

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>



**MEAP Edition**  
**Manning Early Access Program**

Copyright 2009 Manning Publications

For more information on this and other Manning titles go to  
[www.manning.com](http://www.manning.com)

©Manning Publications Co. Please post comments or corrections to the  
Author Online forum: [http://www.manning-  
sandbox.com/forum.jspa?forumID=442](http://www.manning-sandbox.com/forum.jspa?forumID=442)

**Table of Contents:**

Chapter One: Getting Started

Chapter Two: ActionScript 3, XML and E4X

Chapter Three: Hello Spark: Primitives, Components, FXG and MXML Graphics and Video

Chapter Four: Beyond Basics: Spark Containers and Layouts, View States, Effects, Styling and the component lifecycle

Chapter Five: Halo Flex 4: Using Halo Components like DataGrid, ViewStack, TabNavigator Accordion and Title Window

Chapter Six: Building User Friendly Forms using Flex Formatters and Validators

Chapter Seven: Building a Real Application using Cairngorm: SocialStalkr, A Twitter plus Yahoo Maps Validators

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

# 1

## Getting Started

In this chapter, we're going to get Flex 4 and build a few basic Flex 4 apps, in order to learn the basic structure and syntax of a Flex 4 application. We'll also get a high-level picture of the Flex 4 ecosystem. First, however, we'll cover what this book is all about.

Briefly, this book is divided into seven chapters. The first six chapters contain about 25 workshop sessions, and every example is standalone. These examples are toy examples which are focused on exactly what you're trying to learn in that workshop session--and nothing else. In the last chapter, we will build a real Flex 4 application. This chapter is essential, since it provides you with the big picture that toy examples can't provide. How do all the parts fit together? Furthermore, it will teach you Cairngorm, which is the dominant Flex 4 application framework.

Oh yeah, the application: it's called SocialStalkr, and it's a Twitter and Google Maps mashup. (Can you get any more Web 2.0-compliant than that?) Besides, there are actual

**A TWITTER  
MASHUP APP?  
DO I LOOK LIKE  
ASHTON KUCHER?**



books entirely about Twitter now; so, it's like you're getting a free book here.

Finally, a word about the format. Yes, there are some cartoons in this book. If you're over 30 (like me) and used to read Slashdot you may recognize the User Friendly characters--and if you read reddit you'll have a pretty good idea what cartoon the stick figure is inspired by.

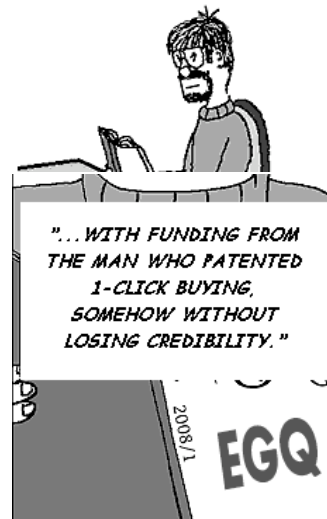
In both cases, the text of the cartoons Co. Please post comments or corrections to the

Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

**PITR RECEIVES THE EVIL  
GENIUS QUARTERLY PROFILE  
OF 37signals, A WEB 2.0  
COMPANY WITH FUNDING  
FROM AMAZON'S JEFF BEZOS.**



**"37signals WINS THE 'DR EVIL'  
AWARD FOR INSPIRING APPLE-  
LIKE DEVOTION BY PRODUCING  
SHINY, EXPENSIVE TODO LISTS  
AND CHAT ROOMS..."**



cartoons is all mine--so if the jokes fall flat, blame me! (Also, the only official deal Manning has is with User Friendly; the other one is just me paying tribute to a cartoon I have greatly enjoyed for a long time.) Anyway, I have two goals for this book: first, to teach Flex 4 in a way that exposes you to lots of real-world Flex problems in an accessible way, and second to have a bit of fun with this book, without being cutesy, distracting or insulting. The purpose of the cartoons in this introductory section are to provide a mildly amusing backstory; in the rest of the book they will actually attempt to be used to draw attention to some important concepts.

You can think of this book as a 2 or 3 day workshop, transcribed into book form--*and much cheaper!* (I am actually planning a number of 2 or 3 day workshops based on this book material, so this is deliberate.) The stick figure character is a (much thinner!) stand-in for me teaching the workshop, and the cartoon characters are your classmates. The questions they will ask or the opinions they express may be your own!

Finally, a favour: this is a very experimental book, which I have already rewritten once, so your feedback is very welcome. *Please email any book format feedback, good or bad, to me directly at [peter@ruboss.com](mailto:peter@ruboss.com).* (Seriously.)

With that out of the way, let's get started!

## Why Flex 4?

Chances are you already know why you want to use Flex, so I won't bore you. If you don't, here's the 1-paragraph version:

Flex 4 is a sexy framework that lets you write code that feels more like coding a desktop application—except it runs inside the Flash player! Because it targets the Flash player, you can build new Rich Internet Applications (RIAs) without worrying about browser compatibility nonsense, JavaScript, CSS, and so on. Because Flex 4 targets one platform (Flash 10), we don't have to worry about platform compatibility issues. The Write Once, Run Anywhere (WORA) dream that client-side Java programmers had—before it turned into “write once, debug everywhere”—can finally be realized, but with Flex. Flex achieves what previous technologies such as Java



©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

applets failed miserably in attempting: applications that feel like desktop applications, but which run inside any modern web browser on Windows and Mac. We can use Flex 4 to build RIAs today that look and feel more like Web 3.0 than many of the “me too [point oh]” sites you see copying 37signals and each other today.

## **Flex 4 Overview**

Now that you’re all excited, let’s take a deep breath and get an overview of the Flex 4 platform. This section will present a high-level overview; don’t worry if you don’t understand a particular point here, it will be explained later.

In Flex 4, we write code in MXML (XML files with a .mxml extension; M for Macromedia) and ActionScript (text files with a .as extension) files and compile them into a SWF file (that is, a Flash movie), which runs in the Flash player. This SWF is referenced by an HTML file, so that when a user with a modern web browser loads the HTML file, it plays the Flash movie (prompting the user to download Flash 10 if it’s not present). The SWF contained in the web page can interact with the web page it’s contained in and with the server it was sent from.

Even if you’ve never created a Flash movie in your life, don’t consider yourself a designer, and wouldn’t recognize the Timeline if you tripped over it, you can use Flex to create attractive applications that run in the Flash player. Flex development is easily learned by any intermediate-level developer with either web (HTML and JavaScript) or desktop UI (such as Windows Forms or Java Swing) programming experience. A Flex 4 application is just a Flash movie (SWF), which lives inside a web page loaded by a web browser that has Flash 10 installed.

**37SIGNALS SELLINK  
SUBSCRIPTIONS TO  
TODO LISTS...**

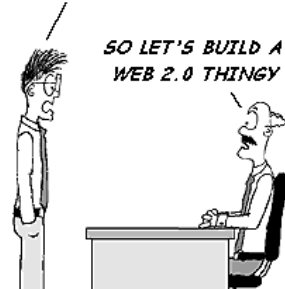


**...CHARGING HOW MUCH?**

**MORE THAN WE CHARGE  
OUR CUSTOMERS FOR  
INTERNET ACCESS**



**...AND THAT'S JUST FOR  
ONE OF THEIR PRODUCTS!**



©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

## Flex vs. Ajax? Flex and Ajax?



Now that we have seen what Flex is, let's consider the main alternative to Flex: Ajax. (Silverlight doesn't count—yet—since it does not have a large enough installed base to be a pragmatic choice for a consumer facing application.)

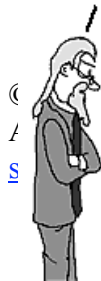
### AREN'T YOU IGNORING JAVA FX?

Yes, just like everyone else is. (Oh, snap!)

The question of when to use Flex, when to use Ajax and when to use both is extremely controversial. There is no one right answer: it depends on many factors, including the size of your application, the skills of your developers, how important search engine optimization is, and so on. Furthermore, as both Flex and Ajax frameworks evolve, the answer itself evolves. That said, there is one question which I like to use: "Are you building a publication or an application?" The more application-like what you are building is, the better a fit Flex usually is. (Another way of thinking about this is to ask yourself if you could visualize your app being or competing with a desktop application.)

## Getting Flex 4 and Flash Builder 4

FLASH BUILDER?  
BUT MY IDE IS  
VIM ON LINUX...



So, since this is a book about Flex 4, and since you're presumably interested enough in Flex 4 to be reading or browsing it, the next step is for us to actually download Flex 4 and play with it. Somewhat confusingly, Flex 4 applications are built using something called Flash

cations Co. Please post comments or corrections to the forum: <http://www.manning-um.jspa?forumID=442>

Builder 4. (In Flex 1, 2 and 3 this was called Flex Builder.) The marketing rationale for this is as follows: Flex applications can be built in conjunction with designers using something called Flash Catalyst (which used to be code named "Thermo"), so it makes sense for them to both be called Flash Something, to emphasize that they play nicely together. Second, Flex applications are compiled into Flash movies (SWFs), just as Flash applications developed in Adobe Flash CS4 are compiled into SWFs. However, since the code editor in Flex Builder 3 was so much better than in Flash CS4, many Flash developers were using Flex Builder to build SWFs, without actually using the Flex Framework. Or, they would use Flex Builder and Flash CS4 together. So, Adobe realized that since they had Flash CS4 and Flash Catalyst, they should rename Flex Builder to Flash Builder to fit in.

Looking a little deeper, the real reason Flex Builder could be renamed Flash Builder is that Flex has been so successful in making the Flash Platform something that is considered suitable for Enterprise use, not something which is dismissed as for just games and annoying ads. So, instead of the Flex name having to essentially run away from the Flash brand, it can be used for the Open Source Flex Framework and all the corporate branding can be Flash.

Anyway, we're going to download Flash Builder 4 from <http://labs.adobe.com/technologies/flashbuilder4/>. There is a trial version available which should last long enough for you to follow along with this book. Download the standalone version, not the Eclipse Plugin version, if you want to follow along with the book verbatim.

Flash Builder is Adobe's Eclipse-based IDE for building Flex applications. However, you can also do Flex development using the Flex SDK without Flash Builder: just use your favorite text editor and the command line compiler that comes with the SDK. This book won't go into how to do that, since most people will end up using Flash Builder.

#### **SHOULD I USE FLASH BUILDER OR JUST THE FLEX SDK?**

While the SDK is free, using it isn't as easy as using Flash Builder. So, even if you plan to use the SDK, I recommend using

©Manning Publications Co. Please post comments or contact  
Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

**ERWIN, WHAT'S EASIER TO BUILD GUIs IN THAN AJAX?**

**OH! I KNOW THIS ONE! ALMOST ANYTHING!!**



**HOW ABOUT FLASH?**



**WOW, I DIDN'T KNOW YOU'D BECOME A DESIGNER?!**



**YOU'RE RIGHT, A.J. IS THE DESIGNER, NOT ME...**

**UM... SILVERLIGHT?**



Flash Builder to learn Flex: not only do you get to defer your decision long enough to finish the book (during which time you may decide you like Flash Builder enough to pay for it), you also will learn Flex faster with Flash Builder providing code completion support and the design mode which lets you drag and drop Flex components to lay out a UI.

## **Beginning the Workshop**

With this completed, we begin the workshop sessions. These workshop sessions will be grouped into chapters containing thematically related sessions. However, each of these is standalone, so they all begin on new pages.

Each workshop session is a separate project in the code zip file. You can follow along with each workshop session by creating a new project in Flash Builder 4. Just choose File > New > Flex Project... to trigger the New Flex Project dialog. Enter a name for the project in the Project name field and click Finish. It doesn't really matter what you name the projects; I named them all session01, session02, ... in the code zip file. However, note that the project name is used to name the main application, so if you want the main application to be named appropriately, use CamelCase for the name. (In chapter 7, for example, the main application is SocialStalkr so that's what the project should be named.)

*NO WAY: YOU KNOW SID AND MICROSOFT...*



*HOW ABOUT FLEX? I HEARD BRUCE ECKEL EVEN SWITCHED...*



*HMM. GOOD POINT. I'LL EMAIL THE CHIEF AND GET YOU GUYS SOME WORKSHOP TRAINING...*

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

## Session 1 - Hello! Flex

SO, I HEARD THAT THE CHIEF IS HIRING AN EXPENSIVE FLEX CONSULTANT TO GIVE US A WORKSHOP.



SO, PETER WAS SAYING THAT THE CHIEF SHOULD JUST PAY US TO READ.



ARE YOU THE WEB 2.0-SAVVY FLEX CONSULTANT WHO IS GIVING US A WORKSHOP?



This is the first workshop session. In it, we build our first complete Flex application, shown below. Note that the code listing titles show you the exact path to the files in the code zip file which you can download from <http://manning.com/armstrong3/>.

### session01/src/Hello.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/halo">
  <s:SimpleText text="Hello! Flex 4"
    fontSize="128"/>
</s:Application>
```

The root of a Flex application is the Application tag. What this application does is simply contain a SimpleText component with the text "Hello! Flex 4". (For now, ignore the various xmlns lines; we'll get to that later in this chapter.) Running this application gives us our first Flex application.

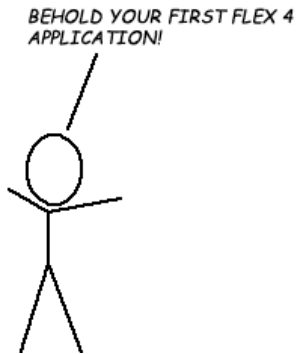


### Key Points

- Flex applications start with an Application tag.
- Flex applications can be built with very little code.

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

## Session 2 - Dispatching and Listening for Events



After you get past all the hype about Rich Internet Applications, Flex development is really distinguished by the pervasiveness of two main things: events and data binding. In this session, we will see how events work; in the next, we will learn data binding. It may seem odd to dive right in to events and data binding before even looking at how Flex applications are structured, but since events and data binding are everywhere in Flex, it's actually preferable to confront them right away so that the other examples aren't mysterious.

In this section, we will create an app which contains three Buttons: button1, button2 and button3. All Buttons dispatch a MouseEvent called "click" (actually MouseEvent.CLICK) when they are clicked; we will show how to handle this in three different ways in this session. What we will do is make clicking one of the buttons add the text "Button 1 clicked" (or 2 or 3) to a SimpleText text component. For layout, we will put these Buttons and the SimpleText into a VGroup, as shown below.

LITERALLY, IN OUR FIRST HOUR OF THE WORKSHOP WE'VE PRODUCED A SEVEN LINE APPLICATION.



### session02/src/Hello.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  initialize="init()">
<fx:Script><![CDATA[ #1
  private function init():void {
    button3.addEventListener(MouseEvent.CLICK, handleClick); #2
  }
  private function handleClick(event:MouseEvent):void { #3
    if (event.target == button2) {
      simpleText.text += 'Button 2 clicked\n';
    }
  }
}
]]>
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

```

    } else if (event.target == button3) {
        simpleText.text += 'Button 3 clicked\n';
    }
}
]]></fx:Script>
<s:VGroup width="100%"> #4
    <s:Button id="button1" label="Button 1"
        click="simpleText.text += 'Button 1 clicked\n'"/> #5
    <s:Button id="button2" label="Button 2"
        click="handleClick(event)"/> #6
    <s:Button id="button3" label="Button 3"/>
    <s:SimpleText id="simpleText"/>
</s:VGroup>
</s:Application>

```

## Annotation Bubbles

**#1** We are inlining ActionScript 3 code for the first time. The `<![CDATA[ and ]]>` is essential inside the `<fx:Script>` tag (so you can type code instead of XML), so Flash Builder adds it for you after you type `<fx:Script>`.

**#2** We call the `init()` function in response to the initialize event being dispatched by the Application

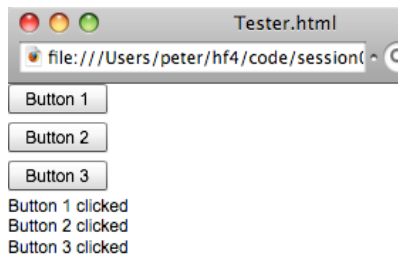
**#3** The `handleClick` event handler takes a `MouseEvent` event parameter that it uses the `target` of to determine which button was clicked.

**#4** We are putting the Buttons and the `SimpleText` into a `VGroup`, which arranges its children vertically. (Had we used an `HGroup`, they would have been laid out horizontally.) `VGroup` and `HGroup` are `Group` subclasses.

**#5** We can also add a handler for the click event in the attribute value

**#6** The click event automatically creates a variable called `event` which is of type `MouseEvent`. We pass this into the `handleClick` method.

Running the application and clicking the `button1`, `button2` and `button3` Buttons in order produces the screenshot shown.



## Key Points

- Events can be handled either in the attributes directly or in explicitly written event

©Manning Publications Co. Please post comments or corrections to the

Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

handler functions.

- Event handlers can be attached in MXML attribute values or in ActionScript `addEventListener` calls.
- Event objects have a `target` property, which is the source of the Event.
- In MXML, components can be nested in Group objects. `HGroup` and `VGroup` are subclasses of Group that lay out their children horizontally and vertically.

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

## Session 3 - The Bindable Annotation and Data Binding

Now that we have seen how to use events, let's look at data binding. Data binding is the most unique thing about Flex. It's a very powerful feature that is easy to use, and Flex 4 has added two-way data binding to make this even easier. Data binding is also easy to abuse, with negative consequences for performance--this is covered in great depth in an excellent one hour presentation entitled "Diving in the Flex Data Binding Waters"<sup>1</sup> by Michael Labriola. (Once you are comfortable enough with Flex 4 and data binding that you are curious about how it works under the covers, I highly recommend spending the time to watch this presentation.)

SO THE SMILING MAN SAYS THAT YOU'RE JUST LOWERING OUR EXPECTATIONS.



In this session, we will learn the basics of one-way and two-way data binding. We will start by building an example that uses one-way data binding twice, to copy the text of two text inputs into each other. We will also create a SimpleText that uses data binding to show the length of the String in the textInput1.

### session03/src/OneWay.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/halo">
  <s:layout>
    <s:VerticalLayout paddingLeft="5" paddingTop="5"/>
  </s:layout>
  <s:TextInput id="textInput1" text="{textInput2.text}"/> #1
  <s:TextInput id="textInput2" text="{textInput1.text}"/>
  <s:SimpleText text="# chars: {textInput1.text.length}"/>
</s:Application>
```

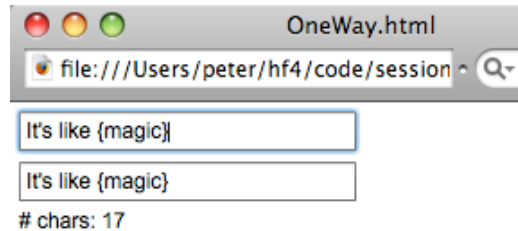
### Annotation Bubbles

**#1 Each TextInput's text property is bound (with the {} syntax) to the other TextInput's text property.**  
Running the application, we can type text in and see the following screen.

---

<sup>1</sup> <http://link.brightcove.com/services/player/bcpid1733261879?bclid=1729365228&bctid=1741212660>

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>



Well, that was pretty cool! However, let's not stop there: let's get even lazier. What if we want to bind the text properties of both TextInputs to each other, but we want to type even less? Flex 4 introduces a new feature to Flex: two-way data binding. Let's see how that works.

#### session03/src/TwoWay.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/halo">
  <s:layout>
    <s:VerticalLayout paddingLeft="5" paddingTop="5"/>
  </s:layout>
  <s:TextInput id="textInput1" text="@{textInput2.text}"/> #1
  <s:TextInput id="textInput2"/>
  <s:SimpleText text="# chars: {textInput1.text.length}"/>
</s:Application>
```

## Annotation Bubbles

**#1 The textInput1 text property binding has the even more magical @{} syntax, which means "bind this both ways." Note that the text property of the textInput2 has no binding now.**

Running the application, we see the identical output to before.

Binding isn't just for stupid UI tricks, however: it's primarily used to get data in and out of ActionScript 3 model objects. So, let's create a model object now, and then see how binding works with it:

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

### session03/src/model/Task.as

```
package {
    public class Task {
        [Bindable]
        public var name:String;          #1

        public function Task(name:String = "") {
            this.name = name;
        }
    }
}
```

## Annotation Bubbles

**#1 The Bindable annotation on the name variable ensures it can be the source of a data binding.**

Next, let's create a new application that uses this model. (You can copy-paste-modify the OneWay.mxml application to save time.)

### session03/src/BindingToModel.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/halo">
<fx:Script>
<![CDATA[
    [Bindable]
    private var _task:Task = new Task("Learn Binding");          #1
]]>
</fx:Script>
<s:layout>
    <s:VerticalLayout paddingLeft="5" paddingTop="5"/>
</s:layout>
<s:TextInput id="textInput1" text="{_task.name}"                #2
    focusOut="_task.name = textInput1.text;"/>
<s:TextInput id="textInput2" text="{_task.name}"
    focusOut="_task.name = textInput2.text;"/>
</s:Application>
```

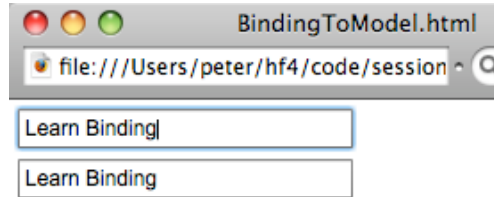
## Annotation Bubbles

**#1 We create a new Task variable \_task which is Bindable, and initialize the name property to Learn Binding.**

**#2 The \_task variable's name property (which is also Bindable) is bound to the text property of both textInput1 and textInput2.**

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

Running the application, we see that the TextInputs are both bound to the name property. Typing in either of them and then focusing out (by, say, typing the Tab key) assigns the text to the model's name property which then updates the other TextInput's text.



But we're feeling lazy; why don't we try using two-way data binding?

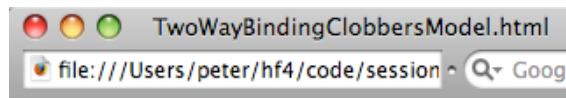
### session03/src/TwoWayBindingClobbersModel.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/halo">
<fx:Script>
<![CDATA[
  [Bindable]
  private var _task:Task = new Task("Don't do this!");    #1
]]>
</fx:Script>
<s:layout>
  <s:VerticalLayout paddingLeft="5" paddingTop="5"/>
</s:layout>
<s:TextInput id="textInput1" text="@{_task.name}"/>      #2
<s:TextInput id="textInput2" text="@{_task.name}"/>
</s:Application>
```

## Annotation Bubbles

- #1 Initialize a new Task just like before.
- #2 Use two-way data binding with interesting results.

Running the application we see a surprise. (Well, if you looked at the file name, it's not a surprise.)



©  
At  
[sar](#)

ents or corrections to the

The `_task.name` got clobbered by the initially blank text of the TextInputs! (Note that typing in them does work, however.) So, be careful, especially with two-way data binding.

Now that we have been introduced to events and data binding, we can look at what a slightly larger Flex application looks like, so that we learn the structure of Flex applications.

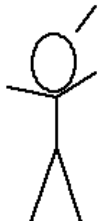
## **Key Points**

- Data binding "magically" copies the value of one property to another property. (Well, it's not magic: `PropertyChangeListeners` are used. But for the purposes of chapter 1 of this book, it's magic.)
- For data binding to work and not generate compiler warnings, the `[Bindable]` annotation must be used on the property which is the source of the data binding as well as on the variable which contains the reference to the Object with the property in question.
- Two-way saves time when dealing with UI components, but be careful when using it with models.

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

## Session 4 - Flex Application Structure Overview by Example: ActionScript and MXML Files and Packages

NEXT, WE'RE GOING TO BUILD AN APPLICATION THAT ACTUALLY DOES SOMETHING! YOUR COMPANY DIDN'T PAY THE BIG WORKSHOP BUCKS FOR NOTHING, YOU KNOW...

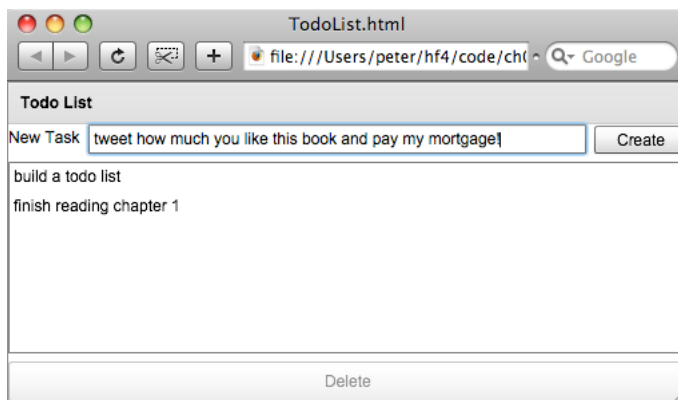


This application has a panel with a title saying "Todo List", a label with the text "New Task", a text input to enter the new task name, a button for the user to click to create the new task, a big

area to list tasks, and a typically-overstuffed delete button. (We need at least one gratuitously huge button if we're going to be web 2.0 compliant!) So, let's start by laying out this UI using MXML. (There is no benefit to the panel itself, but it just looks nicer than just putting the controls directly in the Application—it lets us pretend we're putting *some* thought into design.)

Next, we're going to build this UI. We will lay out the application UI with MXML, and add logic with ActionScript. (XML is a good markup language for laying out a UI, but it is a terrible language to write real procedural logic in. Thankfully, Flex doesn't force us to do that!) We will see how we can introduce behavior by adding ActionScript 3 code to both our MXML file (in a Script tag) and to a standalone ActionScript 3 file.

Now that we have seen the required hello world example (this is a hello book after all!), let's get a sense of the structure of a real Flex application by building one. Specifically, we're going to build a Todo list. We want to build something like this screenshot.



PHEW! AT LEAST MY JOB AS A DESIGNER IS SAFE...



©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

Note that the purpose of this workshop session is just to show the big picture; the details of how everything works are not especially important right now. (I will explain them at a very high level so that the example makes a bit of sense, but I don't want us to get bogged down.) I just want this session to give you a sense of the big picture, of why we are here and what we're trying to do. We'll untangle the details in the workshop sessions ahead.

First, we create a new Task ActionScript class, which is inside a text file with a .as extension. In Flex 4 you write code in one of two ways, in MXML (.mxml) files or in ActionScript 3 (.as) files. (You can actually create multiple classes in one file, but we'll ignore that for now.)

We are going to create this class inside a package, specifically the com.pomodo.model package. Flex supports packages just like languages such as Java do, and the backward domain name syntax is a convention just as it is in Java. So, next we create a com\pomodo\model directory structure in the src directory. (Pomodo is just a meaningless fake company name, derived from the Italian word "pomodoro"—I like to do book examples using the Pomodo name since I own the pomodo.com domain name and thus can do what I want with it. So, since pomodo.com would be its domain name, the backward domain name for use in package names is com.pomodo.)

#### **session04/src/com/pomodo/model/Task.as**

```
package com.pomodo.model { #1
    public class Task {
        [Bindable]
        public var name:String; #2

        public function Task(name:String = "") { #3
            this.name = name;
        }
    }
}
```

## **Annotation Bubbles**

**#1** The com.pomodo.model package. ActionScript 3 typically uses the same “backwards domain name” convention as Java, so we create a com.pomodo.model.Task class in com\pomodo\model.

**#2** We are also creating a variable called name of type String, which we are indicating can be the source of a data binding with the [Bindable] annotation. Briefly, this annotation means that other code can be automatically notified when the value changes. (Binding will be the subject of many workshop sessions later on in this book.)

**#3** We are creating a constructor, with a default value an empty String for the Task name. In this constructor, we set the name variable to the name passed in, using the this keyword to disambiguate which name we are referring to.

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

## NOTE

ActionScript 3 supports packages with fewer restrictions than it did in ActionScript 2. It also supports namespaces. There are many details about what you can and can't do with classes, packages, and namespaces; we'll keep things simple and use the "one class per file" and "package in its folder" approach because it's the most straightforward.

Next, we lay out the UI and add code to create and destroy Tasks.

### session04/src/ToDoList.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/halo">
<fx:Script> #1
<![CDATA[
  import mx.collections.ArrayCollection; #2
  import com.pomodo.model.Task;

  [Bindable]
  private var _tasks:ArrayCollection = new ArrayCollection(); #3

  private function createTask():void { #4
    _tasks.addItem(new Task(newTaskTI.text));
  }

  private function deleteSelectedTask():void { #5
    _tasks.removeItemAt(taskList.selectedIndex);
  }
]]>
</fx:Script>
  <s:Panel title="Todo List" width="100%" height="100%"> #6
    <s:VGroup width="100%" height="100%"> #7
      <s:HGroup width="100%" verticalAlign="middle"> #8
        <s:SimpleText text="New Task"/> #9
        <s:TextInput id="newTaskTI" width="100%" #10
          enter="createTask() "/>
        <s:Button label="Create" click="createTask() "/> #11
      </s:HGroup>
      <s>List id="taskList" width="100%" height="100%" #12
        labelField="name"
        dataProvider="{_tasks}"/> #13
      <s:HGroup width="100%">
        <s:Button label="Delete" width="100%" height="30">
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

```

        enabled="{taskList.selectedItem != null}"           #14
        click="deleteSelectedTask() "/>
    </s:HGroup>
</s:VGroup>
</s:Panel>
</s:Application>

```

## Annotation Bubbles

- #1 Create an `fx:Script` element to hold ActionScript code inside a CDATA block
- #2 Next, we add imports, which make the classes accessible in this class.
- #3 We then create a `_tasks` variable of type `ArrayCollection`, and initialize it to a new `ArrayCollection`. This variable is the source of a data binding in the List, so we need to mark it with the `[Bindable]` annotation.
- #4 The `createTask` function calls the `addItem` method of the `_tasks` `ArrayCollection` with a new `Task` whose name is the value of the `newTaskTI.text`. This function has a void return value, meaning it returns nothing.
- #5 The `deleteSelectedTask` function calls the `removeItemAt` method of the `_tasks` `ArrayCollection`, removing the task whose index is the `taskList.selectedIndex`.
- #6 Next, we create a `Panel` whose title is "Todo List". It has a width and height of 100%, meaning it will take up the full width and height that are left after respecting the padding of the parent `Application`.
- #7 The first component we create inside the `Panel` is a `VGroup` which lays out its children vertically.
- #8 The first of these children is an `HGroup` which lays out its children horizontally. Inside the `HGroup`, we create a `SimpleText`, a `TextInput` and a `Button`.
- #9 New Task `SimpleText` label
- #10 The `TextInput` has an id of `newTaskTI`. In MXML, the id property of a component becomes its variable name (the MXML file is a class, and the id is the name of a global variable inside that class). If we don't provide an id for a component, Flex provides one for us—but then we don't know what it is, so we can't refer to the component in our code. Sometimes this is fine: We don't need to refer to the `Button`, so we don't bother giving it an id. Note that the `newTaskTI` calls `createTask` whenever it broadcasts the enter Event.
- #11 We also modify the `Create` button to call `createTask` when it broadcasts its click Event.
- #12 The `taskList` is editable.
- #13 The `taskList` has a `labelField` of name (since that is the property of the `Task` which we want displayed) and has its `dataProvider` bound to the `_tasks`.
- #14 Finally, the `Delete` button has its `enabled` property bound to whether there is a non-null `selectedItem` in the `taskList` (thus preventing a user from trying to delete a nonexistent `Task`) and has its click Event handled and trigger the `deleteSelectedTask` function.

## Key Points

- Flex applications typically consist of many MXML and ActionScript components, which are stored in `.mxml` and `.as` files. These components are organized into packages.
- MXML is used for UI layout and ActionScript (both in MXML Script blocks and in ActionScript files) is used for behavior. You can even put ActionScript code (for

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

example function calls) inside the values of MXML attributes, such as `click="deleteSelectedTask ()"`.

- These components communicate via data binding and by manually dispatching events.

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

## Session 5 - Spark, Halo and Flex 4 Namespaces

WHAT'S WITH ALL THESE xmlns: THINGS YOU'VE BEEN HAVING US TYPE IN THE FIRST THREE WORKSHOP ITEMS?



VERY OBSERVANT, ESPECIALLY THIS EARLY IN THE WORKSHOP. IT'S ACTUALLY NOT SO BAD. THERE'S A NAMESPACE FOR THE NEW "SPARK" COMPONENTS (WHICH GETS THE S PREFIX), A NAMESPACE FOR THE OLD "HALO" COMPONENTS (THIS GETS THE MX PREFIX), AND A NAMESPACE FOR CORE FLEX STUFF (THIS GETS THE FX PREFIX). THIS SEEMS COMPLEX, AND THERE IS A BIT OF PAIN IN LEARNING, BUT IT'S BETTER THAN WHAT ADOBE ORIGINALLY PROPOSED.



WOW, HOW BAD WAS THE ORIGINAL IDEA?

Fx PREFIXES. EVERYWHERE. YOU'D GO TO THE FxDentist FOR AN FxRootCanal. RYAN STEWART WOULD DRILL PERSONALLY!



Before we go further, we should confront something which stares at you when you first look at a Flex 4 application: Namespaces, and why we need three of them. To understand this, we need to understand the story of Flex components.

Once Upon a Time (in Flex 1.0, 1.5, 2 and 3), all the components were something called "Halo" components, since they had a nice glow. (If my memory is correct, in Flex 1 and 1.5 this used to be green; in Flex 2 and 3 it was blue.) Anyway, since there was one set of components, they were all in the same namespace. In Flex 3, this namespace was <http://www.adobe.com/2006/mxml>, so Flex applications

looked like this:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute">
  <mx:Button label="Hello World"/>
</mx:Application>
```

Simple, isn't it. One namespace, which gets assigned the prefix mx, letting you write things like mx:Button, mx:Application, etc. This simplicity hid a big problem, however: The Halo components were not easily "skinnable" by designers. To skin them beyond what you control with CSS, you'd typically have to subclass them and do a lot of custom coding. Since Flash and Flex are making their way into a lot of design-oriented development shops like agencies, this kind of thing happens more than the average Java or .NET developer would expect. And nobody wants to be wrestling with the core components under agency timetables and deadlines.

So, since Adobe understands designers better than most large companies, one of the themes Adobe had for Flex 4 is "Design in Mind". In

IN LOLCAT TERMS, "DESIGN IN MIND" MEANS "I'M IN UR SDK, REWRITIN UR COMPONENTZ"



©Manning Publications Co. Please post comments or correction  
 Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

marketing terms, this means that Adobe cares about designer-developer workflow and is striving to optimize this.

In this workshop session, we're going to see what a Flex 4 application using only Halo components looks like. We will build the same application as the previous workshop session, a Todo List. Those of you who have developed Flex 3 applications before will find this code very familiar.

#### session05/src/ToDoList.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application                                #1
    xmlns:fx="http://ns.adobe.com/mxml/2009"   #2
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/halo">
<fx:Script>
<![CDATA[
    import mx.collections.ArrayCollection;
    import com.pomodo.model.Task;

    [Bindable]
    private var _tasks:ArrayCollection = new ArrayCollection();

    private function createTask():void {
        _tasks.addItem(new Task(newTaskTI.text));
    }

    private function deleteSelectedTask():void {
        _tasks.removeItemAt(taskList.selectedIndex);
    }
]]>
</fx:Script>
<mx:Panel title="Todo List" width="100%" height="100%" #3
    layout="vertical">
    <mx:HBox width="100%" verticalAlign="middle">
        <mx:Label text="New Task"/>
        <mx:TextInput id="newTaskTI" width="100%"
            enter="createTask()"/>
        <mx:Button label="Create" click="createTask()"/>
    </mx:HBox>
    <mx>List id="taskList" width="100%" height="100%"
        labelField="name"
        dataProvider="{_tasks}"/>
    <mx:ControlBar width="100%">
        <mx:Button label="Delete" width="100%" height="30"
            enabled="{taskList.selectedItem != null}"
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum: <http://www.manning-sandbox.com/forum.jspa?forumID=442>

```

        click="deleteSelectedTask()"/>
    </mx:ControlBar>
</mx:Panel>
</mx:Application>

```

## Annotation Bubbles

#1 For those who are new to Flex, the root tag is an mx:Application, since it is a Halo application. (If you look back at the previous workshop sessions you will see that the root tag we have been using has been s:Application for a Spark Application.)

#2 We create the three XML namespaces. First, the fx prefix for core Flex namespace (<http://ns.adobe.com/mxml/2009>). Second, the s prefix for new Spark components namespace ([library://ns.adobe.com/flex/spark](http://ns.adobe.com/flex/spark)). Third, the mx prefix for old Halo components namespace ([library://ns.adobe.com/flex/halo](http://ns.adobe.com/flex/halo)).

#3 Finally, we create a bunch of Halo components, like Panel (#5), HBox (#6), Label (#7), List (#8) and ControlBar (#9). HBox and VBox are the Halo functional equivalent of HGroup and VGroup in Spark.

Next, we create a Task class which is identical to the Task class in workshop session

#3:

### session05/src/com/pomodo/model/Task.as

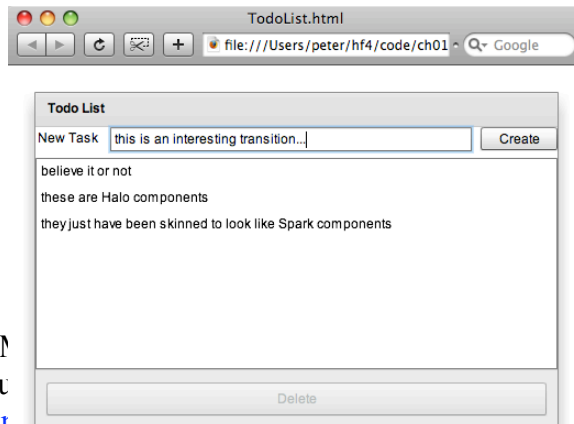
```

package com.pomodo.model {
    public class Task {
        [Bindable]
        public var name:String;

        public function Task(name:String = "") {
            this.name = name;
        }
    }
}

```

Running this application, we see the application shown below:



©  
At  
sar

SO, YOU'RE TELLING US THAT WE SHOULD USE THE NEW "SPARK" COMPONENTS SO A.J. OUR DESIGNER CAN SKIN THEM?



ASSUMING THAT IS TRUE, WHY HAVE THE HALO COMPONENTS?

WELL, FIRST OF ALL, FOR BACKWARD COMPATIBILITY WITH FLEX 3 APPS. SECOND, AND I WISH I WAS JOKING HERE, ADOBE DIDN'T GET DONE. FLEX 4 IS GOING TO SHIP WITHOUT A FULL SET OF SPARK COMPONENTS! SO, YOU'LL NEED TO USE SOME OF THE OLD HALO COMPONENTS LIKE DataGrid SINCE THERE WON'T BE ANY ALTERNATIVE. SO, YOU CAN'T IGNORE THE FACT THAT THESE NAMESPACES EXIST, AND THAT THERE ARE OLD AND NEW COMPONENTS. ISN'T LEGACY CODE GREAT!

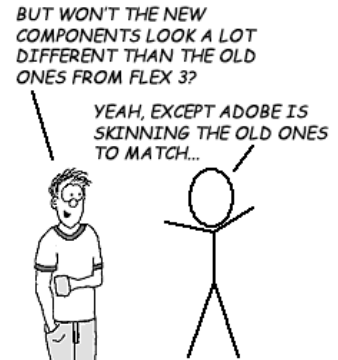


ments or corrections to the

Those of you who have Flex 3 experience may be surprised: This doesn't look like a Halo application in Flex 3! The reason for this is simple: Adobe understands that you need to mix and match Halo and Spark components in Flex 4, so they changed the style of the Halo components in Flex 4 to match the Spark components.

## Key Points

- Flex 4 applications typically use three XML namespaces, since Flex 4 is introducing an entirely new set of components (the Spark components).
- The old school Halo components are what were used in Flex 1-3. They have the mx prefix by convention, since that's what was used in Flex 1-3. The namespace for the Halo components is `library://ns.adobe.com/flex/halo`. You still need to use the Halo components where there are no Spark equivalents yet, such as DataGrid.
- The new Spark components use, by convention, an s prefix for the new namespace of `library://ns.adobe.com/flex/spark`. These components have "design in mind", which will allow designers and developers to work together in a more harmonious way.
- The fx prefix is for the core Flex namespace (<http://ns.adobe.com/mxml/2009>).



## What's Next?

*GOING FROM THIS FAST-PACED INTRODUCTION TO LEARNING ACTIONSCRIPT 3 MAY SEEM A LITTLE DISAPPOINTING, BUT I'LL KEEP IT REALLY BRIEF AND ASSUME YOU KNOW HOW TO PROGRAM. I WON'T BORE YOU WITH AN INTRODUCTION TO OBJECT ORIENTED PROGRAMMING.*



In this chapter we have had a very quick tour of Flex 4, gotten up and running, learned the basics of events and data binding and even understood what the heck all these xmlns things mean. In the next chapter full of workshop sessions we will go a bit slower, learning ActionScript 3 and the Flex 4 fundamentals.

No, I won't bore you with an introduction to object-oriented programming; in this book I'm assuming you are a software developer already--just not a Flex one (yet)

©  
Ⓛ

Please post comments or corrections to the <http://www.manning-sandbox.com/forum.jspa?forumID=442>