CHAPTER 13

Amazon at the Bookstore

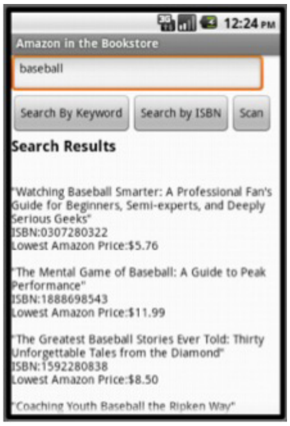
Suppose that you're browsing books at your favorite bookstore and want to know how much a book costs on Amazon.com. With the Amazon at the Bookstore app, you can scan a book or enter an ISBN, and the app will tell you the current lowest price of the book at Amazon.com. You can also search for books on a particular topic.

Amazon at the Bookstore demonstrates how you can use App Inventor to create apps that talk to web services (aka, application programming interfaces, or APIs). This app will get data from a web service created by one of the authors of this book. By the end of this chapter, you'll be able to create your own custom app for talking to Amazon. The application has a simple user interface with which the user can enter keywords or a book's ISBN (international standard book number—a 10- or 13-digit code that uniquely identifies a book) and then lists the title, ISBN, and lowest price for a new copy at Amazon. It also uses the `BarcodeScanner` component so that the user can scan a book to trigger a search instead of entering text (technically, the scanner just inputs the book's ISBN for you).

What You'll Learn

In this app (shown in Figure 13-1), you'll learn:

- How to use a barcode scanner within an app.
- How to access a web information source (Amazon's API) through the `TinyWebDB` component.
- How to process complex data returned from that web information source. In particular, you'll learn how to process a list of books in which each book is itself a list of three items (title, price, and ISBN).



Search Results

"Watching Baseball Smarter: A Professional Fan's Guide for Beginners, Semi-experts, and Deeply Serious Geeks"
ISBN:0307280322
Lowest Amazon Price:$5.76

"The Mental Game of Baseball: A Guide to Peak Performance"
ISBN:1888698543
Lowest Amazon Price:$11.99

"The Greatest Baseball Stories Ever Told: Thirty Unforgettable Tales from the Diamond"
ISBN:1592280838
Lowest Amazon Price:$8.50

"Coaching Youth Baseball the Ripken Way"

Figure 13-1. Amazon at the Bookstore running in the emulator

You'll also be introduced to source code that you can use to create your own web service API with the Python programming language and Google's App Engine.

What is an API?

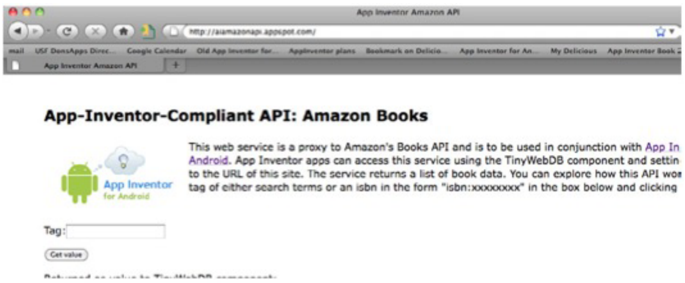
Before you start designing your components and programming the app, let's take a closer look at what an application programmer interface (API) is and how it works. An API is like a website, but instead of communicating with humans, it communicates with other computer programs. APIs are often called "server" programs because they typically serve information to "client" programs that actually interface with humans—such as an App Inventor app. If you've ever used a Facebook app on your phone, you're using a client program that communicates with the Facebook API server app.

In this chapter, you'll create an Android client app that communicates with an Amazon API. Your app will request book and ISBN information from the Amazon API, and the API will return up-to-date listings to your app. The app will then present the book data to the user.

The Amazon API you'll use is specially configured for use with App Inventor. We won't get into the gory details here, but it's useful to know that as a result of this configuration, you can use the `TinyWebDB` component to communicate with Amazon. The good news is you already know how to do that! You'll call `TinyWebDB.GetValue` to request information and then process the information returned in the `TinyWebDB.GotValue` event handler, just as you do when you use a web database. (You can go back to the MakeQuiz and TakeQuiz apps in Chapter 10 to refresh your memory, if needed.)

Before creating the app, you'll need to understand the Amazon API's protocol, which specifies the format for your request and the format of the data returned. Just as different human cultures have different protocols (when you meet someone, do you shake hands, bow, or nod your head?), computers talking to one another have protocols, as well. The Amazon API you'll be using here provides a web interface for exploring how the API works before you start using it. Although the API is designed to talk to other computers, this web interface makes it possible for you to see just how that communication will happen. Following these steps, you can try out what particular `GetValue` calls will return via the website, and know that the API interface will behave exactly the same when you ask it for data via the `TinyWebDB` component in App Inventor. Let's get underway:

1. Open a browser and go to *http://aiamazonapi.appspot.com/*. You'll see the website shown in Figure 13-2.



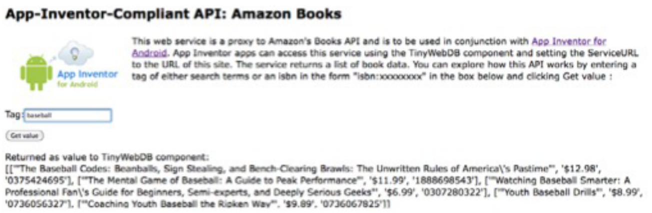
App-Inventor-Compliant API: Amazon Books

This web service is a proxy to Amazon's Books API and is to be used in conjunction with App In
Android. App Inventor apps can access this service using the TinyWebDB component and settin
to the URL of this site. The service returns a list of book data. You can explore how this API wo
tag of either search terms or an isbn in the form "isbn:xxxxxxxx" in the box below and clicking

Tag:

Figure 13-2. The web interface for the App Inventor Amazon API

2. On this web page, you can try the one function you can call with this API: getvalue. Enter a term (e.g., "baseball") in the Tag field and then click "Get value." The web page will display a listing of the top five books returned from Amazon, as shown in Figure 13-3.



App-Inventor-Compliant API: Amazon Books

This web service is a proxy to Amazon's Books API and is to be used in conjunction with App Inventor for Android. App Inventor apps can access this service using the TinyWebDB component and setting the ServiceURL to the URL of this site. The service returns a list of book data. You can explore how this API works by entering a tag of either search terms or an isbn in the form "isbn:xxxxxxxx" in the box below and clicking Get value :

Tag: baseball

```
Returned as value to TinyWebDB component:
[['"The Baseball Codes: Beanballs, Sign Stealing, and Bench-Clearing Brawls: The Unwritten Rules of America\'s Pastime"', '$12.98',
'0375424695'], ['"The Mental Game of Baseball: A Guide to Peak Performance"', '$11.99', '1888698543'], ['"Watching Baseball Smarter: A
Professional Fan\'s Guide for Beginners, Semi-experts, and Deeply Serious Geeks"', '$6.99', '0307280322'], ['"Youth Baseball Drills"', '$8.99',
'0736056327'], ['"Coaching Youth Baseball the Ripken Way"', '$9.89', '0736067825']]
```

Figure 13-3. Making a call to the Amazon API to search for books related to the tag (or keyword) "baseball"

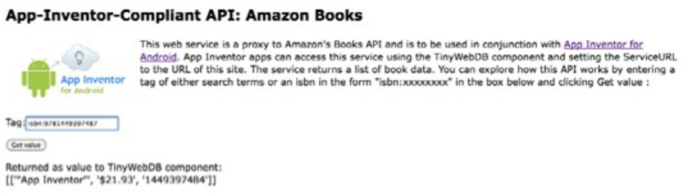
The value returned is a list of books, each one enclosed in brackets [like this] and providing the title, cost, and ISBN.

If you look closely, you'll see that each book is in fact represented as a sublist of another main list. The main list (about baseball) is enclosed in brackets, and each sublist (or book) is enclosed in its own set of brackets within the main brackets. So, the return value from this API is actually a list of lists, with each sublist providing the information for one book. Let's look at this a bit more closely. Each left bracket ([) in the data denotes the beginning of a list. The first left bracket of the result denotes the beginning of the outer list (the list of books). To its immediate right is the beginning of the first sublist, the first book, as demonstrated here:

["The Baseball Codes: Beanballs, Sign Stealing, and Bench-Clearing Brawls: The Unwritten Rules of America's Pastime," '$12.98,' '0375424695']

The sublist has three parts: a title, the lowest current price for the book at Amazon, and the book's ISBN. When you get this information into your App Inventor app, you'll be able to access each part by using `select list item`, with index 1 for the title, index 2 for the price, and index 3 for the ISBN. (To refresh your memory on working with an index and lists, revisit the MakeQuiz app in Chapter 10.)

3. Instead of searching by keyword, you can search for a book by entering an ISBN. To perform such a search, you enter a tag in the form “isbn:00000000000,” where the list of 0s represent an actual ISBN number (see Figure 13-4). The double brackets ([[) in the result [[‘“App Inventor,”’ ‘$21.93,’ ‘1449397484’]] denote that a list of lists is still returned, even though there is only one book. It might seem a bit strange now, but this will be important when we access the information for our app.



App-Inventor-Compliant API: Amazon Books

This web service is a proxy to Amazon's Books API and is to be used in conjunction with App Inventor for Android. App Inventor apps can access this service using the TinyWebDB component and setting the ServiceURL to the URL of this site. The service returns a list of book data. You can explore how this API works by entering a tag of either search terms or an isbn in the form "isbn:xxxxxxxx" in the box below and clicking Get value :

Tag: isbn:9781449397487

Returned as value to TinyWebDB component:
[['"App Inventor"', '$21.93', '1449397484']]

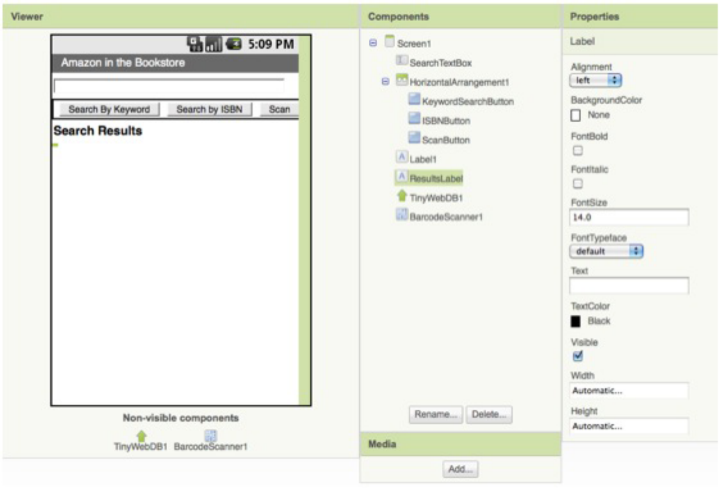
Figure 13-4. Querying the Amazon API by ISBN instead of keyword

Getting Started

Connect to the App Inventor website and start a new project. Then name it “AmazonBooks”, and set the screen’s title to “Amazon at the Bookstore”. Then, connect your device or emulator for live testing.

Designing the Components

The user interface for the Amazon book app is relatively simple: give it a `Textbox` for entering keywords or ISBNs, two buttons for starting the two types of searches (keyword or ISBN), and a third button for letting the user scan a book (we’ll get to that in a bit). Then, add a heading label and another label for listing the results that the Amazon API returns, and finally two non-visible components: `TinyWebDB` and a `BarcodeScanner`. Check your results against Figure 13-5.



Label

Alignment: left
BackgroundColor: None
FontBold: ☐
FontItalic: ☐
FontSize: 14.0
FontTypeface: default
Text:
TextColor: Black
Visible: ✓
Width: Automatic...
Height: Automatic...

Figure 13-5. The Amazon at the Bookstore user interface shown in the Designer

Table 13-1 lists all the components you'll need to build the UI shown in Figure 13-5.

Table 13-1. Component list for the Amazon at the Bookstore app

Component type	Palette group	What you'll name it	Purpose
`Textbox`	User Interface	`SearchTextBox`	The user enters keywords or ISBN here.
`HorizontalArrangement`	Layout	`HorizontalArrangement1`	Arrange the buttons in a line.
`Button`	User Interface	`KeywordSearchButton`	Click to search by keyword.
`Button`	User Interface	`ISBNButton`	Click to search by ISBN.
`Button`	User Interface	`ScanButton`	Click to scan an ISBN from a book.
`Label`	User Interface	`Label1`	The header "Search Results."
`Label`	User Interface	`ResultsLabel`	Where you'll display the results.
`TinyWebDB`	Storage	`TinyWebDB1`	Talk to Amazon.com.
`BarcodeScanner`	Sensors	`BarcodeScanner1`	Scan barcodes.

Set the properties of the components in the following way:

1. Set the `Hint` of the `SearchTextBox` to "Enter keywords or ISBN".
2. Set the properties of the buttons and labels so that they appear as shown in Figure 13-5.

3. Set the `ServiceURL` property of the `TinyWebDB` component to *http://aiamazonapi.appspot.com/*.

Programming the App's Behavior

For this app, you'll specify the following behaviors in the Blocks Editor:

Searching by keyword
: The user enters some terms and clicks the KeywordSearchButton to invoke an Amazon search. You'll call `TinyWebDB1.GetValue` to make it happen.

Searching by ISBN
: The user enters an ISBN and clicks the `ISBNButton`. You'll package the prefix "isbn:" with the number entered and run the Amazon search.

Barcode scanning
: The user clicks a button and the scanner is launched. When the user scans an ISBN from a book, your app will launch a search by ISBN.

Processing the list of books
: At first, your app will display the data returned from Amazon in a rudimentary way. Later, you'll modify the blocks so that the app extracts the title, price, and ISBN from each book returned and displays them in an organized way.

Searching by Keyword

When the user clicks the `KeywordSearchButton`, you want to grab the text from the `SearchTextBox` and send it as the tag in your request to the Amazon API. You'll use the `TinyWebDB.GetValue` block to request the Amazon search. When the results come back from Amazon, the `TinyWebDB.GotValue` event handler will be triggered. For now, just display the result that is returned directly into the `ResultsLabel`, as shown in Figure 13-6. Later, after you see that the data is indeed being retrieved, you can display the data in a more sophisticated fashion.

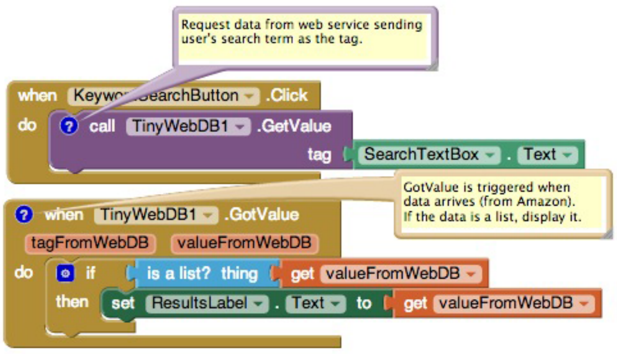
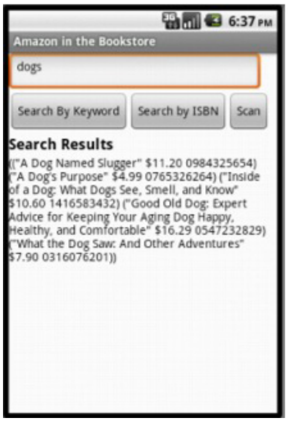

Request data from web service sending user's search term as the tag.
when KeywordSearchButton .Click
do call TinyWebDB1 .GetValue
tag SearchTextBox . Text
GotValue is triggered when data arrives (from Amazon). If the data is a list, display it.
when TinyWebDB1 .GotValue
tagFromWebDB valueFromWebDB
do if is a list? thing get valueFromWebDB
then set ResultsLabel . Text to get valueFromWebDB

Figure 13-6. Send the search request to the API and put results in the ResultsLabel

How the blocks work

When the user clicks the KeywordSearchButton, the TinyWebDB1.GetValue request is made. The tag sent with the request is the information the user entered in the Search TextBox. If you completed the MakeQuiz app (Chapter 10), you know that TinyWebDB1.GetValue requests are not answered immediately. Instead, when the data arrives from the API, TinyWebDB1.GotValue is triggered. In GotValue, the blocks check the value returned to see if it's a list (it won't be if the Amazon API is offline or there is no data for the keywords). If it is a list, the data is placed into the ResultsLabel.

Test your app *Enter a term in the search box and click Search By Keyword. You should get a listing similar to what is shown in Figure 13-7. (It's not terribly nice-looking, but we'll deal with that shortly.)*



Search Results

(("A Dog Named Slugger" $11.20 0984325654) ("A Dog's Purpose" $4.99 0765326264) ("Inside of a Dog: What Dogs See, Smell, and Know" $10.60 1416583432) ("Good Old Dog: Expert Advice for Keeping Your Aging Dog Happy, Healthy, and Comfortable" $16.29 0547232829) ("What the Dog Saw: And Other Adventures" $7.90 0316076201))

Figure 13-7. Keyword search result for "dogs"

Searching by ISBN

The code for searching by ISBN is similar, but in this case the Amazon API expects the tag to be in the form "isbn:xxxxxxxxxxxxx" (this is the protocol the API expects for searching by ISBN). You don't want to force the user to know this protocol; the user should just be able to enter the ISBN in the text box and click Search by ISBN, and the app should add the "isbn:" prefix behind the scenes with make text. Figure 13-8 shows the blocks to do that.

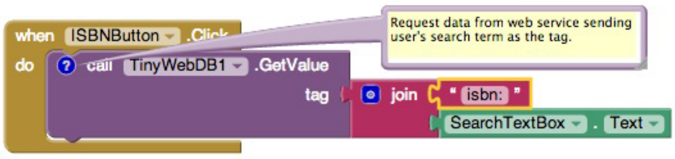

when ISBNButton .Click
do call TinyWebDB1 .GetValue
tag join "isbn:"
SearchTextBox . Text
Request data from web service sending user's search term as the tag.

Figure 13-8. The app prefixes "isbn:" to the search so it will look up a particular book

How the blocks work

The `join` block concatenates the "isbn:" prefix with the information the user has input in the `SearchTextBox` and sends the result as the tag to `TinyWebDB1.GetValue`.

Just as with keyword search, the API sends back a list result for an ISBN search—in this case, a list of just the one item whose ISBN matches the user's input exactly. Because the `TinyWebDB.GotValue` event handler is already set up to process a list of books (even a list with only one item), you won't have to change your event handler to make this work.

Test your app *Enter an ISBN (e.g., 9781449397487) in the `Search TextBox` and click the `ISBNButton`. Does the book information appear?*

Don't Leave Your Users Hanging

When you call a web service (API) with `TinyWebDB1.GetValue`, there can be a delay before the data arrives and `TinyWebDB1.GotValue` is triggered. It is generally a good idea to let users know the request is being processed to reassure them that the app hasn't hung. For this app, you can place a message in the `ResultsLabel` each time you make the call to `TinyWebDB1.GetValue`, as shown in Figure 13-9.

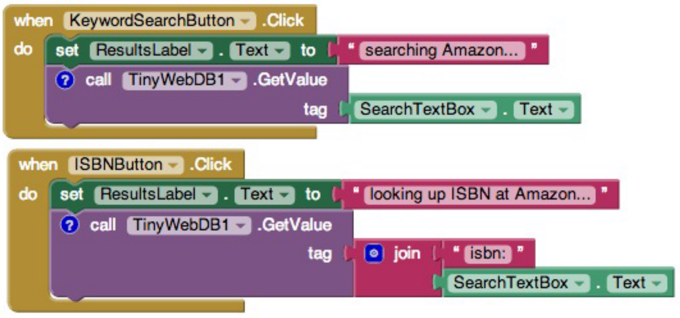

when KeywordSearchButton .Click
do set ResultsLabel . Text to "searching Amazon..."
call TinyWebDB1 .GetValue
tag SearchTextBox . Text
when ISBNButton .Click
do set ResultsLabel . Text to "looking up ISBN at Amazon..."
call TinyWebDB1 .GetValue
tag join "isbn:"
SearchTextBox . Text

Figure 13-9. Adding a message to let the user know what is happening

How the blocks work

For both the keyword and ISBN searches, a "Searching Amazon..." message is placed in `ResultsLabel` when the data is requested. Note that when `GotValue` is triggered, this message is overwritten with the actual results from Amazon.

Scanning a Book

Let's face it: typing on a cell phone isn't always the easiest thing, and you tend to make a mistake here and there. It would certainly be easier (and result in fewer mistakes) if a user could just launch your app and scan the barcode of the book. This is another great built-in Android phone feature that you can tap into easily with App Inventor.

The function `BarcodeScanner.DoScan` starts up the scanner. You'll want to call this when the `ScanButton` is clicked. The event handler `BarcodeScanner.AfterScan` is triggered as soon as something has been scanned. It has one argument, `result`, which contains the information that was scanned. In this case, you want to initiate an ISBN search using that result, as shown in Figure 13-10.

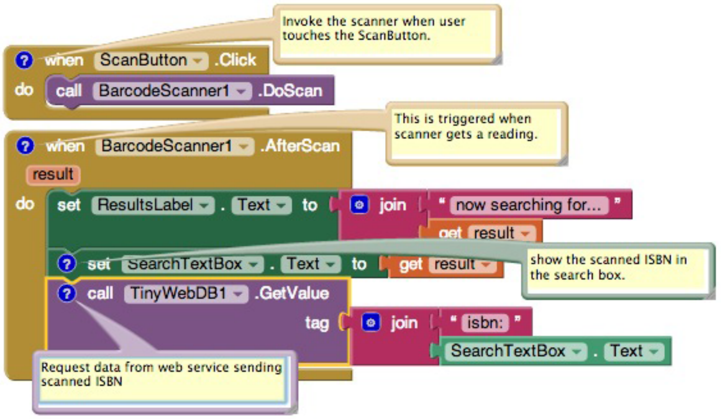

Invoke the scanner when user touches the ScanButton.
when ScanButton .Click
do call BarcodeScanner1 .DoScan
This is triggered when scanner gets a reading.
when BarcodeScanner1 .AfterScan
result
do set ResultsLabel . Text to join "now searching for..." get result
set SearchTextBox . Text to get result
show the scanned ISBN in the search box.
call TinyWebDB1 .GetValue
tag join "isbn:" SearchTextBox . Text
Request data from web service sending scanned ISBN

Figure 13-10. Blocks for initiating an ISBN search after a user scans

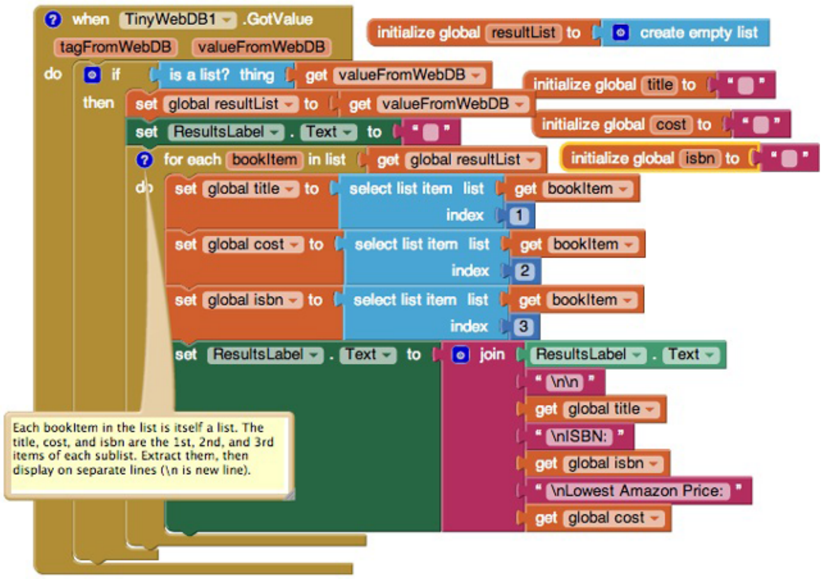
How the blocks work

When the user clicks the `ScanButton`, `DoScan` launches the scanner. When something has been scanned, `AfterScan` is triggered. The argument result holds the result of the scan—in this case, a book's ISBN. The user is notified that a request has been made, the result (the scanned ISBN number) is placed in the `SearchTextBox`, and `TinyWebDB1.GetValue` is called to initiate the search. Again, the `TinyWebDB1.GotValue` event handler will process the book information returned.

Test your app *Click the ScanButton and scan the barcode of a book. Does the app display the book information?*

Improving the Display

A client app like the one you're creating can do whatever it wants with the data it receives—you could compare the price information with that of other online stores, or use the title information to search for similar books from another library. Almost always, you'll want to get the API information loaded into variables that you can then process further. In the `TinyWebDB.GotValue` event handler you have so far, you just place all the information returned from Amazon into the `ResultsLabel`. Instead, let's process the data by 1) putting the title, price, and ISBN of each book returned into separate variables, and 2) displaying those items in an orderly fashion. If you've completed some of the earlier chapters, you're probably getting the hang of defining variables and using them in your display, so try building out the variables you think you'll need and the blocks to display each search result on its own separate line. Then, compare what you've done with Figure 13-11.



```
when TinyWebDB1.GotValue
  tagFromWebDB  valueFromWebDB
do  if  is a list? thing  get valueFromWebDB
    then  set global resultList to  get valueFromWebDB
          set ResultsLabel.Text to " "
          for each bookItem in list  get global resultList
          do  set global title to  select list item  list  get bookItem
                                                     index 1
              set global cost to  select list item  list  get bookItem
                                                    index 2
              set global isbn to  select list item  list  get bookItem
                                                    index 3
              set ResultsLabel.Text to  join  ResultsLabel.Text
                                              "\n\n"
                                              get global title
                                              "\nISBN: "
                                              get global isbn
                                              "\nLowest Amazon Price: "
                                              get global cost

initialize global resultList to  create empty list
initialize global title to " "
initialize global cost to " "
initialize global isbn to " "
```

Each bookItem in the list is itself a list. The title, cost, and isbn are the 1st, 2nd, and 3rd items of each sublist. Extract them, then display on separate lines (\n is new line).

Figure 13-11. Extracting the title, cost, and ISBN of each book, and then displaying them on separate lines

How the blocks work

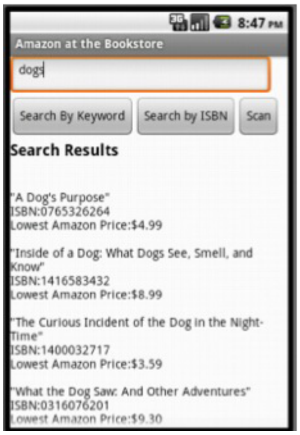
Four variables—`resultList`, `title`, `cost`, and `isbn`—are defined to hold each piece of data as it is returned from the API. The result from the API, `valueFromWebDB`, is placed into the variable `resultList`. This app could have processed the argument `valueFromWebDB` directly, but in general, you'll put it in a variable in case you want to process the data outside the event handler. (Event arguments like `valueFromWebDB` hold their value only within the event handler.)

A `for each` loop is used to iterate through each item of the result. Recall that the data returned from Amazon is a list of lists, with each sublist representing the information for a book. So, the placeholder of the `for each` is renamed `bookitem`, and it holds the current book information (a list) on each iteration.

Now we have to deal with the fact that the variable `bookitem` is a list—the first item is the title; the second, the price; and the third, the ISBN. Thus, we use `select list item` blocks to extract these items and place them into their respective variables (`title`, `price`, and `isbn`).

After the data has been organized into variables, you can process it however you'd like. This app just uses the variables as part of a `join` block that displays the title, price, and ISBN on separate lines.

Test your app *Try another search and check out how the book information is displayed. It should look similar to Figure 13-12.*



Search Results

"A Dog's Purpose"
ISBN:0765326264
Lowest Amazon Price:$4.99

"Inside of a Dog: What Dogs See, Smell, and Know"
ISBN:1416583432
Lowest Amazon Price:$8.99

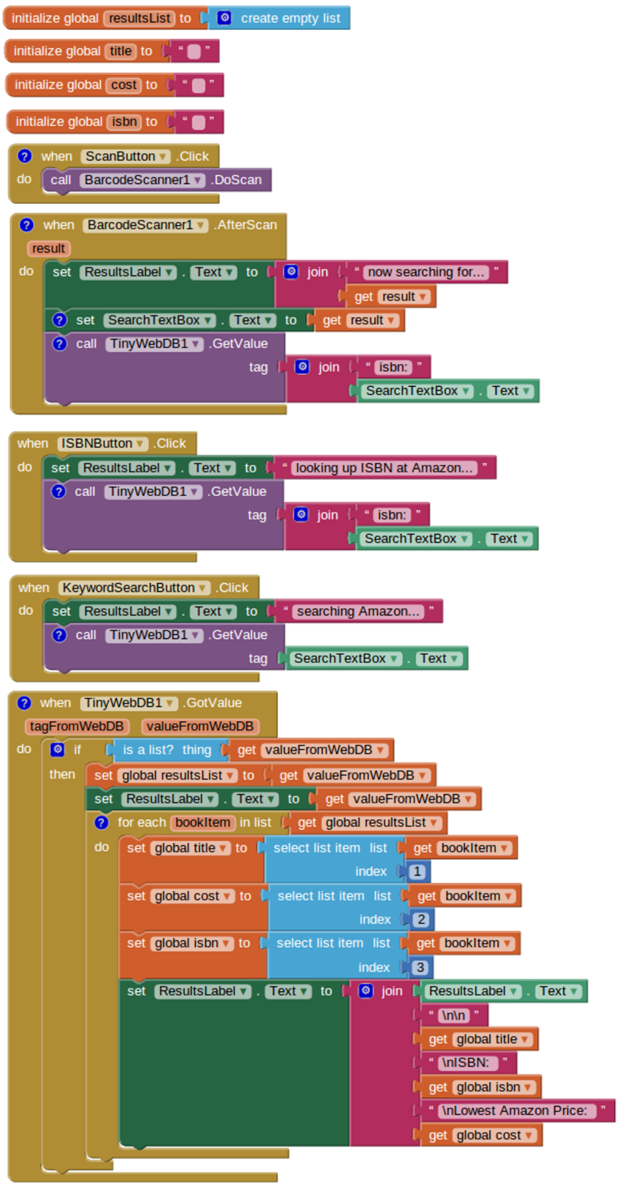
"The Curious Incident of the Dog in the Night-Time"
ISBN:1400032717
Lowest Amazon Price:$3.59

"What the Dog Saw: And Other Adventures"
ISBN:0316076201
Lowest Amazon Price:$9.30

Figure 13-12. The search listing displayed in a more sophisticated fashion

The Complete App: Amazon at the Bookstore

Figure 13-13 shows the final block configuration for Amazon at the Bookstore.



```
initialize global resultsList to create empty list
initialize global title to " "
initialize global cost to " "
initialize global isbn to " "

when ScanButton .Click
do  call BarcodeScanner1 .DoScan

when BarcodeScanner1 .AfterScan
  result
do  set ResultsLabel . Text to  join  " now searching for... "
                                      get result
    set SearchTextBox . Text to get result
    call TinyWebDB1 .GetValue
                         tag  join  " isbn: "
                                    SearchTextBox . Text

when ISBNButton .Click
do  set ResultsLabel . Text to " looking up ISBN at Amazon... "
    call TinyWebDB1 .GetValue
                         tag  join  " isbn: "
                                    SearchTextBox . Text

when KeywordSearchButton .Click
do  set ResultsLabel . Text to " searching Amazon... "
    call TinyWebDB1 .GetValue
                         tag  SearchTextBox . Text

when TinyWebDB1 .GotValue
  tagFromWebDB  valueFromWebDB
do  if  is a list? thing  get valueFromWebDB
    then  set global resultsList to get valueFromWebDB
          set ResultsLabel . Text to get valueFromWebDB
          for each bookItem in list  get global resultsList
          do  set global title to  select list item  list  get bookItem
                                                     index  1
              set global cost to  select list item  list  get bookItem
                                                    index  2
              set global isbn to  select list item  list  get bookItem
                                                    index  3
              set ResultsLabel . Text to  join  ResultsLabel . Text
                                                " \n\n "
                                                get global title
                                                " \nISBN: "
                                                get global isbn
                                                " \nLowest Amazon Price: "
                                                get global cost
```

Figure 13-13. The complete Amazon at the Bookstore app

Customizing the API

The API you connected to, *http://aiamazonapi.appspot.com*, was created with the programming language Python and Google's App Engine. App Engine lets you create and deploy websites and services (APIs) that live on Google's servers. You only pay for App Engine if your site or API becomes popular and attracts lots of hits.

The API service used in this tutorial provides only partial access to the full Amazon API and returns a maximum of five books for any search. If you'd like to provide more flexibility—for example, have it search for items other than books—you can download the source code for the service from *http://appinventorapi.com/amazon/* and customize it. Such customization does require knowledge of Python programming, so beware! But, if you've been completing the App Inventor apps in this book, you might just be ready for the challenge. To get started learning Python, check out the interactive version of the book *How to Think Like a Computer Scientist: Learning with Python* and then check out the section on App Inventor API building in Chapter 24 of this book.

Variations

After you get the app working, you might want to explore some of the following variations:

- As is, the app hangs if the search doesn't return any books (for instance, when the user enters an invalid ISBN). Modify the blocks so that the app reports when there are no results.
- Modify the app so that it only displays books under $10.
- Modify the app so that after you scan a book, its lowest Amazon price is spoken aloud (use the `TextToSpeech` component discussed in the No Text While Driving app in Chapter 4).
- Download the *http://aiamazonapi.appspot.com* API code and modify it so that it returns more information. For example, you might have it return the Amazon URL of each book, display the URL along with each listed book, and let the user click the URL to open that page. As mentioned earlier, modifying the API requires Python programming and some knowledge of Google's App Engine. For more information, check out Chapter 24.

Summary

Here are some of the concepts we covered with this app:

- You can access the Web from an app by using `TinyWebDB` and specially constructed APIs. You set the `ServiceURL` of the `TinyWebDB` component to the API URL and then call `TinyWebDB.GetValue` to request the information. The data isn't immediately returned but can instead be accessed within the `TinyWebDB.GotValue` event handler.

- The `BarcodeScanner.DoScan` function launches the scan. When the user scans a barcode, the `BarcodeScanner.AfterScan` event is triggered and the scanned data is placed in the argument `result`.
- In App Inventor, complex data is represented with lists and lists of lists. If you know the format of the data returned from an API, you can use `for each` and `select list item` to extract the separate pieces of information into variables, and then perform whatever processing or set up the display however you'd like using those variables.