



**MEAP Edition
Manning Early Access Program**

Copyright 2008 Manning Publications

For more information on this and other Manning titles go to
www.manning.com

Table of Contents

Part One – Appcelerator Basics

- Chapter 1 – Introducing Appcelerator
- Chapter 2 – Developing on the client
- Chapter 3 – Developing on the server

Part Two – Developing PlayDate

- Chapter 4 – Iteration Zero: Planning and basic navigation
- Chapter 5 – Iteration One: Implementing login and signup
- Chapter 6 – Iteration Two: Implementing the user profile page
- Chapter 7 – Iteration Three: Implementing the friends page
- Chapter 8 – Iteration Four: Implementing the events page
- Chapter 9 – Iteration Five: Bringing it all together on the splash page

Part Three – Beyond the Basics

- Chapter 10 – Using the command line utility
- Chapter 11 – Security tips
- Chapter 12 – JavaScript functions and variables
- Chapter 13 – Data visualization
- Chapter 14 – Consuming web services
- Chapter 15 – Localization
- Chapter 16 – Using the Appcelerator widget framework

1

Introducing Appcelerator

User experience matters.

I wish I could claim that the above statement was a mind blowing realization unique to this book, but the tech companies making headlines (and gobs of cash) these days have realized that investment in user experience pays dividends. The following passage from a widely acclaimed book on user interaction design sums up this idea effectively:

If we design and construct products in such a way that the people who use them achieve their goals, these people will be satisfied, effective and happy and will gladly pay for the products and recommend that others do the same.

- Cooper, Reimann, and Cronin from *About Face 3 – The Essentials of Interaction Design*

As developers responsible for creating user engaging user experiences for the web, our goal is to create interfaces that allow our users to achieve their goals in as simple and self-evident a manner as possible. If we are successful in that task, our application will be successful and our users will become our most effective sales force. A successful application will be visually attractive, feature a fluid and responsive controls, and remain laser focused on helping the user do what they came into the application to do as quickly and easily as possible. A successful application will also deliver all of this while maintaining the reach and accessibility that have made web applications so popular and useful. That's the definition of a Rich Internet Application (RIA), and that is what we need to build to fulfill increasingly complex user interface requirements.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=461>

So how then do we go about building RIAs that are going to be cost effective to design and develop? Exploring this problem leads us to quickly discover that the server-generated HTML plus Ajax approach of traditional web applications is not going to scale well to create rich user interfaces for the web.

As an application requires more asynchronous data access, fluid and responsive control, and event-driven user interface, we find ourselves jumping through hoops to counteract the stateless nature of the web. We start to invent contrivances like "page flow" and abuse the HTTP session to maintain UI state on the server. The standard MVC frameworks of the web have evolved to give us great tools like Ruby on Rails, Struts 2, and Grails – but the server-centric programming models of these and other popular web frameworks simply do not give us the fluid and responsive interface that we need. What we really need is an event-driven, client/server style architecture where we can easily access server-side data asynchronously and maintain UI state and responsiveness on the client.

So where does that leave us as web developers? Is a patchwork of third party JavaScript toolkits, Ajax, and an MVC-style web framework going to give us the functionality we are looking for? Would such a solution be even remotely maintainable? Do we need to abandon the browser altogether and start programming applications for the Flash Player, Silverlight, or JavaFX? Do we need to double or triple the amount of UI code we write by coding our front-ends in Java for Google Web Toolkit? The answers to those questions are no, heck no, no, and no. There is a way to use the browser as the event-driven client application platform we need – all this while still leveraging the hard-earned HTML and JavaScript knowledge we have gained through a decade in the trenches of development for the web.

1.1 Appcelerator – RIA for the open web

The Appcelerator RIA framework (<http://www.appcelerator.org>) provides us with the tools to tame the browser as a client application platform and create client/server style architecture in a web application. Appcelerator is free and open source, and integrates out of the box with many major server-side programming languages and frameworks, like Java EE, .NET, PHP, Ruby via Rails or Merb, Perl, Python, and Google App Engine (also Python based). And since Appcelerator applications are written in HTML with some JavaScript on the client side, RIAs with Appcelerator run in the browser without need of a plug-in.

Appcelerator provides a message-oriented framework which allows UI components in the web browser and server-side services to easily communicate with one another asynchronously. In the web browser, Appcelerator provides three main components: the message broker, the Web Expression Language, and the Widget Framework. The widget framework is similar to other RIA widget libraries out there in that it provides a set of rich components (like data grids, windows, buttons, etc) that can be used out of the box in your application. Where it is different is that it is designed to be extensible, and provides facilities for wrapping other JavaScript widget libraries (we'll get in to that more in depth as we go along). However, what makes Appcelerator extraordinary is the combination of the Web Expression Language, message broker, and service broker running on the server side. These

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=461>

components provide the message oriented framework that makes your life as an RIA developer much simpler. A high level diagram of how the pieces of the Appcelerator RIA framework is shown in Figure 1.1.

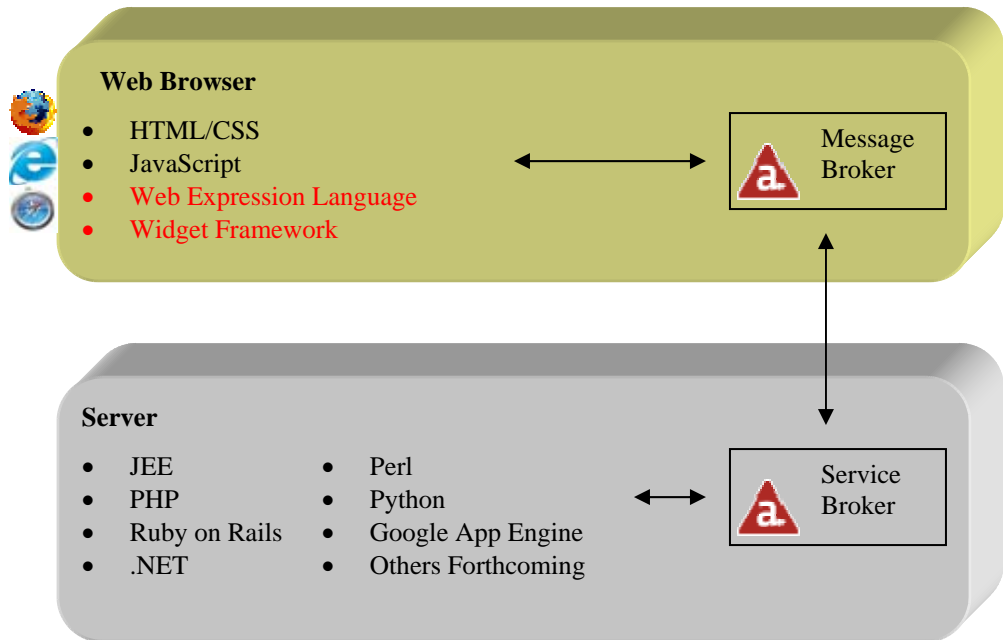


Figure 1.1 – High level architectural diagram of Appcelerator.

So what does an Appcelerator RIA look like? Without going into too much detail, we will see in Figure 1.2 that UI components (either standard HTML elements or Appcelerator Widgets) on an HTML page in a web browser communicate with one another by publishing and subscribing to messages which are handled by the Appcelerator Message Broker running in the browser. When a request for server-side data is made, the message broker makes asynchronous requests to the Appcelerator Service Broker running on the server. The service broker delegates to the proper service method, which handles the request, executes business logic, and returns a response message with a JSON data payload. Any UI components that have subscribed to that response message will be alerted when it is returned, and can bind data from the message payload to on-screen elements. All serialization and de-serialization is handled by the framework. Working in concert, the message broker and the service broker give us the seamless client/server architecture we need to build RIAs.

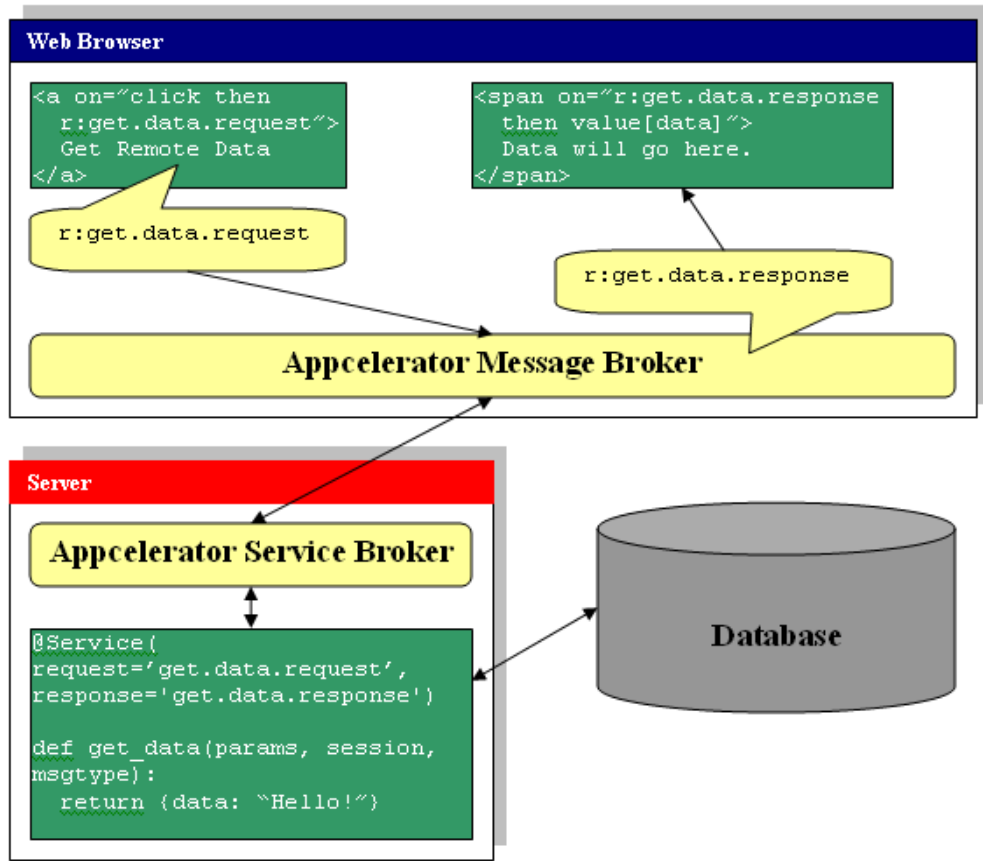


Figure 1.2 – Basic control flow of an Appcelerator application

You probably noticed that mysterious ‘on’ attribute in the HTML elements in Figure 1.2. That’s Appcelerator’s breakthrough innovation for handling events and publishing and subscribing to messages, which they call the Web Expression Language (WEL). The WEL is designed to insulate you from having to write reams of JavaScript to accomplish basic UI programming tasks, and empowers developers to easily publish and subscribe to messages. We’ll cover that in great detail later in the book, but thanks to the natural language (well, kind of natural) syntax you can probably already get the gist of what those expressions are doing. Despite innovations like the WEL, Appcelerator really only builds on the foundation put in place by the open standards of the web. As we learn more about Appcelerator, you will find the learning curve to be a gentle climb because as a web developer, you are already familiar with most of the techniques and standards the framework uses.

I called Appcelerator RIA for the open web because it provides us with the tools to create event-driven, client/server web applications using the open standards of the web: HTML, JavaScript, and CSS. Appcelerator applications are (on the client side) simple HTML files containing Appcelerator extensions like the 'on' attribute we saw in Figure 1.1 and Appcelerator widgets, like modal dialogues, data grids, and many more. Appcelerator delivers all of this without the use of a browser plug-in or an intermediate compiler while leveraging the tools and open technologies of development for the web.

1.1.1 Do we really need another JavaScript toolkit/widget library?

No, we don't. But Appcelerator isn't just a JavaScript toolkit or widget library. The Appcelerator guys like to talk about their product as "RIA + SOA" because Appcelerator provides not only an extensible widget library, but also a message queue and framework for creating a service-oriented application. JavaScript libraries like ExtJS, Prototype, and Scriptaculous are good at providing you with an a la carte selection of effects, widgets, and JavaScript extensions for your UI, but none of them provide an actual framework for creating client/server RIAs. By providing easy communication between UI components and services, Appcelerator provides a fully integrated solution for the web tier.

1.1.2 Can I really love Appcelerator? I've had my heart broken before...

Given the fact that many RIA frameworks and technologies promise a simple solution for creating attractive UIs and deliver something that is anything but simple, I understand your skepticism. But even so, I think you'll find there's a lot to love about Appcelerator.

YOU ALREADY KNOW APPCELERATOR

On the client side, Appcelerator applications consist primarily of HTML, CSS, some simple JavaScript, and Web Expression Language statements. If you're a web developer, it's safe to assume you possess the former three skills, and a lot of folks find the WEL highly intuitive and easy to pick up due to its natural language style. Along with that, you get to choose your favorite server-side language as a companion, according to your preference - each supported server back end features tight integration with the client-side pieces of Appcelerator. That means that rather than replacing your web programming knowledge, you are supplementing it and extending it with a few new syntactical wrinkles and a shift in the way you think about writing web applications.

MESSAGE PUBLISHING AND SUBSCRIBING BAKED RIGHT IN

Event handling can be tedious when developing RIA, but the Appcelerator Web Expression Language makes it extremely simple. Are there any Adobe Flex developers using the Cairngorm micro-architecture (<http://labs.adobe.com/wiki/index.php/Cairngorm>) in the house? If so, you can attest to having to write literally hundreds of lines of ActionScript to code up a new event in your UI. First there's the value object, then the event, then the command, then the business delegate, then registering the event with the controller... the code is boilerplate and is fairly easy to copy/cut/paste, but writing it gets old fast when your application has more than a dozen or so events (and what RIA doesn't?).

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=461>

With Appcelerator, if you want an event fired and back end service request made when a button is clicked, you'd write something like this:

Listing 1.1 – Submitting the contents of a form to a back end service in Appcelerator

```
<input id="loginSubmit" type="button" value="Log In"
      on="click then r:authen.login.request" fieldset="loginFields"/>
```

The single tag in Listing 1.1 does a bunch of things for you in one to two lines of code, depending on how you format your HTML. It marshalls all of the data from form fields from a set of fields called 'loginFields' and puts that data in a JSON object. When the button is clicked, a remote message is sent with the data from the form and routed to the service method on your server back end that handles requests to 'authen.login.request'. The response from that service invocation will be sent via a message called 'authen.login.response', which other UI elements can subscribe to with another single line of code. If you have some background with other RIA frameworks (like Adobe Flex, as mentioned earlier), I'm sure you already realize how handy it is to have all that busywork handled for you.

APPCCELERATOR IS ACTUALLY FREE, OPEN SOURCE, AND BASED ON OPEN STANDARDS

Appcelerator is licensed under the GNU Public License and is free to download and use. Appcelerator also provides an Eclipse-based IDE (which is based on the very cool Aptana IDE and is still in the Alpha development stage at the time of this writing), which is also free of charge. And as we have mentioned earlier, since Appcelerator is based on HTML, CSS, and JavaScript, the standard and runtime environments at the core of Appcelerator are open as well. There are commercial services and licensing options available from Appcelerator Inc., but there's nothing up anyone's sleeve in terms of charging you for an IDE, a feature set, or anything along those lines. Appcelerator is closer to free as in beer rather than free as in free trial period. This and the other preceding strengths we have discussed set Appcelerator apart from its competitors in a variety of ways, which we will take a moment to examine now.

APPCCELERATOR IS THE BEST WEB FRAMEWORK AVAILABLE FOR MANY PLATFORMS

For many programming languages and technology stacks, there are several mature and battle tested web frameworks available which offer a means to create MVC style web applications with varying levels of pain. But the utility and availability of web frameworks is not equal among all platforms. Java has a number of well established web frameworks to choose from, and Ruby has Rails, which many call the language's killer application. But in my opinion (and I know many would disagree), the web frameworks available for PHP, .NET, Python, and Perl are not as advanced as those available in Java (also Groovy) and Ruby. Appcelerator provides a great alternative in those situations, particularly in one of Appcelerator's newest incarnations, which is Google App Engine (a Python based cloud computing platform – learn more at <http://code.google.com/appengine/>). The web

framework that is included by default with App Engine (again, in my opinion) leaves much to be desired, and better results can be achieved by using Appcelerator.

1.2 Appcelerator and other RIA frameworks

The Rich Internet Application space is exploding right now, as more and more software development organizations realize the business value of RIAs. As a result, there are many vendors out there trying to get their piece of the animated pie chart that is the RIA market. Let's take a look at how Appcelerator compares to some of the players in this space.

1.2.1 Heavyweight, plug-in based frameworks

The browser has issues. JavaScript, as a language, has issues. The browser was designed as a standardized way of viewing documents on the web, not as a client application platform. We discussed some of these shortcomings earlier in this chapter as we enumerated the shortcomings of server-generated HTML web applications. In response to this, a few major RIA frameworks have emerged that use plug-ins to provide an environment in the browser to address some of these issues – in exchange for some accessibility (in the sense that users will need a special plug-in to access your application), you get a more robust operating environment for your application.

Arguably, the three biggest players on the RIA for a browser plug-in scene are Adobe Flex (Flash Player plug-in), Microsoft Silverlight, and JavaFX. Silverlight and JavaFX are still relatively new kids on the block, but given the behemoths behind each technology I have no doubts that developers will flock to their platforms just by sheer force of gravity. JavaFX seems to be little more than a press release at this point, but Silverlight is gaining some traction and is already in beta testing for its second release.

While I will concede that Silverlight and JavaFX are likely to make a splash soon, they are not going to factor in to the discussion at this stage in their development. Adobe Flex is the most mature product of the lot, and is still the market leader, so I will use it as the basis for comparison. I actually like Flex quite a bit, for a variety of reasons. The Flash Player is ubiquitous, and the Adobe CS3 product suite enables developers and designers to work together effectively. The Flash Player also has real-time data streaming capabilities and a high speed data-transfer protocol (AMF3) that boasts 5x to 10x performance increases in back end data requests when compared to a standard Ajax request for XML over HTTP. On the development side of things, MXML (Flex markup language) and ActionScript work well together to provide a useful (if verbose and at times inelegant) syntax.

So if the browser is so broken and Flex is so awesome, why bother doing RIAs for the browser at all? Shouldn't we all start learning MXML and ActionScript for Flex and Flash? Well, if that sounds like a lot of work (and a lot of added cost) to you, that's because it is. The first semi-hidden cost of Flex (and the other heavy-weight plug-n based frameworks) is in the overall complexity of the solution. I charted the unscientific graph in figure 1.2 on the

richness of a UI compared to the overall complexity of the solution in a blog posting of mine on this topic a few months ago (See Figure 1.3).

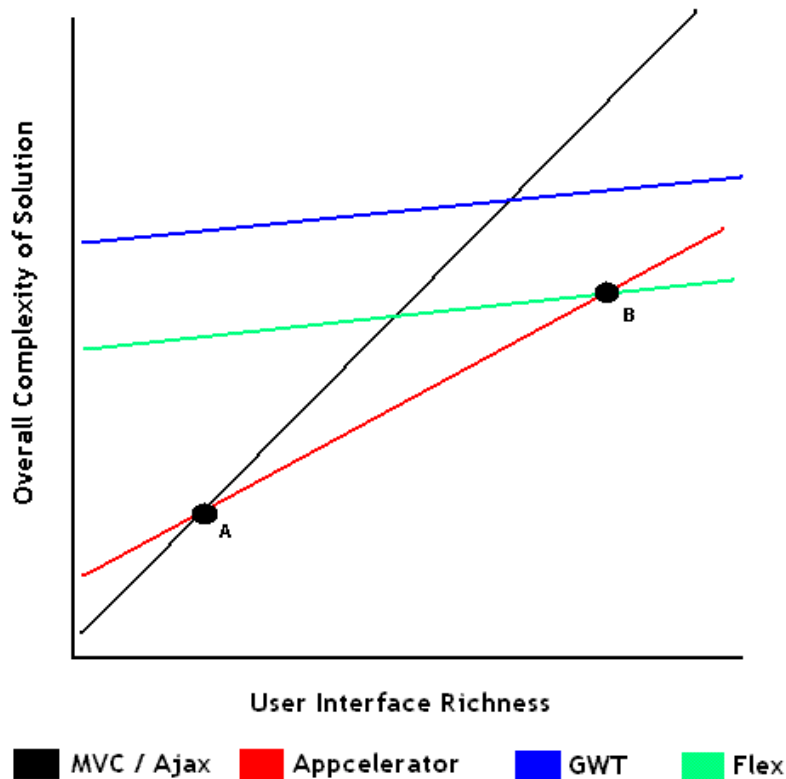


Figure 1.3 – Overall complexity of solution compared to user interface richness

In terms of lines of code (and the complexity of that code), the curve for Adobe Flex starts high and stays high. You will write a lot of ActionScript to do event handling in a Flex application (if you follow good coding practices), and for all but the most complex applications, you will feel like you just swatted a mosquito with a baseball bat.

There is also a great deal of upfront investment necessary to create a Flex application. The first is in training – if you or your developers are web application developers coming from an HTML/CSS background, there is a non-trivial learning curve to start developing Flex applications. You will also need to invest in some Adobe software products to be productive – their Flex Builder IDE starts at 250 bucks in USD (the pro version with charting components

and performance profiling is 700 big ones), and that's not including the CS3 products you will need to make your UI flashy (pun intended). To be fair, you could use the free Flex SDK and compile your code manually (or via Ant or Rake), but given the minor changes you need to incrementally test when coding UI, that gets old fast – trust me. Also, if you are going to take advantage of the AMF3 binary data transfer protocol, get ready to descend into some serious XML configuration fun and introduce a little extra complexity to your product. All this is fine if you really and truly need the horsepower that Flex brings to the table, but it becomes an encumbrance if your app really doesn't need to chart 7,000 records of data that get updated in real time.

In an Appcelerator application, things that should be easy are easy. Whether it is showing or hiding an element with a Fade effect or marshalling and submitting form data, you can get it done with a few lines of code. And thanks to the messaging services available via the framework, the browser becomes a viable RIA platform that makes a plug-in unnecessary for 95% of applications. JavaScript and the browser may have their issues, but the browser remains the most universally accessible web application execution environment out there, and is abandoned at the developer's peril.

However, there comes a point when the browser simply breaks down as an application platform (marked at point B in Figure 1.2). If you need streaming media, the ability to simultaneously transfer and work with several thousand records of data, or pixel-perfect control over the layout and look of your UI, Flex is going to be a good choice. But unless your UI and application will reach that level of complexity, I am of the opinion that the investment is not worth the return, and that you will be much happier (and more productive) using Appcelerator.

1.2.2 Comparing other JavaScript toolkits

I mentioned earlier that Appcelerator is not just a JavaScript toolkit – while it is possible to use the Appcelerator web SDK to add richness to any web page, Appcelerator provides much more than a widget library and some effects. Appcelerator's message and service brokers transform the browser into an event-driven RIA platform that features tight integration with many popular server-side programming languages. Many JavaScript libraries are a grab bag of functionality that you have to piece together yourself, whereas Appcelerator seeks to provide a full stack solution from UI to web service invocation.

The sheer volume of JavaScript widgets and frameworks in the market today is staggering, leading many applications to become an amalgamation of third party JavaScript widgets and libraries. But some of these widget libraries are pretty darn cool, so it would be a shame to ignore them in favor of what Appcelerator provides out of the box. Luckily, Appcelerator provides an integration framework for JavaScript widgets that allows developers to wrap their favorite widgets (or libraries) for usage with Appcelerator and the Web Expression Language. We'll cover how to do that later in the book, but for now just get excited about the fact that the Appcelerator community adds new third party widgets to the

framework every day (okay, maybe not every day, but often), and that it's easy to do it yourself.

1.2.3 Appcelerator's sweet spot

Appcelerator's sweet spot is the space between content-centric, page based web applications with minimal UI requirements and web applications like stock market analytics, graphics editing, and desktop-style applications that are data intensive and require complex graphics and client-side business logic. That gap is large and important, and features most of the applications that are emblematic of Web 2.0 like Facebook, Flickr, or Basecamp. Most of these types of applications are written today using server-centric MVC style programming, and as a result can be insanely complex and require some serious JavaScript chops to develop and maintain.

Appcelerator provides a sane alternative to create these kinds of applications by reducing the amount of code needed to make web sites dynamic and responsive. And by assuming that all data access will be service-oriented and asynchronous, Appcelerator makes creating these data-driven RIAs simple and fun. Simply put, Appcelerator pushes the browser past its current breaking point in the world of MVC plus Ajax and transforms it into a viable client application platform. But that's enough chit-chat. If you've made it this far, you're probably ready to kick the tires on a Rich Internet Application with Appcelerator. Let's slide behind the wheel and see how it handles.

1.3 Bootstrapping ourselves for development

Before we start slinging code, we have a bit of work to do to get our machines set up to create and manage Appcelerator projects. But luckily for us, one of the nicest bits about working with Appcelerator is how easy it is to get started. If you haven't done so already, fire up your browser and head out to <http://www.appcelerator.org>. If it is your first trip to the website and you haven't signed up for the developer network, do so now. You will need the credentials you use to log in later on when we install Appcelerator.

1.3.1 The Appcelerator Developer Network

After signing up for the Appcelerator Developer Network, you will be greeted with a page displaying your shiny new developer profile, shown in figure 1.4.



Figure 1.4 – The Appcelerator Developer Network: Your link to the Appcelerator Community

The development team for Appcelerator hasn't just built an RIA framework – they've created a support system for your RIA development to be successful. Take a few moments to cruise around the site – maybe upload a photo of yourself (or a picture of Stewie Griffin if you feel he better represents you) and browse the “Groups” section to see current conversations going on between Appcelerator developers and the core team. You'll notice quickly that nearly every thread is responded to by an Appcelerator developer or a core team member within an hour or so – this kind of responsiveness to the community is a hallmark of the Appcelerator project. It's a good feeling to know that if you're stuck, you won't go without help for long.

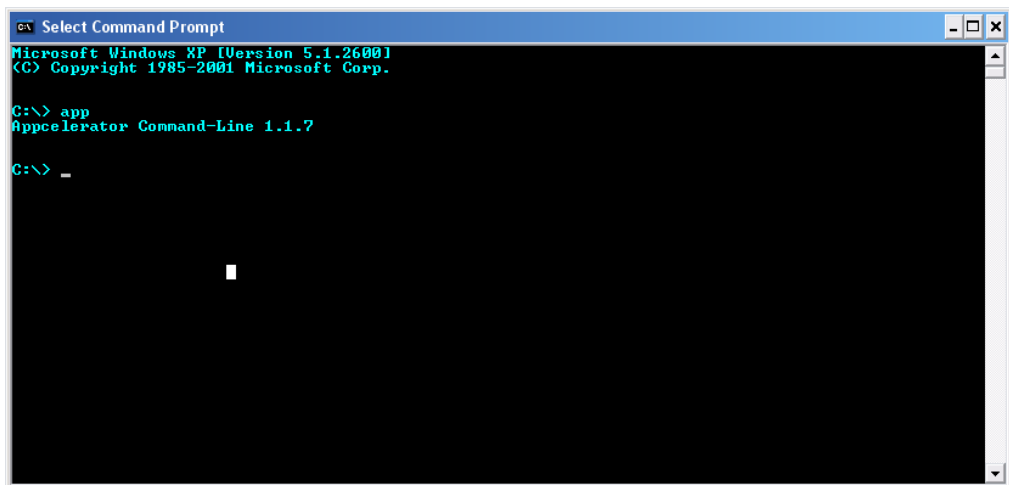
Also under groups, you will find one entitled “Appcelerator Book”. If you guessed that this group is the place to post questions and discussion about this text, then you win a gold star. I am out on the dev network nearly every day, and I try to demonstrate the same kind of responsiveness to questions that the rest of the core development team does. If you have any questions or there are bits of this book that might require some clarification, feel free to post those items there and I will respond as soon as humanly possible.

The dev network is also your one stop shop for any and all things Appcelerator – reference documentation, demos, their Subversion repository, and more. Right now, we're interested in downloading the Appcelerator SDK Installer, so let's download that now. Click on the “Products” link at the top of the page at the dev network and download the Appcelerator SDK for your platform.

1.3.2 Installing the Appcelerator SDK

Once you have successfully downloaded the installer, go ahead and launch it. The installer is fairly straightforward – you can very happily accept the defaults and let the installer do its thing. One thing to note is that the Appcelerator installer will automatically install a Ruby interpreter for your system if one is not already installed. In our case this is a good thing, because we are going to create a Ruby on Rails (<http://www.rubyonrails.org>) application for our “Hello World” example. You can also at this point specify a directory into which Appcelerator will install its self.

When the installer finishes, it will launch a browser window play a “getting started” movie from Appcelerator CTO Nolan Wright. You can watch it if you like, but we will be going over that same information in just a few moments, with a little extra context. So now we have installed the Appcelerator SDK and the Command Line Interface. The CLI gives us access to the `app` command from our command prompt. Open up a terminal and type `app` now to make sure everything is working nicely, as seen in figure 1.5.

A screenshot of a Windows Command Prompt window titled "Select Command Prompt". The window shows the following text: "Microsoft Windows XP [Version 5.1.2600] (C) Copyright 1985-2001 Microsoft Corp." followed by a prompt "C:\> app" and the output "Appcelerator Command-Line 1.1.7". Below this, there is another prompt "C:\> _" and a small white square cursor on the next line.

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\> app
Appcelerator Command-Line 1.1.7

C:\> _
```

Figure 1.5 – Typing the “app” command shows us which version of the CLI we have installed and working

Typing `app help` will give a listing of all the various commands that are available to us, but for now we only care that the CLI is working and that we are now ready to create a basic Appcelerator application.

1.4 Say Hello to Appcelerator

We will now use the `app` command to create a simple “Hello World” RIA. For convenience, create a directory now where you will house the example applications for this book. Change into this directory now and execute the following command: `app create:project . helloworld ruby`. So what does this command do? It tells Appcelerator to create a new

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=461>

project in the current directory called “helloworld” and says that we will be using Ruby on Rails as our service integration point. If you’ve never developed a Ruby on Rails application, don’t panic – you won’t need any specific Rails knowledge at any point in this book. For your first project, you will be asked if you have an Appcelerator Developer Network account. We do indeed, so provide your e-mail address and password to the CLI to continue. If you are behind a proxy server, say “Yes” when prompted.

At this point, you will see a lot of download progress flying by (figure 1.6). The latest versions of the web SDK, widgets, and other Appcelerator components are being downloaded and installed on your machine. If you do not have Rails and its dependent gems (libraries) installed, you will be prompted to do so during the project creation process. If you are prompted to install a SQLite gem (the default database for Rails), choose option 2, the pure Ruby version.

```

Command Prompt - app create:project . helloworld ruby
(23/23) ruby: 31% |ooooooooooooooooooooo| 9.3KB 617.0
(23/23) ruby: 34% |ooooooooooooooooooooo| 10.3KB 683.7
(23/23) ruby: 37% |ooooooooooooooooooooo| 11.1KB 740.4
(23/23) ruby: 41% |ooooooooooooooooooooo| 12.1KB 390.5
(23/23) ruby: 44% |ooooooooooooooooooooo| 13.1KB 422.8
(23/23) ruby: 47% |ooooooooooooooooooooo| 14.1KB 455.0
(23/23) ruby: 51% |ooooooooooooooooooooo| 15.1KB 487.3
(23/23) ruby: 52% |ooooooooooooooooooooo| 15.4KB 496.3
(23/23) ruby: 55% |ooooooooooooooooooooo| 16.4KB 528.5
(23/23) ruby: 59% |ooooooooooooooooooooo| 17.4KB 560.8
(23/23) ruby: 62% |ooooooooooooooooooooo| 18.4KB 593.0
(23/23) ruby: 65% |ooooooooooooooooooooo| 19.4KB 625.3
(23/23) ruby: 66% |ooooooooooooooooooooo| 19.7KB 634.2
(23/23) ruby: 70% |ooooooooooooooooooooo| 20.7KB 666.5
(23/23) ruby: 73% |ooooooooooooooooooooo| 21.7KB 698.7
(23/23) ruby: 77% |ooooooooooooooooooooo| 22.7KB 731.0
(23/23) ruby: 80% |ooooooooooooooooooooo| 23.7KB 763.3
(23/23) ruby: 81% |ooooooooooooooooooooo| 23.9KB 772.2
(23/23) ruby: 84% |ooooooooooooooooooooo| 24.9KB 542.1
(23/23) ruby: 88% |ooooooooooooooooooooo| 25.9KB 563.9
(23/23) ruby: 91% |ooooooooooooooooooooo| 26.9KB 585.6
(23/23) ruby: 94% |ooooooooooooooooooooo| 27.9KB 607.4
(23/23) ruby: 98% |ooooooooooooooooooooo| 28.9KB 629.1
(23/23) ruby: 100% |ooooooooooooooooooooo| 29.4KB 639.6
KB/s ETA: 00:00:00

Installed ruby 1.0.4
Using service ruby 1.0.4
Rails, json, and sqlite3 must be installed to create a project.
Install dependencies now? [Yn]

```

Figure 1.6 – The first time you create an Appcelerator project all the required goodies are installed

After the download process is complete, you should have a new directory created called “helloworld”. This directory contains a full Ruby on Rails application directory structure, plus some additions to facilitate the development of Appcelerator applications. Let’s take a look at some of the more interesting pieces that get created (also in figure 1.7):

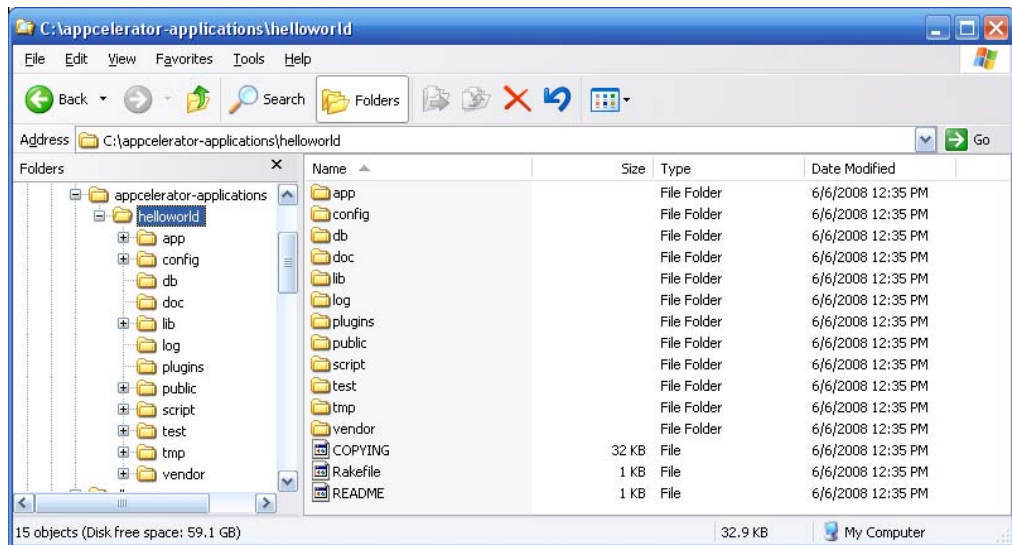


Figure 1.7 – The directory structure of an Appcelerator/Rails application

- `$ROOTDIR/app/services` – this directory contains all the Appcelerator service code we will write. These services will handle authentication, data access, and any server-side logic we need. For applications that use other server-side back-ends, like Java or PHP, this directory location will change, but there is always one directory where your service interfaces live.
- `$ROOTDIR/public` – all of your publicly accessible files should go here. Images, stylesheets, JavaScripts, and all the HTML files that will form the client side piece of our application will live here. Again, the location of this directory will vary depending on the server back end you choose to use, but every project will have a place where all files that are publicly served by the web server will live.
- `$ROOTDIR/public/images` and `$ROOTDIR/public/javascripts` – Appcelerator places some additional assets in each of these folders. Under the images directory, we are given a few handy graphical assets, including a few styles of spinner GIFs. The javascripts directory contains the JavaScript file you will need to include on your pages for Appcelerator to function.
- `$ROOTDIR/public/swf` and `$ROOTDIR/public/widgets` – in addition to the standard 'images', 'javascripts', and 'stylesheets' directories that are present in every Rails application, Appcelerator also creates the 'swf' and 'widgets' directories to house widgets and in some cases the compiled SWF files.

Most of the other files and folders in the new directory have to do with specific Ruby on Rails

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=461>

functionality. These will be familiar to you if you have done any RoR development, but at this point we don't need to be aware of the details surrounding them. It just turns out that RoR provides us with a couple of things that are going to make our lives easier for this (and future) examples of how to build Appcelerator applications. All Rails apps have an embedded web server called WEBrick and are configured to work out of the box with the lightweight SQLite database. That means we don't have to worry about configuring the server side piece at all. In fact, our Appcelerator application is already set up for us to take a look at! Let's fire it up and see that everything is working properly.

1.4.1 Hello, Appcelerator!

Open up a command prompt to your Appcelerator application's home directory and type the command `app run:project`. This will fire up WEBrick and make our application available on <http://localhost:3000/> - navigate there now in your web browser and you should see something like figure 1.8.

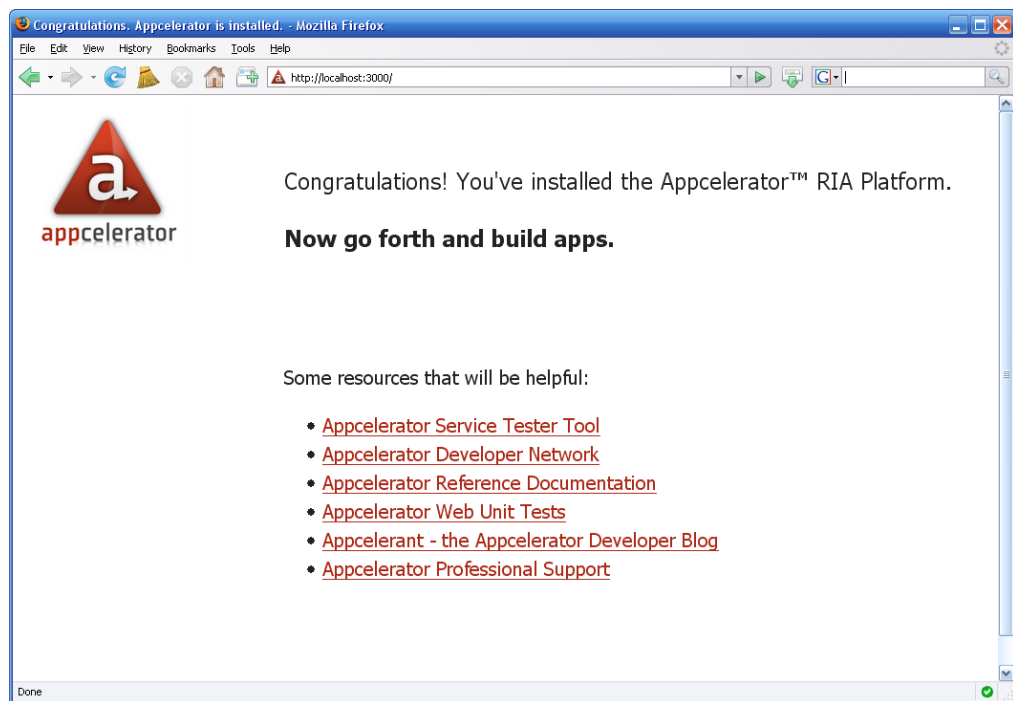


Figure 1.8 – Congratulations! Appcelerator is installed and ready to go.

Not terribly interesting, but it's good to see you can follow directions ;). Let's do a basic example using the test service that is built in to every new Appcelerator application. Open up `$ROOTDIR/public/index.html` in your favorite text editor and delete the contents of the
©Manning Publications Co. Please post comments or corrections to the Author Online forum:
<http://www.manning-sandbox.com/forum.jspa?forumID=461>

<body> tag. In its place, we will create a simple UI to call that service and display the results. Inside the body tag, place the code from listing 1.2.

Listing 1.2 – Hello World with Appcelerator

```
<h1>Hello Appcelerator!</h1>

<!--
  This is a basic form with a single input field.
  The "on" attributes of the basic HTML elements
  are syntactical elements of Appcelerator's Web
  Expression Language
-->
<form on="enter then click[id=submit]">
  <label>Enter A Message:</label><br/>
  <input id="message" type="text" fieldset="messageData"/>
  <input id="submit" type="button" fieldset="messageData" value="Say"
    on="click then r:app.test.message.request"/>
</form>

<br/>

<span style="display:none"
  on="r:app.test.message.response then
    value[message] and show and effect[Highlight]">
</span>
```

Save the file and fire up a web browser. If you navigate to <http://localhost:3000/>, the default web server port for Ruby on Rails applications, you should see a rather ugly form, as in figure 1.9.

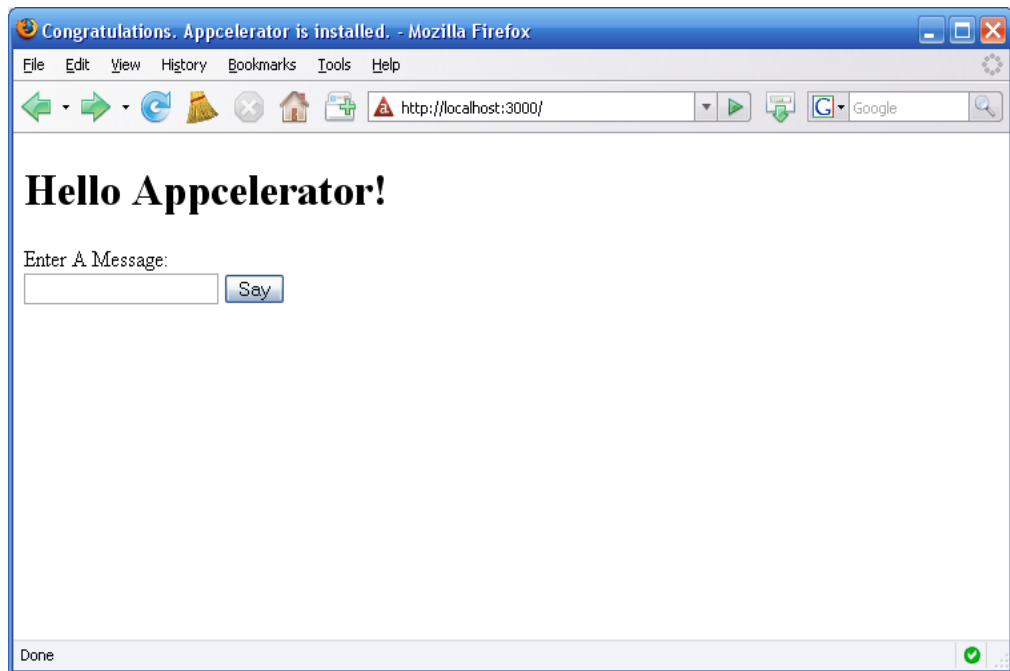


Figure 1.9 – A simple form with Appcelerator messaging

When you start to play around with it, you will notice that when you hit enter or click the “say” button, your message is submitted to the server. A response is then sent back with the message “I got from you: [your message]” and a span element appears with a highlight effect. Not bad for a small chunk of code. In the next chapter, we’ll start to get into how all of this magic works, and begin to understand the basics of messaging and back end services.

1.5 Summary

In this chapter, we have explored the general utility of Rich Internet Applications and some of the prevailing strategies for developing applications in this space. We came to the conclusion that MVC plus Ajax is not going to cut it, and that browser plug-ins have their share of problems also. We learned a bit about how Appcelerator turns the browser into an event-driven client application platform, and how we can use open web standards to develop RIAs. In the next chapter, we will focus on the techniques and tools necessary to develop the client side pieces of an Appcelerator RIA.