Flash Builder and Flash Catalyst

The New Workflow

Steven Peeters

friendsof™
DESIGNER TO DESIGNER™
an Apress® company

Flash Builder and Flash Catalyst
The New Workflow

ISBN-13 (pbk): 978-1-4302-2835-6

ISBN-13 (electronic): 978-1-4302-2836-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Credits

Publisher and President:
Paul Manning

Lead Editor:
Ben Renow-Clarke

Technical Reviewer:
Peter Elst

Editorial Board:
Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, Matthew Moodie, Duncan Parkes, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh

Coordinating Editor:
Laurin Becker

Copy Editor:
Mary Behr, Sharon Terdeman

Compositor:
Bronkella Publishing LLC

Indexer:
BIM Indexing & Proofreading Services

Artist:
April Milne

Cover Designer:
Anna Ishchenko

Photo Credit:
Steven Peeters

Contents at a Glance

Contents at a Glance iii
Contents iv
About the Author xii
About the Technical Reviewer xiii
Acknowledgments xiv
Introduction xv
Chapter 1: Flex and AIR: Taking RIAs to the Next Level 1
Chapter 2: Flex 4 SDK: Overview of the New Features 39
Chapter 3: Flash Builder 4: The programming environment 71
Chapter 4: Flash Catalyst: bridging the gap 99
Chapter 5: Choosing the Best Workflow 133
Chapter 6: Case Study: An E-Commerce Site 187
Chapter 7: Managing RIA Projects 241
Chapter 8: Frameworks 259
Chapter 9: Best Practices 289
Index 319

Contents

Contents at a Glance iii

Contents iv

About the Author xii

About the Technical Reviewer xiii

Acknowledgments xiv

Introduction xv

Chapter 1: Flex and AIR: Taking RIAs to the Next Level 1

Taking Advantage of Flash Technology 1

- Mainframe 1
- Client-Server 2
- Web Applications 2
- Rich Internet Applications 3

Why Should You Use the Flash Platform? 4

Where is Flex Used? 5

- Flex on the Web 5
- Flex in the Enterprise 6
- Testing Flex applications 7

Connecting Applications to a Back End 7

- AMFPHP 8
- Zend AMF 10
- LiveCycle Data Services 12
- BlazeDS 25
- GraniteDS 28
- ColdFusion Data Services 31

Adobe AIR: Why Do We Want Desktop RIAs? 36

Summary 37

Chapter 2: Flex 4 SDK: Overview of the New Features 39
Component Architecture 39
View States 42
FXG 49
Spark-ling Effects 51
Property Effects 53
Transform Effects 54
3D Effects 54
Pixel-Shader Effects 54
Filter Effects 56
Advanced CSS Selectors 56
Use of Namespaces 56
Type Selector 58
Class Selector 59
Universal Selector 60
ID Selector 61
Descendant Selector 63
Pseudo Selector 65
Two-Way Binding 67
Summary 70
Chapter 3: Flash Builder 4: The programming environment 71
What is Flash Builder 4? 71
What Happened to Flex Builder? 72
The New Features 73
Backward Compatibility 73
Improved Project Properties 74
Package Explorer 76
Code Generation 77
Refactoring Support 79
View States in Source Mode 81
Improved Design Editor 84

Improved Back-End Connectivity 89
Data/Services Panel 89
ZendAMF Integration 91
Drag-and-Drop Binding 92
Debugging 94
Conditional Breakpoints 94
Changing Values at Runtime 96
Network Monitor 96
Call Hierarchy View 97
Summary 98
Chapter 4: Flash Catalyst: bridging the gap 99
Introduction to Flash Catalyst 99
Interface overview 100
The artboard 102
The Pages/States panel 102
The Layers panel 103
The Library panel 105
The Wireframe Components panel 108
The HUD 109
The Interactions panel 111
The Properties panel 114
Text properties 117
The Design-Time Data panel 118
The Timelines panel 120
Building applications using Flash Catalyst 124
Building applications from scratch 124
Integration with Creative Suite 5 124
Round tripping between Flash Builder and Flash Catalyst 127
A tool for the "devigner" 127
Building AIR applications with Flash Catalyst? 128
Summary 130

Chapter 5: Choosing the Best Workflow 133

Simple Design, Simple Functionality 133

A Product Manual Application 133

Conclusion 140

Simple Design, Complex Functionality 140

A Photo Resizer 140

Conclusion 148

Simple Design, Connected to a Back End 148

A Twitter Search Client 149

Conclusion 159

Complex Design, Simple Functionality 159

A Static Website 160

Conclusion 165

Complex Design, Complex Functionality 165

A Simulator 165

Conclusion 176

Complex Design, Connected to a Back End 176

A Solar Panel Monitor 177

The Interaction Designer Perspective 179

The Developer Perspective 180

Conclusion 184

Summary 185

Chapter 6: Case Study: An E-Commerce Site 187

Understanding the Project 187

Analyzing the Project 188

Functional Analysis 188

Team Composition 190

Designing the Application 191

Creating the Application's Layout Structure 192

Creating the Tabs to Log In 194

Adding Design Elements 195

Adding the Navigation Bar 196
Adding the Content 198
The Contact Page 201
The Login Page 201
The Registration Page 202
Bringing the Application to Life 203
The Components 204
The Pages 212
Hooking Up the Application to the Back End 218
Creating the Database 218
Writing the Back-End API 219
Adding Dynamic Data to the Application 224
Summary 238
Chapter 7: Managing RIA Projects 241
Projects and Complexity 241
Project Team 243
Team Compositions 243
Collaboration has Changed 247
Agile Development 248
Methodologies 248
Iterative Development 250
FlexPMD: A Useful Tool 251
Summary 258
Chapter 8: Frameworks 259
The Example Application 260
Mate 261
The Code 263
Conclusion 268
PureMVC 269
The Code 271
Conclusion 277

Swiz 278

The Code 280

Conclusion 285

Summary 287

Chapter 9: Best Practices 289

Project Setup 289

Use Workspaces 290

Determine the Target Platform 290

Determine the Back End Technology 290

Deploy Release Builds 290

Place External Libraries in the libs Folder 290

Use Runtime Shared Libraries 291

Externalize the Flex framework 291

Organize Code into Packages 292

Synchronize with Back End Packages 293

Plan the Illustrator/Photoshop File Structure 293

Use an Assets Directory 294

Organize Your Assets 294

Use an XML File for the Parameters 294

Coding and Naming Conventions 295

Don't Link to External Image Files 295

Name All Design Layers 295

Design Only One Item for a Data List Component 295

Create Flash Catalyst Components Before Defining View States 296

Rename Flash Catalyst Components in the Library Panel 296

Reverse Engineer the Application's URL as a Package Structure 297

Use Plural Names for Packages 297

Use Appropriate Package Names 298

Use Singular Names for Classes 298

Use Titlecase or Camelcase for Class Names 298

Append the Class Type to the Class Name 298

Consider Appending "Base" to Base Class Names 298

Consider Appending "Abstract" to Abstract Class Names 298
Maintain Order in Your Classes 299
Use Uppercase for Constants 300
Use Camelcase/Titlecase for Variables 300
Give Variables Proper Names 300
Put an Underscore in Front of Private Variables 300
Use Implicit Getters and Setters 300
Name Implicit Getters and Setters according to the Property 301
Use Implicit Getters and Setters Internally 301
Set Argument Types and Return Types Strictly 302
Always Name the Argument of Event Handlers "Event" 302
Use a Verb in the Method Name 302
Override the Clone() Method for Custom Events 302
Keep Order in Your MXML Files 303
Use Whitespace 304
Format Your Code 304
Every Visual Component Should Have an id Attribute 304
Set the id Attribute as the First One in an MXML Tag 304
Group MXML Tag Attributes on One Line 305
Use an External CSS File 305
Maintain Order in Your CSS File 305
Be Consistent in Naming Styles 306
Limit the Number of CSS Files 306
Define the CSS Files in the Main Application File 307

Collaboration 307

Talk to Each Other 307
Formalize Decisions 307
Use a Source Versioning Tool 307
Only Commit Code that Actually Works 308
Mark Duplicate Assets in Your Design 308
Design on a Single Artboard 309
Give Design Layers Proper Names 309
Don't Define All Components in Your Design 309

Don’t Define All Component States in Your Design 310
Create Reusable “Black Box” Components 310
Consider Using Code Generation............311
Create Library Projects 311
Create a Manifest File for Your Library............311
Test, Test, Test 312
Use Different Deployment Levels............312
Set Up a Bug Reporting/Change Request Environment............313
Use a Unit Testing Framework............313
Use an Application Framework............314
Know When Not to Use an Application Framework 314
Perform Code Reviews............314
Comment Your Code 314
Validate Input Before Sending It to the Back End............315
Use the Same Local Server Settings For Everyone............315
Configure Your Application Externally............315

Summary 317

Index 319

About the Author

Steven Peeters is an Adobe Certified Instructor who works for multimediacollege (Adobe Authorised Training Center in Belgium and Luxembourg) and has 10+ years of development experience with different companies and technologies such as C, C++, Java, Flex, AIR, etc. He is passionate about all things related to Flex, AIR and ColdFusion and he teaches courses on those topics on a regular basis. In between teaching courses he also manages and works on technically complex projects to keep improving his skills.

As a Belgian ColdFusion User Group manager, Steven is also dedicated to the community and he also shares his knowledge regularly on his own website `http://www.flexpert.be` and on his company's blog `http://www.multimediacollege.be/blog`.

When he's not behind the computer you can find Steven spending quality time with his family or catching up on articles from about all areas in the science world.

Adobe | CERTIFIED INSTRUCTOR

Adobe | USER GROUP MANAGER

About the Technical Reviewer

Peter Elst is a freelance Flash Platform consultant and Founding Partner of Project Cocoon Multimedia, based in Belgium and India. As an Adobe Community Professional, author, and speaker at various international conferences, Peter is a well known and respected member of the Flash Community. Whenever he has the time you'll find him posting on his personal blog: `www.peterelst.com`

Acknowledgments

First and foremost I would like to thank my loving wife Mieke and my two little boys Ilyan and Rhune for allowing me the time and space to write this book. I know it’s been a long journey and I haven’t been able to spend as much time as I wanted with you, but I promise I’ll make it up to you all.

I also like to thank my colleagues Frederik, Alwyn and Marije for inspiring me to write this book and for helping me out with the designs I’ve made.

Next, I would like to thank some people from Adobe: Andrew Shorten, Ryan Stewart, Steven Heintz and Tim Buntel (who is no longer a part of the Adobe team unfortunately). All of you helped me out a lot in gaining in-depth insight in these new products.

Of course I mustn’t forget Peter Elst who did a wonderful job in reviewing the technicalities of this book. Your comments and remarks have been very helpful.

And finally I would like to thank the Friends of ED team for their wonderful assistance and for allowing me the opportunity to write this book.

Introduction

Hello and welcome to *Flash Builder and Flash Catalyst: the New Workflow*. I've written this book to guide you through the process of deciding how to tackle a new project.

As you probably know, developing a Rich Internet Application is not always an easy task to do. There are several aspects to keep in mind. One of the key aspects is the project workflow. Not every application will have the same workflow applied to it during its creation. A lot of it depends on the size of the project, whether or not you need to connect your application to some kind of back end technology, how many people you are working with and, let's not forget this one, how the application is designed.

If you have been developing RIAs in the past you are likely to have spoken unspeakable words when receiving the final design for the application from the designer on your team, especially if the designer is not familiar with the capabilities and limits of the technology that is used for actually developing the application. Or maybe you had to redo a couple of days work because of some "minor" design changes. You will no doubt have had a couple of frustrating moments in your life due to these kinds of problems.

Most of these problems, however, are based on the fact that designer and developers speak totally different languages. You may think they speak English or Dutch or even Swahili, but in fact, designers are talking in colors and pixels, while the developers speak in view states, loops and variables. With the arrival of Flash Catalyst the two profiles can now speak the same language and understand what they are trying to do. Designers can create basic Flex applications by working only with pixels and colors, whereas developers can now simply take those applications and extend them with a database connection, for example.

This is what this book is about. The first 4 chapters will guide you through the basics and new features of Flex 4, Flash Builder 4 and Flash Catalyst so you're up-to-speed on the capabilities and limits of those products. In chapters 5 and 6 I will take you on a tutorial based tour of the various possible workflows you can have for your applications. These chapters will help you understand how these products work together and how they improve the interaction between designers, interaction designers and developers when creating Rich Internet Applications. Choosing the best workflow will make your development process much more efficient and will help you use the available resources to their maximum potential.

In the last parts of this book I will explain to you the best practices in creating RIAs. These best practices come not only from official sources such as Adobe related articles. Most of them come from my own experience as a developer and project manager. One of the best practices, depending on the size of your project of course, is the usage of application frameworks. They will help you to collaborate with other developers and maximize extensibility, scalability and maintainability for your application. I've dedicated an entire chapter to the application frameworks, explaining the differences and similarities between the ones that are most commonly used by Flex developers.

Layout conventions

To keep this book as clear and easy to follow as possible, the following text conventions are used throughout.

Important words or concepts are normally highlighted on the first appearance in **bold type**.

Code is presented in `fixed-width font`.

New or changed code is normally presented in **`bold fixed-width font`**.

Pseudo-code and variable input are written in *`italic fixed-width font`*.

Menu commands are written in the form **Menu ➤ Submenu ➤ Submenu**.

Where we want to draw your attention to something, We’ve highlighted it like this:

> *Ahem, don’t say I didn’t warn you.*

Sometimes code won’t fit on a single line in a book. Where this happens, we use an arrow like this: ➥.

```
This is a very, very long section of code that should be written all on the same ➥
line without a break.
```

Chapter 1

Flex and AIR: Taking RIAs to the Next Level

In this chapter I'll explain a little about creating Rich Internet Applications (RIAs) and desktop applications using Internet technology, where the technology came from and how it is used today. I'll try to provide some basic understanding of Adobe Flex for the novice user. I will also look briefly at how Flash Catalyst changes that workflow and how it can make you more productive when creating Rich Internet Applications. Finally, I'll present a brief overview of some of the technologies that are commonly used for connecting a Flex application to a back end, because I'm going to be using different kinds of back-end technologies in the examples throughout this book.

Taking Advantage of Flash Technology

Let's start with a brief overview of where the technology comes from and what we have already accomplished, as shown in Figure 1-1.

Mainframe

A very long time ago, the interactive programming business started out on a **mainframe**. Who hasn't seen the famous black screens with green or orange letters on them? In fact, some mainframe programs are still out there doing their jobs. Even today there are still new programs being written for a mainframe environment.

In this type of environment, applications run on a central machine and you need a direct line to that machine to be able to access the program. Users access programs using a computer known as a *thin client,* which relies on the mainframe for processing and storage, and are of little use in stand-alone mode. The problem is, this means that only the employees of the company, or of other companies that have a leased-line connection have access. It also means that most of the programs are written only for internal use, though that actually has a couple of advantages for the configuration: there are typically not that many simultaneous connections, security is tight, and downtimes are manageable because you know which users are currently using the application.

Client-Server

Early in the nineties, things started changing. The price of a personal computer (PC) had dropped enough to get companies interested in distributed environments. This type of environment came to be known as **client-server**, because it involves a centrally located server that users access from client machines (their PCs) to run their programs. This may sound similar to what I just described, but don't be fooled. There are some similarities but there are also some major differences. The similarities lie in the fact that a server is placed at a central location and that server is necessary to run certain programs. But in the mainframe era, users were equipped with what is called a *thin client*. This is a computer that is of little use in a stand-alone mode. It always needs a mainframe to connect to. In a client-server setup it is assumed the user has tasks to do that don't require a connection to a server. Just think about writing analyses and reports, creating flowcharts or other everyday tasks. The PC, with its own hard drive and memory, enabled the user to work without having to acquire server access. Many of the programs used every day are installed on every computer, and most of the time, some services and databases are all that is located on the server.

This architecture also has benefits, in particular the fact that servers can be clustered to provide better performance and availability. Clustering is useful when one or more applications on the server may need to have a lot of simultaneous connections available.

By using the Internet as a global network between the company branches, the server could be located halfway across the globe, while the nice-looking applications could run on their local machines, with no lag in performance due to network latency issues while going to another screen or calling a dialog box, because this was all done using the local machine's capabilities.

A huge step forward during this time was the development of an attractive Graphical User Interface (GUI). All of a sudden, those monotone screens were disappearing and making way for appealing, easier-to-use interfaces for programs that need to be used every day by the same users. The way people interact with an application determines how they experience it. *User friendly* became the new buzzword and the whole user experience was given a boost.

However, these advantages came at a cost. Every time a new version of a program had to be installed, it had to be installed on all the connected computers individually. Although programs were later developed that could do this automatically (e.g. overnight), the process was often quite labor-intensive and not always practical. Examples of this type of software are *Remote Installer* from Emco Software (http://www.emco.is/products/remote-installer/features.php) and Desktop Central produced by Manage Engine (http://www.manageengine.com/products/desktop-central/windows-software-installation.html).

Web Applications

In the mid-nineties, the World Wide Web was coming out of the shadows and more and more people were connecting to the wonderful global network that was the Internet. Suddenly, the whole world was connected. Initially, almost all of the content that was available was in text format, basically some text on a page with maybe a few images. Interactivity essentially meant the user clicking on some part of the text and a new page being loaded. Animations were scarce and consisted mainly of animated GIF (Graphics Interchange Format) images.

Even though this was a step back from the client-server technology, because there was no way to build a rich GUI experience, companies recognized an opportunity to reach a much broader audience. The first **web applications** were born as companies opened up these services to everybody and the race towards globalization made a gigantic leap. Suddenly, the whole world could become a client.

Web applications accessed through the Internet have an advantage, one that was common to all mainframe applications. Since the application resides on a shared server, you only have to install or update the application once! The next time the user visited your web application, he would automatically be working with the latest version.

Rich Internet Applications

Meanwhile, web sites continued to develop, adding a variety of elements and functionality. Flash technology was growing from being a way to add animations to a web site, to being a tool to add interactivity to the web application. Some web sites were built entirely in Flash; others used only small bits and pieces of Flash on their web sites. But interest in Flash kept growing until in 2004, Macromedia, which was acquired by Adobe Systems Inc. in 2005, launched a new technology called Flex. This technology is actually an extension of the existing Flash platform in the sense that it uses the same basic functionality. But Flex is a framework with a Software Development Kit (SDK); it uses the existing ActionScript language to create library components that can be used *out of the box*. Over time, more and more library components became available and they also became more elaborate in their functionality. The applications built with this technology became known as **Rich Internet Applications (RIAs)**. I'm not saying that there weren't any RIAs existing before Flex emerged. It is just that around this time Adobe launched the term, which is now widely accepted to indicate these kinds of applications.

The *rich* part refers to the fact that the technology allows attractive, user-friendly applications to be built. The design for such applications, however, was quite challenging in the past for Flex developers, because it is not always easy to programmatically re-create a nifty design. But with the arrival of Flash Catalyst, the process has been made a lot easier. Flash Catalyst lets you start developing your application directly from the Photoshop or Illustrator design file, creating a much closer connection between the design and development team.

The *Internet* part of the term refers to the reach of these applications. Since the applications are accessible via the Internet, they are available to every computer in the world that has been connected to the internet. All any user needs to run a Flex application is a proper version of the Flash Player. Which version is required will be discussed in the next section.

With rich Internet applications, you get the *best of both worlds*, because you have the power to create a graphical, fully interactive application with the ease of having to install or update it only once on the server, while still reaching to the ends of the world.

Of course, there are some competitors like OpenLazlo, Silverlight, and Ajax (and I'm probably forgetting a few others), but they all seem to be copying a lot of the Flex functionality. This is not necessarily a bad thing, because copying and improving capabilities is often the way technologies are advanced. The competition results in better components, smaller file sizes, reduced development times, and improved tools.

These factors have contributed to the development of Flex. Today, Flex is widely adopted by lots of companies all over the world. It has been used both in simple administrative programs as well as for real-time and mission-critical applications. In fact, I've worked at the Port of Antwerp for a couple of years and during my time there, we went from traditional client-server development to using Flex as a front end to the existing back end, as well as creating completely new applications to replace old mainframe applications. Those new applications needed to work in real-time, to monitor the traffic in the harbor, for example.

Another big company that has adopted Flex as their development tool is Coggno. (This is a company that specializes in providing a platform for e-learning. So, you can develop your online content, upload it to the Coggno site, and they will take care of licensing and secure payments for you, so you don't have to go through all of that development yourself. And they use Flex as the tool to allow you to create and manage your online content. The fact that Adobe thrives in a worldwide community also lets that community suggest new or improved features. If the community shouts loud enough, Adobe listens and starts working on those issues, which is why we are already using the fourth release of the Flex SDK.

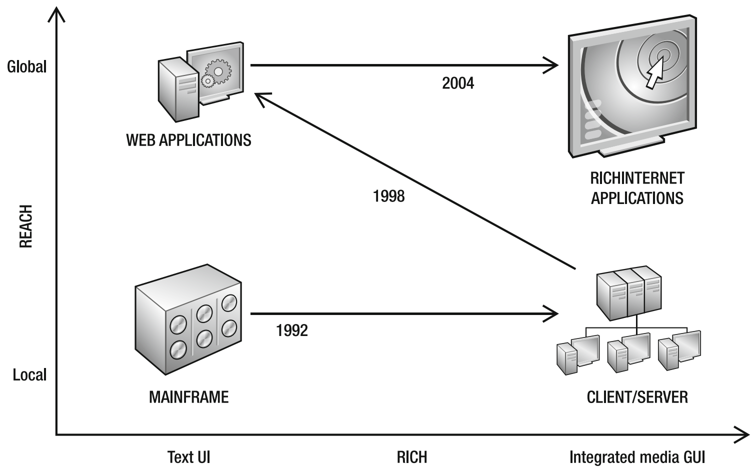

Figure 1-1. A schematic overview of applications types over time

Why Should You Use the Flash Platform?

To install a rich Internet application, you can choose from a number of technologies: Flash Player, Java plug-in, Shockwave (which is a plug-in for Director), Silverlight plug-in, and more. An important factor to take into account is the worldwide installation numbers, and even though the numbers in Figure 1-2 have been taken from the Flash Player web site, they have been gathered from a representational sample and extrapolated to the entire Internet-connected population.

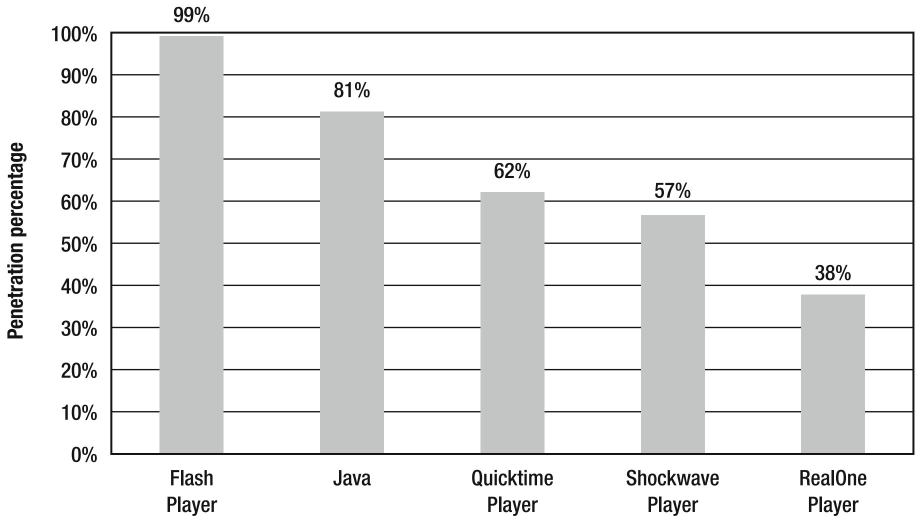

Figure 1-2. Interactive plug-in penetration statistics (December 2009)

Now, what we can learn from these statistics is that the Flash player is the most widely distributed plug-in for interactive content, not surprising since it has been around for over 15 years. (Unfortunately, at the moment of writing, no statistics were available for the Silverlight plug-in.) If you dig a little deeper into the version penetration of the Flash Player, you can see just how fast a new version of this player is adopted. The latest major version was released in October 2008 and it has already been adopted by almost 95% of all mature markets. I'm sure this number will continue to climb toward 99%, just like version 9 of the player.

Version 9.0.0 is all it takes to develop and run Flex 3 projects, but for Flex 4 you must target version 10 of the Flash Player. This means that in theory your business could be missing out on up to 5% of its client potential because of users who don't have the required version installed. But most Flash web sites and web applications have a detection mechanism that gives the user the option to automatically install the latest version before continuing. This doesn't always do the trick because some companies make it impossible for a user to install a new version, and there's not much a developer can do about that. However, the June 2009 statistics for enterprise penetration of Flash player 10 show that almost 75% of enterprises are already running this version. As this number continues to grow, the threshold for implementing Flex 4 applications will gradually disappear.

Adobe is also putting great effort in getting the Flash Player on the mobile platform. With the release of version 10.1, the company promises the player will run on several different mobile operating systems. You used to have a Flash Player version for the PC and a Flash Lite version for the mobile devices. Now, while the PC version continued to develop quite rapidly, Flash Lite could be compared to having Flash Player 7 on the device. That was a huge difference in performance, let alone the fact that you can't use ActionScript 3.0 in that version. But as I said, the new 10.1 release synchronizes the two platforms, so you should be able to create Flex 4 applications for mobile devices. To do that, though, you won't use the ordinary version of the Flex SDK. At the moment of writing this book Adobe is currently working on a special "mobile optimized" version of the Flex framework, with the working name of "Slider." This version will reflect the constraints of a mobile platform, such as limited memory, less powerful CPUs, and different input mechanisms. The first edition of the mobile Adobe Flex SDK is expected to be available in 2010. If you want to know more about Slider, take a look at the white paper "Flex and mobile: a vision for building contextual applications," available at `http://download.macromedia.com/pub/labs/flex/mobile/flexmobile_whitepaper.pdf`.

Where is Flex Used?

Since I'm talking about the use of the Flash Player for both companies and individuals, let's take a closer look at where Flex applications are being used. The usage of this technology can be divided into two large groups: web applications that are generally available to the public, and business applications. Let's take a closer look.

Flex on the Web

Don't get me wrong here. A lot of business applications are available through the Internet. Just think about **e-commerce sites**. That is basically the first category I'd like to talk about.

When these kinds of applications are brought to the Web using Flex, they have a couple of advantages over standard HTML web sites. A major benefit is that through use of the Flash technology, visitors can be turned into buyers more easily. Now, you may think "What does Flash have to do with the content I'm offering?" Well, the short answer is "nothing." But if you know Flash technology and what it can do, you know the effect it can have on the way a user experiences your site. It's fairly easy to make some kind of promotional article stand out from the crowd using an animation for example. In other words, you draw the attention of the visitor to a certain item you want to sell. I am not pretending to know a lot about marketing, but I do know this is exactly how you can persuade a visitor to actually buy something from your site.

Another fairly common application type is the **tracking system**. If you've ever ordered something from a major e-commerce site, you probably know what I'm talking about. After your purchase, you generally receive a confirmation mail that includes a link to some transport company's web site where you can track your order. You can see where it's at and sometimes even where it's going next and how it's going to get there—by plane, train or delivery truck. Whenever a key transport step is finished, the information is updated. And sometimes this even happens in real-time, which means that the web site is updated as soon as the package information is updated on the server. I'll talk a bit more about real-time possibilities when I discuss LiveCycle Data Services.

Besides the advantage of having real-time communication possibilities, tracking web sites created with Flex have another advantage over traditional HTML-based sites. With the latter, whenever you refresh the data on the tracking page, the entire page needs to be regenerated and sent over the Internet in its entirety. This means that even elements that haven't changed, such as images, headers, footers, and so forth, are also sent, increasing the network load and consuming unnecessary bandwidth. When using Flex, only the actual data that has been changed will be sent over the network connection. So, in this way, you are actually conserving bandwidth.

Flex is also great for creating **widgets,** small tools that are added to the user interface, typically to provide some specific functionality, such as weather forecasts, stock values, live tennis scores, mortgage simulators, and so on. You can use Flash technology to create interesting animations between different values. And of course you can combine these animations with real-time server-side pushed data to create eye-catching interactive widgets.

Flex can also be used to build **a complete web site.** It can be quite useful when you require some kind of data table, for example, since Flex has a Datagrid component built-in. But I've tried this myself a couple of times and found that inevitably you end up with a web application instead of merely an informative web site. This is because the Flex SDK was developed for creating applications and almost all of its components are geared toward that goal. Of course, there are some good Flex web sites out there. I'm thinking of Adobe TV (`http://tv.adobe.com`), which contains a nice set of technical video tutorials, and Parleys (`http://www.parleys.com`), which contains a lot of recorded presentations and even has an offline version. Still, if you want a Flash-based web site, you're better off just using Flash, even if you have the need for some minor back-end communication.

Flex in the Enterprise

When we take a closer look at how Flex is used in the enterprise, it's not all that different, but still there are some very big differences. E-commerce and tracking sites can be considered enterprise applications, but the type I'd like to discuss are the ones that are not available to the general public. These are business-to-business (B2B) applications or tools that are intended only for internal use.

Here are examples of some of the kinds of enterprise applications I've worked on:

- A document management system. This application was essentially a tool to keep all project-related files together on a central server. Project folders were created and contained everything from the analysis and design to zipped project source code and build script. The tool let you search across projects for certain file names and allowed for versioning of the uploaded files.
- A real-time ship-tracking system for a harbor. Barge captains, terminal operators, agents, garbage collectors, and customs authorities could all access certain parts of the same Flex applications to organize the entire path of a barge from the point it enters the harbor until it leaves again.

- An application for gathering statistical data. This one allowed neurologists to enter an initial diagnosis and keep track of their patients' progress for a certain treatment. All the data is then gathered in a central location for statistical analysis concerning these treatments.
- A building maintenance application. This tool allowed users to register certain malfunctions or structural problems in a building. The problems were directed to the proper areas and assigned to an appropriate worker. The person requesting the fix was able to keep track of the progress on his request.

Of course this is not an exhaustive list of possibilities, but it gives you an example of what's out there, because most of the time, such applications are not visible to the public.

Testing Flex applications

As you know, testing is an important part of application development, and so it is with Flex applications, especially in the enterprise. You have to run tests on different levels: locally, combined with other team members' changes, and acceptance testing. The first two levels of testing are done iteratively for every change and for every release to another level. During this iterative testing it is imperative that you perform regression tests as well. So, with every design change, with every code change, you should test not only your changes, but also the surrounding components to see whether your changes have introduced any unwanted side effects. For acceptance testing, the application runs in an environment that is supposed to resemble its intended environment; this is done just before going into production.

Of course, testing an application is not something you do only with an enterprise application. Even for the simplest of applications, I urge you to test everything thoroughly before even thinking about deploying it or handing it over to your client. But I will talk more about how you can test your applications properly later on in this book.

Connecting Applications to a Back End

Since the kinds of applications we'll be discussing will need some kind of back-end communication, let me introduce to you a few methods of communication with a certain kind of back end. In the examples throughout the book, I'll use several of these back end types, so it's important that you understand their differences and similarities when implementing them in a Flex application. Later in this book I will discuss in detail how Flash Builder 4 will make your life as a developer a lot easier than it used to be with Flex 3. The new version of the product has extensive built-in capabilities to help you connect to different kinds of back-end technologies. Add the drag-and-drop binding features and you are set for some real rapid development.

I'm not going to list all the possible technologies you could use to connect your Flex application, but I'll try to give you an overview and some example code for the most common ones. All of these have advantages and disadvantages, just as with any other technology. So you will have to decide which technology best suits your needs for a certain project.

Sometimes deciding which technology to use is difficult, because you have to make a decision which you can't really base on anything when this client-server workflow is completely new to you. Sometimes it's quite easy to make that decision because you already have developers experienced in ColdFusion, PHP, or Java on your staff. You should take advantage of that experience and choose the back-end technology they are comfortable with. And sometimes, of course, you don't even have a choice, because you need to use the existing back end and you have to adopt the communication type that can work with that particular technology.

AMFPHP

AMFPHP is a free, open source PHP implementation of the Action Message Format (AMF) that allows you to connect Flash and Flex applications to a PHP back end. It can't be compared to LiveCycle Data Services (LCDS) or BlazeDS (which I'll both discuss in a bit) because it only has support for using the `<mx:RemoteObject>` tag. And even though it hasn't been updated in ages, it still performs beautifully.

AMFPHP does have some things going for it. First of all, BlazeDS and LCDS don't allow you to use PHP as your back-end technology. Moreover, as a kind of middleware layer between Flex and the back-end code, it is very lightweight. In fact, I think it's the most lightweight solution out there at this time, mainly because AMFPHP doesn't include or impose any development framework. You are free to choose the one you want to use, or you can simply not use a specific framework at all. AMFPHP also lets you map your ActionScript classes directly to your PHP classes. This works much the same as in LCDS and BlazeDS. The only difference is that in your PHP class you have to explicitly state the fully qualified class name in a property called `$_explicitType`. This property is then mapped to the `[RemoteClass]` metadata tag in your ActionScript class definition.

Let me give you a small example of how this would look with some basic classes. Assume we have to fetch a list of people (such as registrations via a website) to display them in some kind of grid. The basic PHP `PersonDTO` class might look like this:

```
<?php
  class PersonDTO {
    var $id;
    var $firstname;
    var $lastname;
    var $phone;
    var $email;

    var $_explicitType = “com.domain.project.valueObjects.PersonDTO”;
  }
?>
```

In this example, the `PersonDTO`.php file should be located in a package (which is basically a directory) on the server called `com.domain.project.valueObjects`. Now, where does this package structure start from? Let's take a look at the installation procedure for AMFPHP, which is pretty straightforward. All you need to do is go to the download section of the AMFPHP site (`http://www.amfphp.org`), which will redirect you to SourceForge. There you can find and download the latest version—1.9 beta2. (Or you go directly to `http://sourceforge.net/projects/amfphp/files/amfphp/amfphp%201.9%20beta2/amfphp-1.9.beta.20080120.zip/download`.) Just unzip this file into the web root of your server (htdocs for Apache servers). All this does is install a separate directory into which you can place your services and valueObjects, as you can see in Figure 1-3.

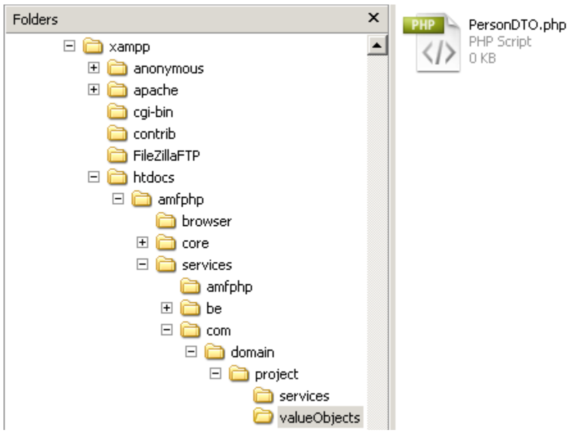

Figure 1-3. Directory structure in amfphp when installed in xampp (the Apache server package)

Now, we still have to link our ActionScript class to that PHP class. To make it easy on myself here, I'm going to use publicly accessible attributes in the class. Normally, you shouldn't do this; you should declare all properties `private` and create some implicit getters and setters for them. These implicit getters and setters are specific ActionScript implementations that differ from other programming languages in both their notation and usage. Instead of defining the getter as `getProperty()`, you need to put a space between the words, creating a `get property()` method. The advantage is that you can still use the dot-notation when working with these properties as if they were public properties. But you are still in fact executing a method, so additional calculations or other functionality will be executed when setting a value, for example. Occasionally, however, it can be handy to use these `public` properties for quickly testing something, which is why I'll show this in the first example. The other examples will use `private` properties.

```
package com.domain.project.valueObjects {
  [Bindable]
  [RemoteClass(alias="com.domain.project.valueObjects.PersonDTO")]
  public class PersonDTO {
    public var id:Number;
    public var fistname:String;
    public var lastname:String;
    public var phone:String;
    public var email:String;
  }
}
```

Notice that the `[RemoteClass]` metadata tag contains an `alias` property that is set to the exact same value as the `$_explicitType` property of the PHP class. This is necessary for the system to work properly. I'll explain more about this tag and its purpose in the section on Flash Remoting with LiveCycle Data Services.

In your back-end method that retrieves this list of people, you have to create an `array()` and fill it with `PersonDTO` objects. The AMFPHP middleware layer will make sure that this list is then converted to an `Array` of `PersonDTO` objects in ActionScript.

Please note that the return value from such a PHP method is an Array and not an ArrayCollection. If you wish to use the result in a binding expression (e.g. dataProvider in a DataGrid) you still have to convert it to an ArrayCollection.You can't just cast it to an ArrayCollection, so you have to convert it like this:

var arr:ArrayCollection = new ArrayCollection(event.result as Array);

The really great feature of AMFPHP is its browser. This is a Flex application that allows you to browse through the different services you've created. An introspection module generates a list of `LinkButton` components for each method in the selected service. By clicking on this button, you can fill out any necessary parameters and make the actual function call. In this way, you can verify that you don't have any errors in your code, because you will get compilation errors from your PHP code. You can also check the result value for the function because it is displayed in the same browser application. So, all in all, AMFPHP is a great tool to verify your back-end functionality apart from your Flex application, which can be very helpful for locating bugs or errors.

Zend AMF

Zend AMF is another way of communicating between Flex or Flash and PHP in a back-end environment. It is quite similar to AMFPHP in the sense that it uses the Action Message Format (AMF) protocol, but ZendAMF is essentially the new implementation and is officially supported by both Adobe and Zend.

ZendAMF is part of the Zend framework, so you'll need to download the framework from `http://frameswork.zend.com/download/latest`. If you're using Flash Builder 4, you have the option of installing the framework on your server automatically when using the Data/Services panel. Thus, I won't go into too much detail here; I'm just going to explain briefly how ZendAMF works without Flash Builder. We'll get to the practical details in Chapter 3, where I'll explain more on how to use the framework from within Flash Builder 4.

After installing the Zend framework on your web server, you need to create a bootstrap file. This file is like the `gateway.php` file used with AMFPHP. It is the endpoint you'll be connecting to from Flash or Flex. This file should include the `Server.php` file you'll find in the Zend framework directory. This class will actually be doing most of the work for us when calling a back-end method. Assuming the framework is located in a directory next to the browser root (the htdocs folder on Apache), the bootstrap file might look like this:

```
<?php
  // Turn on the error reporting. This should be turned off in a
  // production environment.
  error_reporting(E_ALL|E_STRICT);
  ini_set("display_errors", "on");

  // Extend the include path to include the framework directory
  ini_set("include_path", ini_get("include_path") . ":../frameworks");

  // Include the server classes we are going to use
  require_once "Zend/Amf/Server.php";
  require_once "PersonServices.php";
  // Initialize the server object and tell it which service class it wants to
expose.
  // You can expose multiple classes by repeating the setClass() method call for
each
  // class.
  $server = new Zend_Amf_Server();
  $server->setClass(“PersonServices”);
?>
```

The `PersonServices.php` file that is included in this file defines the functionality made available to the Flash and Flex applications. Let’s assume that this services class contains a method available to retrieve all the people from the database.

```
<?php
  class PersonServices {
    public function __construct() {
      // Set up the connection details
      mysql_connect(“localhost”, “root”, “”);
      mysql_select_db(“flex4”);
    }

    public function getAllPersons() {
      $result = mysql_query(“SELECT * FROM person”);
      $arr = array();
      while($row = mysql_fetch_assoc($result)) {
        // Just attach the associative row object to the end of the array
        array_push($arr, $row);
      }

      return $arr;
    }
  }
```

You may have noticed that I did not include the closing PHP tag. Because of potential problems with whitespace characters in class files, Zend actually recommends leaving out this closing tag as a best practice.

Just as with AMFPHP, you need to create a `services-config.xml` file that contains the destination to which you want to connect. However, this file is a little bit different for ZendAMF since you’re not connecting to the `gateway.php` file anymore. Instead, you’re connecting to the bootstrap file you just created. Everything else stays the same.

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
  <services>
    <service id="amfphp-flashremoting-service"
             class="flex.messaging.services.RemotingService"
             messageTypes="flex.messaging.messages.RemotingMessage">
      <destination id="zend">
```

```
        <channels>
          <channelref="my-zend"/>
        </channels>
        <properties>
          <source>*</source>
        </properties>
      </destination>
    </service>
  </services>
  <channels>
    <channel-definition id="my-zend"class="mx.messaging.channels.AMFChannel">
      <endpoint uri="http://localhost/zendamf_remote/"
                class="flex.messaging.endpoints.AMFEndpoint"/>
    </channel-definition>
  </channels>
</services-config>
```

Once you've created this `config` file, you need to include it in the compiler settings of your Flex project. From that point on, all you need to do is create an `<mx:RemoteObject>` in your Flex application with the `zend` destination and call the `getAllPersons` method of the back-end class, as shown in the code example below. The results of this call will be transferred back to the Flex front end in the form of a `result` event, while potential faults will be returned as a `fault` event.

```
<mx:Script>
  private function resultHandler(event:Resultevent):void {
    for(var i:uint = 0; i < event.result.length; i++) {
      // Just print out the firstname and lastname fields concatenated
      trace(event.result[i].firstname + " " + event.result[i].lastname);
    }
  }

  private function faultHandler(event:FaultEvent):void {
    Alert.show(event.fault.faultString, event.fault.faultCode);
  }
</mx:Script>

<mx:RemoteObject id="service" destination="zend"
                 result="resultHandler(event)"
                 fault="faultHandler(event)"/>
```

What I've just shown you is a tiny example of how to use the ZendAMF implementation of the AMF protocol for connecting to a PHP back end. This is only a small part of the entire Zend framework, so I suggest you take a closer look at the rest of the framework as it is very useful if you're working with a PHP back end. You'll find the documentation at `http://framework.zend.com/docs/overview`.

LiveCycle Data Services

LiveCycle Data Services is the main way of connecting your back end to a Flex application if you want to use Adobe technology. It offers a lot of possibilities and features, as you can see in Figure 1-4.

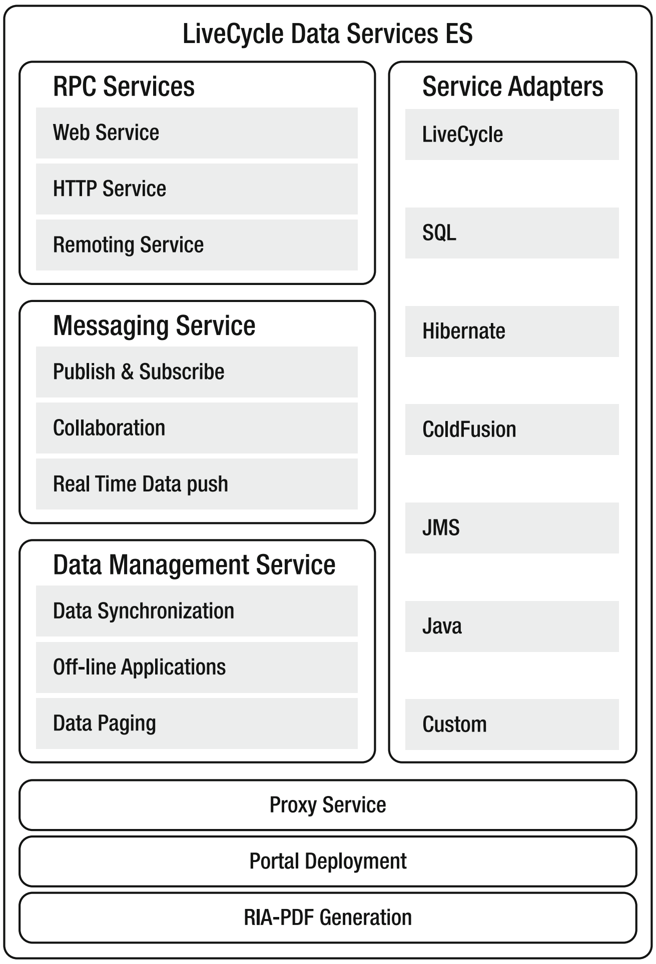

Figure 1-4. A schematic overview of LiveCycle Data Services

Proxy Service

The first feature I'd like to take a closer look at is the solution to **cross-domain issues**. Let me first explain the problem. The Flash Player has a few security restrictions, including the fact that you can't just access any file on any sever using a Flash (or Flex) application. Your web application is located on your web server and the Flash player only allows you to access remote files located on the same server as your application. If you want to access a file on another server, you'll get a "`Security error accessing URL`."

There are two solutions to this problem. The first is rather simple: you just have to place (or modify) a `crossdomain.xml` file on the root of the web server you're trying to access. The content of this cross-domain file is:

```
<cross-domain-policy>
  <allow-access-fromdomain="IP Address or range" />
</cross-domain-policy>
```

The problem with this solution is that this file needs to be located on the remote server and unless this is a server you control, you probably won't have access to that location. I'm sure you can imagine the security risks if you at company X would have access to the web root of company Y, which could be a competitor, the Department of Defense or even NASA.

That's why there's a second solution—the **ProxyService**. This is really just a work-around for the problem, but it's a good one. The main configuration file for LiveCycle Data Services is `services-config.xml`, `and it` includes a `proxy-config.xml` configuration file in which you can define a destination to be used when accessing a remote file. Here's an example of a `proxy-config.xml` file.

LCDSIntro ➤ lcds-config ➤ proxy-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<service id="proxy-service"
         class="flex.messaging.services.HTTPProxyService"
         messageTypes="flex.messaging.messages.HTTPMessage,
                       flex.messaging.messages.SOAPMessage">
  <properties>
    <connection-manager>
      <max-total-connections>100</max-total-connections>
      <default-max-connections-per-host>2</default-max-connections-per-host>
    </connection-manager>
    <allow-lax-ssl>true</allow-lax-ssl>
  </properties>

  <adapters>
    <adapter-definition id="http-proxy"
                        class="flex.messaging.services.http.HTTPProxyAdapter"
                        default="true"/>
    <adapter-definition id="soap-proxy"
                        class="flex.messaging.services.http.SOAPProxyAdapter"/>
  </adapters>

  <default-channels>
    <channel ref="my-http"/>
  </default-channels>

  <!--Default destination using a dynamic URL-->
  <destination id="DefaultHTTP">
    <properties>
      <dynamic-url>
        http://remote-machine:port-number/*
      </dynamic-url>
    </properties>
  </destination>

  <!--Named destination using a fixed URL-->
```