macromedia® white paper

Macromedia Flex: the Presentation Tier Solution for Enterprise Rich Internet Applications

November 2003

Macromedia, Inc.
600 Townsend Street
San Francisco, CA 94103
415–252–2000

Contents

The user experience for browsing web content is excellent, however, it has become evident that the user experience of interacting with moderately complex web applications is generally less satisfying. The web's page-based model and lack of client-side intelligence makes transactions that should be simple, like online shopping, too difficult for many users; more complex user interactions, like those in traditional client-server and desktop applications, are almost impossible. This technology has led to a generation of web applications that are difficult to use, costly to support, and, in many cases, largely ineffective.

In response to the need for better user experiences, a new category of Internet applications is emerging: Rich Internet Applications (RIAs). RIAs combine the responsiveness and interactivity of desktop applications with the broad reach and ease of distribution of web applications. RIAs drive increased return on investment (ROI) by simplifying and improving the user interaction of web applications. RIAs make it possible to deliver applications that provide a richer, more interactive, and responsive user experience.

Macromedia is widely recognized as a leader in the emerging market for RIAs. The Macromedia Flash client is available now on 98% of browsers, so almost everyone can use RIAs based on Flash. The Flash client has a lightweight, cross-platform, cross-device runtime that is neutral for both application server development and deployment platforms and for client operating systems (Windows, MacOS, and Linux). Applications that target the Flash Player can run identically on all the major operating systems today, and are backward compatible with previous versions of Windows and the Macintosh OS.

Macromedia Flex (previously code-named *Royale*) is a new server product from Macromedia that makes the full power of RIAs accessible to enterprise application developers. Flex has a standards-based architecture that complements current enterprise developer tools, methodologies, and design patterns. This white paper provides an overview of the Macromedia Flex product.

Evolution of the Presentation Tier

Users and IT organizations are increasingly frustrated with the limitations of HTML as a way to deliver applications on the web. These technical and business forces are driving the emergence of a new solution that overcomes the limitations of HTML, while preserving the centralized deployment-management model and adding the richness and control of desktop applications.

Benefits of Great User Experience

Providing a superior customer experience is a key differentiator for most successful companies, and the explosion of the Internet has increased the percentage of customer and partner interactions that occur online. This means that online experiences are replacing personal interactions and this trend is rapidly increasing. While on the surface, this trend is potentially beneficial, in terms of driving down costs however, the benefits derived by any individual organization are uncertain. This is because switching costs on the Internet are low; thus, the quality of the user experience becomes one of the key factors in building and maintaining online relationships.

Great user experience has always been the hallmark of the most dominant brands in business. Organizations that understand the benefit of great user experiences will focus their efforts on attracting and retaining online relationships.

Reach versus Richness

As the industry has progressed through the major transitions in system architecture, the presentation capabilities of the client have ebbed and flowed. Figure 1 highlights the evolution of the application experience offered by each stage of computing, starting with mainframes and concluding with the emergence to Rich Internet Applications (RIAs).

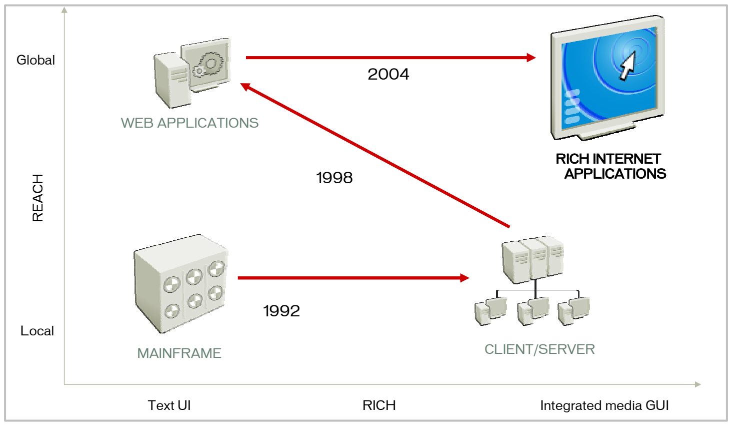


Figure 1: The Evolution of Rich Internet Applications

Figure 1 shows the following changes in client applications:

- Interactive applications started with host-based computing, typically mainframes and minicomputers. The business driver for this stage of computing was back-office automation (for example, payroll). The applications were distributed locally (intracompany, private network), and richness of the user interface was limited to text.
- The client-server wave followed and was driven by the need to provide access to information and applications to the department level within the organization. Application reach was still available locally in the organizations, but the richness of the applications was substantially increased through the graphical user interface and the use of client-side processing.
- Client-server applications were followed by web applications, which were driven by the requirement to provide low-cost and broad-reach access to information and applications. The global reach of web applications, along with a host-based model for centralized application administration and management, addressed the key limitations of the client-server model—but at a steep cost in user experience. From a processing perspective, the web application model transformed client computers into dumb terminals. Key elements of user interaction that are necessary to provide great user experience, such as direct manipulation (for example, drag and drop), client-side processing (for example, client-side sorting, filtering, and data validation), and local storage, were lost.
- Recently, the RIA model has gained momentum, as organizations realize that this new model delivers significant business benefits, productivity enhancements, and cost reductions. RIAs combine the responsiveness and interactivity of desktop applications with the broad reach and ease of distribution of web-based applications. RIAs are the next step in the evolution of system architecture, and are designed to maximize *reach* and *richnes*s.

Evolution of Tools, Standards, and Patterns

Enterprises require products that incorporate the tools, standards, and patterns that are used within their application development and deployment cycle. These organizations are demanding that new technologies, especially those on the presentation tier, meet or exceed the following requirements:

- **Provide a familiar programming model:** Enterprise developers have become familiar with object-oriented languages (such as Java and C#) for business logic, and tag-based languages (such as JSP, ASP, and CFML) for user interface development. New products must build on these existing models to leverage skills and low-cost adoption.
- **Use existing infrastructure:** Organizations have invested heavily in application server technology. Research from leading industry analysts points to a future in which the majority of companies employ J2EE and .NET within their organizations. Using and complying with this infrastructure are also requirements within most organizations.

- **Adopt standard protocols and APIs:** One of the many positive results of the web has been the adoption of a broad spectrum of standards across the entire technology stack. This includes, but is not limited to, industry standards, such as HTML/HTTP(S), XML, SOAP/web services, CSS, SVG, as well as J2EE and .NET APIs. Incorporating these standards, where appropriate, is a requirement for most organizations.
- **Retain existing tools:** Key to the adoption of presentation tier solutions by the developer community is ensuring that the developer can use their existing editor or IDE to write their application. This drives the requirement that a presentation tier solution allows code authoring in leading IDEs, like Eclipse, Borland JBuilder, JetBrains IntelliJ IDEA, Microsoft Visual Studio, as well as leading web application development products like Macromedia Dreamweaver.
- **Support key design patterns:** Enterprise developers also tend to work with different design patterns; for example, the model-view-controller (MVC) pattern that is prevalent in J2EE and .NET enterprise application development. These patterns must be able to be implemented within a presentation tier solution.

As a result of the requirements for architectural consistency and standards-bases solutions, RIAs have been somewhat inaccessible to most enterprise application developers–until now.

The Flex Solution

The Macromedia Flex presentation server is a new Macromedia product that makes Rich Internet Applications accessible to development organizations that require a solution that leverages their existing skills, patterns, and infrastructure. The Flex presentation server sits in the presentation tier of an organizations' N-tier application model, and augments the existing HTML generation with executable code that runs on the client.

Flex Applications

The main difference between Flex applications and traditional HTML applications is the ability to offload processing that is best suited to run on the client, such as field validation, data formatting, sorting, filtering, tool tips, integrated video, behaviors, and effects.

Flex makes it possible for developers to create applications that provide users with immediate responses, smooth transitions between states and displays, and continuous workflow without unnecessary interruptions.

Flex Development

The Flex development model will be familiar to developers who have worked with JSP, ASP/ASP.NET or similar technologies. Flex uses a tag-based language for user interface definition and a similar server-side just-in-time compilation and caching for rendering the client application. The Flex development environment consists of a rich library of user interface components, an XML-based markup language to declaratively lay out these components, and an object-oriented programming language to handle the way a user interacts with the application.

Flex produces RIAs that are rendered on the client using the Macromedia Flash Player, but are developed using industry standards and a familiar development paradigm.

Flex Deployment and Administration

Deploying a Flex server on the J2EE platform is very straightforward, because Flex is a native Java application. Flex application deployment on the J2EE platform is handled with Java web archive (WAR) files. From an administration perspective, the Flex XML Schema- and file-based application model mean that Flex applications can be incorporated easily into the administration and application lifecycle tools already in use in the enterprise. While executing in the Flash Player, the Flex application can interact with server-side functionality, such as databases, SOAP web services, and other server-side services.

Flex and the N-Tier Model

In addition to augmenting the current presentation tier, Flex does not require any changes to current business and integration tiers. The Flex presentation server executes within the application server, and provides integration and management capabilities for Flex applications. The Flex integration capabilities make it easy to leverage existing code and information using web services, direct database access, Enterprise JavaBeans (EJBs), or other methods. Flex also coexists with presentation tier tools, and you can quickly add Flex applications or components to Java Server Pages (JSP).

Flex Product Overview

Figure 2 shows an overview of the Flex server architecture. The Flex presentation server deploys on the existing J2EE or .NET application server and consists of the Flex application framework and the Flex presentation runtime services.

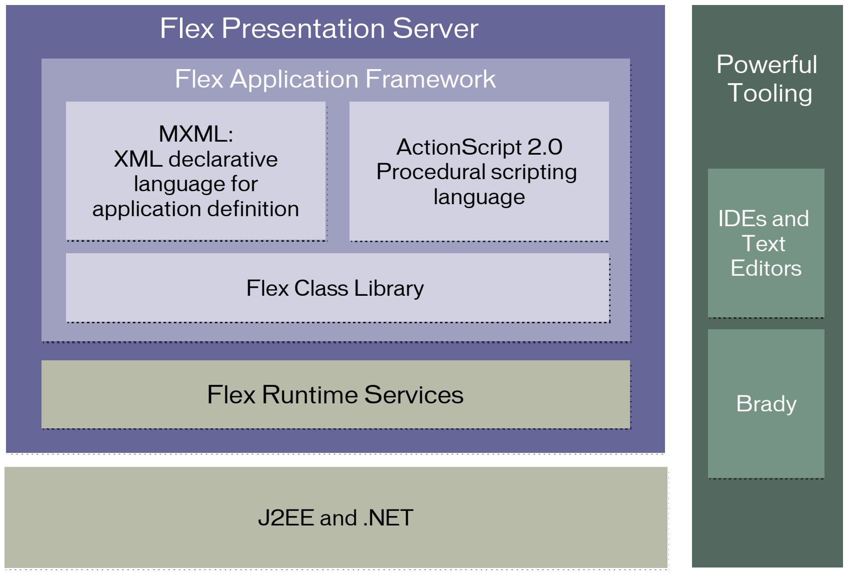

Flex Presentation Server
Flex Application Framework
MXML: XML declarative language for application definition
ActionScript 2.0 Procedural scripting language
Flex Class Library
Flex Runtime Services
J2EE and .NET
Powerful Tooling
IDEs and Text Editors
Brady

Figure 2: Flex Architecture

Flex Application Framework

The Flex application framework contains the Flex markup language (code-named *MXML*), ActionScript 2.0, and the underlying Flex class library.

The Flex developer works within the application framework using a combination of the MXML language and ActionScript 2.0. MXML is used to declaratively define the structure and contents of a Flex application. ActionScript 2.0 is the procedural language that is used to perform runtime control and data processing in your Flex application.

The Flex application framework also includes a class library, which contains Flex components, managers and behaviors. With the Flex component-based development model, developers can incorporate pre-built components, create new components, or combine pre-built components into composite components.

MXML: The Flex Markup Language

Like HTML, MXML is a markup language that describes user interfaces that provide content and functionality. Unlike HTML, MXML provides declarative abstractions for presentation tier logic and bindings between the user interface and server-side data. MXML helps maximize developer productivity and application reusability by cleanly separating presentation and business logic concerns.

MXML development is based on the same iterative process used for other types of web application files such as HTML, Java Server Pages (JSP), Active Server Pages (ASP), and ColdFusion Markup Language (CFML). Developing an MXML application is as easy as opening a text editor, typing some XML tags, saving the file, and opening the file's URL in a web browser.

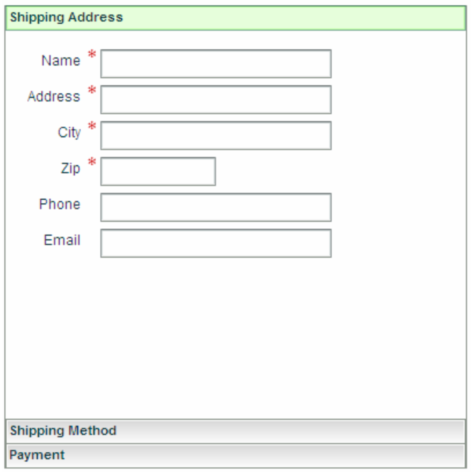


Figure 3: An accordion-style form, created with MXML

```
<!-- MXML source code for the Accordion shown in Figure 3-->
?xml version="1.0" encoding="iso-8859-1"?>
<mx:Application width="400" height="400" xmlns:mx="http://www.macromedia.com/2003/mxml">
 <mx:model id="checkoutModel">
  <checkout>
   <name>{name.text}</name>
   <address>{address.text}</address>
   <city>{city.text}</city>
   <zip>{zip.text}</zip>
   <email>{email.text}</email>
   <phone>{phone.text}</phone>
   <cardHolder>{cardHolder.text}</cardHolder>
   <cardNumber>{cardNumber.text}</cardNumber>
   <expirationMonth>{cardExpirationMonth.value}</expirationMonth>
   <expirationYear>{cardExpirationYear.value}</expirationYear>
  </checkout>
 </mx:model>

 <mx:ZipCodeValidator field="checkoutModel.checkout.zip"/>
 <mx:EmailValidator field="checkoutModel.checkout.email"/>
 <mx:PhoneNumberValidator field="checkoutModel.checkout.phone"/>
 <mx:CreditCardValidator field="checkoutModel.checkout.cardNumber"/>

 <mx:Accordion widthFlex="1" heightFlex="1">

  <mx:Form label="Shipping Address">

   <mx:FormItem label="Name" required="true">
    <mx:TextInput id="name" width="200"/>
   </mx:FormItem>
```

MXML Code for Accordion Form in Figure 3

Also, MXML files are ordinary XML files, so you have a wide choice of development environments. You can develop within a simple text editor, a dedicated XML editor, or an integrated development environment (IDE) that supports text editing. Depending on the capabilities of your editor, you may also have structured editing, code coloring, and code hinting, because MXML conforms to the W3C XML Schema definition.

ActionScript 2.0

The procedural programming language for Flex development is ActionScript 2.0, a strongly typed, object-oriented programming language. ActionScript 2.0 is similar to the core JavaScript programming language and is based on the JavaScript standard (ECMAScript profile 262 edition 4).

Flex developers use ActionScript 2.0 to enhance the user experience of the applications whenever appropriate. For instance, developers use ActionScript 2.0 to define event listeners and handlers, set or get the values of component properties, and handle call back functions.

Flex Class Library

Flex includes a rich class library, which contains Flex components (containers, and controls), data binding, behaviors, and other features. All Flex user interface components follow the Macromedia Halo and MX elements, which maximizes the effectiveness and consistency of user experiences.

Flex Components

The component-based development model is used to create Flex applications. Developers can use the pre-built components included with Flex, they can extend components to add new properties and methods, and they can create new components as required by the application.

Flex components are extremely flexible and provide the developer with great control over the component's appearance, how the component reacts to user interactions, the font and font size of any text included in the component, the size of the component in the application, and many other characteristics.

Flex components support the following characteristics:

- **Events:** Application or user actions that require a component response.
- **Behaviors:** Visible or audible changes to the component triggered by an application or user action.
- **Skins:** Symbols that control a component's appearance.
- **Styles:** Set of characteristics, such as font, font size, and text alignment.
- **Size:** Height and width of a component. All components have a default size.

The Flex class library supplies two types of components: containers and controls. When developers build an application using Flex, they describe its user interface using controls and containers. *Controls* are user interface components that handle user interactions and display data that users can manipulate directly through the use of a Button, DataGrid, or TreeControl. A *container* defines a region of Flash Player drawing surface, and controls the layout for everything in the container, including other containers and controls. Examples of containers are a Form container used for data entry, a Box, and a Grid.

Most controls have the following characteristics:

- **MXML API** for declaring the control and the values of its properties and events
- **ActionScript API** for calling the control's methods and setting its properties and events at runtime
- **Customizable look and feel** using styles, skins, and fonts

Flex Behaviors

The Flex class library also provides pre-built behaviors. These pre-built behaviors make it easy for developers to add animation, motion, and sound to their application in response to some user or programmatic action. For example, a developer can use behaviors to cause a dialog box to bounce slightly when it receives focus, or to play a sound when the user enters an invalid value.

A *behavior* is a combination of a trigger paired with an effect. A *trigger* is an action, such as a mouse click on a component, a component getting focus, or a component becoming visible. An *effect* is a visible change to the component occurring over a period of time, measured in milliseconds. Examples of effects are fade, move, resize, pause, or wipe transition.

Developers can define multiple effects for a single trigger, as well as customize effects or make composite effects to meet the specific needs of their application.

Flex Presentation Runtime Services

Flex includes a suite of services for RIA compilation and caching, enterprise resource integration, and deployment-runtime needs. All Flex services are designed to minimize redundancy and to fully exploit existing enterprise resources; for example, server-side application logic, authentication, and session management are provided through integration with the underlying J2EE or .NET platforms, instead of introducing redundant services layers.

Complementing J2EE and other existing technologies, Flex is a completely native application on the J2EE application server, and integrates with the platform for web services, data and application integration, security, and other resources. Flex works with J2EE application servers, including IBM WebSphere, BEA WebLogic, Macromedia JRun, and the servlet container Apache Tomcat.

Flex application compilation is done in a just-in-time model, with no user experience disruption. A Flex application is compiled when it is first referenced and is then cached for subsequent invocations; Flex applications are automatically recompiled if any related files are updated.

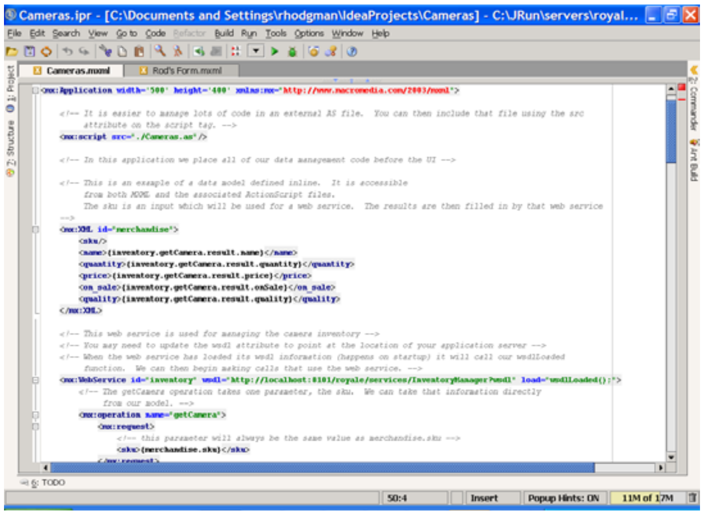
The Flex integration-related services include the following:

- **Web services proxy:** The server-side web services proxy extends the native Flash model by securely supporting web services outside the application's originating domain.
- **Server-side Java object access:** Access is available for application, data, and directory integration, including Java objects, JavaBeans, EJBs, and JNDI objects. The forthcoming .NET release of Flex will support equivalent .NET components and services.
- **Shared session support:** Flex application sessions can be shared between HTML and Flash application contexts.
- **Authentication:** Flex supports J2EE single sign-on services for authentication.
- **Flash Player detection and updating:** Flex requires Flash Player 7 in order to take advantage of its security and performance features. Flex includes services for detecting and optionally updating Flash Player installations, a pivotal requirement for enterprises that have "lock-down" policies for client device configurations.
- **Deferred instantiation:** Flex includes several runtime options to optimize interactivity by loading application elements on demand or under developer control, rather than all at once.

Flex Authoring

Flex application developers can work with several classes of tools. The first tool class includes any generic text editor; for example, vi, emacs or Microsoft Notepad. MXML files are normal text files, so developers can use any text editor The Flex text file-based approach also makes Flex applications easy to manage with existing source code control systems.

Integrated development environments (IDEs) that support the W3C XML Schema Definition Language 1.0 comprise the second class of tool. Examples of these tools include Eclipse, IBM WebSphere Studio Application Developer, Borland JBuilder, JetBrains IntelliJ IDEA, and Altova xmlspy. Because Flex conforms to the W3C XML Schema Definition, tools in this class automatically support MXML code hinting, color coding, and other useful IDE features. Figure 4 shows a Flex application being edited in IntelliJ IDEA.


Cameras.ipr - [C:\Documents and Settings\rhodgman\IdeaProjects\Cameras] - C:\JRun\servers\royal...
File Edit Search View Go to Code Refactor Build Run Tools Options Window Help
Cameras.mxml
Rod's Form.mxml
1: Project
2: Structure
2: Commander
Ant Build
6: TODO
50:4
Insert
Popup Hints: ON
11M of 17M

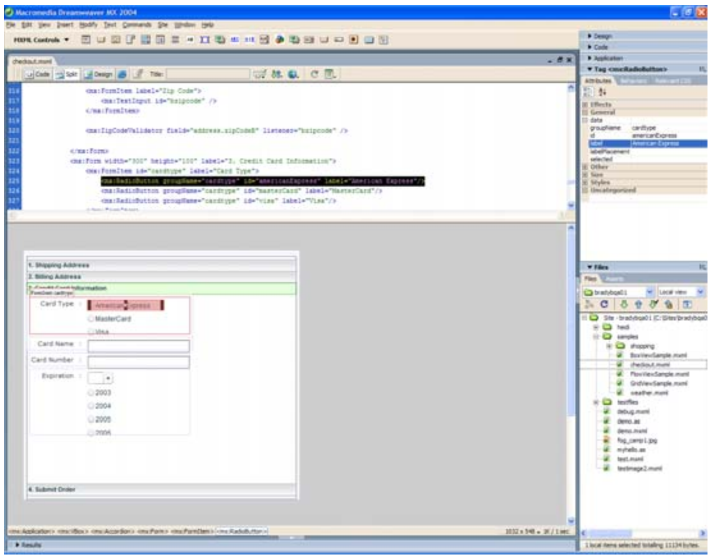
```
<mx:Application width='500' height='400' xmlns:mx="http://www.macromedia.com/2003/mxml">

    <!-- It is easier to manage lots of code in an external AS file.  You can then include that file using the src
         attribute on the script tag. -->
    <mx:script src="./Cameras.as" />

    <!-- In this application we place all of our data management code before the UI -->

    <!-- This is an example of a data model defined inline.  It is accessible
         from both MXML and the associated ActionScript files.
         The sku is an input which will be used for a web service.  The results are then filled in by that web service
    -->
    <mx:XML id="merchandise">
        <sku/>
        <name>{inventory.getCamera.result.name}</name>
        <quantity>{inventory.getCamera.result.quantity}</quantity>
        <price>{inventory.getCamera.result.price}</price>
        <on_sale>{inventory.getCamera.result.onSale}</on_sale>
        <quality>{inventory.getCamera.result.quality}</quality>
    </mx:XML>

    <!-- This web service is used for managing the camera inventory -->
    <!-- You may need to update the wsdl attribute to point at the location of your application server -->
    <!-- When the web service has loaded its wsdl information (happens on startup) it will call our wsdlLoaded
         function.  We can then begin making calls that use the web service. -->
    <mx:WebService id="inventory" wsdl="http://localhost:8101/royale/services/InventoryManager?wsdl" load="wsdlLoaded();">
        <!-- The getCamera operation takes one parameter, the sku.  We can take that information directly
             from our model. -->
        <mx:operation name="getCamera">
            <mx:request>
                <!-- this parameter will always be the same value as merchandise.sku -->
                <sku>{merchandise.sku}</sku>
            </mx:request>
```

Figure 4: Flex Application Development using IntelliJ IDEA

Products in the third class of tool used are more tightly integrated with the Flex class framework and other Flex features. One example is the future Macromedia product, code-named *Brady*, built on Dreamweaver MX 2004, (see Figure 5) Brady offers the following additional features to Flex developers: (in addition to the XML Schema-based integration features previously mentioned):

- **Visual layout:** Flex components appear on a Brady tool palette, and developers can build applications in a visual design pane without requiring MXML hand-coding.
- **Round-trip editing:** As Figure 5 shows, acclaimed round-trip editing capabilities in Dreamweaver apply to Flex applications, so changes made in the design pane are instantly reflected in the code pane (and vice versa).
- **Extended debugging and tracing/monitoring services:** Brady adds support for debugging client-side ActionScript and data connectivity in Flex applications.

- **Data binding and services:** Integration with the Flex server lets users browse the available data sources, such as web services, and bind user interface elements to data.



```
<mx:FormItem label="Zip Code">
    <mx:TextInput id="bzipcode" />
</mx:FormItem>

<mx:ZipCodeValidator field="address.zipCodeB" listener="bzipcode" />

</mx:Form>
<mx:Form width="300" height="100" label="3. Credit Card Information">
    <mx:FormItem id="cardtype" label="Card Type">
        <mx:RadioButton groupName="cardtype" id="americanExpress" label="American Express"/>
        <mx:RadioButton groupName="cardtype" id="masterCard" label="MasterCard"/>
        <mx:RadioButton groupName="cardtype" id="visa" label="Visa"/>
```

Figure 5: Flex Application Development using Brady

Macromedia is also working with other IDE vendors and organizations to bring the Brady level of Flex integration to other popular IDEs.

Flex and Standards

Macromedia contributes to and supports industry standards and Flex extends the following industry standards:

- J2EE and .NET platforms: The de facto standards for enterprise Internet applications.
- XML: The MXML language is XML-based and consistent with related standards initiatives, such as W3C XForms.
- XML Schema: The MXML schema conforms to W3C XML Schema Definition Language 1.0, an XML language for describing and constraining the content of XML documents. The benefits of supporting this standard are the integration (code coloring and code hinting) with the leading IDEs and usable with other XML tools such as XSLT.
- XML namespaces: The xmlns attribute in an MXML tag that specifies an XML namespace. XML namespaces let you refer to more than one XML tag vocabulary in the same XML document.
- The Flex event model: The event model is a subset of Document Object Model (DOM) Level 3 Events, a W3C working draft. DOM Level 3 defines an event system that allows platform and language-neutral registration of event handlers, describes event flow through a tree structure, and provides basic contextual information for each event.

- ECMAScript, in the form of Macromedia ActionScript: ActionScript is similar to the core JavaScript programming language and is based on the same standard (ECMAScript profile 262 edition 4) as JavaScript.
- Web services: Flex applications support web service requests and results that are formatted as Simple Object Access Protocol (SOAP) messages and are transported over Hypertext Transfer Protocol (HTTP).
- Cascading Style Sheets (CSS), supported by MXML styles.
- Java objects: MXML tags interact with server-side Java objects, including plain old Java objects (POJOS), JavaBeans, EJBs, and objects available through the Java Naming and Directory Interface (JNDI).
- SVG (Scalable Vector Graphics) for shapes and other vector drawings.
- SWF, Macromedia Flash file format, a published specification.

Summary

Macromedia Flex is a presentation tier solution for enterprise RIAs. It represents a significant milestone in the evolution of the presentation tier, and builds on the standards, tools, methodologies, and design patterns that are prevalent among enterprise application developers.

Macromedia is focused on delivering a lightweight, cross-platform, cross-device runtime that works across multiple application server platforms (J2EE and .NET) and client operating systems (Windows, Mac OS, Linux, etc). Applications that target the Flash Player can run on the major operating systems today, and are backward compatible with earlier versions of Windows and the Mac OS. The Flash client is available on 98% of browsers, so RIAs based on Flash can be used by anyone.

To learn more about Macromedia Flex, and how you can be involved with the latest developments, go to http://www.macromedia.com/software/flex.

Summary Feature List

Feature	Description
MXML	The Flex markup language that describes the user interfaces and provides declarative abstractions for presentation tier logic and bindings between the user interface and server-side data.
ActionScript 2.0	A strongly-typed, object-oriented programming language for procedural programming within Flex. ActionScript is similar to the core JavaScript programming language and is based on the same standard (ECMAScript profile 262 edition 4) as JavaScript.
Containers	Define a region of drawing surface and controls the layout for everything in the container, including other containers and controls.
Controls	A suite of form-based controls that handle user interactions and display data that can be directly manipulated by the user.
Data model	Defines data representation and storage including data binding validation and formatting.
Data communication services	A set of communication services for connecting with server-side data over XML.
Web Services	Flex provides MXML tags for interacting with web services that define their interfaces in a Web Services Description Language (WSDL) document available as a URL.
Remote Object Services	Flex provides MXML tags for interacting with server-side Java objects, including plain old Java objects (POJOS), JavaBeans, Enterprise JavaBeans (EJBs), and objects available through the Java Naming and Directory Interface (JNDI).
Behaviors	Enhanced user experience through animation, motion, sound, and effects.
Event model	An event system that allows registration of event handlers, describes event flow and provides basic contextual information for each event.
Style sheets	Cascading Style Sheets (CSS) is a standard mechanism for declaring text and visual styles in Flex.
Managers	Managers provide support for high-level Flex application tasks such as History management and ToolTip management.