Rotate By X:" width="70"/>
                    <s:TextInput id="rotateX" text="45.0" width="50"/>
                </s:HGroup>
                <s:HGroup verticalAlign="middle">
                    <s:Label text="Rotate By Y:" width="70"/>

```

```
<s:TextInput id="rotateY" text="15.0" width="50"/>
</s:HGroup>
<s:HGroup verticalAlign="middle">
    <s:Label text="Rotate By Z:" width="70"/>
    <s:TextInput id="rotateZ" text="15.0" width="50"/>
</s:HGroup>
<s:Button label="Apply Properties"
    click="applyRotateProperties()"/>

</s:VGroup>
<s:Image id="imageFlex"
    source="http://www.tutorialspoint.com/images/flex-
mini.png"
    height="200" width="200"/>
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



## Flex – AnimateProperties Effect

This Animate effect animates an arbitrary set of properties between values. Specify the properties and values to animate by setting the motionPaths property.

### Class Declaration

Following is the declaration for **spark.effects.Animate** class:

```
public class Animate
    extends Effect
```

### Public Properties

S.N.	Property & Description
1	disableLayout : Boolean

	If true, the effect disables layout on its targets' parent containers, setting the containers autoLayout property to false, and also disables any layout constraints on the target objects.
2	easer : IEaser  The easing behavior for this effect.
3	interpolator : IInterpolator  The interpolator used by this effect to calculate values between the start and end values of a property.
4	motionPaths : Vector.<MotionPath>  A Vector of MotionPath objects, each of which holds the name of a property being animated and the values that the property takes during the animation.
5	repeatBehavior : String  The behavior of a repeating effect, which means an effect with repeatCount equal to either 0 or > 1.

## Public Methods

S.N.	Method & Description
1	Animate(target:Object = null)  Constructor.

## Events

S.N.	Event & Description
1	<b>effectRepeat</b>  Dispatched when the effect begins a new repetition, for any effect that is repeated more than once.
2	<b>effectUpdate</b>

	Dispatched every time the effect updates the target.
--	--

## Methods Inherited

This class inherits methods from the following classes:

- mx.effects.Effect
- flash.events.EventDispatcher
- Object

## Flex Animate Effect Example

Let us follow the following steps to check usage of Animate Effect in a Flex application by creating a test application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            private function applyAnimateProperties():void {
                animateEffect.play();
            }
        ]]>
```

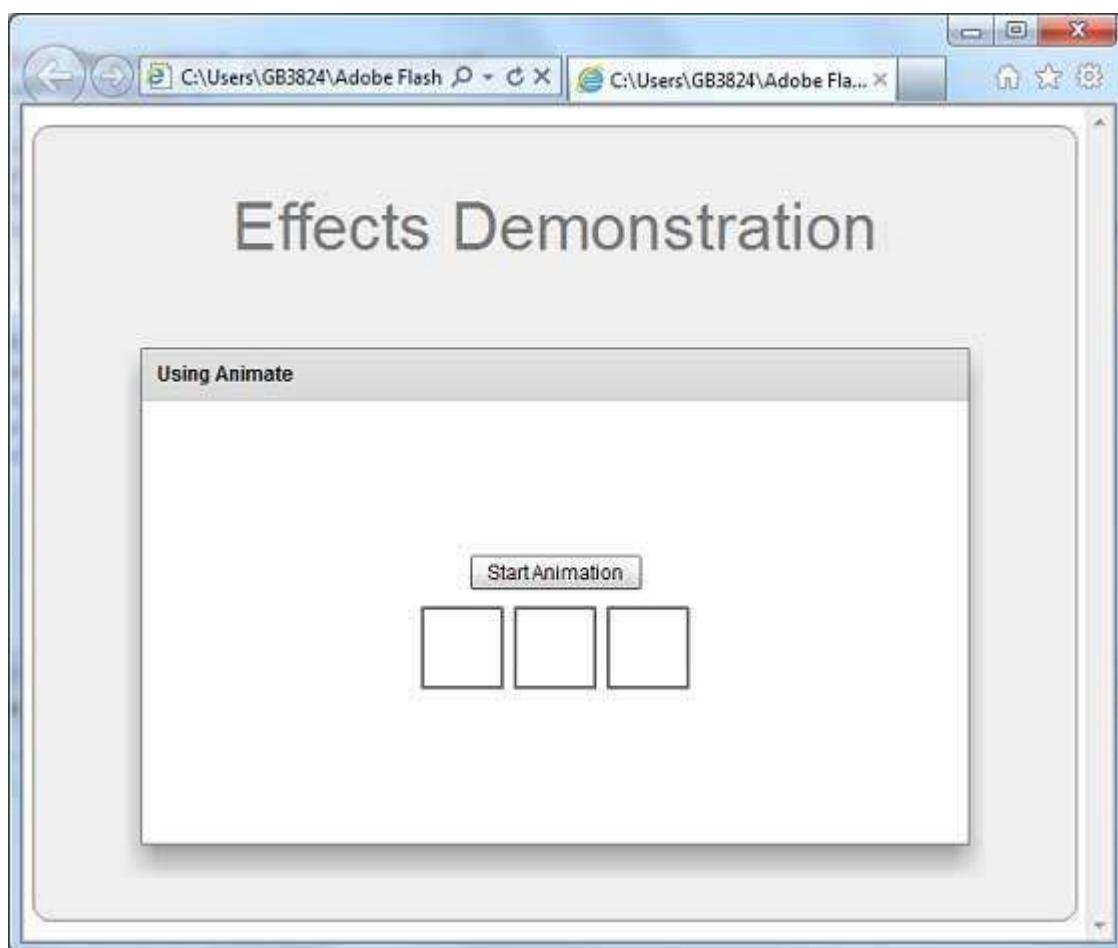
```

</fx:Script>
<fx:Declarations>
    <s:Animate id="animateEffect" duration="750"
        target="{mainHGroup}" >
        <s:SimpleMotionPath valueFrom="1" valueTo="15"
            property="gap" />
        <s:SimpleMotionPath valueFrom="0" valueTo="-50"
            property="z" />
    </s:Animate>
</fx:Declarations>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Effects Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>
        <s:Panel id="animatePanel" title="Using Animate"
            width="500" height="300" >
            <s:layout>
                <s:VerticalLayout gap="10" verticalAlign="middle"
                    horizontalAlign="center"/>
            </s:layout>
            <s:Button label="Start Animation"
                click="applyAnimateProperties()"/>
            <s:HGroup id="mainHGroup">
                <s:BorderContainer width="50" height="50"
                    borderWeight="2" color="0x323232" />
                <s:BorderContainer width="50" height="50"
                    borderWeight="2" color="0x323232" />
                <s:BorderContainer width="50" height="50"
                    borderWeight="2" color="0x323232" />
            </s:HGroup>
        </s:Panel>
    </s:VGroup>

```

```
</s:BorderContainer>  
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in Flex - Create Application chapter. If everything is fine with your application, it will produce the following result:



# 15. Flex – Event Handling

Flex uses concept of event to pass data from one object to another depending upon the state or user interaction within the application.

ActionScript has a generic **Event** class which defines much of the functionality needed to work with events. Every time an event occurs within a Flex application, three types of objects from the Event class hierarchy are created.

Event has the following three key properties

Property	Description
Type	<b>type</b> states about what kind of event just happened. This may be click, initialize, mouseover, change, etc. The actual values will be represented by constants like MouseEvent.CLICK.
Target	The <b>target</b> property of Event is an object reference to the component that generated the event. If you click a Button with an id of clickMeButton, the target of that click event will be clickMeButton
Currenttarget	The <b>currentTarget</b> property varies container hierarchy. It mainly deals with flow of events.

## Event Flow Phases

An event goes through three phases looking for event handlers.

Phase	Description
Capture	In the capture phase, the program will start looking for event handlers from the outside (or top) parent to the innermost one. The capture phase stops at the parent of the object that triggered the event.
Target	In the target phase, the component that triggered the event, is checked for an event handler.
Bubble	The Bubble phase is reverse of capture phase, working back through the structure, from the target component's parent on up.

Consider the following application code:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
```

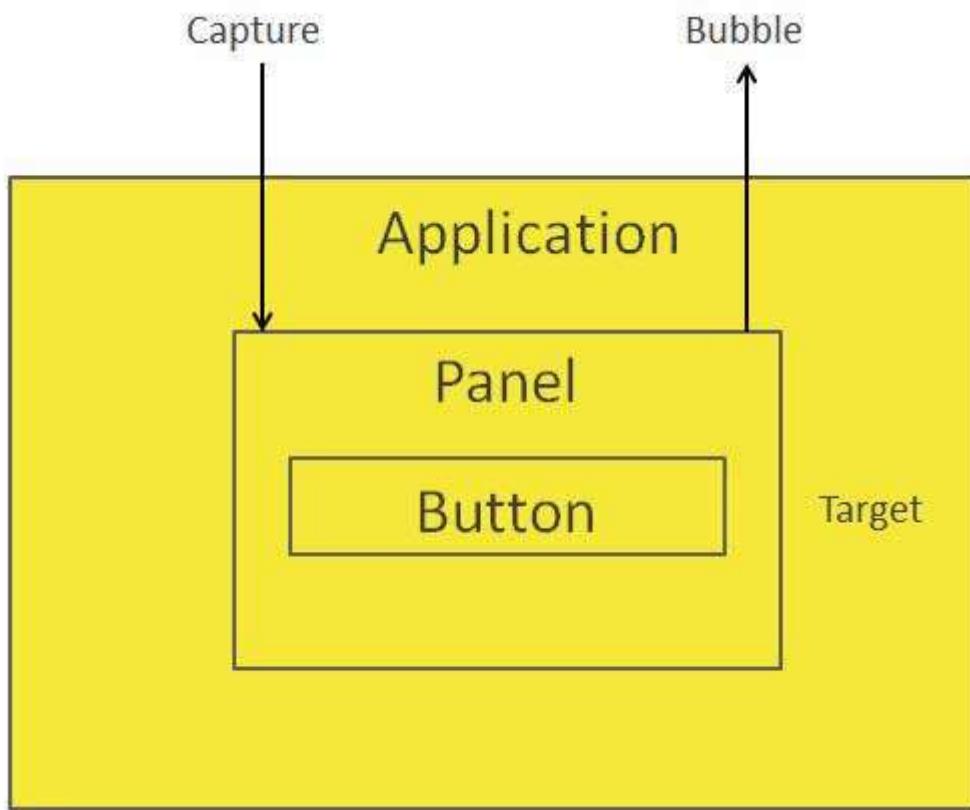
```

width="100%" height="100%
minWidth="500" minHeight="500" >
<s:Panel>
  <s:Button id="clickMeButton" label="Click Me!" click="doAction( );"/>
</s:Panel>
</s:Application>

```

When the user clicks the Button, he or she has also clicked the Panel and the Application.

The event goes through three phases looking for event-handler assignments.



Let us follow the steps below to test event handling in a Flex application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxmxml file: **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%" minWidth="500" minHeight="500"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            protected function reportEvent(event:MouseEvent):void
            {
                var target:String = event.target.id;
                var currentTarget:String = event.currentTarget.id;
                var eventPhase: String;

                if(event.target is Button){
                    var button:Button = event.target as Button;
                    target = button.label + " Button";
                } else if(event.target is HGroup){
                    var hGroup:HGroup = event.target as HGroup;
                    target = hGroup.id + " HGroup";
                }else if(event.target is Panel){
                    var panel:Panel = event.target as Panel;
                    target = panel.id + " Panel";
                }

                if(event.currentTarget is Button){
                    var button1:Button = event.currentTarget as Button;
                    currentTarget = button1.label + " Button";
                }else if(event.currentTarget is HGroup){
                    var hGroup1:HGroup = event.currentTarget as HGroup;
                    currentTarget = hGroup1.id + " HGroup";
                }else if(event.currentTarget is Panel){
                    var panel1:Panel = event.currentTarget as Panel;
                    currentTarget = panel1.id + " Panel";
                }
            }
        ]]>
    </fx:Script>

```

```

        currentTarget = panel1.id + " Panel";
    }

    var eventPhaseInt:uint = event.eventPhase;

    if(eventPhaseInt == EventPhase.AT_TARGET){
        eventPhase = "Target";
    } else if(eventPhaseInt == EventPhase.BUBBLING_PHASE){
        eventPhase = "Bubbling";
    }else if(eventPhaseInt == EventPhase.CAPTURING_PHASE){
        eventPhase = "Capturing";
    }

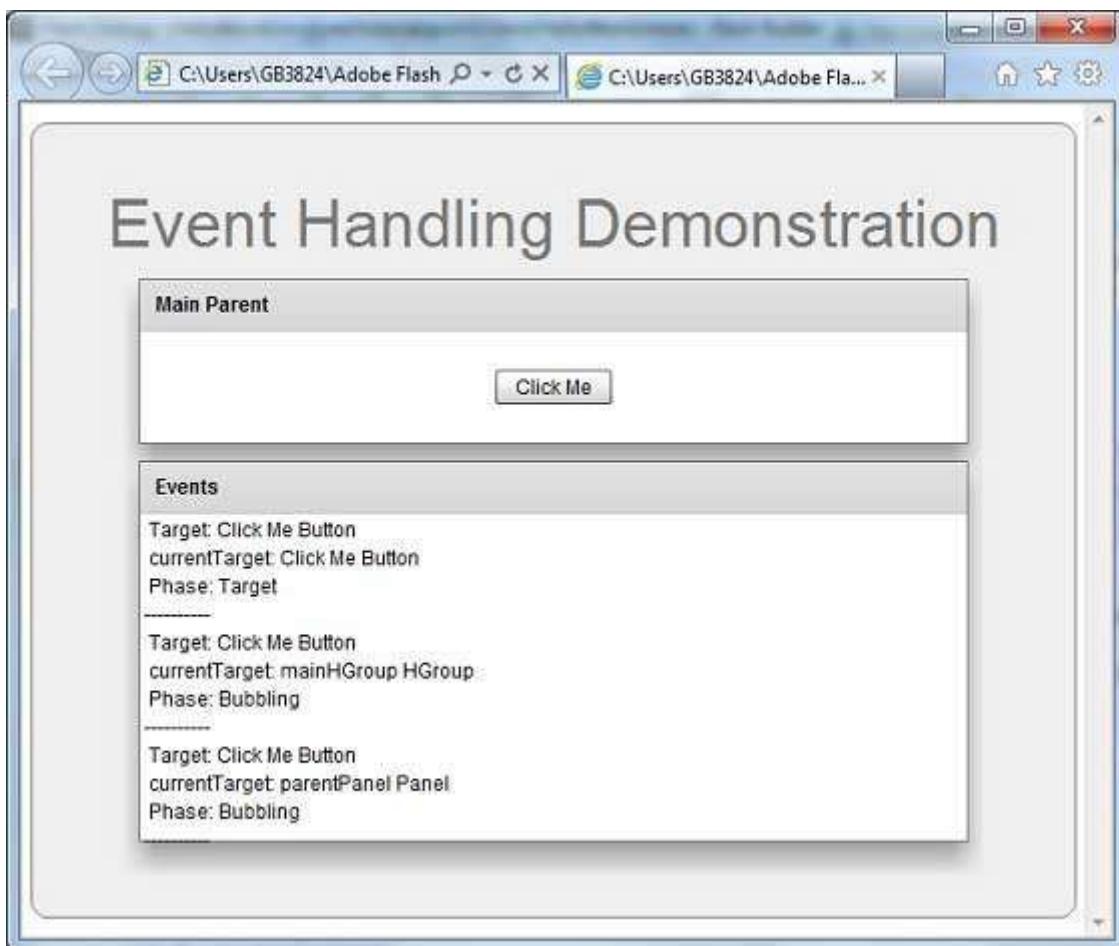
    reports.text += " Target: " + target + "\n currentTarget: " +
    currentTarget + "\n Phase: " + eventPhase + "\n-----\n";

}
]]>
</fx:Script>
<s:BorderContainer width="630" height="480" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="10"
horizontalAlign="center" verticalAlign="middle">
<s:Label id="lblHeader" text="Event Handling Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="parentPanel" title="Main Parent"
click="reportEvent(event)" width="500"
height="100" includeInLayout="true" visible="true">
<s:layout>
    <s:VerticalLayout gap="10"
        verticalAlign="middle" horizontalAlign="center"/>
</s:layout>
    <s:HGroup id="mainHGroup" click="reportEvent(event)">
        <s:Button label="Click Me"
            click="reportEvent(event)"/>
    </s:HGroup>
</s:Panel>

```

```
<s:Panel id="reportPanel"
    title="Events" width="500" height="230">
    <mx:Text id="reports" />
</s:Panel>
</s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, then it will produce the following result:



# 16. Flex – Custom Controls

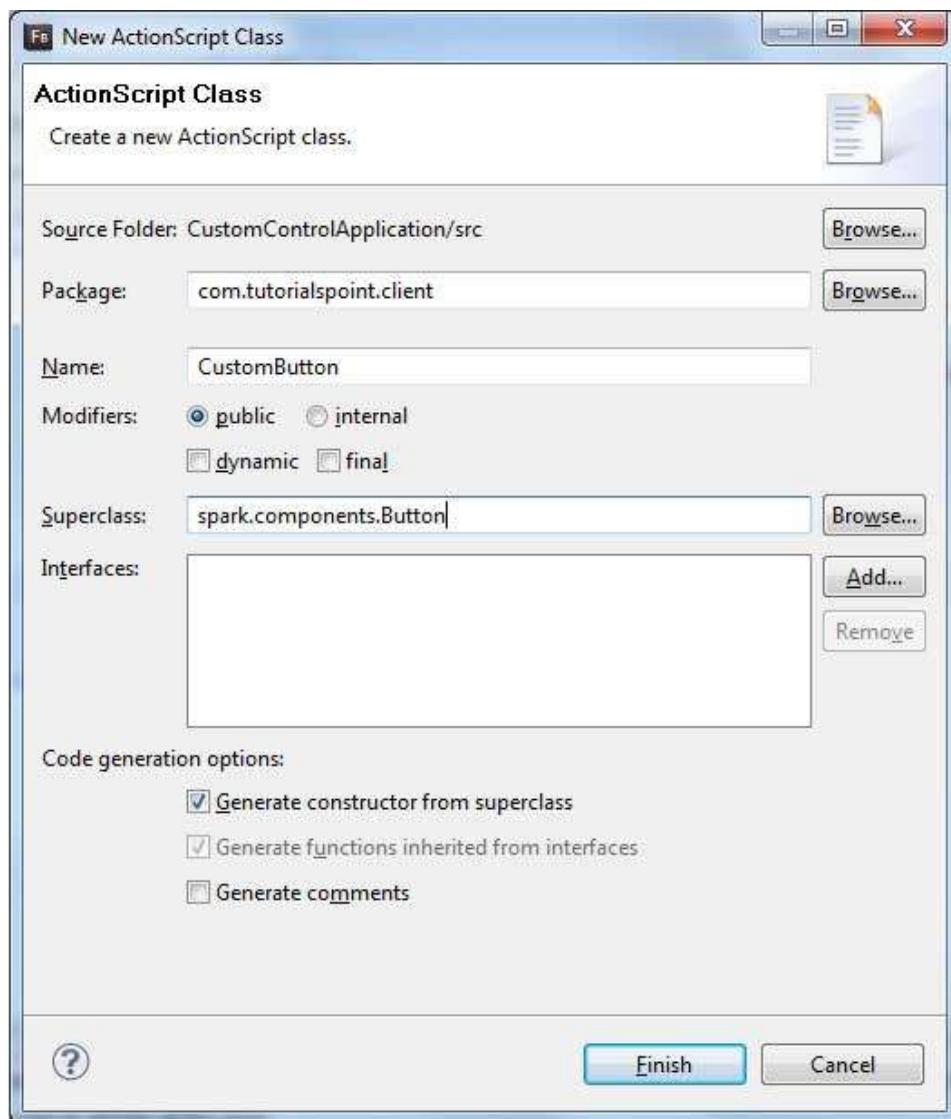
Flex provides two ways to create custom components.

- Using ActionScript
- Using MXML

## Using ActionScript

You can create a component by extending existing component. To create a component using Flash Builder, Click on **File > New > ActionScript Class**.

Enter the details as shown below:



Flash Builder will create the following CustomButton.as file.

```

package com.tutorialspoint.client

{
    import spark.components.Button;

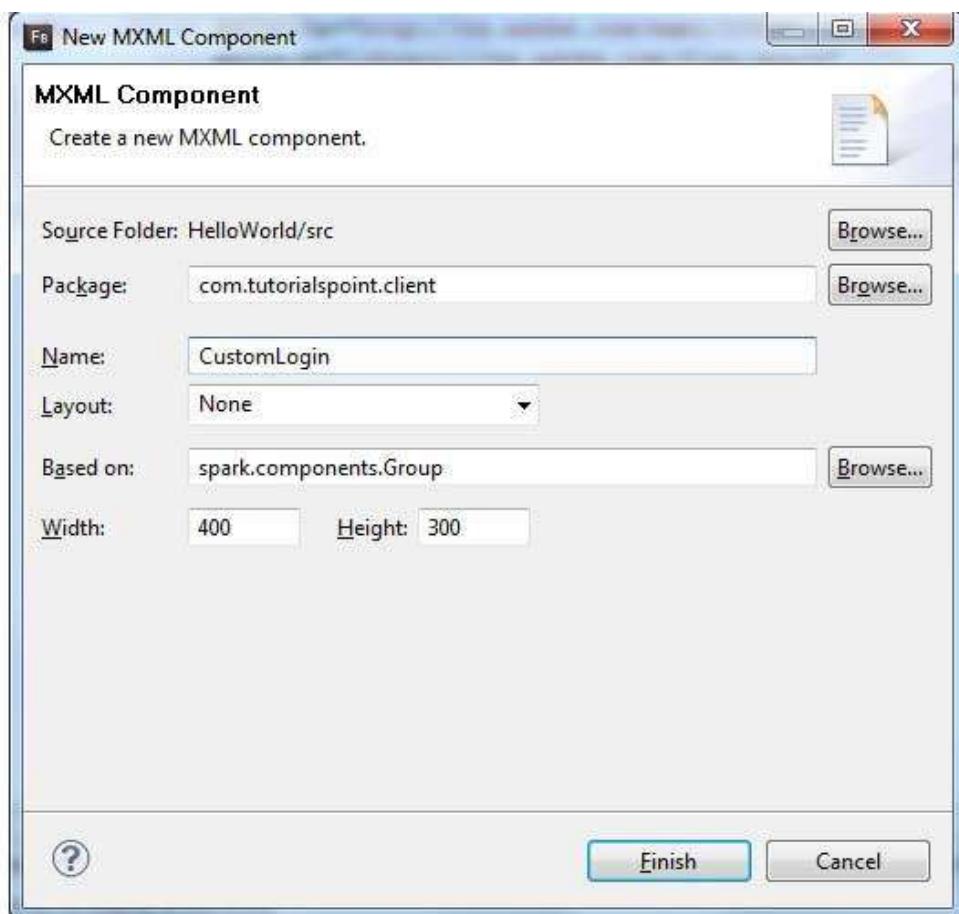
    public class CustomButton extends Button
    {
        public function CustomButton()
        {
            super();
        }
    }
}

```

## Using MXML

You can create a component by extending existing component. To create a component using Flash Builder, Click on **File > New > MXML Component**.

Enter the details as shown below.



Flash Builder will create the following CustomLogin.mxml file.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Group xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="400" height="300">

</s:Group>
```

Let us follow the following steps to test custom controls in a Flex application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
4	Create <i>CustomLogin.mxml</i> and <i>CustomButton.as</i> component as explained above. Modify these files as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/client/CustomLogin.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Group xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx" width="400" height="300">
    <s:Form>
        <s:FormItem label="UserName:>
            <s:TextInput width="200" />
        </s:FormItem>
        <s:FormItem label="Password:>
            <s:TextInput width="200"
                displayAsPassword="true" />
        </s:FormItem>
        <s:FormItem>
            <s:Button label="Login" />
        </s:FormItem>
    </s:Form>
</s:Group>
```

```
</s:Form>
</s:Group>
```

Following is the content of the modified mxml file **src/com.tutorialspoint/client/CustomButton.as**.

```
package com.tutorialspoint.client
{
    import spark.components.Button;

    public class CustomButton extends Button
    {
        public function CustomButton()
        {
            super();
            this.setStyle("color","green");
            this.label = "Submit";
        }
    }
}
```

Following is the content of the modified mxml file **src/com.tutorialspoint/client/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    xmlns:client="com.tutorialspoint.client.*"
    initialize="application_initializeHandler(event)"
    >
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            protected function application_initializeHandler(event:FlexEvent):void
            {
                //create a new custom button
                var customButton: CustomButton = new CustomButton();
        
```

```

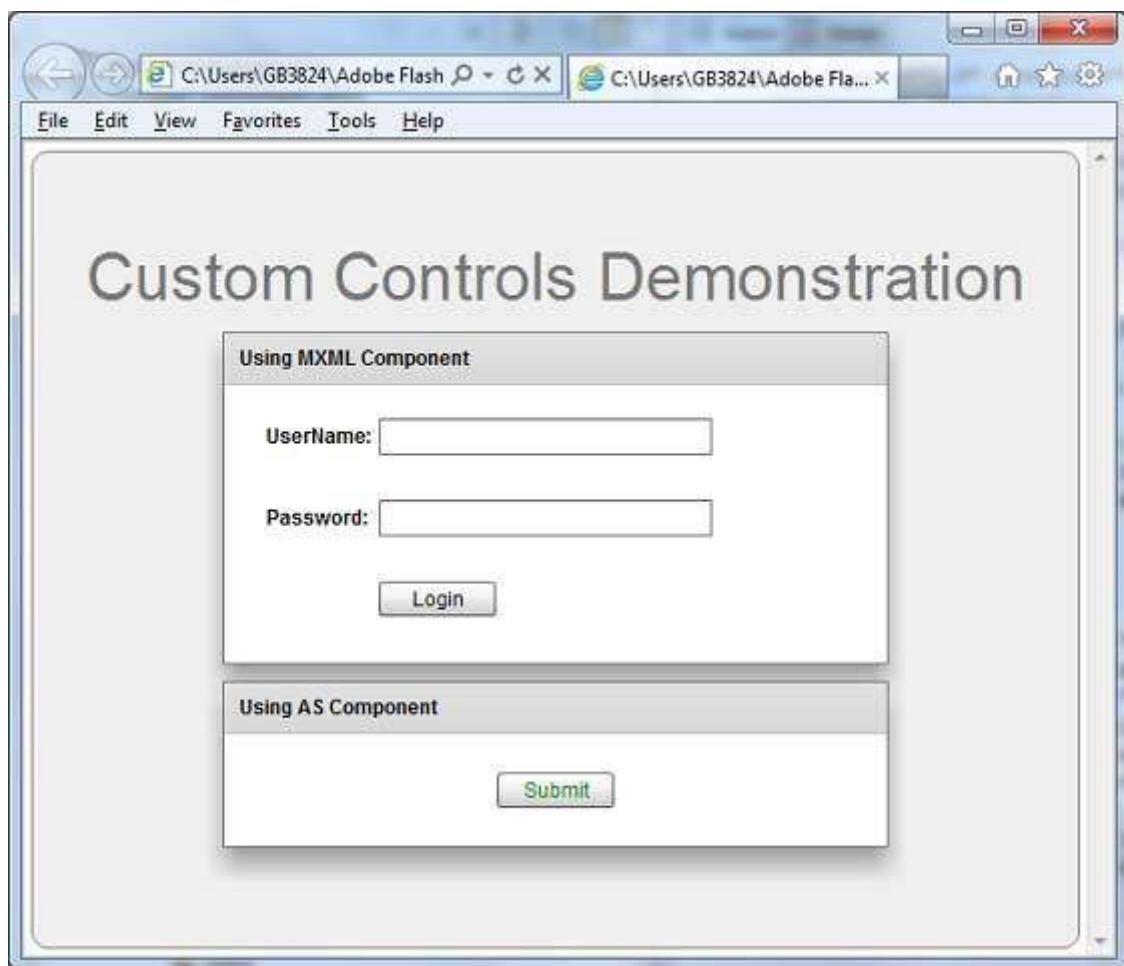
        asPanel.addElement(customButton);
    }

]]>
</fx:Script>
<s:BorderContainer width="630" height="480" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="10"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" text="Custom Controls Demonstration"
            fontSize="40" color="0x777777" styleName="heading"/>

        <s:Panel title="Using MXML Component" width="400" height="200">
            <client:CustomLogin>
            </client:CustomLogin>
        </s:Panel>
        <s:Panel title="Using AS Component" width="400" height="100">
            <s:VGroup id="asPanel" width="100%" height="100%" gap="10"
                horizontalAlign="center" verticalAlign="middle">
                </s:VGroup>
            </s:Panel>
        </s:VGroup>
    </s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, then it will produce the following result:



# 17. Flex – RPC Services

Flex provides RPC services to provide server side data to client side. Flex provides a fair amount of control on to server side data.

- Using Flex RPC services, we can define user actions to be executed on server side.
- Flex RPC Services can be integrated with any server side technologies.
- One of Flex RPC Service provide inbuilt support for compressed binary data to be transferred over the wire and is pretty fast.

Flex provides the following three types of RPC Services

RPC Service	Description
HttpService	<mx:HTTPService> tag is used to represent an HTTPService object in an MXML file. When you make a call to HTTPService object's send() method, it makes an HTTP request to the specified URL, and an HTTP response is returned. You can also use the HTTP HEAD, OPTIONS, TRACE, and DELETE methods.
WebService	The <mx:WebService> tag is used to get access to the operations of SOAP-compliant web services.
RemoteObject	The <mx:RemoteObject> tag is used to represent an HTTPService object in an MXML file. This tag gives you access to the methods of Java objects using Action Message Format (AMF) encoding.

We're going to discuss HTTP Service in detail. We'll use an XML source file placed at server and access it at client side via HTTP Service

## Items.xml

```
<items>
    <item name="Book" description="History of France"></item>
    <item name="Pen" description="Parker Pen"></item>
    <item name="Pencil" description="Stationary"></item>
<items>
```

## HTTPService Declaration

Now declare a HTTPService and pass it url of the above file

```
<fx:Declarations>
    <mx:HTTPService id="itemRequest"
        url="http://www.tutorialspoint.com/flex/Items.xml" />
```

```
</fx:Declarations>
```

## RPC Call

Make a call to itemRequest.send() method and bind values from lastResult object of itemRequest webservice to Flex UI component.

```
...
itemRequest.send();
...
<mx:DataGrid id="dgItems" height="80%" width="75%" dataProvider="{itemRequest.lastResult.items.item}">
<mx:columns>
    <mx:DataGridColumn headerText="Name" dataField="name"/>
    <mx:DataGridColumn headerText="Description" dataField="description"/>
</mx:columns>
</mx:DataGrid>
```

## RPC Service Call Example

Now, Let us follow the steps to test RPC services in a Flex application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file: **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    minWidth="500" minHeight="500" creationComplete="init(event)">
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
```

```

import mx.events.FlexEvent;
import mx.rpc.events.FaultEvent;
import mx.rpc.events.ResultEvent;

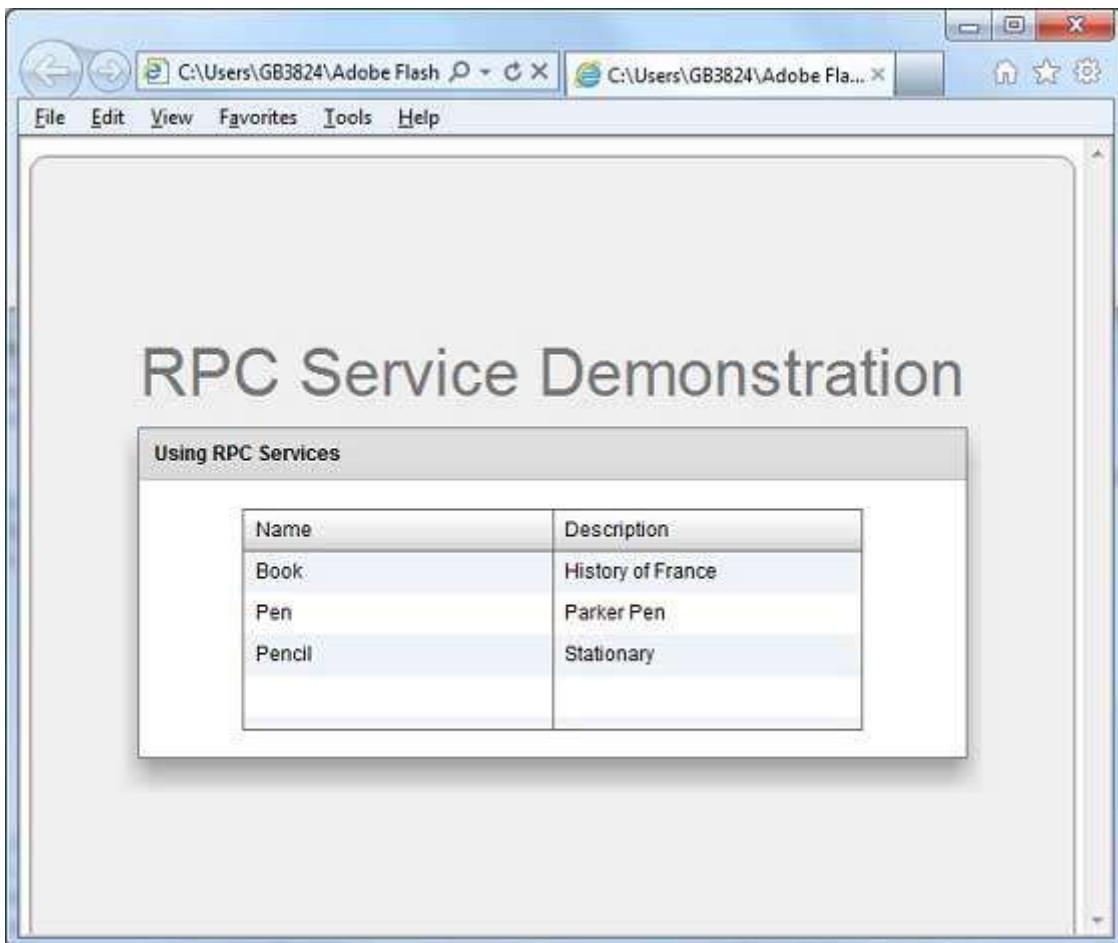
protected function init(event:FlexEvent):void
{
    itemRequest.send();
}

]]>
</fx:Script>
<fx:Declarations>
<mx:HTTPService id="itemRequest"
url="http://www.tutorialspoint.com/flex/Items.xml" />
</fx:Declarations>
<s:BorderContainer width="630" height="480" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="10"
horizontalAlign="center" verticalAlign="middle">
<s:Label id="lblHeader" text="RPC Service Demonstration"
fontSize="40" color="0x777777" styleName="heading"/>
<s:Panel id="parentPanel" title="Using RPC Services"
width="500" height="200" >
<s:layout>
<s:VerticalLayout gap="10"
verticalAlign="middle" horizontalAlign="center"/>
</s:layout>
<mx:DataGrid id="dgItems" height="80%" width="75%"
dataProvider="{itemRequest.lastResult.items.item}">
<mx:columns>
<mx:DataGridColumn headerText="Name"
dataField="name"/>
<mx:DataGridColumn headerText="Description"
dataField="description"/>
</mx:columns>
</mx:DataGrid>
</s:Panel>

```

```
</s:VGroup>  
</s:BorderContainer>  
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



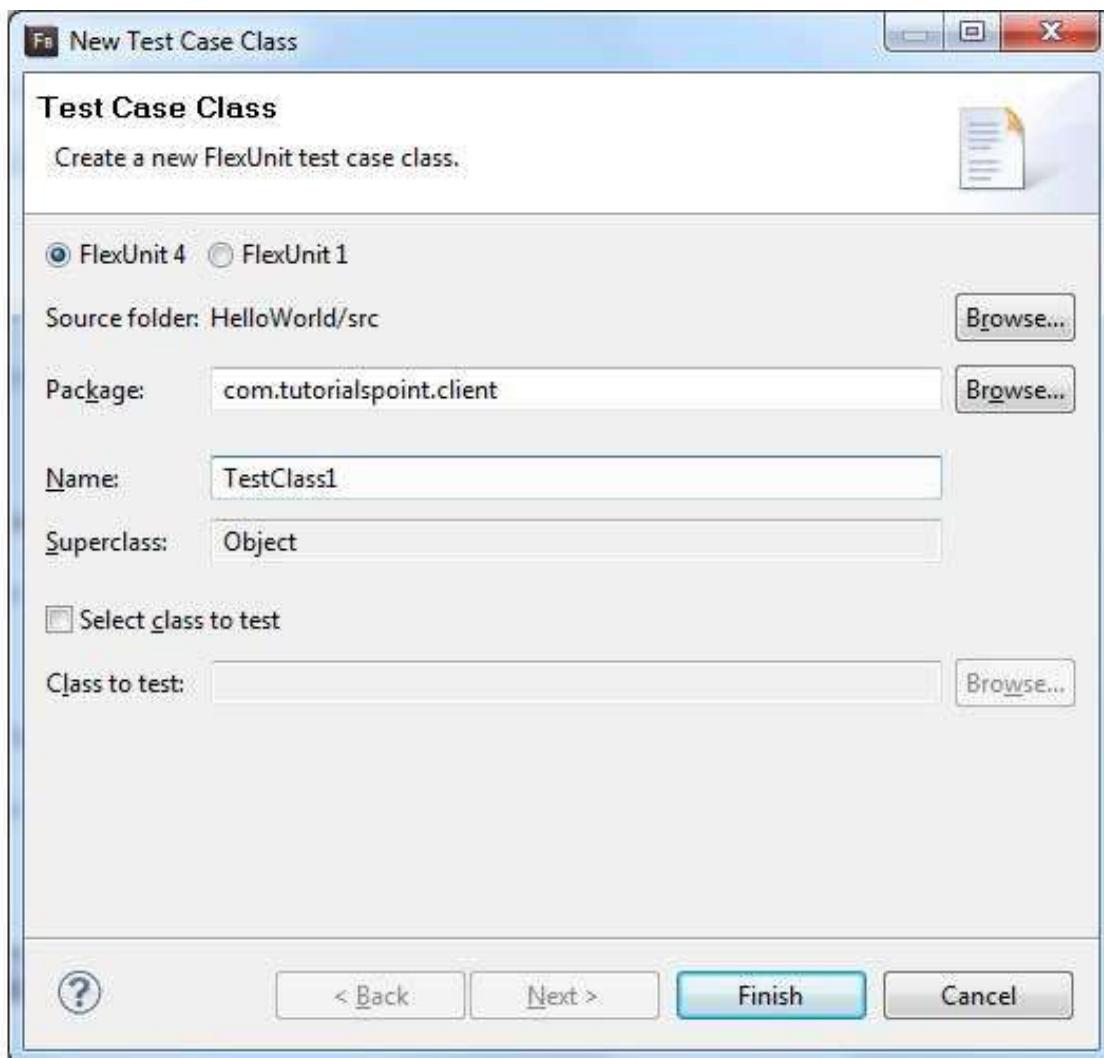
# 18. Flex – FlexUnit Integration

Flash Builder 4 has an excellent inbuilt support for FlexUnit integration in Flex development Cycle.

## Create a Test Case Class

You can create a Test Case Class using Flash Builder Create Test Class wizard. Running test cases is a breeze with Flash Builder as you will see in this article.

To create a test case class using Flash Builder, Click on **File > New > Test Case Class**. Enter the details as shown below.



Flash Builder will create the following TestClass1.as a file.

```
package com.tutorialspoint.client
```

```
{
public class TestClass1
{
    [Before]
    public function setUp():void {}

    [After]
    public function tearDown():void {}

    [BeforeClass]
    public static function setUpBeforeClass():void {}

    [AfterClass]
    public static function tearDownAfterClass():void {}
}
}
```

## Flex Unit Integration Example

Now, let us follow the steps to test FlexUnit Integration in a Flex application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Create <i>TestClass1.as</i> test case as described above and Modify <i>TestClass1.as</i> as explained below.
4	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified as file **src/com.tutorialspoint/client/TestClass1.as**.

```
package com.tutorialspoint.client
{
    import org.flexunit.asserts.assertEquals;

    public class TestClass1
```

```

{
    private var counter: int = 1;

    [Before]
    public function setUp():void
    {
        //this code will run before every test case execution
    }

    [After]
    public function tearDown():void
    {
        //this code will run after every test case execution
    }

    [BeforeClass]
    public static function setUpBeforeClass():void
    {
        //this code will run once when test cases start execution
    }

    [AfterClass]
    public static function tearDownAfterClass():void
    {
        //this code will run once when test cases ends execution
    }

    [Test]
    public function testCounter():void {
        assertEquals(counter, 1);
    }
}

```

Following is the content of the modified mxm file **src/com.tutorialspoint/HelloWorld.mxml**.

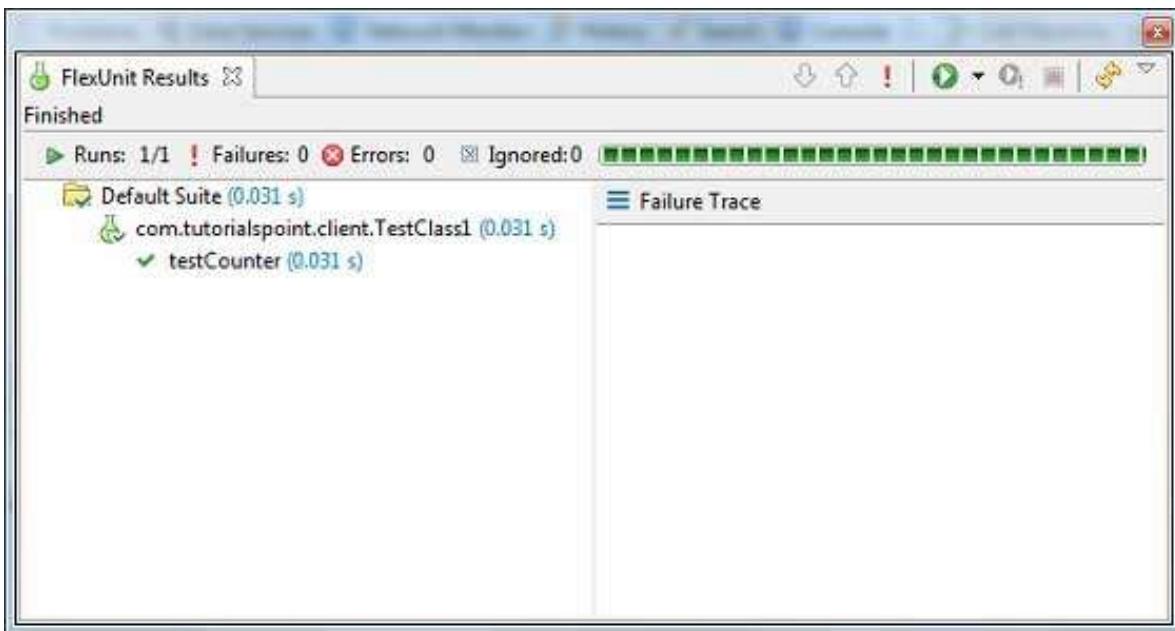
```
<?xml version="1.0" encoding="utf-8"?>
```

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    minWidth="500" minHeight="500">
</s:Application>
```

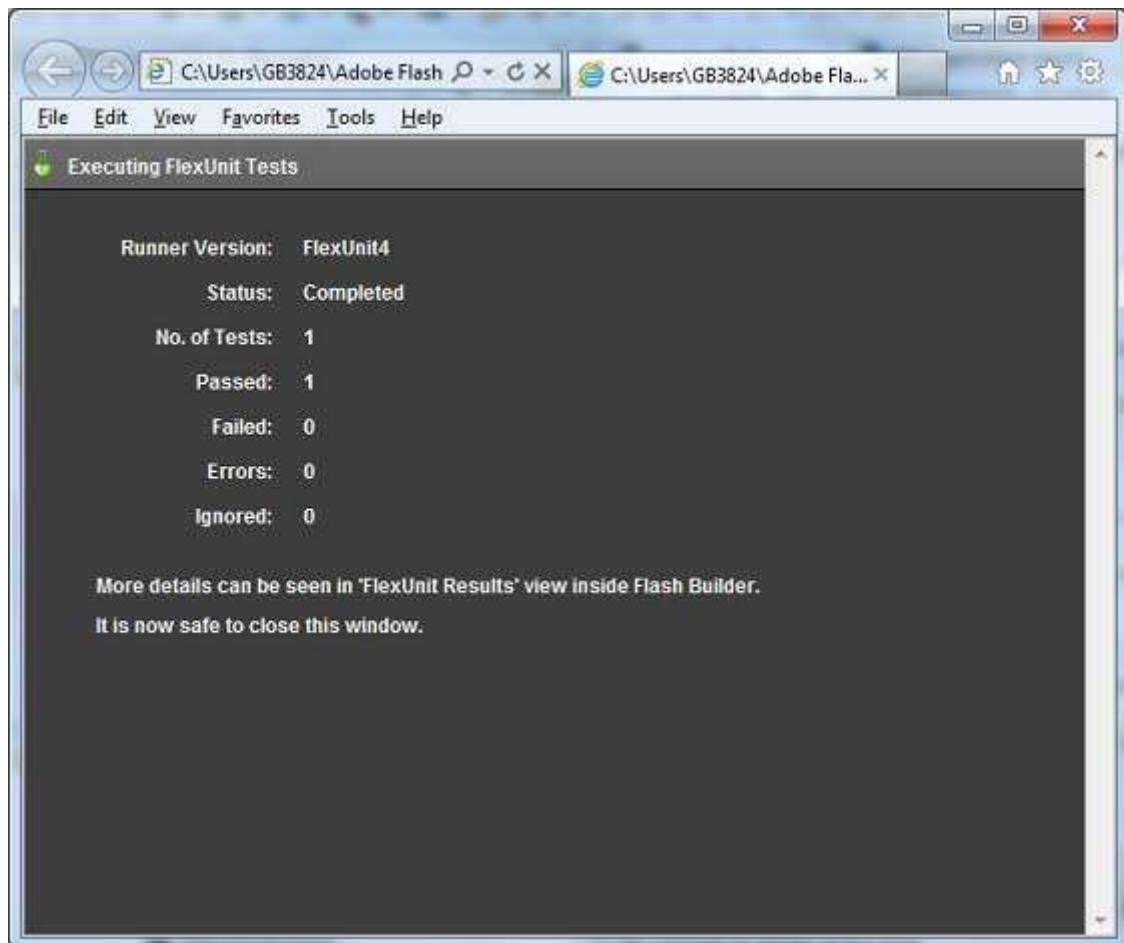
Once you are ready with all the changes done, let us compile in normal mode as we did in [Flex - Create Application](#) chapter.

## Running Test cases

Now Right Click on **TestClass1** in package explorer and select **Run As > FlexUnit Tests**. You'll see the following output in Flash Builder test window.



Flash Builder also shows test results in the browser.



# 19. Flex – Debug Application

Flex provides excellent capability of debugging flex code and Flash Builder 4 has an excellent built-in debugger and debugging perspective support.

- During debug mode, Flex Application runs on Flash Player Debugger version built in Flash Builder 4 which supports debugging capability.
- So developers get an easy and inbuilt debugging configuration in Flash Builder

In this article, we'll demonstrate usage of debugging Flex Client code using Flash Builder. We'll do the following tasks

- Set break points in the code and see them in Breakpoint Explorer.
- Step through the code line by line during debugging.
- View the values of variable.
- Inspect the values of all the variables.
- Inspect the value of an expression.
- Display the stack frame for suspended threads.

## Debugging Example

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%"
    minWidth="500" minHeight="500"
    initialize="application_initializeHandler(event)">
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
```

```

import mx.controls.Alert;
import mx.events.FlexEvent;
protected function btnClickMe_clickHandler(event:MouseEvent):void
{
    Alert.show("Hello World!");
}

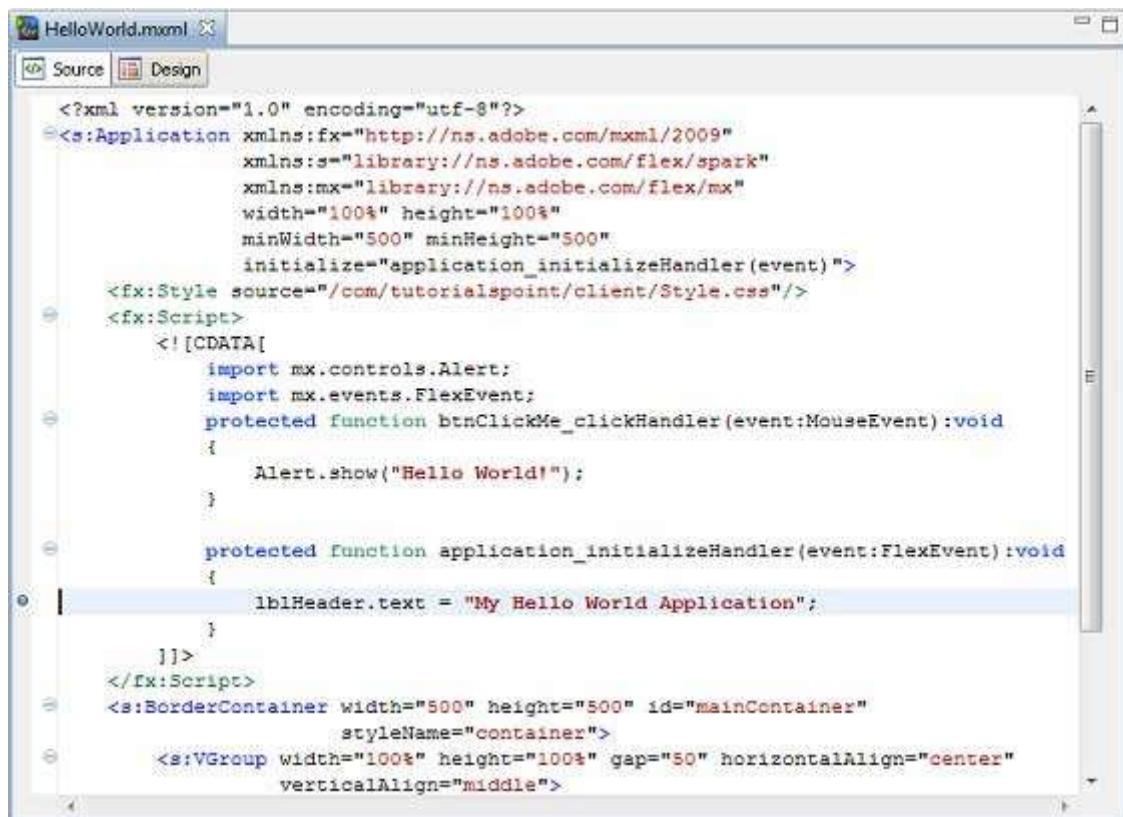
protected function application_initializeHandler(event:FlexEvent):void
{
    lblHeader.text = "My Hello World Application";
}
]]>
</fx:Script>
<s:BorderContainer width="500" height="500" id="mainContainer"
styleName="container">
<s:VGroup width="100%" height="100%" gap="50" horizontalAlign="center"
verticalAlign="middle">
<s:Label id="lblHeader" fontSize="40" color="0x777777"
styleName="heading"/>
<s:Button label="Click Me!" id="btnClickMe"
click="btnClickMe_clickHandler(event)" styleName="button" />
</s:VGroup>
</s:BorderContainer>
</s:Application>

```

Once you are ready with all the changes done, let us compile in normal mode as we did in [Flex - Create Application](#) chapter.

## Step 1 - Place Breakpoints

Place a breakpoint on the first line of application initialize Handler of HelloWorld.mxml



```

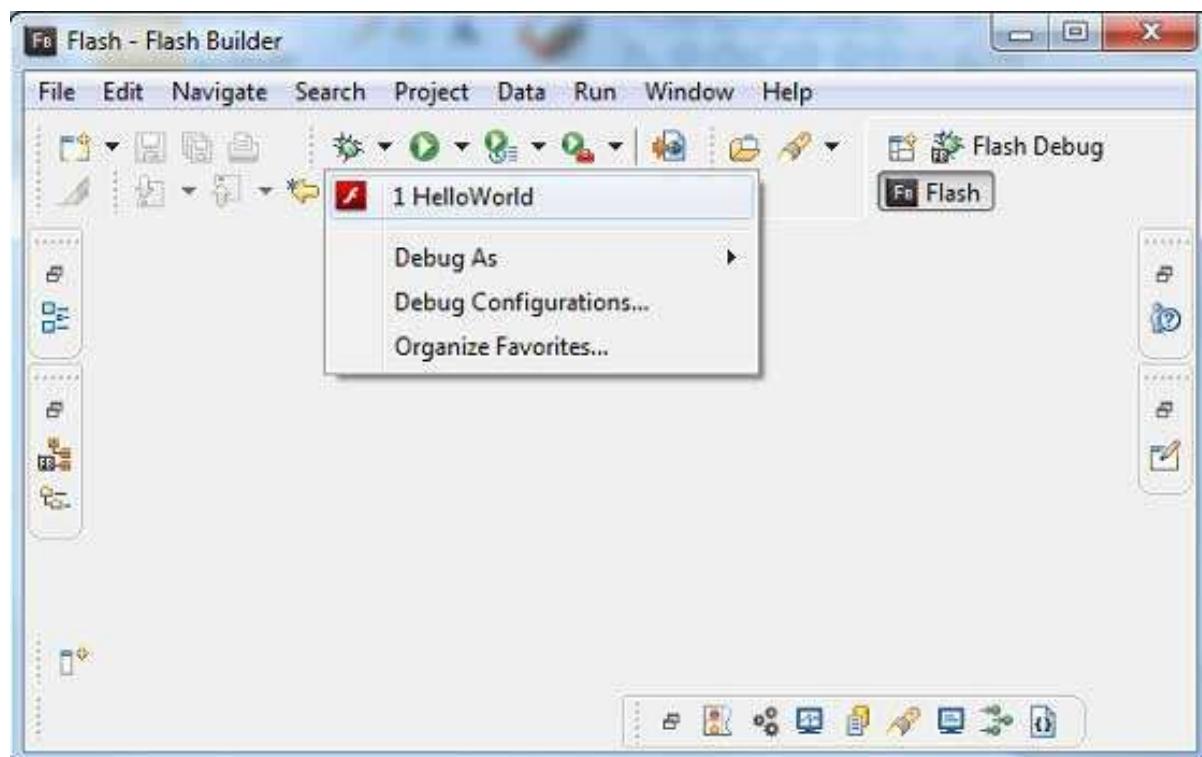
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%"
    minWidth="500" minHeight="500"
    initialize="application_initializeHandler(event)">
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<fx:Script>
    <![CDATA[
        import mx.controls.Alert;
        import mx.events.FlexEvent;
        protected function btnClickMe_clickHandler(event:MouseEvent):void
        {
            Alert.show("Hello World!");
        }

        protected function application_initializeHandler(event:FlexEvent):void
        {
            lblHeader.text = "My Hello World Application";
        }
    ]]>
</fx:Script>
<s:BorderContainer width="500" height="500" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50" horizontalAlign="center"
        verticalAlign="middle">

```

## Step 2 - Debug Application

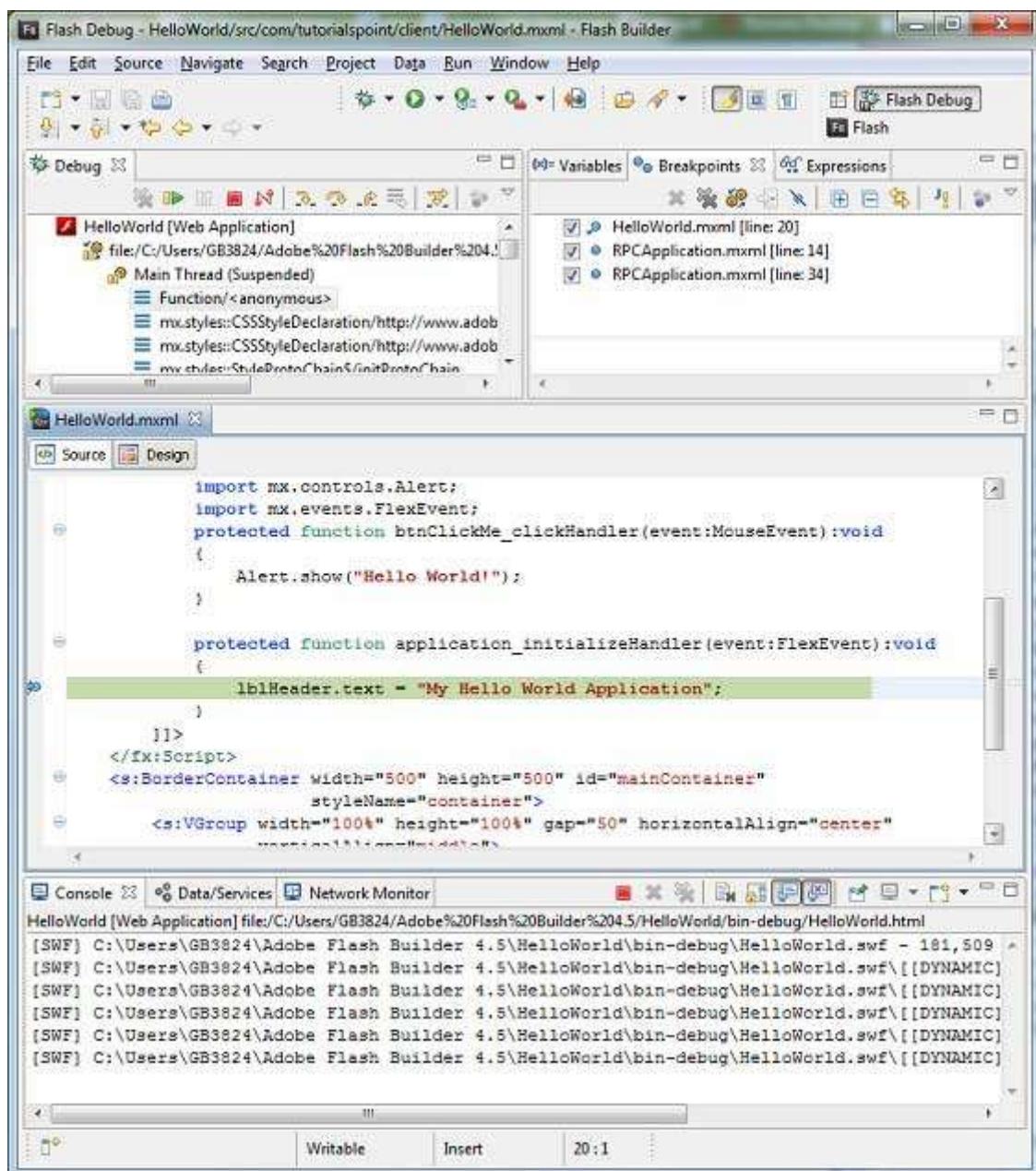
Now click on  Debug application menu and select **HelloWorld** application to debug the application.



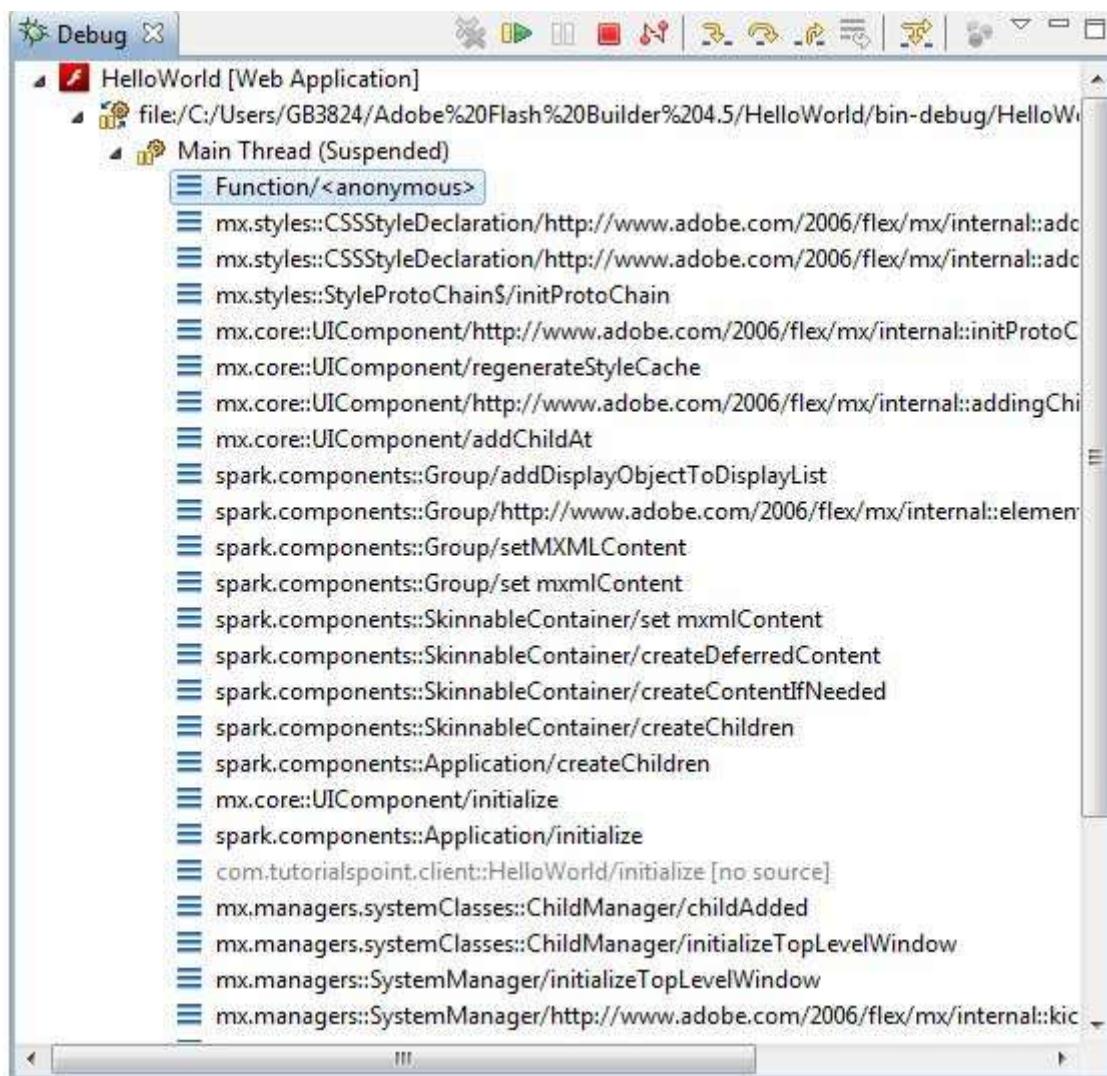
If everything is fine, application will launch in the browser and you will see following debug logs in Flash Builder console

```
[SWF] \HelloWorld\bin-debug\HelloWorld.swf
- 181,509 bytes after decompression
[SWF] \HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]]\1
- 763,122 bytes after decompression
[SWF] \HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]]\2
- 1,221,837 bytes after decompression
[SWF] \HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]]\3
- 1,136,788 bytes after decompression
[SWF] \HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]]\4
- 2,019,570 bytes after decompression
[SWF] \HelloWorld\bin-debug\HelloWorld.swf\[[DYNAMIC]]\5
- 318,334 bytes after decompression
```

As soon as Application launches, you will see the focus on Flash Builder breakpoint as we've placed the breakpoint on first line of application\_initialize Handler method.



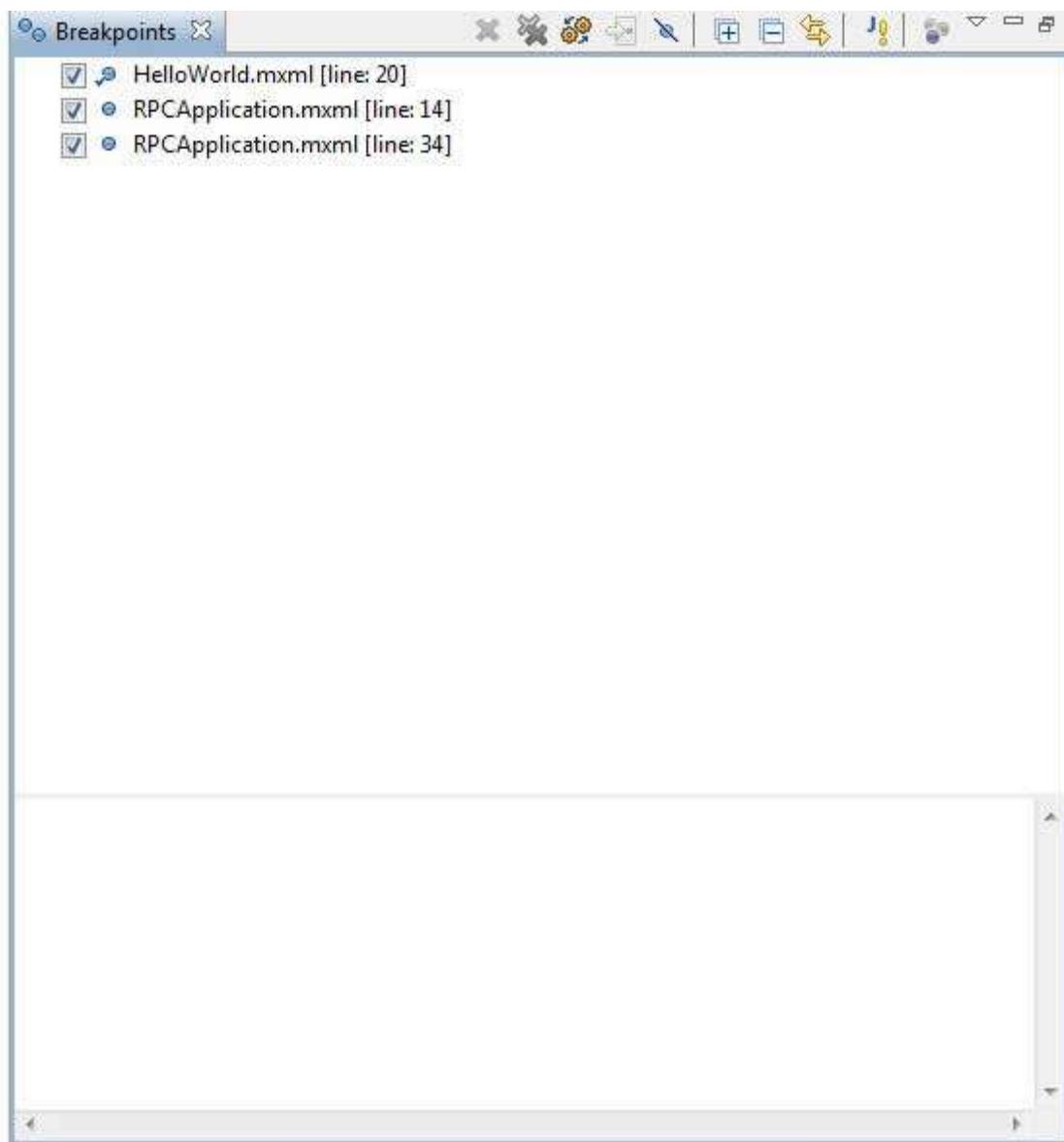
You can see the stacktrace for suspended threads.



You can see the values for expressions.

Name	Value
<code>="? "lblHeader.text"</code>	My Hello World Application
<code>="? "lblHeader"</code>	spark.components.Label (...)
<code>_alpha</code>	1
<code>_blendMode</code>	"normal"
<code>_blendShader</code>	<setter>
<code>_height</code>	0
<code>_parent</code>	spark.components.VGrou...
<code>_scaleX</code>	1
<code>_scaleY</code>	1
<code>_transform</code>	flash.geom.Transform (@...)
<code>_visible</code>	false
<code>_width</code>	0
<code>_x</code>	0
<code>_y</code>	0
<code>accessibilityDescription</code>	""
<code>accessibilityEnabled</code>	true
<code>accessibilityImpl</code>	spark.accessibility.TextBas...
<code>accessibilityName</code>	""
<code>accessibilityProperties</code>	flash.accessibility.Accessi...
<code>accessibilityShortcuts</code>	""
<code>activeEffects</code>	[] (@cc064c1)
<code>advanceStyleClient</code>	null
<code>_affectedProperties</code>	Object (@cc3d1a9)
<code>alpha</code>	1
<code>_alpha</code>	1
<code>automaticRadioButton</code>	null
<code>automationDelegates</code>	null
<code>_automationDelegates</code>	null
<code>automationEnabled</code>	true
<code>automationName</code>	""

You can see the list of breakpoints placed.



Now keep pressing F6 until you reach the last line of application\_initializeHandler() method. As reference for function keys, F6 inspects code line by line, F5 steps inside further and F8 will resume the application. Now you can see the list of values of all variables of application\_initializeHandler() method.

Name	Value
↳ this	com.tutorialspoint.client.HelloWorld (@c9880a1)
↳ [inherited]	
↳ _328988860btnClickMe	spark.components.Button (@ccbf0a1)
↳ _530004280mainContainer	spark.components.BorderContainer (@cc160a1)
↳ _583847907lblHeader	spark.components.Label (@ccb60b1)
↳ btnClickMe	spark.components.Button (@ccbfoa1)
↳ lblHeader	spark.components.Label (@ccb60b1)
↳ [inherited]	
↳ elementFormat	null
↳ embeddedFontContext	null
↳ mainContainer	spark.components.BorderContainer (@cc160a1)
↳ [inherited]	
↳ backgroundFill	null
↳ _backgroundFill	null
↳ borderStroke	null
↳ _borderStroke	null
↳ _moduleFactoryInitialized	true
↳ event	mx.events.FlexEvent (@cd40539)
 spark.components.Label (@ccb60b1)	

Now you can see the flex code can be debugged in the same way as a Java Application can be debugged. Place breakpoints to any line and play with debugging capabilities of flex.

# 20. Flex – Internationalization

Flex provides two ways to internationalize a Flex application, We'll demonstrate use of Compile time Internationalization being most commonly used among projects.

Technique	Description
Compile Time Internationalization	This technique is most prevalent and requires very little overhead at runtime; is a very efficient technique for translating both constant and parameterized strings; simplest to implement. Compile Time internationalization uses standard properties files to store translated strings and parameterized messages, and these properties files are compiled directly in the application.
Run Time Internationalization	This technique is very flexible but slower than static string internationalization. You need to compile the localization properties files separately, leave them external to application, and load them at run time.

## Workflow of internationalizing a Flex Application

---

### Step 1 – Create folder structure

Create a locale folder under src folder of Flex project. This will be the parent directory for all of the properties files for the locales that the application will support. Inside the locale folder, create subfolders, one for each of the application's locales to be supported. The convention for naming a locale is

```
{language}_{country code}
```

For example, en\_US represents English of the United States. The locale de\_DE represents German. The sample application will support two common languages: English, and German

### Step 2 – Create properties files

Create properties file containing the messages to be used in the application. We've created a **HelloWorldMessages.properties** file under **src > locale > en\_US** folder in our example.

```
enterName=Enter your name  
clickMe=Click Me  
applicationTitle=Application Internationalization Demonstration  
greeting>Hello {0}
```

Create properties files containing translated values specific to locale. We've created a **HelloWorldMessages.properties** file under **src > locale > de\_DE** folder in our

example. This file contains translations in german language. \_de specifies the german locale and we're going to support german language in our application.

If you are creating properties file using Flash Builder then change the encoding of the file to UTF-8. Select the file and then right-click in it to open its properties window. Select Text file encoding as **Other UTF-8**. Apply and Save the change.

```
enterName=Geben Sie Ihren Namen
clickMe=Klick mich
applicationTitle=Anwendung Internationalisierung Demonstration
greeting=Hallo {0}
```

### Step 3 – Specify Compiler options

- Right-click your project and select Properties.
- Select Flex Compiler, and add the following to the Additional Compiler Arguments settings:

```
-locale en_US de_DE
```

- Right-click your project and select Properties.
- Select Flex Build Path, and add the following to the Source Path settings:

```
src\locale\{locale}
```

### Internalization Example

Now Let us follow the following steps to test Internalization technique in a Flex application:

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the Flex - Create Application chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxml file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    minWidth="500" minHeight="500">
```

```

<fx:Metadata>
    [ResourceBundle("HelloWorldMessages")]
</fx:Metadata>
<fx:Style source="/com/tutorialspoint/client/Style.css"/>
<fx:Script>
    <![CDATA[
        import mx.controls.Alert;
        [Bindable]
        private var locales:Array = [{label:"English", locale:"en_US"}, {label:"German", locale:"de_DE"}];

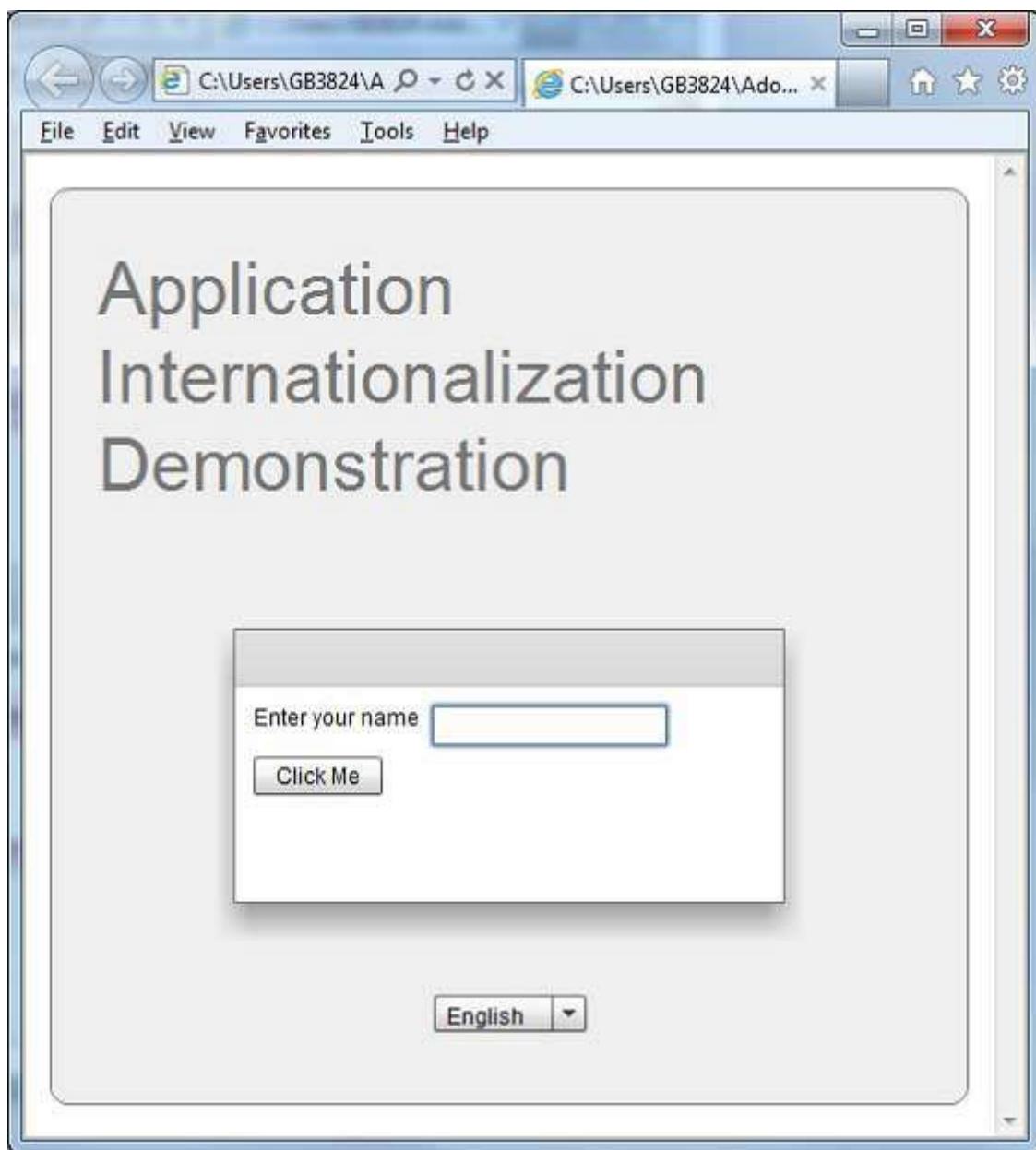
        private function comboChangeHandler():void
        {
            resourceManager.localeChain = [localeComboBox.selectedItem.locale];
        }

        protected function clickMe_clickHandler(event:MouseEvent):void
        {
            var name:String = txtName.text;
            var inputArray:Array = new Array();
            inputArray.push(name);
            Alert.show(resourceManager.getString('HelloWorldMessages','greeting',inputArray));
        }
    ]]
</fx:Script>
<s:BorderContainer width="500" height="500" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center" verticalAlign="middle">
        <s:Label id="lblHeader" fontSize="40"
            color="0x777777"
            text ="{resourceManager.getString('HelloWorldMessages','applicationTitle')}"
            styleName="heading" width="90%" height="150"/>
        <s:Panel width="300" height="150">
            <s:layout>

```

```
<s:VerticalLayout paddingTop="10" paddingLeft="10" />
</s:layout>
<s:HGroup >
  <s:Label
    text="{resourceManager.getString('HelloWorldMessages'
      , 'enterName')}"
    paddingTop="2"/>
  <s:TextInput id="txtName"/>
</s:HGroup>
<s:Button
  label="{resourceManager.getString('HelloWorldMessages','clickMe')}"
  click="clickMe_clickHandler(event)" right="10" />
</s:Panel>
<mx:ComboBox id="localeComboBox"
  dataProvider="{locales}" change="comboChangeHandler()"/>
</s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



Change the language using language drop down and see the result.



# 21. Flex – Printing Support

Flex provides a special class **FlexPrintJob** to print flex objects.

- FlexPrintJob can be used to print one or more Flex objects, such as a Form or VBox container.
- FlexPrintJob prints the object and all objects that it contains.
- The objects can be all or part of the displayed interface.
- The objects can be components that format data specifically for printing.
- The FlexPrintJob class lets you scale the output to fit the page.
- The FlexPrintJob class automatically uses multiple pages to print an object that does not fit on a single page.
- The FlexPrintJob class causes the operating system to display a Print dialog box. You cannot print without some user action.

## Prepare and Send a Print Job

You print output by preparing and sending a print job. Let's create an instance of the FlexPrintJob class

```
var printJob:FlexPrintJob = new FlexPrintJob();
```

Start the print job

```
printJob.start();
```

Flex will cause the operating system to display a Print dialog box. Add one or more objects to the print job and specify how to scale them

```
printJob.addObject(myObject, FlexPrintJobScaleType.MATCH_WIDTH);
```

Each object starts on a new page. Send the print job to the printer

```
printJob.send();
```

## Printing Example

Step	Description
1	Create a project with a name <i>HelloWorld</i> under a package <i>com.tutorialspoint.client</i> as explained in the <i>Flex - Create Application</i> chapter.
2	Modify <i>HelloWorld.mxml</i> as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.

Following is the content of the modified mxm file **src/com.tutorialspoint/HelloWorld.mxml**.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    width="100%" height="100%"
    minWidth="500" minHeight="500"
    initialize="application_initializeHandler(event)">
    <fx:Style source="/com/tutorialspoint/client/Style.css"/>
    <fx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.events.FlexEvent;
            import mx.printing.FlexPrintJob;
            import mx.printing.FlexPrintJobScaleType;
            protected function btnClickMe_clickHandler(event:MouseEvent):void
            {
                // Create an instance of the FlexPrintJob class.
                var printJob:FlexPrintJob = new FlexPrintJob();

                // Start the print job.
                if (printJob.start() != true) return;

                // Add the object to print. Do not scale it.
                printJob.addObject(myDataGrid, FlexPrintJobScaleType.NONE);

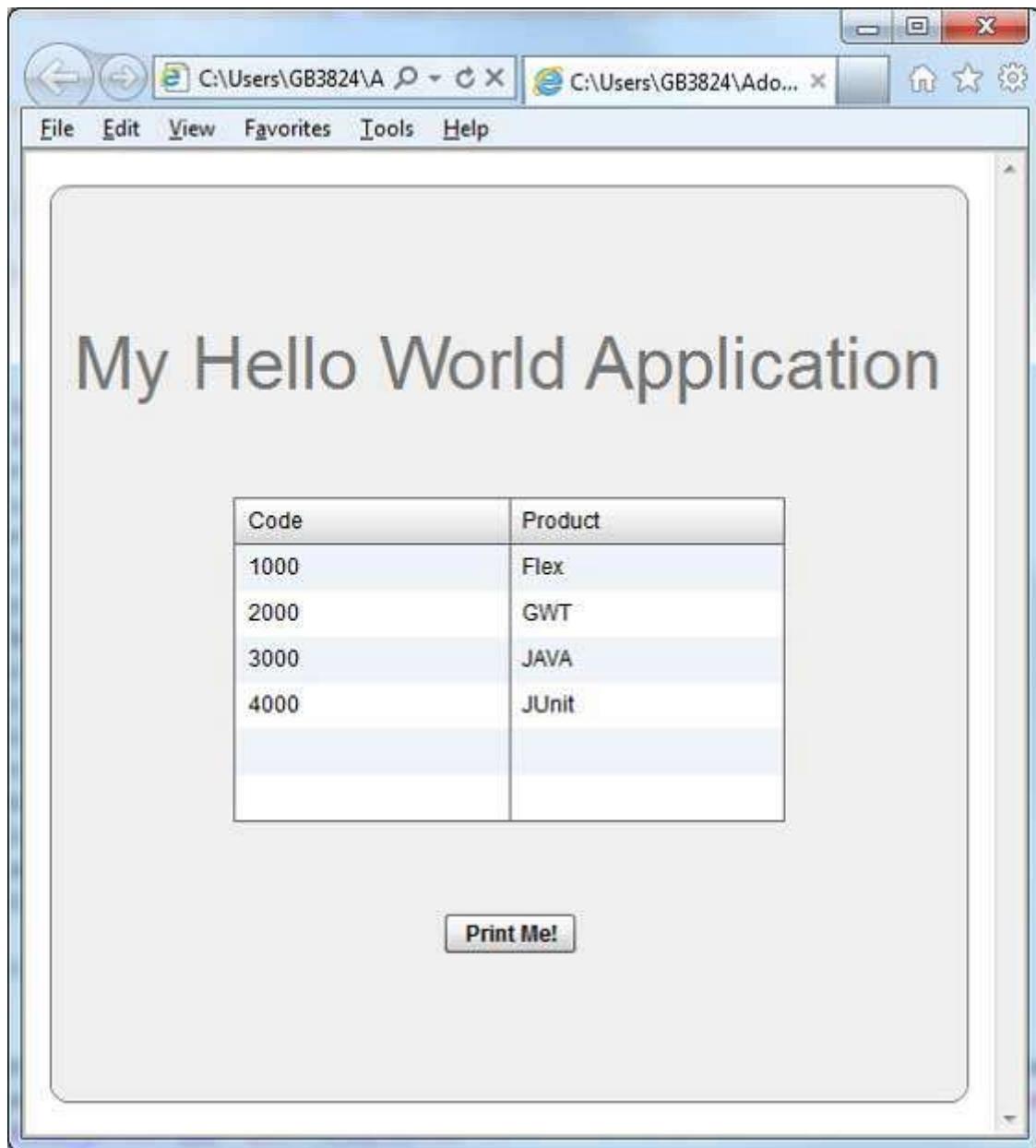
                // Send the job to the printer.
                printJob.send();
            }

            protected function application_initializeHandler(event:FlexEvent):void
            {
                lblHeader.text = "My Hello World Application";
            }
        ]]>
```

```
</fx:Script>

<s:BorderContainer width="500" height="500" id="mainContainer"
    styleName="container">
    <s:VGroup width="100%" height="100%" gap="50"
        horizontalAlign="center"
        verticalAlign="middle">
        <s:Label id="lblHeader" fontSize="40" color="0x777777"
            styleName="heading"/>
        <mx:DataGrid id="myDataGrid" width="300">
            <mx:dataProvider>
                <fx:Object Product="Flex" Code="1000"/>
                <fx:Object Product="GWT" Code="2000"/>
                <fx:Object Product="JAVA" Code="3000"/>
                <fx:Object Product="JUnit" Code="4000"/>
            </mx:dataProvider>
        </mx:DataGrid>
        <s:Button label="Print Me!" id="btnClickMe"
            click="btnClickMe_clickHandler(event)"
            styleName="button" />
    </s:VGroup>
</s:BorderContainer>
</s:Application>
```

Once you are ready with all the changes done, let us compile and run the application in normal mode as we did in [Flex - Create Application](#) chapter. If everything is fine with your application, it will produce the following result:



Click on print me button and you can see the printout of the data grid shown below.

