



An Article on Advanced Flex Techniques  
[By: Flex Team METAOPTION]

## **Advanced Flex**

A rich Internet application combines the benefits of using the Web as a low-cost deployment model with a rich user experience that's at least as good as today's desktop applications. And, since RIAs don't require that the entire page be refreshed to update their data, the response time is much faster and the network load much lower. Think of a globally available client/server application.

Rich Internet applications eliminate the roundtrips and substantially improve system performance

By doing a lot more of the processing on the client than a thin client Web application. Besides, RIAs are stateful: they accumulate the information right on the client! To put it simply, RIA isn't a set of pages controlled by the server; they are actual applications running on the client's computer and communicating with servers primarily to process and exchange data.

The major part of Advance flex is give below:

### **1: Advance Flex Programming**

- Embedding Assets
- Printing
- Creating Modular Applications
- Deep Linking
- Communicating with the Wrapper
- Using Shared Objects
- Localizing Flex Applications

#### **Embedding Assets:**

The process of compiling an asset into your application is called embedding the asset. Flex lets you embed image files, movie files, MP3 files, and TrueType fonts into your applications. Embedded assets are compiled into the SWF file of your Flex application. They are *not* loaded at run time and you do not have to deploy the original asset files with your application.

The alternative to embedding assets is to load them at run time. Assets loaded at run time have to be deployed with your application because they are not compiled into it. This has the advantage of keeping the file size of your Flex application smaller and speeding up its initial loading time.

#### **Printing:**

Adobe Flex applications allow the users print from within the application. The Flex mx.printing package contains classes that facilitate the creation of printing output from Flex applications:

[FlexPrintJob](#): A class that prints one or more objects. Automatically splits large objects for printing on multiple pages and scales the output to fit the page size.

[PrintDataGrid](#) A subclass of the DataGrid control with a default appearance that is customized for printing. The class includes properties and a method that provide additional sizing and printing features.

[PrintAdvancedDataGrid](#) A subclass of the AdvancedDataGrid control with a default appearance that is customized for printing. The class includes properties and a method that provide additional sizing and printing features.

[PrintOLAPDataGrid](#) A subclass of the OLAPDataGrid control with a default appearance that is customized for printing. The class includes properties and a method that provide additional sizing and printing features.

[FlexPrintJobScaleType](#) Defines constants used in the FlexPrintJob addObject() method.

Together, these classes give you control over how the user prints information from the application. For example, your application can print only a selected subset of the information on the screen, or it can print information that is not being displayed. Also, your application can reformat the information and optimize its layout and appearance for printing.

### **Modular Application:**

Modules are SWF files that can be loaded and unloaded by an application. They cannot be run independently of an application, but any number of applications can share the modules.

Modules let you split your application into several pieces, or modules. The main application, or shell, can dynamically load other modules that it requires, when it needs them. It does not have to load all modules when it starts, nor does it have to load any modules if the user does not interact with them. When the application no longer needs a module, it can unload the module to free up memory and resources.

Modular applications have the following benefits:

1. Smaller initial download size of the SWF file.
2. Shorter load time due to smaller SWF file size.
3. Better encapsulation of related aspects of an application.

### **Deep Linking:**

One of the benefits of a Flex application is that the application can smoothly transition from state to state without having to fetch a new page from the server and refresh the browser. By avoiding the constant refreshing of pages, the end-user's experience is more fluid and continuous. In addition, the load on the server is greatly reduced because it need only return the application once, rather than a new page every time the user changes views.

Deep linking relies on communication between the browser and the Flex application. The communication is bidirectional: if a change occurs in the application, the browser must be notified, and if a change in the browser occurs, then the application must be notified. This communication is handled by the [BrowserManager](#) class.

### **Communicating with the Wrapper:**

You exchange data between a Flex application and the HTML page that embeds that application in several ways, depending on the type of integration that your application requires.

A Flex application is loaded in a browser within a wrapper. This wrapper is often an HTML page that can include JavaScript or other client-side logic that the Flex application can interact with.

### **Shared Objects:**

Shared objects behave like cookies. You use SharedObject class to store data on the client's hard disk and retrieve those objects in the same session or in another session. Applications can access their own shared object data only. You can make your shared data object available to other application from the same domain and you cannot share with applications from other domains. Shared objects allow you to write simple objects like Arrays, String and Date. You can create multiple shared objects for one application. Each shared object is associated with a name. To add data to shared objects, you use the *data* property of the shared object. Below is a function, which displays a welcome message if the user is revisiting.

By default, Flash can save locally persistent SharedObject objects of up to 100 KB per domain. When the application tries to save data to a shared object that would make it bigger than 100 KB, Flash Player displays the Local Storage dialog box, which lets the user allow or deny local storage for the domain that is requesting access.

### **Localizing Flex Application:**

Localization is the process of including assets to support multiple locales. A locale is the combination of a language and a country code; for example, en\_US refers to the English language as spoken in the United States, and fr\_FR refers to the French language as spoken in France. To localize an application, you would provide two sets of assets, one for the en\_US locale and one for the fr\_FR locale.

Localization goes beyond just translating Strings used in your application. It can also include any type of asset such as audio files, images, and videos. Because the meanings of colors and images can vary based on the culture,

you can change the styles used by your application, in addition to the language used, based on the locale.

## 2: Flex Application Design for Cairngorm

### Overview of Cairngorm

Although this document attempts to cover the basics of planning a Cairngorm application, it does not go into any deep framework detail. For further information

and code example on Cairngorm see Steven Webster's blog, the Cairngorm Wiki on Adobe Labs, or the Yahoo! FlexCoders group.

What is Cairngorm? Cairngorm is fundamentally a methodology for breaking up your application code by logical functions; by data, by user views, and by the code that controls everything. This is routinely referred to as MVC, or Model, View, and Control.

### The Pieces of Cairngorm

- **Model Locator:** Stores all of your application's Value Objects (data) and shared variables, in one place. Similar to an HTTP Session object, except that its stored client side in the Flex interface instead of server side within a middle tier application server.
- **View:** One or more Flex components (button, panel, combo box, Tile, etc) bundled together as a named unit, bound to data in the Model Locator, and generating custom Cairngorm Events based on user interaction (clicks, rollovers, drag and drop.)
- **Front Controller:** Receives Cairngorm Events and maps them to Cairngorm Commands.
- **Command:** Handles business logic, calls Cairngorm Delegates and/or other Commands, and updates the Value Objects and variables stored in the Model Locator
- **Delegate:** Created by a Command, they instantiate remote procedure calls (HTTP, Web Services, etc) and hand the results back to that Command.
  
- **Service:** Defines the remote procedure calls (HTTP, Web Services, etc) to connect to remote data stores.

### How the Pieces Fit Together

Cairngorm basically works like this: Your client interface is comprised of Views. The Views use Flex binding to display data contained in the Model Locator. The Views generate Events based on user gestures such as mouse click, button press, and drag & drop. Those Events are "broadcast" and "heard" by the Front Controller, which is a map of Events to Commands. Commands contain business logic, create Delegates to perform work, handle responses from Delegates, and update the data stored in the Model Locator. Since Views are bound to the data in the Model Locator the Views automatically update when the Model Locator data is changed. Delegates call Services and hand results back to Commands, and are optional but recommended.

Services make remote data calls and hand the results back to Delegates. The diagram on the following page provides a simple overview of the processing flow in a Cairngorm application.

### 3: Rich Text Editor

The RichTextEditor control lets users enter and format text. The text characteristics that users can vary include the 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 direct children:

- 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.

The RichTextEditor has a default height and width of 300 by 325 pixels and a default minimum height and width of 200 by 220 pixels. If you put a RichTextEditor control in a DividedBox control, make sure that the DividedBox control is large enough to contain the RichTextEditor control at its minimum dimensions. Also, you can explicitly set the RichTextEditor control's minHeight or minWidth property to NaN to let the DividedBox container reduce the control's dimensions to 0.

### 4: Advanced DataGrid

In any GUI tool, one of the most popular components is the one that shows data in a table format like JTable in Java or DataWindow in PowerBuilder. The Adobe Flex 2 version of such a component is called DataGrid. In any UI framework, the robustness of such a component depends on formatting and validating utilities as well as a whole suite of data input controls: CheckBoxes, ComboBoxes, RadioButtons, all sorts of Inputs, Masks, and so on. Using theatrical terminology, the role of the king is played by his entourage. Practically speaking, touching up the DataGrid is touching up a large portion of the Flex framework.

Advanced DataGrid is a new component that builds upon the features of DataGrid and adds many features like multi column sorting interface, cell-level formatting functions, tree view (hierarchical data and grouping data), cell selection, custom rows, column grouping, SummaryCollection and the PrintAdvancedDataGrid.

### 5: Flex charting Components

The ability to display data in a chart or graph can make data interpretation much easier for Flex application users. Rather than present a simple table of numeric data, you can display a bar, pie, line, or other type of chart using colors, captions, and a two-dimensional representation of your data.

Data visualization lets you present data in a way that simplifies data interpretation and data relationships. Charting is one type of data visualization in which you create two-dimensional representations of your data. Flex supports some of the most common types of two-dimensional charts, such as bar, column, and pie charts, and provides you with a great deal of control over the appearance of the charts.

The chart controls all take a `dataProvider` property that defines the data for the chart. A *data provider* is a collection of objects, similar to an array. The chart components use a flat, or list-based, data provider, similar to a one-dimensional array.

A data provider consists of two parts: a collection of data objects and an API. The data provider API is a set of methods and properties that a class must implement so that a Flex component recognizes it as a data provider.

The data provider creates a level of abstraction between Flex components and the data that you use to populate them. You can populate multiple components from the same data provider, switch data providers for a component at runtime, and modify the data provider so that changes are reflected by all components that use the data provider.

## Chart types

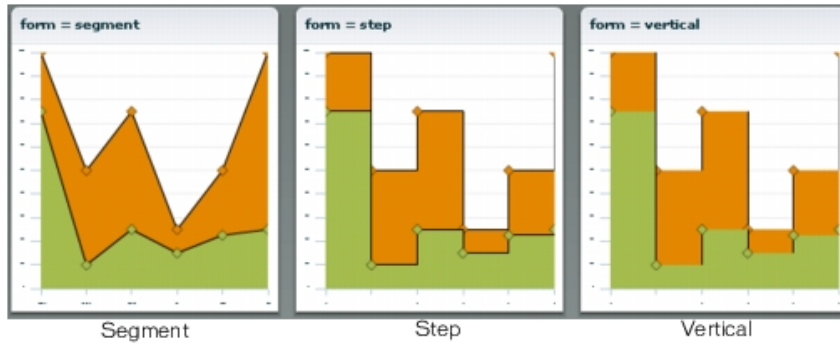
Flex supports many of the most common types of charts, including bar charts, line charts, pie charts, and others. This section describes the set of charts supplied with Flex. In addition to these chart types, you can also extend the `CartesianChart` control to create custom charts.

- Area charts
- Bar charts
- Bubble charts
- Candlestick charts
- Column charts
- HighLowOpenClose charts
- Line charts
- Pie charts
- Plot charts

## Area charts

You use the `AreaChart` control to represent data as an area bounded by a line connecting the values in the data. The area underneath the line is filled in with a color or pattern. You can use an icon or symbol to represent each data point along the line, or show a simple line without icons.

The following figure shows an example of an area chart:



## Bar charts

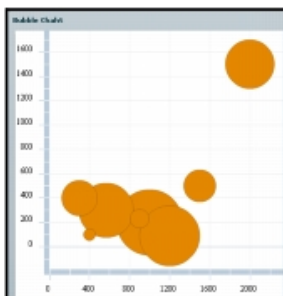
You use the BarChart control to represent data as a series of horizontal bars whose length is determined by values in the data. You can use the BarChart control to represent a variety of chart variations, including clustered bars, overlaid stacked, 100% stacked, and high-low areas.

## Bubble charts

You use the BubbleChart control to represent data with three values for each data point: a value that determines its position along the x-axis, a value that determines its position along the y-axis, and a value that determines the size of the chart symbol, relative to the other data points on the chart.

The `<mx:BubbleChart>` tag takes an additional property, `maxRadius`. This property specifies the maximum radius of the largest chart element, in pixels. The data point with the largest value is assigned this radius; all other data points are assigned a smaller radius based on their value relative to the largest value. The default value is 30 pixels.

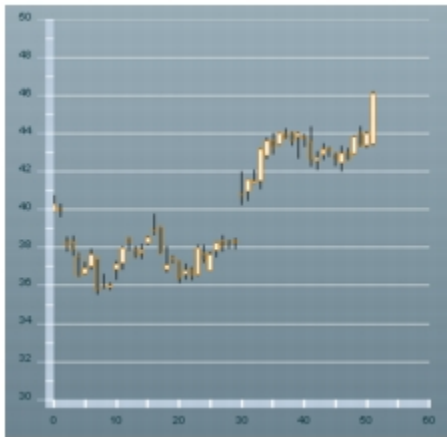
The following figure shows an example of a bubble chart:



## Candlestick charts

The [CandlestickChart](#) control represents financial data as a series of candlesticks representing the high, low, opening, and closing values of a data series. The top and bottom of the vertical line in each candlestick represent the high and low values for the data point, while the top and bottom of the filled box represents the opening and closing values. Each candlestick is filled differently depending on whether the closing value for the data point is higher or lower than the opening value.

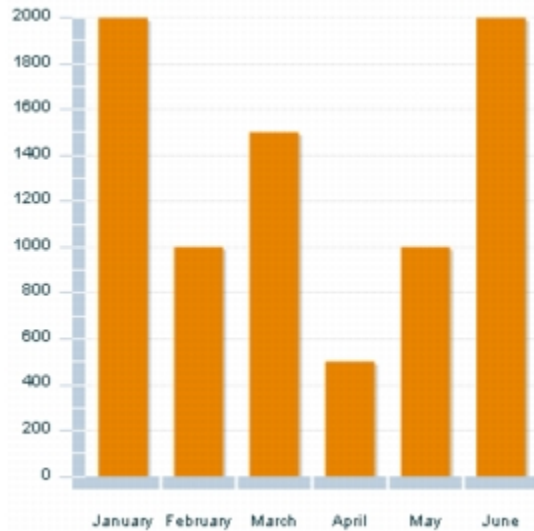
The following shows an example of a CandlestickChart chart:



## Column charts

The ColumnChart control represents data as a series of vertical columns whose height is determined by values in the data. You can use the ColumnChart control to create several variations of column charts, including simple columns, clustered columns, overlaid stacked, 100% stacked, and high-low.

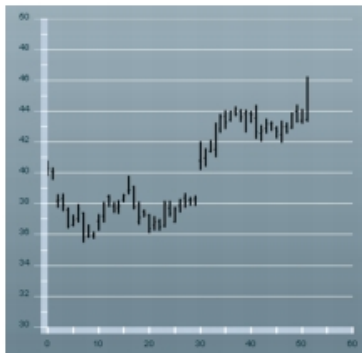
A simple chart shows a single data series, where a series is a group of related data points. For example, a data series might be monthly sales revenues, or daily occupancy rates for a hotel. The following chart shows a single data series that corresponds to the percentage growth in sales over four business quarters:



### HighLowOpenClose charts

The [HLOCChart](#) control represents financial data as a series of lines representing the high, low, opening, and closing values of a data series. The top and bottom of the vertical line represent the high and low values for the data point, while the left tick mark represents the opening values and the right tick mark represents the closing values.

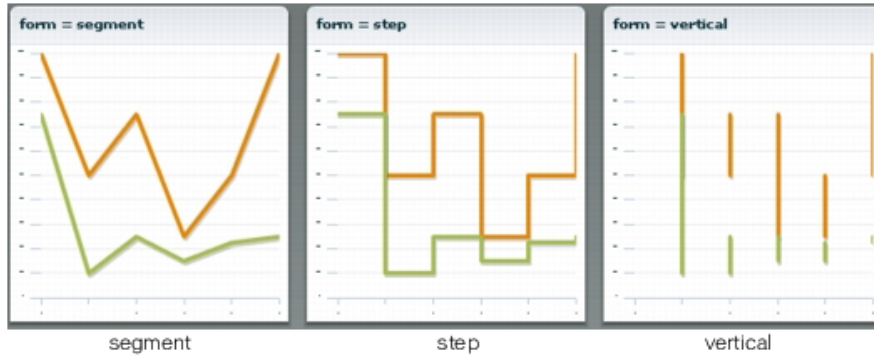
The HLOCChart control does not require a data point that represents the opening value. A related chart is the [CandlestickChart](#) control that represents similar data as candlesticks. You use the [HLOCSeries](#) with the HLOCChart control to define the data for HighLowOpenClose charts. The following example shows a HighLowOpenClose chart:



## Line charts

The LineChart control represents data as a series of points, in Cartesian coordinates, connected by a continuous line. You can use an icon or symbol to represent each data point along the line, or show a simple line without icons.

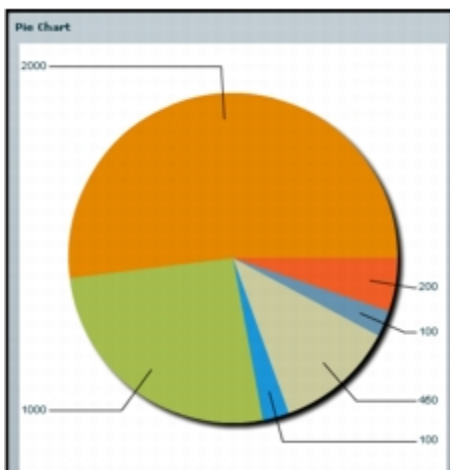
The following figure shows an example of a simple line chart:



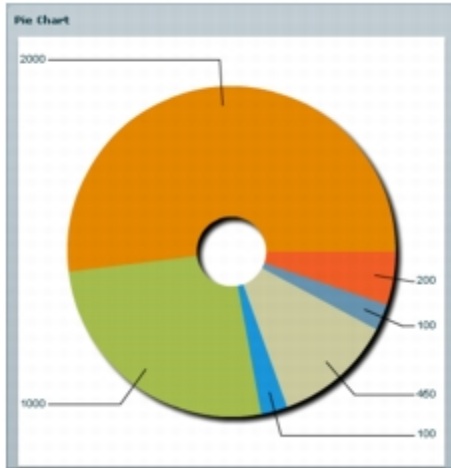
## Pie charts

You use the PieChart control to define a standard pie chart. The data for the data provider determines the size of each wedge in the pie chart relative to the other wedges.

The following figure shows an example of a pie chart:



Flex lets you create doughnut charts out of PieChart controls. Doughnut charts are identical to pie charts, except that they have hollow centers and resemble wheels rather than filled circles. The following figure shows an example of a doughnut chart:



### Plot charts

You use the PlotChart control to represent data in Cartesian coordinates where each data point has one value that determines its position along the x-axis, and one value that determines its position along the y-axis. You can define the shape that Flex displays at each data point with the data series' renderer.

The following figure shows an example of a plot chart with the CircleRenderer:

