CHAPTER 24

Communicating with the Web

Mobile technology and the ubiquitous nature of the Web have changed the world we live in. You can now sit in the park and do your banking, search Amazon.com to find reviews of the book you're reading, and check Twitter to see what people in every other park in the world are thinking about. Mobile phones have moved well past just calling and texting—now, you have instant access to the world's data, too.

You can use your phone's browser to reach the Web, but often the small screen and limited speed of a mobile device can make this problematic. Custom apps, specially designed to pull in small chunks of particularly suitable information from the Web, can provide a more attractive alternative to the mobile browser.

Figure 24-1.

In this chapter, we'll take a look at App Inventor components that access information from the Web. You'll learn how to show a web page within the user interface of your app, and you'll learn about APIs and how to access information from a web service.

Creativity is about remixing the world, combining (*mashing*) existing ideas and content in interesting new ways. Eminem is among many artists over the past few decades who popularized the music mashup when he set his Slim Shady vocal over AC/DC and Vanilla Ice tracks. This kind of "sampling" is now common, and numerous artists, including Girl Talk and Negativland, focus primarily on creating new tracks from mashing together old content.

The web and mobile world are no different: websites and apps remix content from various data sources, and most sites are now designed with such interoperability in mind. An illustrative example of a web mashup is *Housing Maps*, pictured in *Figure 24-1*, which takes apartment rental information from *Craigslist* and mashes it with the Google Maps API.

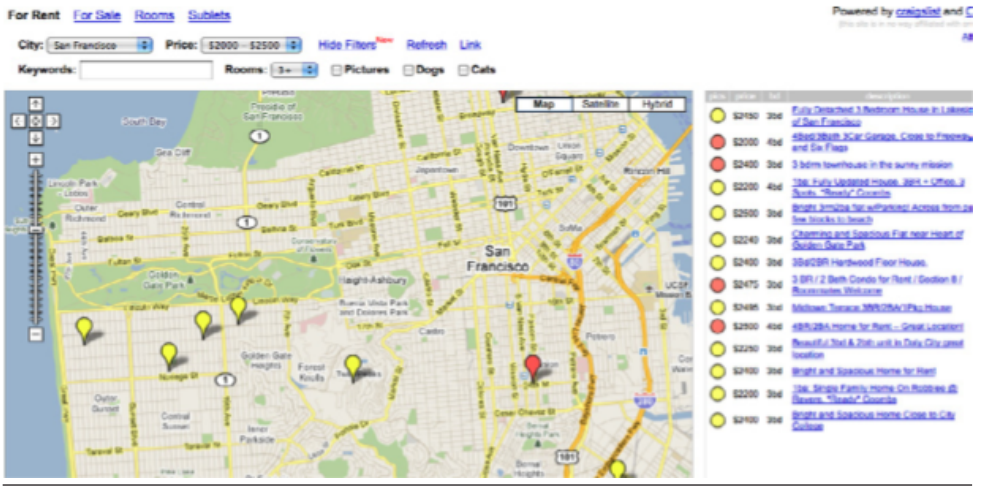

Figure 24-2. Housing Maps mashes information from Craigslist and Google Maps

Mashups akin to Housing Maps are possible because services such as Google Maps provide both a website and a corresponding *web service API*. We humans visit *http://maps.google.com/* in a browser, but apps such as Housing Maps communicate *machine to machine* with the Google Maps API. Mashups process the data, combine it with data from other sites (e.g., Craigslist), and then present it in new and interesting ways.

Just about every popular website now provides this alternative, machine-to-machine access. The program providing the data is called a *web service*, and the protocol for how a *client* app should communicate with the service is called an *application programmer interface*, or API. In practice, the term *API* is used to refer to the web service, as well.

The Amazon Web Service (AWS) was one of the first web services, as Amazon realized that opening its data for use by third-party entities would eventually lead to more books being sold. When Facebook launched its API in 2007, many people raised their eyebrows. Facebook's data isn't book advertisements, so why should it let other apps "steal" that data and potentially draw many users away from the Facebook site (and its advertisements!)? Yet, its openness led Facebook toward becoming a *platform* instead of just a site—meaning that other programs could build on and tap into Facebook's functionality, and no one can argue with its success today. By the time Twitter launched in 2009, API access was an expectation, not a novelty, and Twitter acted accordingly. Now, as shown in *Figure 24-2*, most websites offer both an API and a human interface.

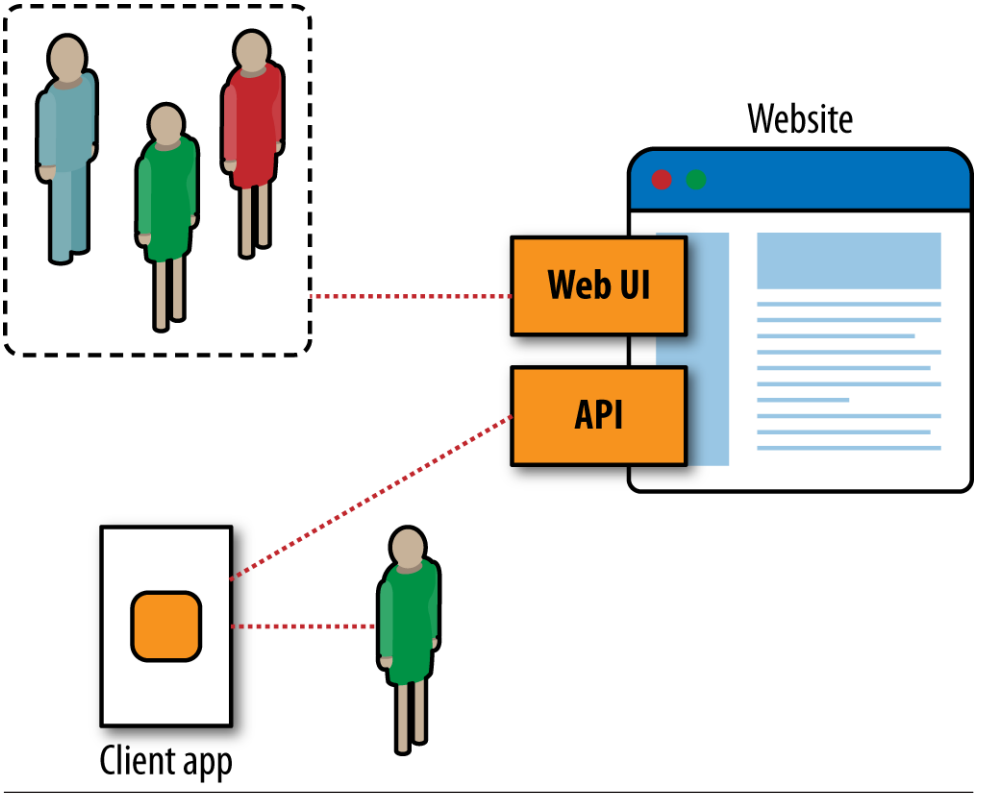

Figure 24-3. Most websites provide both a human interface and an API for client apps

Thus, the Web is one thing to us average humans (a collection of sites to visit). To programmers, it is the world's largest and most diverse database of information.

The WebViewer Component

The `WebViewer` component lets you show a web page within your app. You can show a Google Maps page showing the user's current location, a twitter page showing the most recent trending topics related to your app, or a page from nba.com showing the statistics for your favorite players.

`WebViewer` (see *Figure 24-3*) is like the `Canvas` component in that it defines a subpanel of the screen. But whereas `Canvas` is used for drawings and animations, `WebViewer` shows a web page.

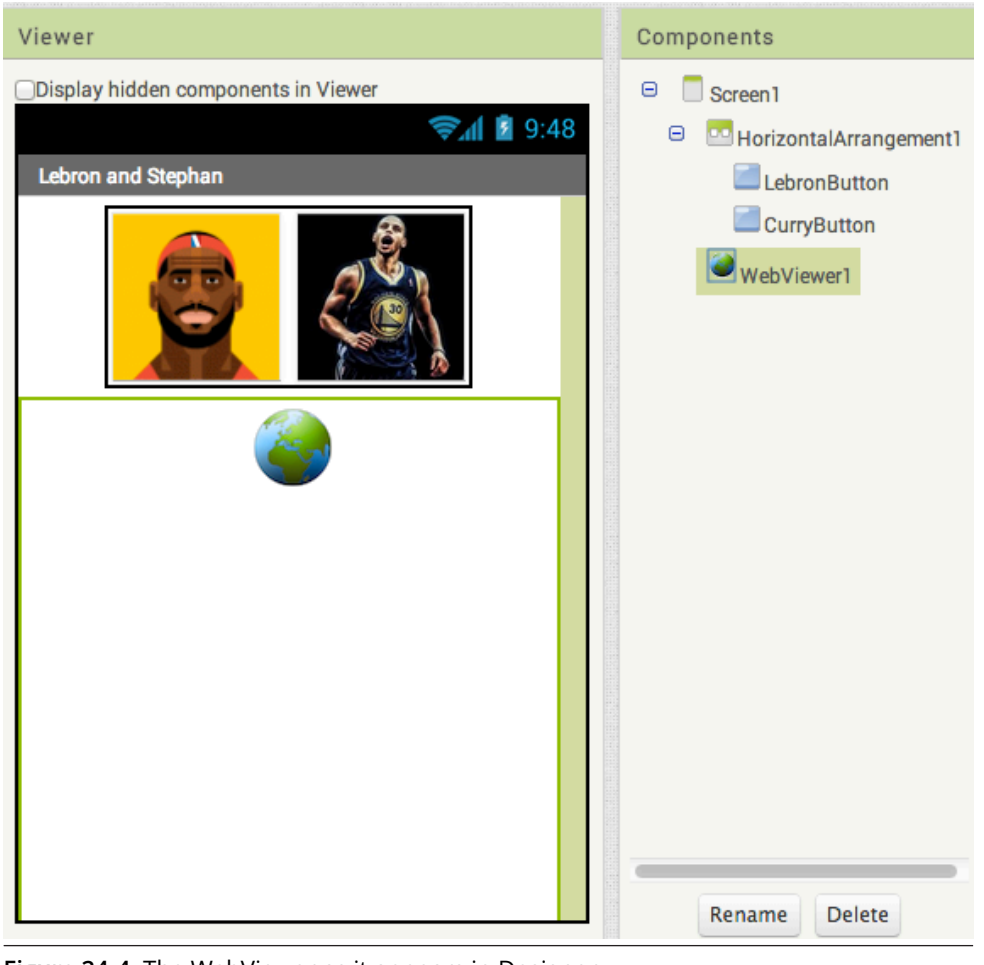

Figure 24-4. The WebViewer as it appears in Designer.

You can drag in a WebViewer from the User Interface drawer. You can then dynamically change the URL that appears, as in *Figure 24-4*, which depicts blocks from an app that shows the stats of NBA players Lebron James and Stephen Curry:

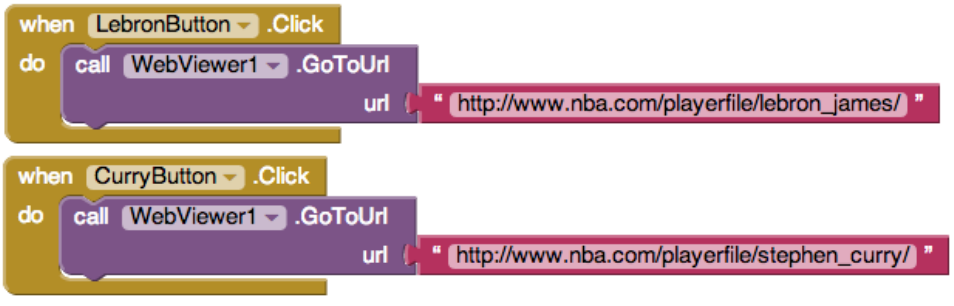

Figure 24-5. Blocks to show the web page for the chosen players

If the user taps the picture of Stephen Curry, the app would show his page from *nba.com* in the `WebViewer`, as in *Figure 24-5*.

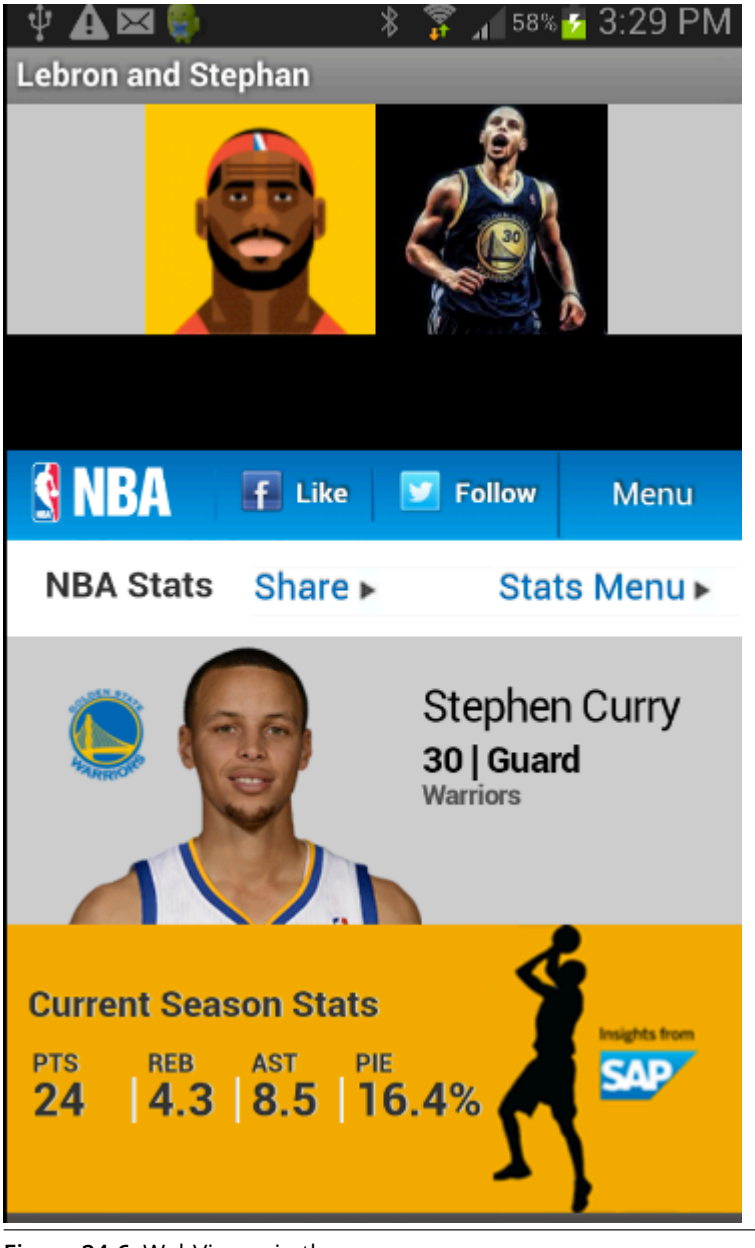

Figure 24-6. WebViewer in the app

The Web Component

Whereas `WebViewer` displays a web page, the `Web` component, a relatively new component in App Inventor, facilitates an app communicating with a web service via the standard Hypertext Transfer Protocol (HTTP). That protocol provides `Get`, `Put`, and

`Post` methods for bringing information into your app. The information arrives not as a displayable page, but as data that you can display or process as you like.

The component is fairly low level, and using it requires some programming expertise. You typically set the `Web.URL` property to specify which web service you will communicate with, and then you call one of the HTTP methods to request some action. It's complicated because you need to understand the API of the web service (the protocol for communication), and you need to understand how to process the information that the web service returns to your app. This processing is known as parsing, and it is an advanced programming technique.

In this chapter, you'll be introduced to the `Web` component through a relatively simple example that accesses financial stock price information from a public API made available by Yahoo Finance. The protocol for talking to this API is fairly simple, and the data returned is in a list of values separated by commas (*comma-separated values*, or CSV), so it serves as a nice introduction to API communication. Unfortunately, most APIs have complicated permission schemes and APIs, and they often return data in formats such as JavaScript Object Notation (JSON) or XML, which require some advanced code to parse.

STOCK MARKET SAMPLE

Figure 24-6 shows the blocks for an app that displays Google stock information when the app launches.

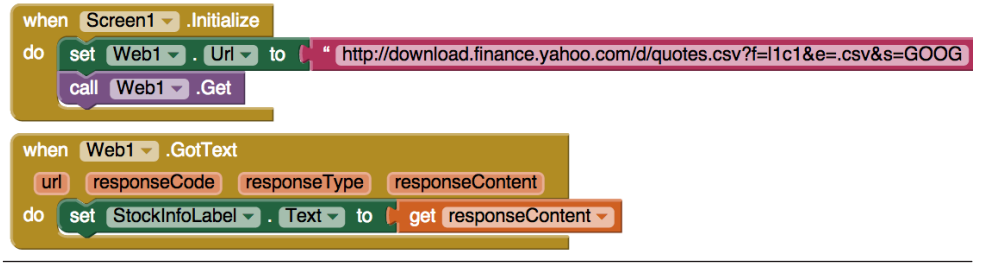

Figure 24-7. Accessing live stock information via the Web component

On `Screen.Initialize`, `Web1.Url` is set to the URL for communicating with Yahoo Finance. When `Web1.Get` is called, the request is made, but no data is returned immediately.

Instead, when Yahoo returns the requested data to your app, the `Web1.GotText` event is triggered, and this is where you can process the returned data. The event parameter `responseContent` holds the data. As just mentioned, the Yahoo Finance API returns data in CSV format. If you build this app and run it, you'll see that the current

Google stock price and the change in the price for the day are displayed in `StockInfoLabel`, separated by commas.

You can customize the Web.Url to get the information for a different company (or companies), and to get various types of stock market information. The Yahoo Finance API, at *https://code.google.com/p/yahoo-finance-managed/wiki/CSVAPI*, specifies how you can change the URL to customize your request, as well as the format of the data it returns.

TinyWebDB and TinyWebDB-Compliant APIs

The `Web` component provides a method for accessing APIs. If an API is fairly simple, such as Yahoo Finance, novice programmers can use the `Web` component to directly access it. But other APIs, like the Amazon API introduced in *Chapter 13*, are more complicated.

For complicated APIs, an experienced programmer can set up a TinyWebDB-compliant web service that can then be used by less experienced App Inventor programmers to access the API. When such a service is set up, other programmers can access the web service with the simple tag-value protocol inherent in the `TinyWebDB.GetValue` function. You send a particular tag as the parameter, and a list or text object is returned as the value. In this way, the App Inventor programmer is shielded from the difficult programming required to *parse* (understand and extract data) standard data formats such as XML or JSON.

"TinyWebDB-compliant" just means a web service that follows `TinyWebDB`'s expected protocol: it expects a specific request, and returns data that TinyWebDB can understand. The Amazon API web service used in *Chapter 13* is an example of such a web service, and can be used as a sample for programmers who would like to set up such a service (e.g., if you're a teacher and want to provide access to some API for your students).

In the past, building APIs was difficult because you not only needed to understand the programming and web protocols, but you also needed to set up a server to host your web service, and a database to store the data. Now, it's much easier because you can leverage cloud-computing tools such as Google's App Engine and Amazon's Elastic Compute Cloud to immediately deploy the service you create. These platforms will not only host your web service, but they'll also let hundreds of users access it before charging you a single dime. As you can imagine, these sites are a great boon to innovation.

The details of creating a TinyWebDB-compliant web service are beyond the scope of this book. But if you're interested, check out the documentation and samples at *http://appinventorapi.com/*.

Summary

Most websites and many mobile apps are not standalone entities; to do their jobs, they rely on the interoperability of other sites. With App Inventor, you can build games, quizzes, and other standalone apps, but soon enough, you'll encounter issues related to web access. Can I write an app that tells me when the next bus will arrive at my usual stop? Can I write an app that texts a special subset of my Facebook friends? Can I write an app that sends tweets? App Inventor provides three components that can talk to the Web: the `WebViewer` for showing a live web page; the `Web` component, for accessing information from an API; and the `TinyWebDB` component to access data in a specially designed web API.

Accessing an API can be complicated; you need to know the protocol for requesting information, and you need to process (parse) the often complex data returned. But the reward for learning how to do this is great; your apps can interact with the world!