O'REILLY® Short Cuts

Flex 3 Early Evaluation: Assessing Flex and Your Project Needs

by EffectiveUI, Inc.

ISBN: 9780596515911

Released: September 21, 2007

Evaluating any new technology can be a challenge. This Short Cut is designed to help web application developers (and their team managers) understand if Flex is the right technology for their company or client. Although the authors of this book have a strong understanding of the Flex framework, this isn't a sales pitch for Flex. The authors believe that Flex is a good tool to solve many business problems, but it is not a cure-all.

After reading this material, you'll know what's good about Rich Internet Applications. You'll know the capabilities of Flex 3, how Flex works, and how much it costs. We'll tell you about the challenges you'll face in developing with Flex, and show you the steps you'll take to create and deploy a Flex 3 application.

Contents

ADOBE DEVELOPER LIBRARY

Adobe

Introduction

As the President of Effective, I have the unique opportunity to listen to how companies like eBay, Ford, Random House, Viacom, GE, and NBC think about the Internet and desktop software. I get to see firsthand how people combine original ideas with innovative technology to completely change the way a company conducts business.

That's what we do at EffectiveUI. We work with companies large and small to strategize, design, and develop Rich Internet and Web 2.0 applications. Five years ago "Rich Internet Application" (RIA) seemed like any another buzzword. But we took it seriously. In fact, we built our business model around RIA technology. Our mission says it all: We create interfaces people want to use.

—Anthony Franco

About This Short Cut

Evaluating any new technology can be a challenge. This O'Reilly Short Cut is designed to help you understand if Flex is the right technology for your company. Although the authors of this book have a strong understanding of the Flex framework, it's not a sales pitch. We believe that Flex is a good tool to solve many business problems, but it's not a cure-all.

After reading this O'Reilly Short Cut, you should understand:

- What's good about Rich Internet Applications.
- The capabilities of Flex 3.
- How Flex works.
- How much Flex costs.
- The challenges you'll face using Flex.
- How simple it is to create and deploy a Flex 3 application.

What Is Flex?

Flex is a free, open-source platform and component library for developing applications that can be deployed on the ubiquitous Flash Player. When you think of Flash, you may think of the player or the Flash development IDE. Flex is just another way of creating Flash content. But that's not all. Flex is specifically geared toward building robust, Rich Internet Applications. The libraries and coding methodologies are designed to bridge the gap for "traditional" developers (Java, C++, .Net) to Flash Player deployed applications.

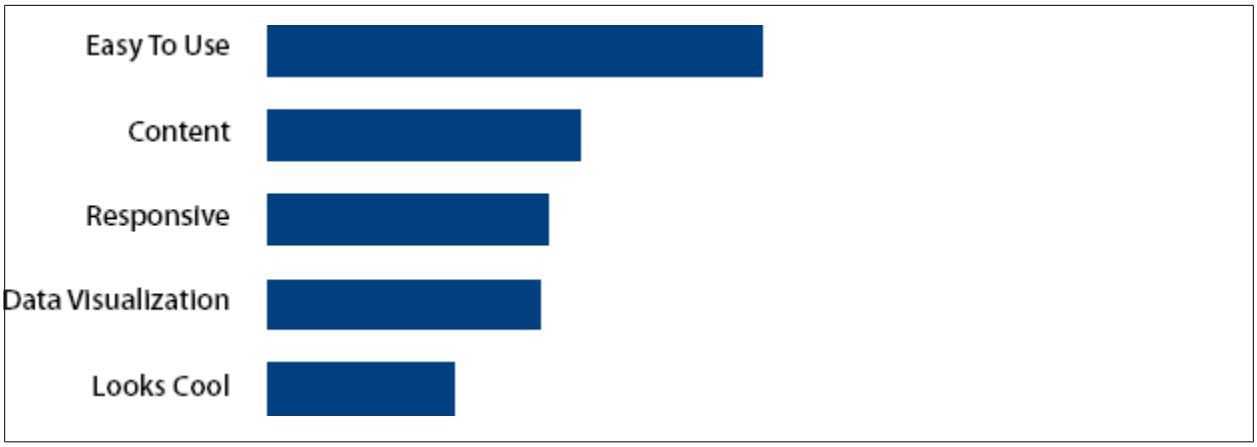

Figure 1. A Forrester Report states that users rank Easy to Use higher than Content as the reason they prefer interactive features in their browsers

Why Are Rich Internet Applications Important?

The Importance of User Acceptance

According to Forrester Research, 70-80 percent of all IT projects fail. That may seem like an unrealistically large percentage, but consider this: The majority of these projects fail not for technical issues, but rather due to a lack of user acceptance. Most projects are launched into production, but nobody uses them. (Insert your favorite "tree in the forest" analogy here.)

If most applications are never used, then what are the key elements of the ones that are? **Figure 1** lists the top five elements of interactive web applications ranked by users in importance. Flex offers a compelling framework to build applications that deliver all five elements.

Ease of Use Is the Key to User Experience

Figure 1 illustrates an important point: People prefer easy interfaces more than anything else in an application, including content. Conversely, the number one reason people abandon applications is because they're not easy to use, regardless of how cool and responsive the interface is or how much of the desired content is made available. The key to a successful application is how easy it is to use and learn.

It's widely admitted that many existing web applications built with HTML, Javascript, and CSS are not easy to use. Although this is partially due to poor design, it's also due to inherent limitations of the underlying technologies. HTML applications usually use a page-based model designed to present content that's linked to other pages, along with HTML forms and some limited Javascript added in to provide a measure of interactivity. This is great for presenting content and col-

lecting some information, but it's not good for building highly interactive applications. Full page refreshes with every interaction confuses people and causes them to lose focus.

Newer Web 2.0 applications such as Google Mail and Calendar try to solve this problem by using Javascript techniques (such as Ajax) to keep folks on a single page and provide them with more predictable desktop application-like responses. This has gone a long way in making web applications easier to use. However, there are inherent limitations in the ad-hoc techniques used to create these types of applications that make them formidable for the average developer. It's taken a Google to produce a truly usable and scalable Ajax application.

Flex's best asset is that it "bakes-in" interactivity and usability. It includes a very well-architected visual component library, designed by Adobe's Experience Design team. Flex controls are visually consistent, and easily skinnable. They use familiar user interface metaphors, they're accessibility compliant, and they're well-documented. Developers can use them to create web applications that provide the ease of use people crave. Furthermore, you can design the application to keep people focused on tasks and comfortable with the interface by using pleasant animation, sound effects, and predictable controls.

Flex Helps Deliver the Content Users Want

As popular internet portals like YouTube and Google illustrate, if an application has content and is designed well in other areas, then people will spend a lot of time there. Content-rich sites are also increasingly turning to Flash media players to deliver content because of the reputation Flash has for delivering high-quality multimedia experiences. These applications raise people's expectations for multimedia content and Flex is designed to help developers deliver such content.

Flex makes it easier for developers to add the Flash media player as a control in their applications. In recent years, the Flash player has been integrated as a standard part of every web browser and other devices, from the mobile phone to the XBox. The longevity and wide distribution of Flash player ensures that application content is experienced immediately by Flex application users.

Flex Applications Are More Responsive

People rate the speed at which an application loads and its responsiveness as key to their user experience. Because Flex applications are designed as full-featured clients, they're more responsive than traditional web applications because they don't always have to hit the server for every interaction. Also, while Flash applications often have a reputation for making people wait for them to download to the browser, Flex applications (being built on the Flash 9 player) are much faster.

The Flash 9 player is 100-fold faster than its predecessor. In addition, Memory Management, Just In Time Compiler, Shared Libraries, and automated Garbage Collection are all added features that help ensure a quickly downloaded and well-performing application.

Flex Delivers Data Visualization with Built-in Charting Controls

Besides multimedia content, popular applications on the Web help people visualize plain old data aggregated from one or more sources. Powerful support is designed into Flex to easily deliver such applications. Flex comes with a set of charting controls to build all types of popular line, bar, bubble, and pie charts. Flex also easily integrates with the backend servers from which these charts are produced. Finally, Flex has the full power of Flash animation, which allows you to create any type of data visualization that a developer or client can imagine. One example is using a ski mountain as a visual metaphor for ski weather data, instead of a boring line chart.

Flex Delivers Style So that Applications Can Look Cool

Flex comes with the same CSS power that traditional web applications have and much more. You can skin virtually any design into the application using a variety of techniques. With the power of sound and motion effects, applications look cool. While this is on the bottom of the list of what people want, it's important for folks who want stylish and elegant applications.

Flex works both Online and Offline

Many traditionally online development platforms are targeting the desktop, including Google Gears. Flex makes desktop development easy by allowing applications to target either the online Flash platform or the desktop-based Adobe AIR platform. Changing from an online to offline requires nothing more than changing two lines of code and recompiling.

Flex Gives Developers the Power to Deliver Superior User Experience

Because of its flexibility and power, Flex can't hold your hand throughout the process. Just as with any powerful development platform, you can design and build a very bad application. Ultimately, you know the users of your application better than anyone else. No matter what platform you choose, you'll need to balance functionality with complexity and usability with business goals.

Most certainly, some of the pros and cons listed in **Figure 2** are subjective and are based on "all things being equal." A strong Java developer can develop a Java application faster than an average Flex developer. Your user base may be Windows XP and Vista only, so XAML may be more compatible. Nonetheless, this chart

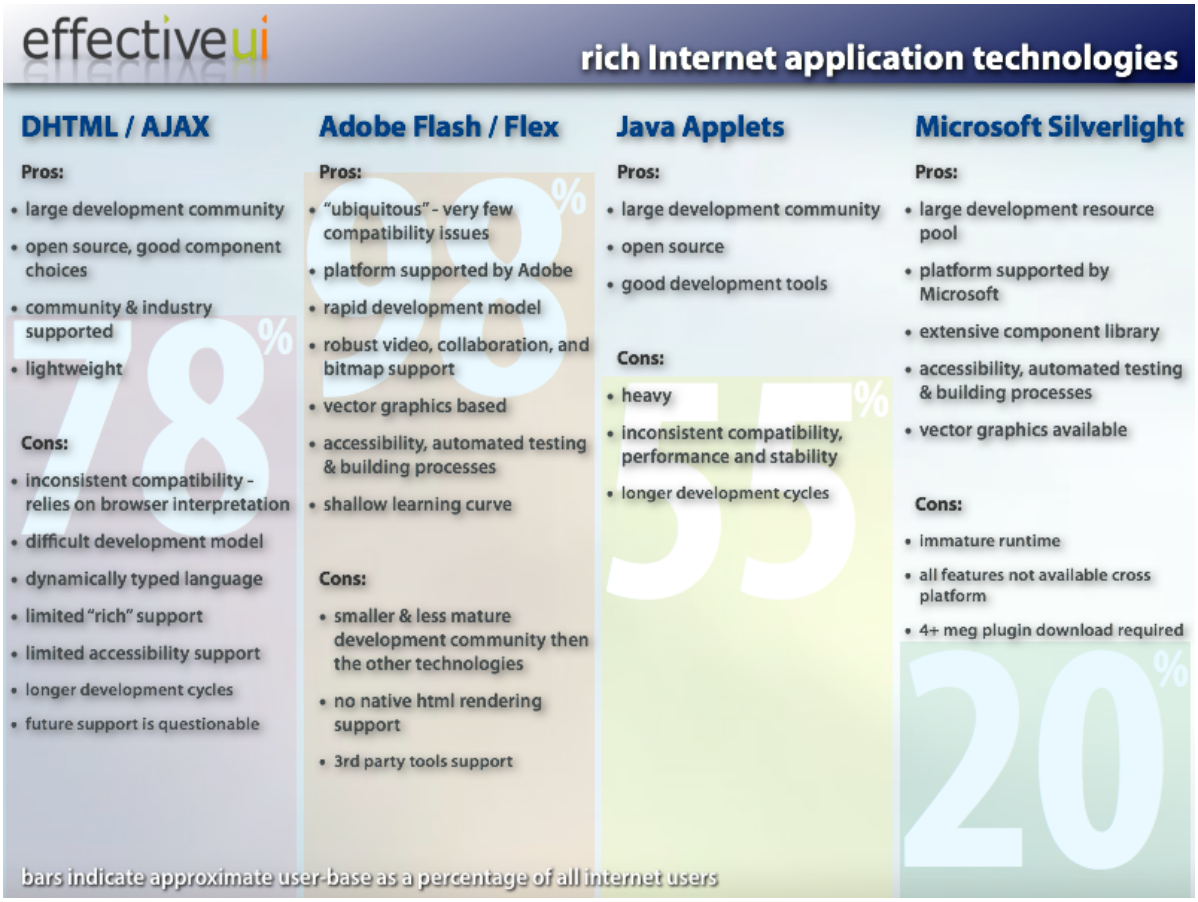

Figure 2.

helps to show that Flex offers a number of advantages. We'll examine Flex's capabilities more in depth later. First, let's look at Flex's background.

Understanding Flex 3 Capabilities

Background

Flash

User interface designers and developers have been severely limited in their choice of tools for online applications since the Web's inception. While backend developers could leverage the power of robust OO languages like Java and .NET, interface developers have struggled through the evolution of table-based designs in HTML to integration with JavaScript and Cascading Style Sheets into meaningful applications. Development has been hindered by a lack of consistent support between browsers and a lack of powerful languages with which to design interfaces.

To address these issues, Macromedia leveraged their animation tool Flash as an internet-deployable way to make interactive movies. As Flash developed, developers were granted more and more freedom to create applications similar to those available on the desktop. Additionally, as the Flash player spread to every browser and device available, developers no longer needed to spend the arduous hours traditionally required to address incompatibility issues in HTML, JavaScript, and CSS.

Flash, however, has always been geared more toward graphic designers than application coders, who are often initially confused by concepts like the timeline or the Flash development environment. To address these issues, Macromedia invented a new way to develop Flash applications using programming languages and metaphors developers were used to: Flex.

Flex 1.0

In early 2004, Macromedia (now Adobe) released Flex 1.0 as a new way to develop Rich Internet Applications. The intention of Flex 1.0 was to leverage the already engaging Flash client with the immense power of Java server-side technology using powerful programming styles familiar to desktop application developers.

Flex 1.0 was released as a J2EE application that compiled Adobe's proprietary MXML and ActionScript into Flash applications (.swf's). Flex 1.0 applications deployed in a browser to any systems running Flash Player 7. Deploying to the Flash Player greatly increased cross-platform compatibility.

Version 1 Flex applications were required to run on a Java-based application server known as Flex Presentation Server. Flex Presentation Server acted as a gateway, allowing the client to communicate with XML Web Services, Remote Objects, and most other SOA's. Although simple to install and configure, pricing started at around $12,000. The server also lacked certain management and analysis features common in competing server products.

MXML, an XML-based scripting language, was a huge step forward for developing on the Flash platform. MXML allows any developer with no prior Flash experience to develop Flex applications. However, due to the fact that MXML compiles down to ActionScript, both languages can be used for greater customization.

The components derived from the Flex 1.0 framework initiated the first steps toward positioning the Adobe Flex platform as a familiar and efficient way to produce immersive presentation tiers.

Flex 1.5

Macromedia promptly responded to most Flex 1.0 shortcomings by releasing Flex 1.5 in late 2004. Four main features were added in Flex 1.5, positioning Flex even closer to the top of the RIA food chain. The important new features included: improved data display and visualization tools, enhanced performance, more versatile styling and skinning, and even more deployment platforms.

In Version 1.5, Adobe also released Flex Charting Components, which provide simple interfaces for displaying data in a number of different charting formats such as pie graphs, bar and column charts, and line graphs. Flex Charting also provides a set of transitions that you can apply to animate the graph as data changes.

For more info on Flex 1.5, visit: ***http://www.adobe.com/macromedia/proom/pr/2004/flex_15_announce.html***

Flex 2.0

In mid-2006, Adobe drastically improved existing Flex technology with the release of Version 2.0. Version 2.0 was significant for two reasons. First, Adobe removed the original licensing and compiling restrictions on Flex and released version 2.0 as Open Source. A command-line Flex compiler was provided free of charge, along with source code for the thousands of rich components present in Flex since its inception.

Next, Adobe updated Flex and MXML to integrate with the new and powerful ActionScript 3 programming languages. ActionScript 3 has many advantages over ActionScript 2, including strong typing and a more consistent use of object-oriented programming metaphors familiar to all application programmers today. A full discussion of the merits of ActionScript 3 is beyond the scope of this document, but with the use of this language and access to the full set of Adobe component library source code, developers were given an even greater degree of freedom in developing components for Rich Internet Applications.

Flex 3

Flex 3 was released in beta form in June of 2007. As of this writing the product is still in beta, though release is targeted for October.

Flex 3 represents a less extreme update to the framework than Flex 2 did, but provides a number of nice enhancements. First, Flex 3 provides deeper integration with other Adobe products, allowing complex animations designed in Flash and complicated designs built in Fireworks to be easily imported to Flex. Second, Flex 3 enables persistent framework caching, which allows applications to be compiled into .swfs as small as 50k. Third, Flex 3 provides several new components, in-

cluding the Advanced DataGird, which has more functionality and is easier to style than the traditional DataGrid. Fourth, Flex 3 provides native support for Adobe AIR applications, making it as easy to build desktop applications as Flex has made it to build them online. Lastly, Flex 3 is released with a new version of Flex Builder (currently codenamed "Moxie") that has many new features (including code-refactoring) that make development easier.

Development in Flex 3.0

You can develop Flex 3.0 applications using one of two basic approaches. First, you can develop them in any preferred text editor and compile them using the free command-line compiler provided by Adobe. This approach gives you the freedom to work with whatever environment and programming tools you're used to without incurring any cost for the use of the environment.

Second, you can license the Flex Builder Development IDE from Adobe for a marginal fee. Flex Builder is built on the Open Source Eclipse environment familiar to many application developers as both a standalone IDE and a plugin for Eclipse. Flex Builder also makes a Design view tool available, which provides a WYSIWYG (What You See Is What You Get) interface to create MXML files, useful in managing the basic elements in Flex application layouts.

Architecture

Flex makes use of a number of programming metaphors familiar to application developers that aren't available or easily accessible in other RIA programming environments. Let's take a look at some of these.

Event-Driven Components

Flex uses the event broadcaster/listener paradigm to communicate between objects. ActionScript 3 makes these concepts available in a way that developers used to the pattern will instantly recognize, particularly those used to implementing it in Java. Other RIA development schemes must rely on more clumsy methods of passing data between objects.

Data Binding

Flex introduces a concept of data binding to communicate between objects. Data binding allows a developer to effectively bind a data field on one object to a field or fields on other object(s) through a few lines of code. Flex implements this by compiling data binding into a combination of an event dispatcher, listener, getter, and setter behind the scenes. This gives you cleaner code and instant access to updated data, allowing you to focus on the real problem the RIA is designed to solve, rather than managing excess, but necessary, code.

Accessibility

Adobe's component libraries promote accessibility by providing a framework that easily supports a wide range of accessibility tools, including screen readers and a variety of input devices, from mouseless workstations to webcams.

Automated Testing

FlexUnit is a free set of application libraries that allows automated testing of Flex applications, much like JUnit for Java.

Flex Data Services

According to Adobe's web site, Flex Data Services is a J2EE application that provides high-performance connectivity with existing server-side data and business logic. Flex Data Services allows for seamless access server side objects in Flex, and integrates seamlessly with other powerful middleware, such as Hibernate and Enterprise Java Beans.

Flex Data Services has several advantages over traditional web services, including data synchronization and serialization. Flex Data Services implements a unique form of data synchronization that allows for instant updates and doesn't rely on polling, making your application's view of your data a simple extension of the database. Flex Data Service's use of serialization makes it easy for developers to manipulate data in the client-side Flex application in the same way they'd manipulate it in Java on the server-side, without having to worry about data being lost or corrupted in between.

Flex Data Services also offers a wide range of pricing options, from a free single-CPU production version to volume licensing. Adobe's offer of a free version ensures that developers can create and test their applications locally without having to purchase a license for yet another piece of software.

Flash Media Server

Flex integrates seamlessly with Flash Media Server, an Adobe product which manages streaming media content. Integration with Flash Media Server ensures that application developers can focus their efforts on making interfaces to video media rather than consuming a lot of time by managing the video content themselves.

Player

The Flash Player has several features that make it attractive as the deployment environment for Rich Internet Applications. Let's discuss a few of these.

Vector-Based

The Flash Player renders objects as vector-based images, meaning that each line and curve is drawn dynamically. This ensures that images don't lose their crispness as the scale changes.

A good example of where this is useful is in displaying a map. In a traditional RIA, the map is displayed in a graphical format such as a JPEG, GlF, or PNG. If someone wants to zoom in on a certain section of the map, another raster image featuring a more detailed view of the area, appropriately scaled for that person's screen, needs to be served to the user. This process causes a noticeable lag in the application.

In direct contrast, the Flash Player simply redraws the image with the appropriate scale and dynamically adds features as necessary, so that all processing is done quickly on the client side. This ensures that the application responds as quickly and effectively as traditional desktop applications.

JIT Compiler

Inconsistent and poor performance is often noted as a major problem in the traditional web browser. Flex offers greatly improved performance by running in the Flash Player, which makes use of a Just In Time (JIT) compiler. Applications are delivered to the player's virtual machine as byte code and then compiled just before execution on the client's machine. This allows for increased performance times and is a tried-and-true method of delivering content used by all Java and .NET applications.

Performance is an important aspect to consider when creating great Rich Internet Applications. A JIT compiler ensures that Flex applications respond to user interaction with the same speed and robustness of a traditional desktop application.

Bitmap Manipulation

Flex provides an extensive bitmap manipulation library that lets developers create applications that dynamically generate and alter bitmap-based images.

Video

The Flash Player provides a built-in video-viewing component, which ensures people have a consistent experience when viewing video. Developers don't need to worry about serving multiple versions of the video to accommodate different players and operating systems, and can embed the video directly into their applications in meaningful ways that enhance the user experience. Due to these advantages, we've seen many video-centric sites (such as YouTube) emerge in the past several years, deploying videos using only the Flash Player.

Transitions and Effects

Flex makes the robust transitions and effects that Flash applications are known for easily accessible to developers without bogging them down in timeline details. Flex provides a complete library of simple effects that developers can leverage to create meaningful transitions that enhance the feel of applications, including complex movement effects, fade effects, sizing effects, and rotation effects.

Since Flash was first developed, effects on web pages have been controversial. Everyone who's used a web site can probably recall visiting sites where a poor implementation of effects lead to a confusing interface. At the same time, when used correctly, effects can greatly enhance user experiences and make applications easier and more interesting to use. Simple, elegant effects and animations have been implemented in desktop applications with great success for at least a decade now, and Rich Internet Applications should aspire to offer the same degree of comfort to their users.

Flex also supports a wide range of sound effects. These are easily implemented through an extensive sound object API.

Printing

Flex applications can easily use the browser's inherent printing capabilities so that people aren't limited to viewing data onscreen, but can print relevant documents and data. Flex developers can also restrict which portions of the application can be printed to help ensure copyright laws on things such as images (a Flex image gallery for a photographer's web site, for example).

Binary Sockets

The request/response model used by traditional web pages leads to slow, unresponsive web pages. Flex uses of binary sockets to enable a real-time bidirectional transfer of data between client and sever. The Flex Messaging Service uses this functionality to provide a complete messaging solution that integrates with existing enterprise solutions like JMS.

Localization

Flex allows limited access to read and write data on a client's local machine. Flex applications are limited to reading only local shared objects they've saved, preventing security holes inherent to other localization schemes. Flex applications can leverage data localization to increase performance times and give a measure of offline support to applications.

Using the Flex 3 Framework

Language

Flex uses two programming languages: MXML and ActionScript.

MXML

MXML is unique to Flex and designed specifically to describe properties in Flex layouts, though it's not limited to describing visual objects. MXML is an extension of XML, where each tag represents a different Flex component. You can nest components inside each other to provide a variety of different layout and interface options. Other nonvisual components, such as web services interfaces, transitions, and style sheet specifications can also be written in MXML.

ActionScript 3.0

ActionScript 3.0 is a rich and robust programming language that's useful for complex data manipulation. It's the latest version of the Flash Player's ActionScript language, which was originally based on the ECMA Script 1 specifications for scripting languages and resembled JavaScript. The current implementation has come a long way, and looks much more like Java than JavaScript.

You can write ActionScript 3.0 methods to process data, react to events, perform transitions, or even draw shapes and components on the screen. It's customarily implemented to enhance the functionality of MXML components.

ActionScript and MXML work very nicely together. While all Flex applications must begin with at least a single MXML file, ActionScript can be nested within MXML components or imported from separate files. In the intended implementation, MXML provides the structure and organization in an application, while ActionScript gives access to advanced features, processing, and data manipulation methods.

Other Development Integration

How to Integrate Flex with Your Server Stack

Flex isn't tied to a specific technology; rather, it defines common interfaces for connecting to multiple independent data sources, which makes it a good player in an SOA (service-oriented architecture). Flex connects seamlessly to most backend services in a service-oriented architecture, whether the underlying technology is Java, PHP, Ruby, or .NET. Flex provides several nonvisual components which you can use to connect to a variety of web-based data sources.

What Java Developers Should Know

Most Java developers will find that their experience in Java makes the transition to Flex application development relatively easy. Flex is specifically designed to use the tools Java developers are familiar with, and the syntax of ActionScript 3 is remarkably similar to that of Java. Additionally, Flex takes advantage of a number of design patterns and software best practices that Java developers will feel comfortable using.

The controller and event listener design patterns utilized by Flex components will be familiar to Java programmers with AWT or Swing development experience. It may be less familiar to JSP developers who are used to JSP tag libraries and templates and not a fully OO client server interaction. For these programmers, it will be a pleasant experience, since they have the full power of object-oriented programming and full suite of UI controls from which to quickly build powerful UIs.

Furthermore, Java developers will find that using Flex with a Java backend creates a much cleaner separation between client and server than in other web-development frameworks like Struts, Tapestry, and JSF. Java developers will discover a number of options to connect a Flex client to their favorite Java server stack, whether it's JBoss or Spring, Hibernate, straight JDBC, or some other server. Whatever the choice for a Java server backend, it's easy to expose Java server objects (either via http or WSDL) to Flex frontend-based web services or directly as Flash remoting objects.

Finally, Java developers with no Flash experience may be unfamiliar with vector graphics, movie player, and the animation Flash roots of Flex. However, many of these have been removed or simplified in the Flex framework and replaced with easy-to-use effects and components. Java developers should learn as much as possible about the Flash roots of Flex to fully utilize the expansive set of rich tools available.

What .NET Developers Should Know

Similar to Java developers, many .NET developers will find themselves comfortable in the underlying language of Flex and ActionScript 3 because of a shared similarity with C# and VB.NET. Like Java developers, .NET developers will feel comfortable with the object-oriented aspects of AS3, from classes and interfaces to getters and setters. The .NET developers familiar with ASP development will experience a similar transition out of the HTML/CSS template world to a full-featured client-side UI experience. For those who've used Windows Forms and VB UI development, the learning curve will be less as they'll be used to writing code behind a set of visual components and traditional client/server design patterns.

One potential obstacle for .NET developers is to see Flex as a non-.NET technology, since most Flex applications are built with Java and PHP server backends. .NET developers will be happy to know that Flex operates well with web services built and deployed on .NET. In fact, it's easy for Flex applications to consume simple http web services created in .NET. Simply create the web service as normal in .NET by marking your functions as WebMethod. This exposes a URL like: ***http://localhost/MyWebService/MyWebService.asmx?WSDL***

In Flex, it's easy to consume a web service with the WebService component, allowing Flex to call any WSDL-based web service. Another option is to create a non-WSDL, http-based service in .NET and use the Flex HTTPService component to call the data. You can also expose .NET objects as Flash Remoting objects that are serialized across the network and consumed by the Flex RemoteObject component. Whatever option you choose, you'll find proven methods of using Flex as a frontend to .NET server-side applications.

Finally, .NET developers who've followed and are learning the Windows Presentation Framework (WPF), on which the user interface for Windows Vista is built, will already be interested in and familiar with animation, multimedia support, 3-D, advanced visual and sound effects, vector images, and many other advanced features of building user experiences with Flash and Flex. They should investigate Flex as an alternative environment in which to build advanced rich user experiences.

What Flash Developers Should Know

If you're a Flash developer who learned how to code in the Flash IDE, you have a slight learning curve ahead of you. To start, Flex obscures the timeline from you and forces a more traditional OOP methodology. If you understand the basic concepts of MVC (Model, View, Controller) architecture, you're on your way to understanding how the Flex component framework works. Each component has the notion of an event listener and event dispatcher.

Licensing Costs

Open Source: $0.00

In contrast to Version 1, Flex 3.0 is free (as was Version 2.0). You can design, build, and deploy commercial Flex 3.0 applications and never spend a dime on licensing. Adobe opened up the platform to gain a broad developer adoption and they hope to earn revenue on developer tools and servers for larger teams working on complex projects.

Flex Builder: $499

Flex Builder is an integrated development environment, or IDE, for Flex Developers. It's built on top of the Open Source Eclipse IDE used primarily for Java development. It has a feature set similar to other IDE's like Microsoft Visual Studio, NET Beans and JBuilder. Flex Builder is by no means required for Flex development (all of the source code can be written by any text editor) but its rich set of development views and tools make it very useful in Flex development. Flex Builder also offers a WYSIWYG design view that allows developers to graphically drag and place components in their application as desired. This is especially useful for new developers who want to quickly learn how the different layout components can be used to make clean interfaces.

Charting Components: $299 or $749 for the FB/Charting bundle

For a small fee, Adobe offers a set of charting components that you can use in production. The charting package contains all standard chart types and all of the classes required for manipulating them. You can test it for free, but your charts will contain the words "Flex Charting Trial" until you purchase a serial number from Adobe.

Flex Data Services: $0.00-$20,000

Flex Data Services is a set of Java libraries provided by Adobe to make it easy for Flex applications to connect to standard J2EE server-side applications. Flex Data Services has several pricing options, from the free single-CPU license up to a $20,000 enterprise edition.

Flash Media Server: $4,500 per CPU

Flash Media Server manages streaming media from your server and integrates well with Flex applications. Adobe currently offers a free evaluation version with no expiration date, but the full commercial version is $4,500 as of the time of this publication.

Putting It All Together: Building an Application in Flex 3

The first part of this section offers step-by-step tutorials for the following:

- Creating a new Flex project
- Laying out a simple application in Design view
- Adding data to a chart using data binding
- Running a Flex application

In order to cover more ground, the second portion walks you through some snippets of code that explain:

- How to create additional views for existing data sources
- How to change styles in a Flex application
- How to add simple transitions

The sample application provided here can also serve as a resource for highlighting a more complex implementation of the Flex presentation tier. All code is documented with explanations for each of the components.

In order to complete the following examples, you need to download and install Flex Builder 3 and the Flex Charting Components. You can find a 30-day trial of both at this location: ***http://www.adobe.com/products/flex/***

First install Flex Builder 3, and then install the charting components. Once you have both installed, start Flex Builder to begin building your first Flex application.

Getting Started with Flex Builder 3

Flex Builder 3 is based on the Eclipse IDE, an Open Source Java-based development environment built by IBM. You can install Flex Builder as a plugin for an existing installation of Eclipse or as a standalone Eclipse application. If you have an existing installation of Eclipse on your system, you may want to install Flex Builder as a plugin. Otherwise, install the standalone view, and you're ready to start coding. Let's begin by creating a new project.

Creating a New Flex Project

1. From the main menu bar select File → New → Flex Project. A wizard appears, walking you through a few easy steps to create a new Flex project.
2. Give your project a name. Any name at all. Feel free to get creative.
3.
4. Leave the server type as "Other/None." We won't need to use a server for this application.
5. Click 'Next.' If you want to change where Flex builder stores the files for this application, feel free to do so.
6. Click 'Finish.'

Your screen should look something like **Figure 3**.

Flex Builder creates the application document, main.mxml. Flex organizes your application as groups of components and "main.xml" actually represents the Application component, which is the outermost level component in any Flex appli-

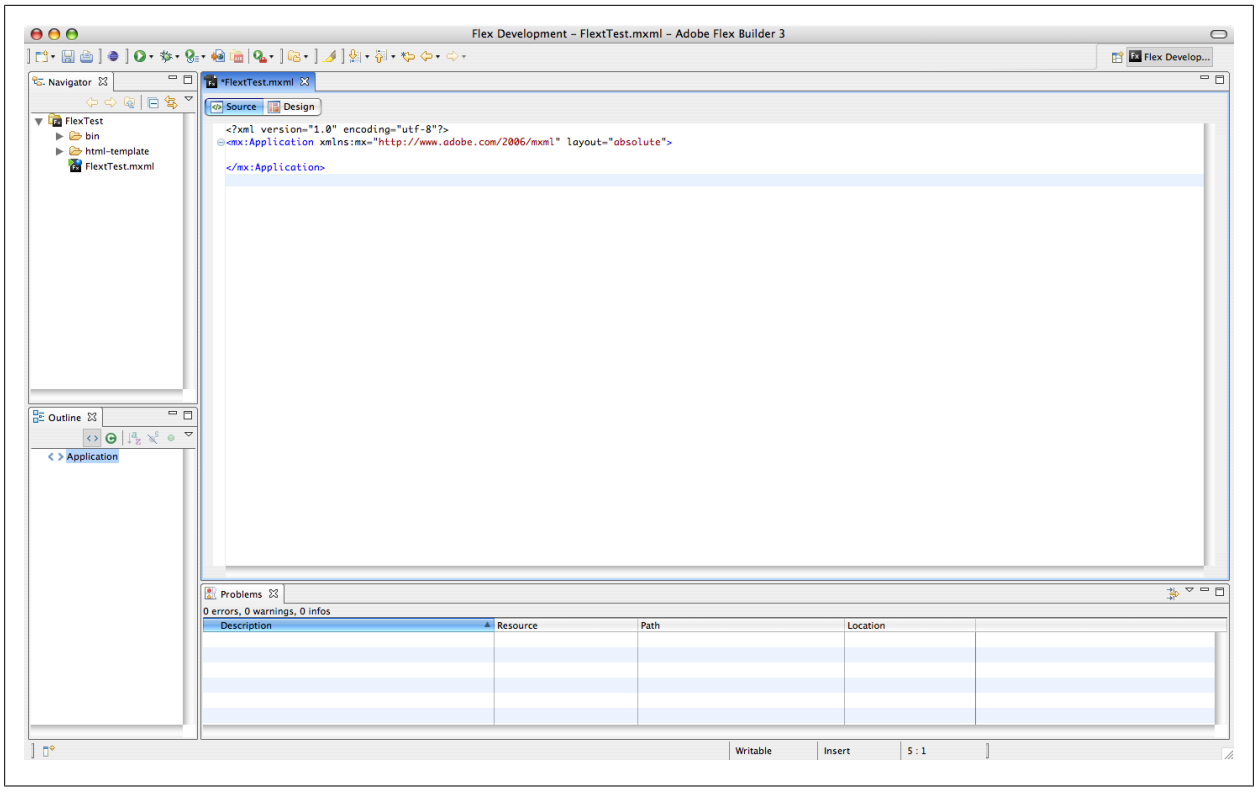

Figure 3.

cation. Later, we'll nest a number of other components inside this Application component.

Laying Out an Application Using Design View

Source or Design View?

Flex offers two views for developing applications, Source view and Design view.

Source view is the traditional method of developing in most IDE's and it gives you direct ability to view and edit the code. Most developers use this view, as it offers maximum flexibility and prohibits an additional layer of abstraction from the developer to her code.

Design view is a WYSIWYG development view. It allows you to drag components from the Components panel onto the stage and see them rendered without having to compile the application. After you've placed a component on the stage, you can change the properties and styles of each component instance using the Flex Properties panel. We'll build this section in Design view just to give you a feel for how it works, and then we'll build the rest of the sections in Code view.

Directly below the tab header for main.mxml, you'll see two buttons labeled Source and Design. Click the Design button and you should see something similar to **Figure 4**.

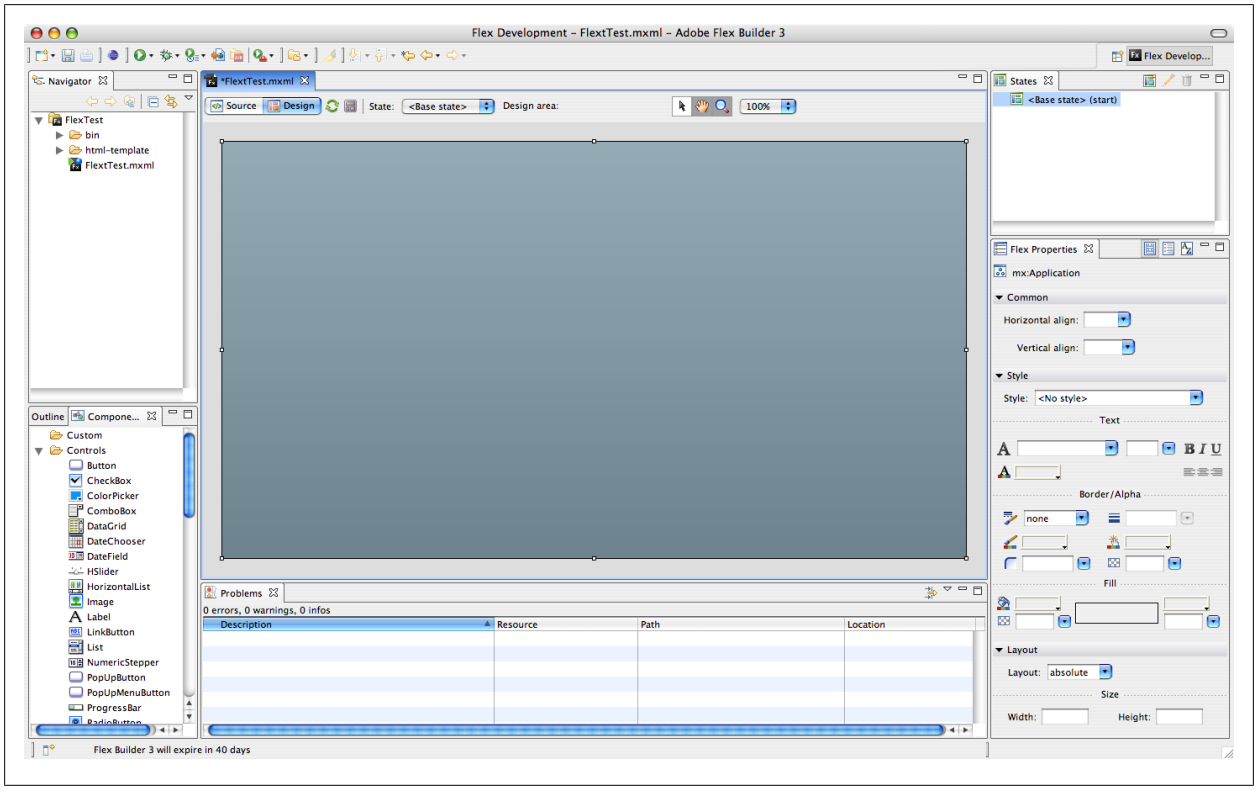

Figure 4.

The Components panel is on the lower-left side of the screen. It contains a list of all the prebuilt Flex components provided by Flex Builder. The Flex Properties panel is on the righthand side of the screen. This is where you can edit each individual component. Located above the Flex Properties panel is the States panel, which allows you to create different view states for your application without writing a single line of code.

For this lesson, we'll concern ourselves only with two panels: Components and Flex Properties.

Flex Properties Panel

There are three view options in the Flex Properties panel: Standard, Category, and Alphabetical (**Figure 5**). Standard is the default view for properties. It has the appearance of an easy-to-read form. Category view displays a list of properties organized in their associated category. Alphabetical view is self-explanatory.

Components Panel

The Components panel is a library of components at your disposal. It's accesible only when you're in Design view. The components are organized by the function

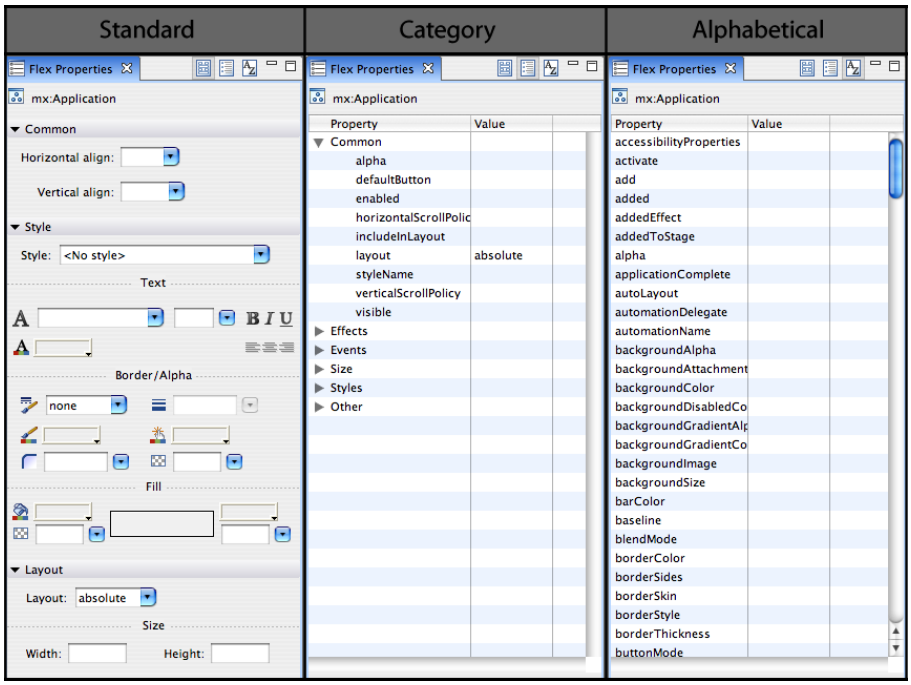

Figure 5.

they serve. Any custom components you write can also be accessed from this panel in the Custom folder (**Figure 6**).

The Layout Manager

Flex relies on a number of layout components to organize the objects in an application. While developers are free to write their own layout components, Adobe has provided a well-rounded set of prebuilt components with various options on how to organize the application. The Flex Layout Manager considers the order of components within a layout component and the type of layout component being used when rendering the application. For instance, an HBox is a simple layout component that displays all of its children horizontally, adding subsequent children to the right of previous children. Similarly, the VBox component displays all of its children vertically, with the first child on top. The separation of layout components from other interactive components makes it easy to control and change the layout of your application without having to touch data processing code.

Our application simulates an RIA used by an automobile distributor to display the number of SUVs, Sedans, and Vans sold for the past few years. We'll display the data first in an editable spreadsheet and then on a chart. Let's get started.

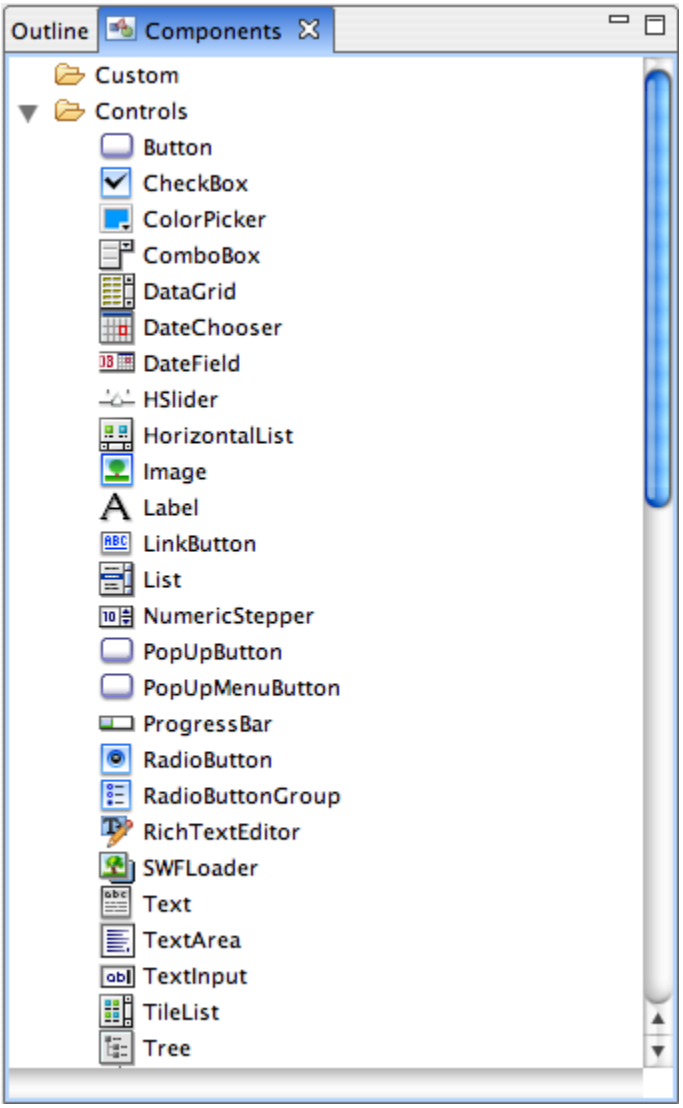

Figure 6.

1. Click anywhere inside of the stage area to select the application object. You should see mx:Application at the top of the Flex Properties window (**Figure 7**).
2. With the Application object selected in Design view, set the following properties from the Properties panel:
   - Horizontal align: *center*
   - Layout: *vertical*
3. Drag a Panel component onto the stage. Using the Properties inspector on the left, set the Title property to Car Sales, the width to 800, and the height to nothing (**Figure 8**).

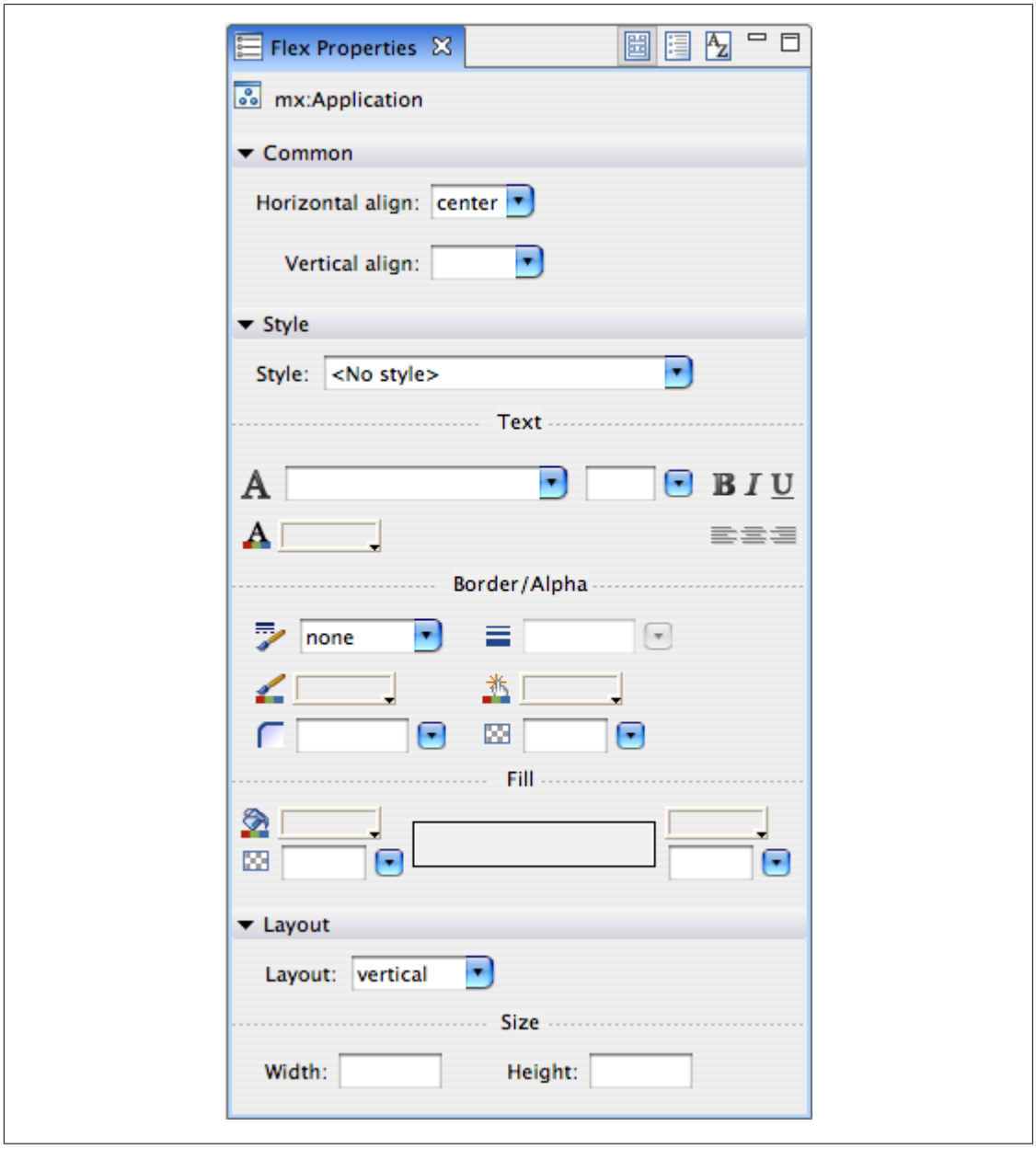

Figure 7.

Setting the height to nothing causes the Flex Layout Manager to size the component to fit its children.

4. Drag another panel onto the stage. Set the following properties on the Properties inspector:
   - width: 100%
   - height: 100%
   - layout: vertical

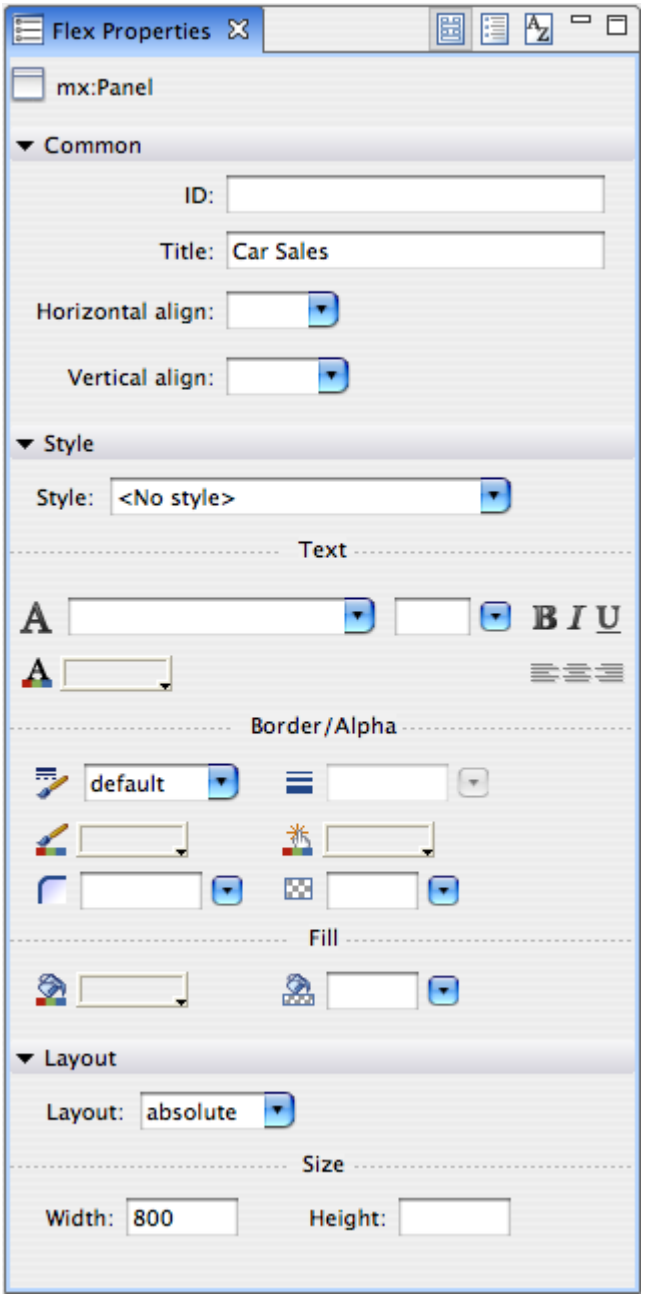

Figure 8.

- title: Car Sales by Year

5. Drag a DataGrid component from the Components panel into the top panel. Set the following properties on the DataGrid:
   - width: 100%
   - editable: true

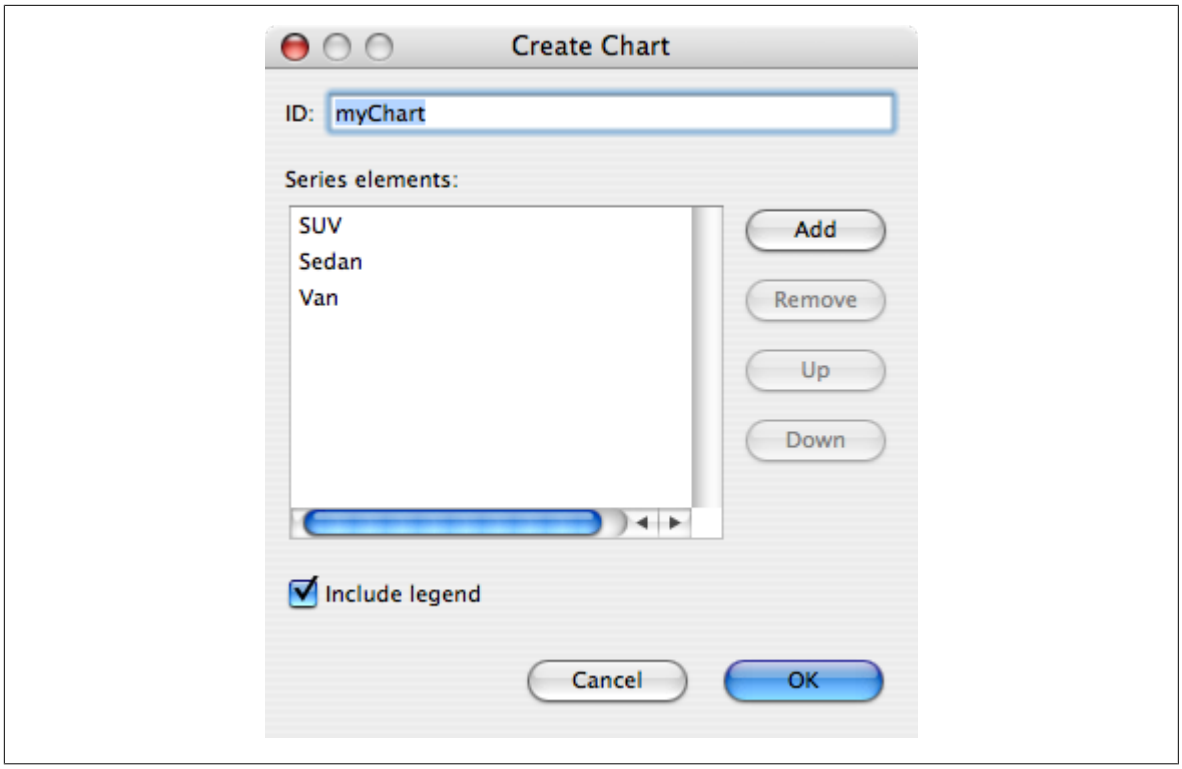

Figure 9.

6. Drag a ColumnChart component from the Components panel into the bottom panel. A wizard appears, prompting you to set a few properties for the ColumnChart. Set the ID to myChart. Select the default series, series1, and click Remove. Then add three series named SUV, Sedan, and Van as shown in **Figure 9**.
7. Set the following properties on the ColumnChart:
   - width: *100%*
   - height: *60%*
   - id: myChart
8. Notice a legend component was added just below the ColumnChart. Drag the legend above the ColumnChart within the panel so that it rests above the ColumnChart.

If you switch back to Source view, you'll see the MXML that Flex Builder generated. All of the components are created as open node tags. You'll also notice that all the component properties are applied as attributes to the nodes.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical">
       <mx:Panel width="800" layout="absolute" title="Car Sales">
```

```
        <mx:DataGrid width="100%" editable="true">
        </mx:DataGrid>
    </mx:Panel>
    <mx:Panel width="100%" height="100%" layout="vertical"
        title="Car Sales by Year">
        <mx:ColumnChart id="myChart">
            <mx:series>
                <mx:ColumnSeries displayName="SUV" yField=""/>
                <mx:ColumnSeries displayName="Sedan" yField=""/>
                <mx:ColumnSeries displayName="Van" yField=""/>
            </mx:series>
        </mx:ColumnChart>
        <mx:Legend dataProvider="{myChart}"/>
```

Now that we have a simple layout, it's time to add some real data to the ColumnChart and DataGrid. The next section shows you how to bind the chart to an ActionScript data source.

Adding Data to the Controls with Data Binding

A useful feature in Flex is the ability to bind a data source directly to a visual component's dataProvider property. The Flex data-binding architecture has a built-in mechanism that automatically updates all binding destinations whenever the source data changes. We now need to add some ActionScript to create the data source.

Creating an ArrayCollection and Binding it to the Controls

1. Add the following block of code directly below the opening <mx:Application> tag:

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

        [Bindable]
        private var chartData:ArrayCollection;

        private function init() {
            chartData = new ArrayCollection();
            chartData.addItem( {year:2003, count:53});
            chartData.addItem( {year:2004, count:34});
            chartData.addItem( {year:2005, count:65});
        }
    ]]>
</mx:Script>
```

2. In order to initiate the chartData ArrayCollection, we need to call the "init()" method when the "initialize" event is dispatched from the Application. Insert

the initialize handler into the <mx:Application> tag, as shown in the last line of the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    horizontalAlign="center"
    initialize="init()">
```

3. Now that we've created a data source, we can bind it directly to the dataProvider property of the ColumnChart and DataGrid using curly bracket notation, as shown here:

```
<mx:DataGrid width="100%" dataProvider="{ chartData }">
```

The data binding for the controls is complete. After only a few more steps, we'll be ready to run the application.

Customizing the ColumnChart

In order for the ColumnChart to render its data properly, we need to make some additions to the chart.

1. Inside the ColumnChart MXML tag, you'll notice a child node named Series. The Series property is an Array of fields within the dataProvider that you'll want to display in the chart. In the case of this dataProvider, we'll want to display three series: "sedan," "suv," and "van." Add the additional <mx:ColumnSeries> tags to the <mx:series> node. The Series node should look like this:

```
<mx:series>
    <mx:ColumnSeries displayName="Sedan" xField="year" yField="sedan"/>
    <mx:ColumnSeries displayName="SUV" xField="year" yField="suv"/>
    <mx:ColumnSeries displayName="Van" xField="year" yField="van"/>
/mx:series>
```

2. As you see in the previous sample, the xField property for each series corresponds to the year field of the dataProvider. Now we need to add a horizontal axis to the chart. Add the following block inside of the of ColumnChart node:

```
<mx:horizontalAxis>
    <mx:CategoryAxis categoryField="year" title="Year" />
</mx:horizontalAxis>
```

Customizing the DataGrid

The DataGrid needs some fine tuning to display data in a readable fashion. Because there are four fields in each item of the dataProvider, we need to make sure that the DataGrid has a column for each field.

1. Inside the <mx:DataGrid> tag, you'll find three existing columns <mx:DataGridColumns>. Add an additional column so there are a total of four.
2. Format the columns to look like this:

```
<mx:columns>
       <mx:DataGridColumn headerText="Year" dataField="year" editable="false"/>
       <mx:DataGridColumn headerText="Sedan's" dataField="sedan"/>
       <mx:DataGridColumn headerText="SUV's" dataField="suv"/>
       <mx:DataGridColumn headerText="Van's" dataField="van"/>
</mx:columns>
```

   The headerText property defines the label that shows up in the header of the column. The dataField property defines which field from the dataProvider it should display. The editable field in the year column is set to false because we don't want to change the year in this case.

Your main.mxml file in its entirety should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
       layout="vertical"
       horizontalAlign="center"
       initialize="init()" width="100%" height="100%">
       <mx:Script>
              <![CDATA[
                     import mx.collections.ArrayCollection;

                     [Bindable]
                     private var chartData:ArrayCollection;

                     private function init():void {
                            chartData = new ArrayCollection();
                            chartData.addItem( {year:2003, sedan:53, suv:32,
van: 23});
                            chartData.addItem( {year:2004, sedan:43, suv:32,
van: 43});
                            chartData.addItem( {year:2005, sedan:23, suv:32,
van: 53});
                     }
              ]]>
</mx:Script>
```

```
<mx:Panel width="800" title="Car Sales">
       <mx:DataGrid width="100%" dataProvider="{ chartData }" editable="true">
              <mx:columns>
                     <mx:DataGridColumn headerText="Year" dataField="year"
                            editable="false"/>
                     <mx:DataGridColumn headerText="Sedan's" dataField="sedan"/>
                     <mx:DataGridColumn headerText="SUV's" dataField="suv"/>
                     <mx:DataGridColumn headerText="Van's" dataField="van"/>
              </mx:columns>
       </mx:DataGrid>
</mx:Panel>

<mx:Panel width="100%" height="100%" title="Car Sales by Year">
             <mx:Legend dataProvider="{myChart}" color="0x000000"/>

             <mx:ColumnChart width="100%" height="100%" id="myChart"
                    dataProvider="{ chartData }" >
                    <mx:horizontalAxis>
                           <mx:CategoryAxis categoryField="year" title="Year" />
                    </mx:horizontalAxis>
                    <mx:series>
                           <mx:ColumnSeries displayName="Sedan" xField="year"
                                  yField="sedan"/>
                           <mx:ColumnSeries displayName="SUV" xField="year"
                                  yField="suv"/>
                           <mx:ColumnSeries displayName="Van" xField="year"
                                  yField="van"/>
                    </mx:series>
             </mx:ColumnChart>
      </mx:Panel>
</mx:Application>
```

Running and Debugging a Flex Application

Now that we've written all that code, it's time to run the application. Running an application in Flex Builder is a very simple process. Just like a Flash application, a Flex application consists of a compiled Flash movie (.swf), which is then embedded into a browser-friendly HTML page. Flex Builder handles initiating a command to the Flex compiler, which then compiles all MXML and ActionScript into an .swf. Flex Builder also automatically generates an HTML page with the Flash movie embedded in it.

Here are several different methods to run your application from Flex Builder:

A. From the main menu, select Run → Run As → Flex Application.
-or-

B. Right-click the main.mxml file in the Navigator. Select Run as → Flex Application

Figure 10.

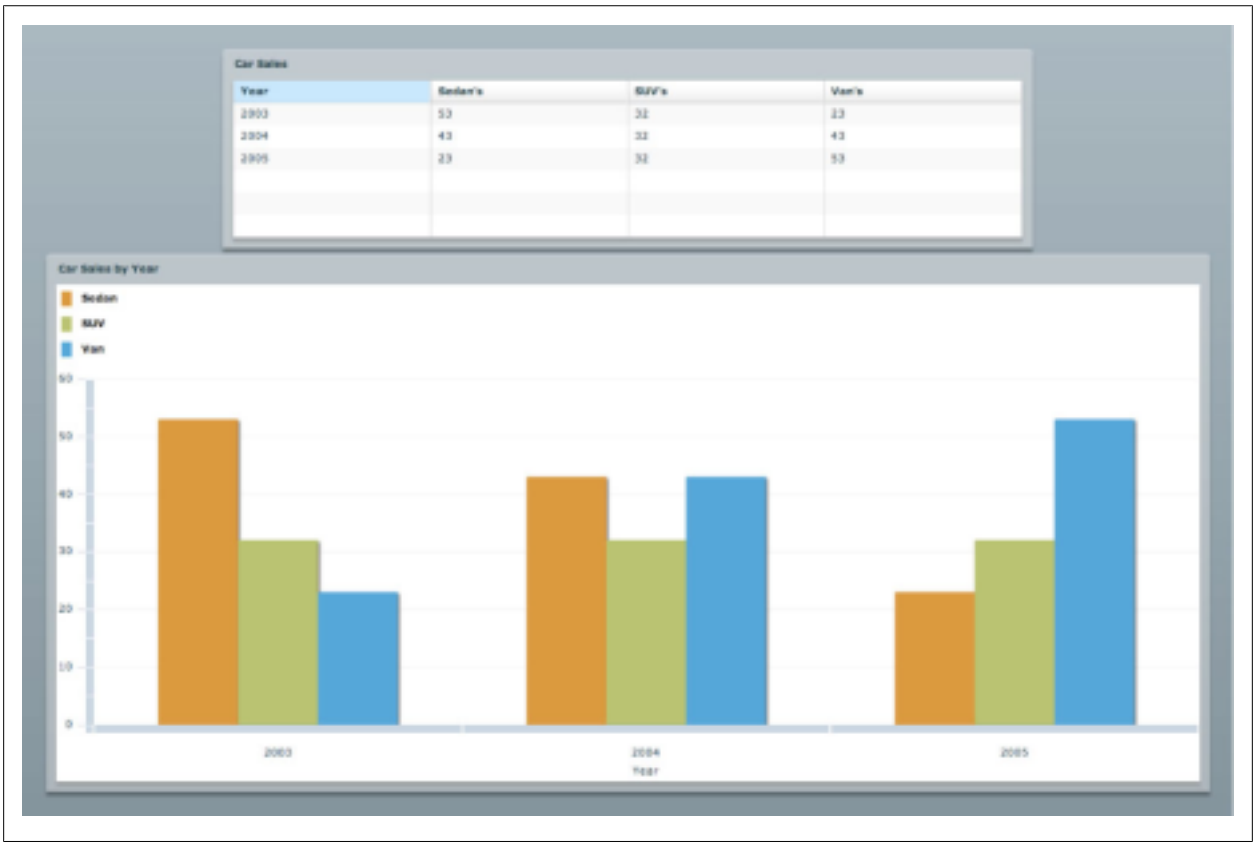

Figure 11.

-or-

C. Press the button that's highlighted below, located in the tool bar that lies at the top of the Flex Builder window (**Figure 10**).

After performing one the previous actions, your default browser should open up with the application running inside of it. It should look like **Figure 11**.

Here, you see a working RIA, with just a few lines of code. This application presents our data in a way that people will understand, and it was easy to code. Try updating the data by making a change to the DataGrid. The graph updates automatically, and we don't have to email a single event handler to process the change, thanks to data binding.

This application works well, but it's not especially pretty. Let's add another set of charts, some styles, and a few effects to spruce it up.

Adding Additional Charts

First, we'll add a set of pie charts. Let's say that our car company wants to display the results for each car type in a separate pie chart. Flex does a great job at helping us separate views from data, so this is easily accomplished.

Let's place the PieCharts on the right of our current chart with each chart in its own separate panel. We'll begin by adjusting our layout to accommodate our new set of charts. First, wrap the lower panel in an HBox. Set the HBox's width to 800 to match the width of our DataGrid above and set the height to 500. Then, after the panel, in the HBox, add a VBox. You can do this using either the Display view or by adding the code by hand. It should look like this when completed:

```
<mx:HBox width="800" height="500">
	<mx:Panel width="100%" height="100%" title="Car Sales by Year">
		<mx:Legend dataProvider="{myChart}" color="0x000000"/>
		<mx:ColumnChart width="100%" height="100%" id="myChart"
			dataProvider="{ chartData }" >
			<mx:horizontalAxis>
				<mx:CategoryAxis categoryField="year" title="Year" />
			</mx:horizontalAxis>
			<mx:series>
				<mx:ColumnSeries displayName="Sedan" xField="year"
yField="sedan"/>
				<mx:ColumnSeries displayName="SUV" xField="year"
yField="suv"/>
				<mx:ColumnSeries displayName="Van" xField="year"
yField="van"/>
			</mx:series>
		</mx:ColumnChart>
	</mx:Panel>
	<mx:VBox width="30%" height="100%">
	</mx:VBox>
</mx:HBox>
```

Next, let's add the PieCharts. Start by adding a single PieChart to the VBox. Add a new Panel to the VBox, and set its title to Sedans. Now add a PieChart to the panel. Add a PieSeries to the PieChart with the field set to sedan, like this:

```
<mx:Panel width="100%" height="100%" title="Sedans">
	<mx:PieChart width="100%" height="100%" dataProvider="{chartData}">
		<mx:series>
			<mx:PieSeries field="sedan"/>
		</mx:series>
	</mx:PieChart>
</mx:Panel>
```

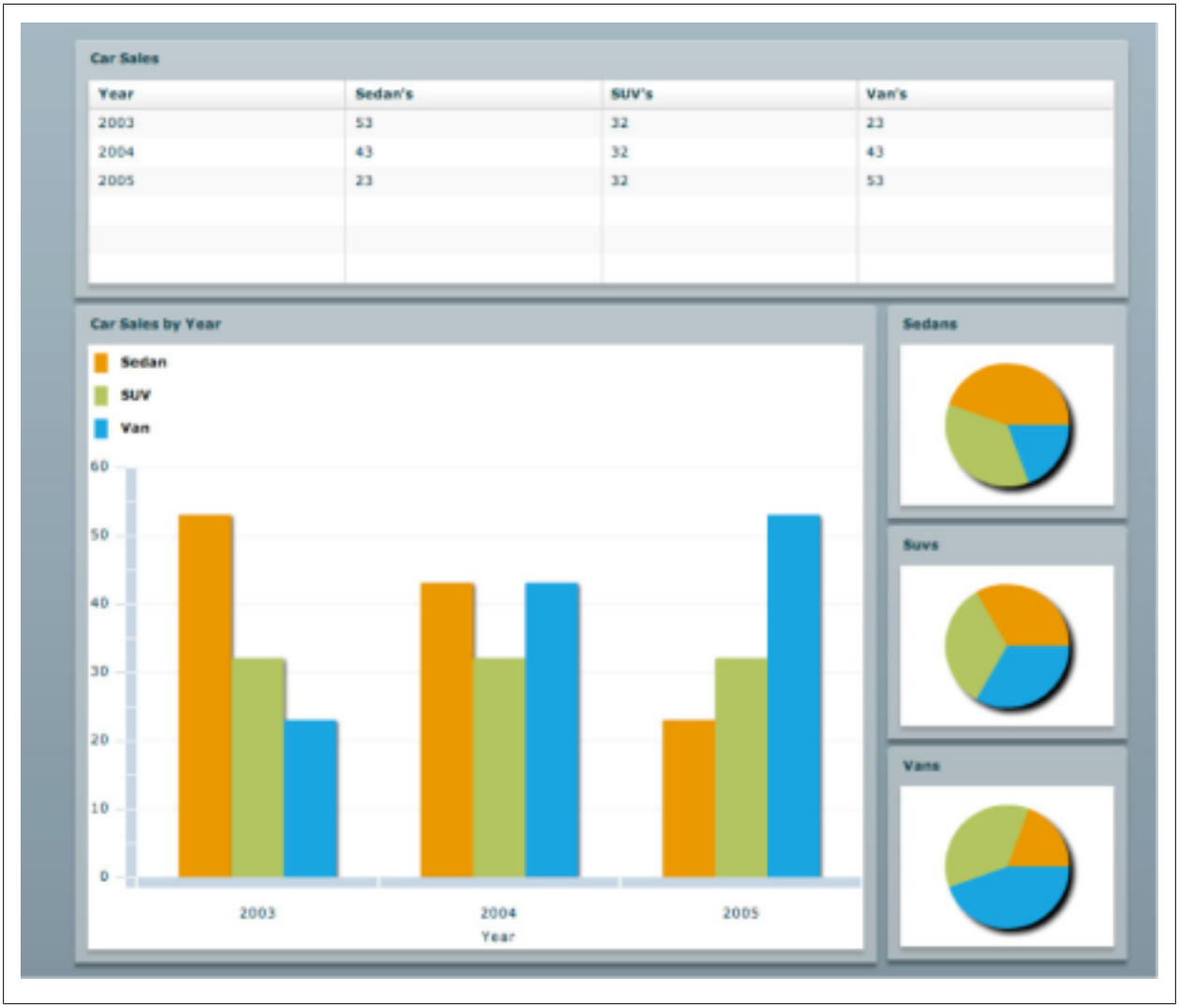

Figure 12.

Repeat those steps for both SUVs and Vans, adding them after the Sedans. Run the application. It should now look like **Figure 12**.

Already, the application's looking much better, but there are a few improvements yet to make. You may have noticed our pie charts don't have any labels specifying the year to which each slice of the pie corresponds. That's because Flex can't determine which field in the data to use in setting our label. To set the PieChart labels, we need to use a labelFunction. The PieChart has a labelFunction field, and all we need to do is write a function and point this attribute to it.

Back in the Script tag, add the following code:

```
private function displayYear(data:Object, field:String,
       index:Number, percentValue:Number):String {
              return data.year;
}
```

You can modify labels in any way, using any data available. All we really need is to return the year field on our data object. For more on using label functions, see Adobe Flex Documentation.

Adding Style

Another problem with our current view is that the colors are misleading. Flex has a set of colors it uses for each series of data by default, and it repeats this in all charts across an application. In our application, this feature is misleading, because the colors used to delineate different car types in our ColumnChart are the same colors used to delineate different years in our PieCharts.

Unfortunately, Flex doesn't provide an easy way to specify colors for a ColumnChart using style sheets, so we need to change the colors by hand. This isn't difficult, though. First, we need to change each of our ColumnSeries tags in the ColumnChart to have both an open and an end tag, rather than the single line tag we used previously. Next, nest a fill object inside of the ColumnSeries object, and then add a SolidColor object with the color you desire. I've chosen to do Sedans in red, SUVs in blue, and Vans in green. The code for the ColumnChart now looks like this:

```
<mx:ColumnChart width="100%" height="100%" id="myChart" dataProvider="{ chartData }" >
	<mx:horizontalAxis>
		<mx:CategoryAxis categoryField="year" title="Year" />
	</mx:horizontalAxis>
	<mx:series>
		<mx:ColumnSeries displayName="Sedan" xField="year" yField="sedan">
			<mx:fill>
				<mx:SolidColor color="#CC0000" />
			</mx:fill>
		</mx:ColumnSeries>
		<mx:ColumnSeries displayName="SUV" xField="year" yField="suv">
			<mx:fill>
				<mx:SolidColor color="#0000CC" />
			</mx:fill>
		</mx:ColumnSeries>
		<mx:ColumnSeries displayName="Van" xField="year" yField="van">
			<mx:fill>
				<mx:SolidColor color="#00CC00" />
			</mx:fill>
		</mx:ColumnSeries>
	</mx:series>
</mx:ColumnChart>
```

Now, let's add the rest of our style changes.

Flex allows a number of different ways to apply styles. I prefer to reference them in a separate file, since this helps keep separation between our data and our view, and because doing so keeps with the best practices for traditional web design.

Let's start by adding a new CSS file to our project. Right-click the project in Flex Builder, and then select New → CSS file. Name the file whatever you want. Next, we need to reference our CSS file from the Flex application. To do this, add a style tag to the application just under the script block, like this:

```
<mx:Style source="Sample.css"/>
```

Back in our style sheet, let's add some styles to the entire application. You add styles in the same way you would with traditional web applications. A full discussion of style sheets is beyond the scope of this document, but a number of simple guides are readily available on the internet.

We'll set the text color to black, to ensure that all of the text in the application stays black. We'll also change the background from the teal green Adobe uses by default to a shade of gray. Flex, by default, applies the background application color style as a gradient, so this makes for a very nice effect:

```
Application{
       color : #000000;
       backgroundColor: #333333;
}
```

Next, let's style our data grid:

```
DataGrid {
   backgroundColor: #0066ff;
   alternatingItemColors: #cccccc, #e9e8e8;
   verticalGridLineColor: #00ccff;
   rollOverColor: #a7dffc;
   dropShadowEnabled: true;
   headerStyleName: "mydataGridHeaderStyle";
}
.mydataGridHeaderStyle {
   fontWeight: bold;
}
```

I made these styles using Adobe's style explorer, which allows developers to graphically pick the styles they'll use in their application. The styles are exported in CSS by the Style Explorer, which is itself a Flex application. You can find the style explorer at ***http://examples.adobe.com/flex2/consulting/styleexplorer/Flex2StyleExplorer.html***.

Finally, let's add styles to our PieCharts. Unlike ColumCharts, we can specify the colors used by PieCharts in CSS. Since we have Sedans red in our ColumnChart, I'm going to use three different shades of red in our Sedans PieChart. Similarly, I'm going to use three shades of blue for SUVs, and three shades of green for Vans. The CSS looks like this:

```
PieSeries{
	labelPosition: outside;
}

.redPie{
	fills:#FF0000, #880000, #330000;
}
.bluePie{
	fills:#0000FF, #000088, #000033;
}
.greenPie{
	fills:#00FF00, #008800, #003300;
}
```

Finally, back in our application, we need to modify the PieSeries tags to reference our new styles. This is done by setting the styleName attribute for each PieSeries accordingly. Here's an example of how this looks for the Sedans series:

```
<mx:PieChart width="100%" height="100%" dataProvider="{chartData}">
	<mx:series>
		<mx:PieSeries styleName="redPie" field="sedan"
labelFunction="displayYear"/>
	</mx:series>
</mx:PieChart>
```

Repeat this same change for both SUVs and Vans, using the bluePie and greenPie styles respectively, and our styling is complete. The application should now look like **Figure 13**.

With just a few simple style additions, our application looks compelling and interesting.

Adding Transitions

Let's make one final update to our application to enhance user experience: a simple transition. Right now, when our data changes, the graph suddenly changes without warning. While most people are probably used to this sort of behavior, they'll appreciate and enjoy using our application more if we make this change a little more interesting, and Flex makes this easy.

Figure 13.

Flex comes with a number of different, built-in transition functions. One of these functions is the SeriesSlide, which slides data in and out of a chart's boundaries. Let's declare two SeriesSlide transition objects in our application: a transition in, and a transition out. Add the following tags just below our style declaration:

```
<mx:SeriesSlide id="slideIn" direction="up" duration="500" />
<mx:SeriesSlide id="slideOut" direction="down" duration="300" />
```

Now we need to add these transitions to the series objects in our application. Set the showDataEffect attribute on each series to slideIn and the hideDataEffect attribute to slideOut. These attributes in the mxml are really events fired by the series object when its data is updated. By setting them to transition objects, Flex implictly creates a set of event listeners and handlers which use the transition object to animate the data moving in and out of our chart. Set the attributes like this:

```
<mx:ColumnSeries displayName="Sedan" xField="year" yField="sedan"
showDataEffect="slideIn" hideDa-
taEffect="slideOut">
```

Set these attributes in all of the ColumnSeries and PieSeries objects, and run the application. Make some changes to the data in the data grid, and watch the application respond. Now people will feel like our application responds to them in a way that makes sense and makes using the application more enjoyable, but at the same time our transition isn't so complicated that it distracts folks from the data our application is built to enjoy.

Finally, here's how the entire application file should look:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
       layout="vertical"
       horizontalAlign="center"
       initialize="init()" width="100%" height="100%">
       <mx:Script>
              <![CDATA[
                     import mx.collections.ArrayCollection;

                     [Bindable]
                     private var chartData:ArrayCollection;

                     private function init():void {
                            chartData = new ArrayCollection();
                            chartData.addItem( {year:2003, sedan:53, suv:32,
van: 23});
                            chartData.addItem( {year:2004, sedan:43, suv:32,
van: 43});
                            chartData.addItem( {year:2005, sedan:23, suv:32,
van: 53});
                     }
                     private function displayYear(data:Object, field:String,
index:Number,
percentValue:Number):String {
             return data.year;
             }
             ]]>
       </mx:Script>

       <mx:Style source="Sample.css"/>

       <!-- Define custom charting effects to run when the chart
dataProvider gets updated. -->
       <mx:SeriesSlide     id="slideIn" direction="up" duration="500" />
       <mx:SeriesSlide     id="slideOut" direction="down" duration="300" />
```

```
        <mx:Panel width="800" title="Car Sales">
                <mx:DataGrid width="100%" dataProvider="{ chartData }" editable="true">
                        <mx:columns>
                                <mx:DataGridColumn headerText="Year" dataField="year"
editable="false"/>
                                <mx:DataGridColumn headerText="Sedan's"
dataField="sedan"/>
                                <mx:DataGridColumn headerText="SUV's" dataField="suv"/>
                                <mx:DataGridColumn headerText="Van's" dataField="van"/>
                        </mx:columns>
                </mx:DataGrid>
        </mx:Panel>
        <mx:HBox width="800" height="500">
        <mx:Panel width="100%" height="100%" title="Car Sales by Year">
                        <mx:Legend dataProvider="{myChart}" color="0x000000"/>

                        <mx:ColumnChart width="100%" height="100%" id="myChart"
dataProvider="{ chart-
Data }" >
                                <mx:horizontalAxis>
                                        <mx:CategoryAxis categoryField="year"
title="Year" />
                                </mx:horizontalAxis>
                                <mx:series>
                                        <mx:ColumnSeries displayName="Sedan"
xField="year" yField="sedan"
showDataEffect="slideIn" hideDataEffect="slideOut">
                                                <mx:fill>
                                                <mx:SolidColor color="#CC0000" />
                                        </mx:fill>
                                        </mx:ColumnSeries>
                                        <mx:ColumnSeries displayName="SUV" xField="year"
yField="suv"
showDataEffect="slideIn" hideDataEffect="slideOut">
                                                <mx:fill>
                                                <mx:SolidColor color="#0000CC" />
                                        </mx:fill>
                                        </mx:ColumnSeries>
                                        <mx:ColumnSeries displayName="Van" xField="year"
yField="van"
showDataEffect="slideIn" hideDataEffect="slideOut">
                                                <mx:fill>
                                                <mx:SolidColor color="#00CC00" />
                                        </mx:fill>
                                        </mx:ColumnSeries>
                                </mx:series>
                        </mx:ColumnChart>
                </mx:Panel>
                <mx:VBox width="30%" height="100%">
```

```
                    <mx:Panel width="100%" height="100%" title="Sedans">
                            <mx:PieChart width="100%" height="100%"
dataProvider="{chartData}">
                                    <mx:series>
                                            <mx:PieSeries styleName="redPie"
field="sedan" labelFunc-
tion="displayYear" showDataEffect="slideIn" hideDataEffect="slideOut"/>
                                    </mx:series>
                            </mx:PieChart>
                    </mx:Panel>
                    <mx:Panel width="100%" height="100%" title="Suvs">
                            <mx:PieChart width="100%" height="100%"
dataProvider="{chartData}">
                                    <mx:series>
                                            <mx:PieSeries styleName="bluePie"
field="suv" labelFunc-
tion="displayYear" showDataEffect="slideIn" hideDataEffect="slideOut"/>
                                    </mx:series>
                            </mx:PieChart>
                    </mx:Panel>
                    <mx:Panel width="100%" height="100%" title="Vans">
                            <mx:PieChart width="100%" height="100%"
dataProvider="{chartData}">
                                    <mx:series>
                                            <mx:PieSeries styleName="greenPie"
field="van" labelFunc-
tion="displayYear" showDataEffect="slideIn" hideDataEffect="slideOut"/>
                                    </mx:series>
                            </mx:PieChart>
                    </mx:Panel>
            </mx:VBox>
            </mx:HBox>

</mx:Application>
```

And that's it. In just 93 lines of code and a few styles, we've built an entire application that's easy to use and conveys our data in a way that's easily understood.

EffectiveUI Authors

Anthony Franco, President
blog: ***http://anthonyfranco.wordpress.com***
web site: **www.effectiveUI.com** [***http://www.effectiveui.com***]
RJ Owen, Flex Architect
John Wright, Flex Architect
Drew McLean, Flex Architect