Make This

Chapter 13 – Using Web Services

Accessing web sites from within apps is very useful. Web sites that provide web services provide an **application programmer interface (API)** to facilitate using a web service within your program. In this tutorial, you construct an app that accesses a web service to return information.

This tutorial teaches the following skills:

- **Using the Web Component**
- **Working with web services**
- **Using if then else logic**

Note: Before attempting this exercise, complete the Chapter 2 and 3 exercises to familiarize yourself with the App Inventor interface, getting your Android device connected, and starting a new project.

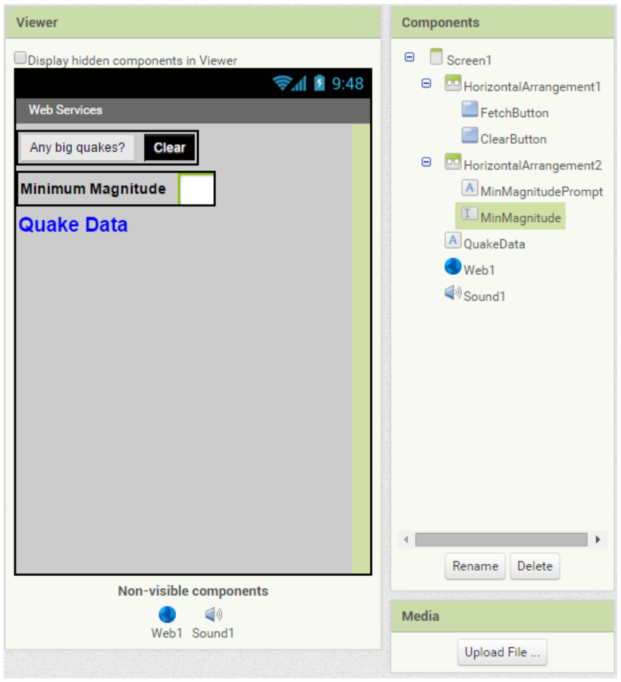
Building the Web Services App

1. Navigate to http://appinventor.mit.edu/explore/. If necessary, sign in with your Google Account.
2. Start a new project named *WebServicesApp*. Change the **Title** of **Screen1** to *Web Services*. Set the **BackgroundColor** to *Light Gray*.
3. Connect App Inventor to your Android device.

Part 1 - Constructing the Interface

The interface allows the user to enter a minimum value for the magnitude of earthquakes. Then the user can press the "Any Big Quakes?" button to pull up the most current data on quakes worldwide that are larger than the magnitude entered by the user. Pressing the Clear button will reset for another search.

Your layout for your interface should look like this:



Build the interface shown above by dragging out the components shown in the following table:

Component	Palette Group	Component Name	Function
HorizontalArrangement	Layout	HorizontalArrangement1	Area to line up two buttons
Button	User Interface	FetchButton	Button to launch communication with web service
Button	User Interface	ClearButton	Button to reset
HorizontalArrangement	Layout	HorizontalArrangement2	Area to line up the label and input text field for magnitude
Label	User Interface	MinMagnitudePrompt	Label for text entry box
TextBox	User Interface	MinMagnitude	Text box to display minimum magnitude value entered
Label	User Interface	QuakeData	Label to identify output area for display of results
Web	Connectivity	Web1	Non-visible component that provides

			communication to a web service
Sound	Media	Sound1	Non-visible component that supports vibration

Set the properties of each component in the following ways:

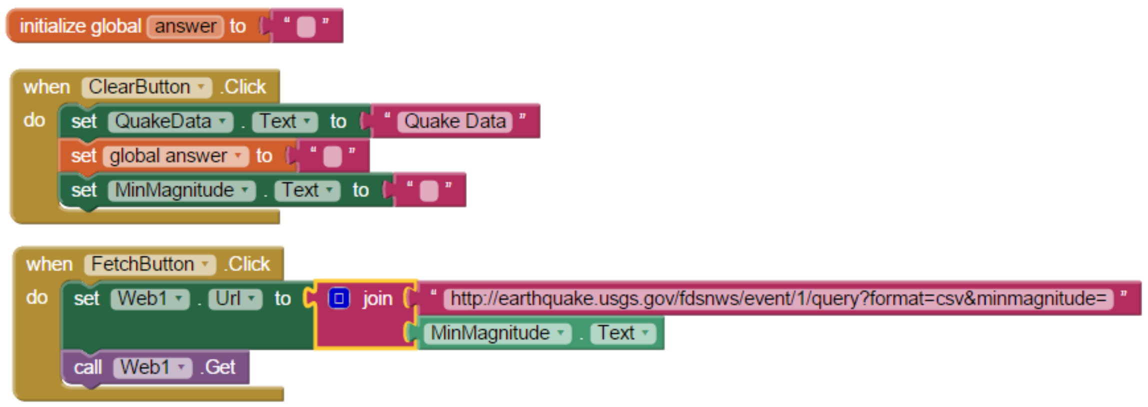
- Change the **Text** for **FetchButton** to *Any big quakes?*
- Change the **Text** for **ClearButton** to Clear. Change the **BackgroundColor** of the button to *Black* and change the **TextColor** to *White*.
- Change the **Text** of **MinMagnitudePrompt** to *Minimum Magnitude.* Click the **FontBold** check box to select it. Change the **FontSize** to 16.
- Change the **Hint** of **MinMagnitude** to *Enter minimum magnitude*. Change the **Width** to 30 pixels.
- Change the **Text** of **QuakeData** to *Quake Data*. Change the **TextColor** to *Blue*. Change the FontSize to 22. Click the **FontBold** check box to select it.

Part 2 – Program Functionality of the App

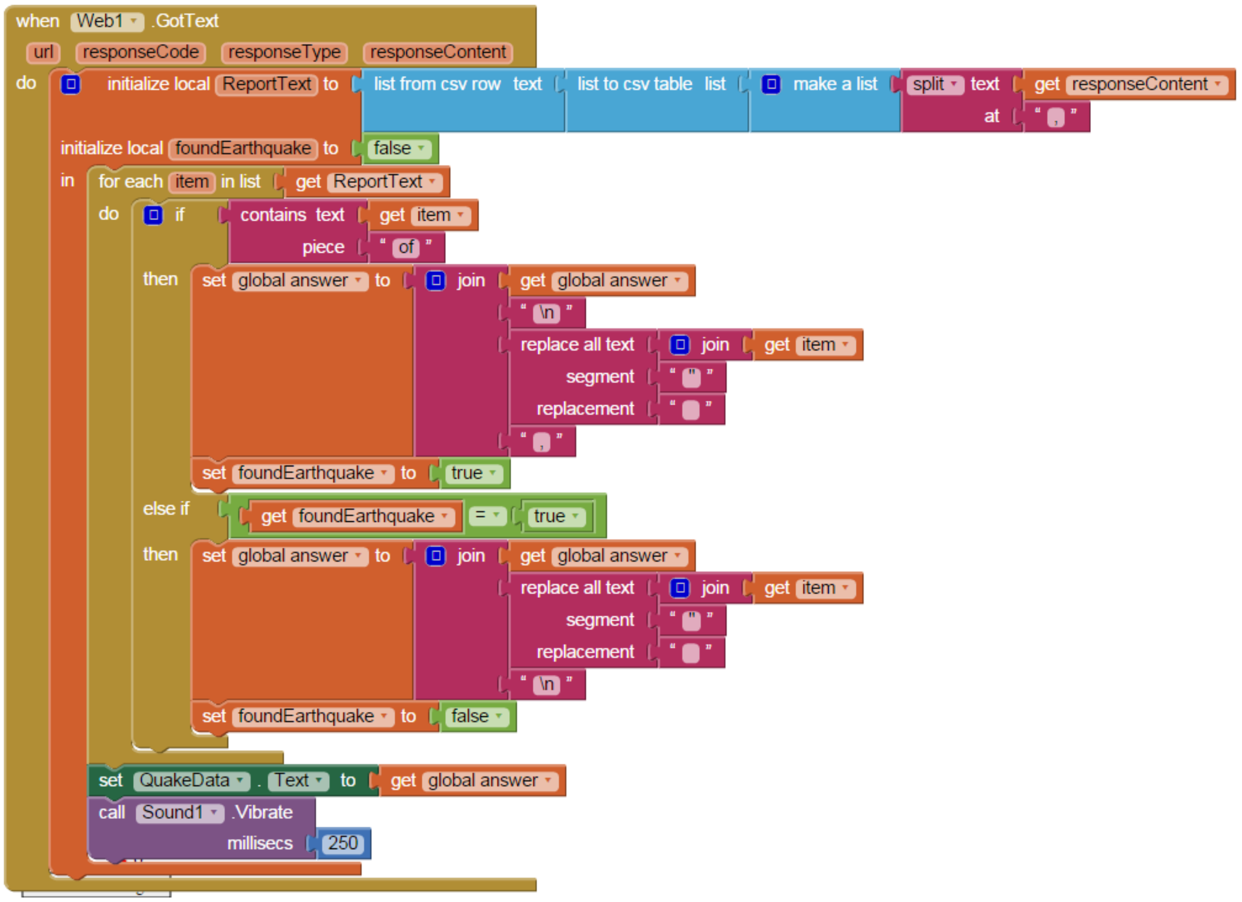
1. Change to **Blocks** view.

To gather the latest earthquake data we will be using a web service provided by the US Geological Service. To access an API, you need to have an understanding of how the API functions. Web sites such as www.programmableweb.com can provide insight, however, often the best way is just to Google the service name followed by "API". Documentation for the USGS Earthquake Hazards API can be found here. We will walk through the special fields we need from this site's information to gather the data we want to display.

The completed blocks for this app are as seen here:

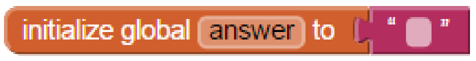
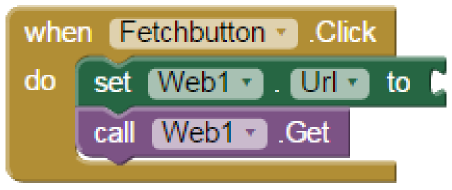
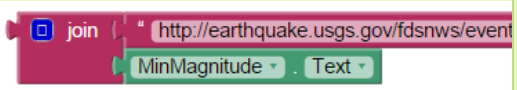


```
initialize global answer to " "

when ClearButton .Click
do  set QuakeData . Text to " Quake Data "
    set global answer to " "
    set MinMagnitude . Text to " "

when FetchButton .Click
do  set Web1 . Url to join " http://earthquake.usgs.gov/fdsnws/event/1/query?format=csv&minmagnitude= "
                          MinMagnitude . Text
    call Web1 .Get
```



```
when Web1 .GotText
url  responseCode  responseType  responseContent
do  initialize local ReportText to  list from csv row text  list to csv table list  make a list  split text  get responseContent
                                                                                                      at  ","
    initialize local foundEarthquake to  false
    in  for each item in list  get ReportText
        do  if  contains text  get item
                       piece  "of"
            then  set global answer to  join  get global answer
                                              "\n"
                                              replace all text  join  get item
                                                       segment  """
                                                   replacement  ""
                                              ","
                  set foundEarthquake to  true
            else if  get foundEarthquake = true
            then  set global answer to  join  get global answer
                                              replace all text  join  get item
                                                       segment  """
                                                   replacement  ""
                                              "\n"
                  set foundEarthquake to  false
        set QuakeData . Text to  get global answer
        call Sound1 .Vibrate
                     millisecs  250
```

2. Build the following blocks (as indicated in the table below) and arrange them as shown above.

Block	**Drawer**	**Function**
 initialize global answer to " "	Variables	Initialize the answer text string to just the empty string
 when Fetchbutton .Click do set Web1 . Url to call Web1 .Get	FetchButton event And Web1	Handle the user clicking on the FetchButton. First get ready to call the webservice by setting the URL field. Then call that program.
 join " http://earthquake.usgs.gov/fdsnws/event MinMagnitude . Text	Text drawer And MinMagnitude text box property	Build the command being sent to the geology service's computer. The URL of that computer is : http://earthquake.usgs.gov/fdsnws /event/1/query?format=csv&min magnitude=

		That is joined with the number the user entered in the text box.

The main work of the app is already complete! Using the Web component, we can talk to a program running on another computer, anywhere in the Internet. We can ask that program a specific question. Here, the computer we are talking with is living at:

http://earthquake.usgs.gov/fdsnws/event/1/query?

The question mark tells the program we are about to ask it a question. We know from the API that if we tell it **format=csv** it will send us back a neatly organize text file of information where all the data is separated by commas (CSV stands for “comma separated values”). From the API we know that if we tell the program **minmagnitude= 7** it will only send us back the earthquakes that were bigger than a 7. We build up that command by pasting together the part that stays the same each time with the part the user will enter, the MinMagnitude.text value.

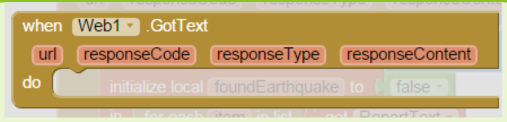
So let’s just try out that command in a regular browser. It will download a file named “query” that you can open in Notepad or Word. Here is some of the query file from today:

```
time,latitude,longitude,depth,mag,magType,nst,gap,dmin,rms,net,id,updated
,place,type
2015-05-17T16:39:10.190Z,34.2158333,-
117.4476667,11.575,1.04,ml,32,43,0.04451,0.17,ci,ci37163863,2015-05-
17T16:43:08.395Z,"4km W of Devore, California",earthquake
2015-05-17T16:36:10.120Z,38.7946663,-
122.7638321,3.73,0.56,md,8,115,0.02795,0.26,nc,nc72450706,2015-05-
17T16:37:43.380Z,"2km NNW of The Geysers, California",earthquake
2015-05-17T16:29:09.000Z,61.6231,-
149.8645,33.3,1.4,ml,,,,0.71,ak,ak11599648,2015-05-17T16:44:48.999Z,"2km
WSW of Houston, Alaska",earthquake
```

The first line tells us what kind of information is in each column. If you don’t understand a column heading, you can get more details from the US Geological Services API. We are just interested in reporting back the place information, which is in the next to last column.

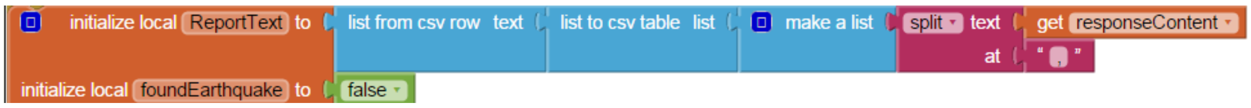
The rest of the processing blocks in this app work to read through the information in the query file and grab out the place information. We will paste that into a big string of text named “answer” and then display it.

We begin by waiting until we get a response from the other computer. We will know because the GotText event for our Web1 component will trigger:

Block	Drawer	Function
 when Web1 .GotText url responseCode responseType responseContent do	Web1	Whenever the Geology Service's program finishes creating the query files and sends it to our app, the GotText event will trigger.

Next we will initialize two local variables, **ReportText** and **foundEarthquake**. Build this statement, pulling from:

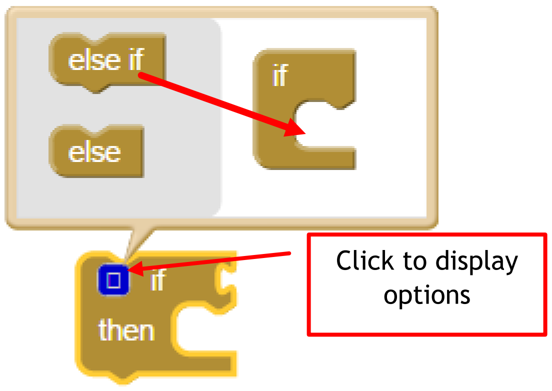
- the Variables drawer for orange tiles,
- the List drawer for **blue** tiles,
- the Logic drawer for green tiles
- and the Text drawer for red tiles



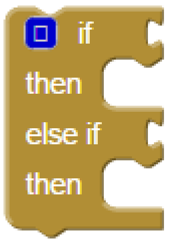
The first line turns the information in the query file into one big list named ReportText that we can process. The second statement sets up a true/false variable named foundEarthquake.

Next build the following blocks by pulling:

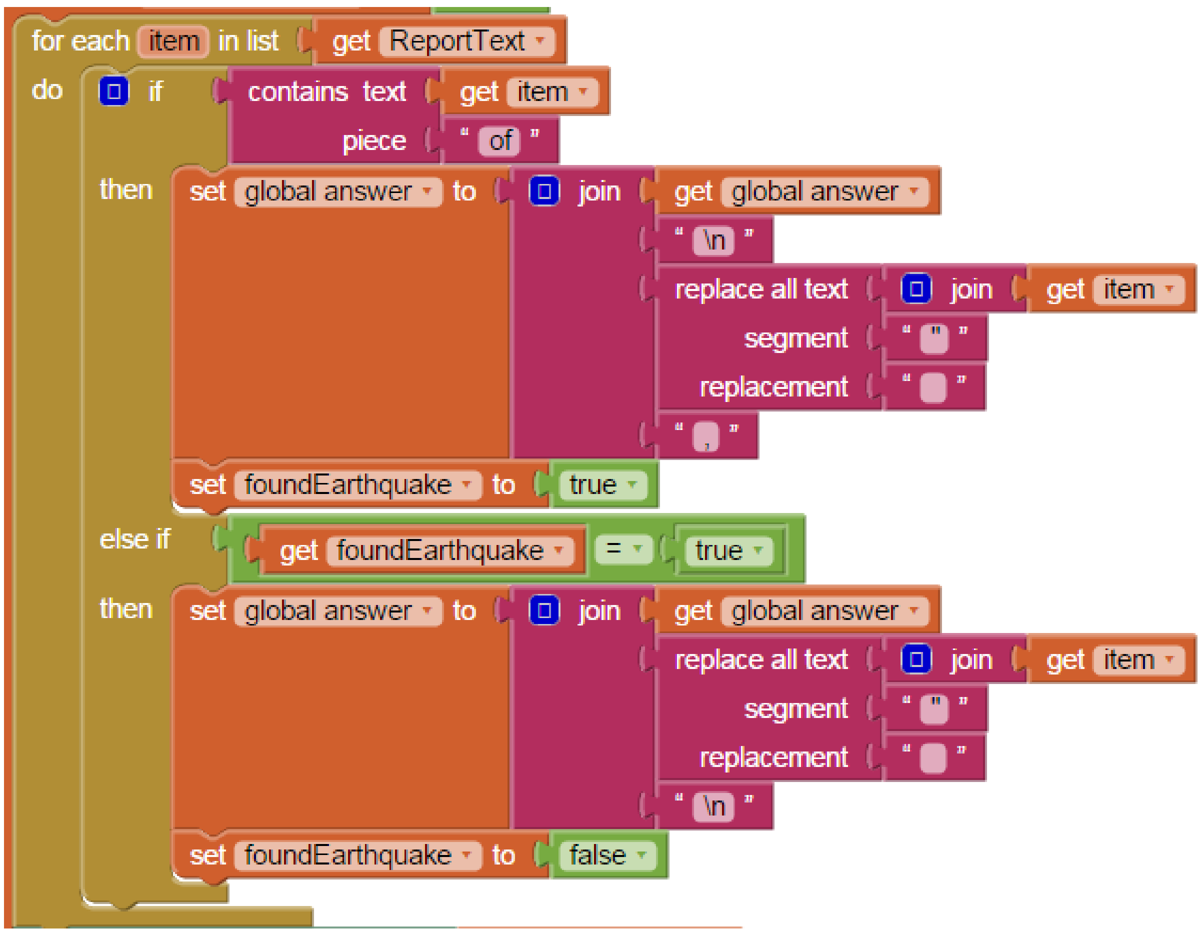
- **For each** statement from **Control** drawer
- **If then else** statement from the **Control** drawer. Once that is clicked in place, use the **blue square** to open up the popup to customize the **if** statement. Drag an **else if** tile into place. This turns the If_then block into an If_then_else_if block.



Resulting block will look like this:



- All **orange** tiles are from the Variables drawer.
- All **red** tiles are from the Text drawer.
- All **green** tiles are from the Logic drawer.



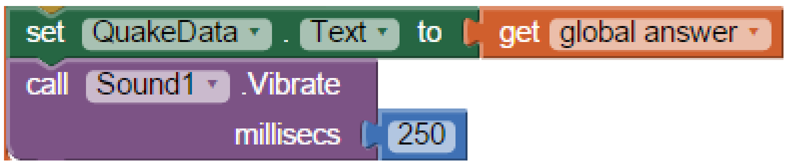
```
for each item in list  get ReportText
do  if  contains text  get item
                 piece  " of "
    then  set global answer to  join  get global answer
                                      " \n "
                                      replace all text  join  get item
                                               segment  " " "
                                           replacement  "   "
                                      " , "
          set foundEarthquake to  true
    else if  get foundEarthquake  =  true
    then  set global answer to  join  get global answer
                                      replace all text  join  get item
                                               segment  " " "
                                           replacement  "   "
                                      " \n "
          set foundEarthquake to  false
```

We process the query results by looking at each item in the list ReportText. Each earthquake location is described as a certain distance away from a city. So one way to find just the place the earthquake occurred data is to look for the word “of”.

If the item has the word “of”, it must be that we have the place information so we add that to the answer variable. We clean it up a bit by replacing the quotation marks with spaces, and use the symbol “\n” or newline to put it on the next line of the answer.

After we read an item that has the word “of”, we know the next item in the list ReportText will be the state or country where the earthquake happened. So if we just found an earthquake entry, then we will also paste the next item onto the end of answer.

Finally we end the large block of code with these blocks:



```
set QuakeData . Text to  get global answer
call Sound1 .Vibrate
           millisecs  250
```

This places the data we retrieved in the QuakeData label and then has our device vibrate for 250 milliseconds.

Now test out your app by asking it to find all earthquakes larger than 6 in magnitude. Make sure the Clear button empties out the data just retrieved.

Extensions to This Project

1. Think of another way to locate and pull the place data from the query file. Hint: the place data appears every 14th item in the list ReportText.
2. Research other web services such as New York Times APIs (documentation at http://developer.nytimes.com/docs). One of the APIs allows you to search the NY Times for articles. Modify your app to allow the user to enter a topic then search for NY Times articles on that topic.
    a. Hint: The API for article search does not return articles in a CSV format like Yahoo! Finance does. Instead it returns values in a JSON format. Check out Copter Labs article on JSON to gain an understanding of how JSON works.
    b. Hint 2: The Web component of App Inventor can handle data retrieved in a JSON format. Check out this article for guidance on how to handle JSON formats in App Inventor.

Resources

- AI2 Connectivity Components: Web
- User Guide for App Inventor 2
- Guide to Understanding Blocks