# Overview

| The Data Map app visualizes data about states. It uses a GeoJSON data file to draw states on the map and a spreadsheet CSV file to read in data about the states into lists. Objectives: In this lesson you will learn to:* Use GeoJSON files to draw polygons on a map,
* Use a local CSV file and read in data from the file into a List of Lists data structure,
* Iterate through a list using a loop to process the data file and display it on the map in InfoBoxes.

[Short Handout](https://docs.google.com/document/d/1wdxBIXOzXy9bAqy1hYF5-SZ7xfPnAK7UZKwz4HK8wbQ/edit?usp=sharing) |  |
| --- | --- |

# Getting Ready

Start App Inventor with the [Data Map App Template](http://ai2.appinventor.mit.edu/?repo=templates.appinventor.mit.edu/trincoll/csp/unit7/templates/DataMap/DataMapTemplate.asc). Once the project opens, use *Save As* to rename your project*.* Note that this app will not work in the version 2.48 Companion app, but it will work with the latest 2.49 and up Companion Apps. It might not work in iOS.

# The Data

Data sources can be included and used in App Inventor apps in 3 different ways:

1. By manually constructing **lists** of data: This is good for small amounts of data, but not a good option if there is a lot of data.
2. By uploading a **data file** into the Media section: This is a good option for larger amounts of data, especially if you can find a file that has the data you need already in it.
3. By using a web **API** to read in the data as it is needed: This is a good option if the data changes frequently

This app uses 2 data files which are already uploaded into the template in the Media section of the template app. The first data file used in this app is a [**Comma-Separated-Values (CSV)**](https://en.wikipedia.org/wiki/Comma-separated_values)file. Any spreadsheet in Excel or Google Sheets can be saved as a **.csv** file which just has the text separated by commas. This is a great format for importing into App Inventor. The .csv file in this app contains state data from this [spreadsheet of state data](https://drive.google.com/open?id=1JbW50ohaUMmZl3h3fo4ntlxW5g8P8NCnuSoBbeCg3J8) from the [Center for Disease Control (CDC)](https://www.cdc.gov/nchs/pressroom/stats_of_the_states.htm) that includes the following data:

| STATE | Population in 2016 from census.gov | Non-insured rate (people who do not have health insurance in every 100K of population in 2016 from CDC.gov) | Firearms Death RATE of death from firearms for every 10-K population in 2016 from CDC.gov | Firearm Deaths totals in 2016 | Drug Overdose death rates per 10K population 2016 | Drug overdose deaths total | CDC URL for state info.  |
| --- | --- | --- | --- | --- | --- | --- | --- |

The second file, [us\_states.geojson](https://docs.google.com/document/d/18zBz7OfIgiDXdhe8JWMQhHyxCRsrUbLsfjE2oGpbFEM/edit?usp=sharing), loads in the polygon shapes of each state to overlay on the map so that we can click on them. [**GeoJSON**](http://geojson.org/)(pronounced geo-jay-sun) is a standard agreed-upon format for geographical information used on the web and in data files. [**JSON**](https://www.json.org/) **(Javascript Object Notation)** is a general format that describes features and values for any object that many web servers use to communicate and store data. This is what the file us\_states.geojson looks like inside. Here’s the description for Alabama:

{ "type": "Feature", "properties": { "GEO\_ID": "0400000US01", "STATE": "01", "NAME": "Alabama", "LSAD": "", "CENSUSAREA": 50645.326 }, "geometry": { "type": "MultiPolygon", "coordinates": [ [ [ [ -86.783628, 34.991925 ], [ -85.605165, 34.984678 ], [ -85.184131, 32.870525 ], [ -84.909052, 32.26354 ], [ -85.141831, 31.839861 ], [ -85.002368, 31.000682 ], [ -87.598928, 30.997457 ], [ -87.394219, 30.641699 ], [ -87.452378, 30.300201 ], [ -87.80056, 30.229365 ], [ -88.028401, 30.221132 ], [ -87.755263, 30.277292 ], [ -87.936717, 30.657432 ], [ -88.061998, 30.644891 ], [ -88.136173, 30.320729 ], [ -88.395023, 30.369425 ], [ -88.471214, 31.851385 ], [ -88.097888, 34.892202 ], [ -88.200064, 34.995634 ], [ -86.783628, 34.991925 ] ] ], [ [ [ -88.124658, 30.28364 ], [ -88.075856, 30.246139 ], [ -88.313323, 30.230024 ], [ -88.141143, 30.255024 ], [ -88.124658, 30.28364 ] ] ] ] } },

Notice that it is a set of latitude and longitude pairs that describe the points of the polygon to draw the state shape.



You can create your own geojson files at <http://geojson.io> and find free public ones online for example at <https://geojson-maps.ash.ms/> .

# Designing the User Interface

For the UI of this app, first put in a Label describing what the user should do, for example “Click on each state to see the 2016 CDC data”. Drag in a Map component from the Maps drawer and set its properties as described in the table below. Drag in a Feature Collection from the Maps drawer and use the use\_states.geojson file for its source. This will create all the states on the map. Also drag in a File component from the Storage drawer to process the CSV spreadsheet data in the code.

| **UI Component**  | **Name** | **Properties** |
| --- | --- | --- |
| Label | Label1 | Text - Click on each state to see 2016 CDC Data |
| Maps/Map | Map1 | * CenterFromString 35.467560, -97.516428 (which is Oklahoma in middle of US)
* Width - Fill Parent
* Height - 50%
* ZoomLevel - 3
 |
| Maps/Feature Collection | Rename:FeatureCollectionStates | Source - us\_states.geojson(This will create all the polygons for the states) |
| Storage/File | File1 | No changes |

# Coding the App

The app will store all the CSV spreadsheet data in a **list of lists** for all the rows in the spreadsheet. Set up an empty list variable called data:

| **Abstraction: List Variables** |  **Values** |
| --- | --- |
| data | Empty list |



Only 2 event handlers are needed for this app. The Screen1.Initialize event handler will read in the file “//StateDataCDC.csv” into the File1 component. The // tells it to get the file from the device running the app since the file comes with the app in its media assets. We will use the File1.ReadFrom procedure to read in the file. Make sure you type in “//StateDataCDC.csv” to match the filename exactly in the text block.



Once File1.ReadFrom completes reading the file, it will call the event handler **File1.GotText**. This event handler returns the read file in the **text** parameter. We will use the [**List from CSV Table** block](http://appinventor.mit.edu/explore/ai2/support/blocks/lists.html#listfromcsvtable) in the Lists drawer to read the file text into our list **data.** You may want to temporarily show the data list in your label to see what it looks like. You will see that it is a complex data abstraction that is a **list of lists.** It is a list of states where each state is a list of data corresponding to each row in the spreadsheet.

 [ [ Alabama, data for Alabama…. ], [Alaska, ….], …. , [Wyoming, …] ]



Notice that the first element in the list is the row of headers in the spreadsheet describing each column. We can remove this first row, so that we start with just the data.



Then, we need a loop to go through the states in our map, and put in information from the data file for each state. The states in the map are in the FeatureCollection component,and the **FeatureCollection.Features** block pulls out all the states in a **List.** We can loop through this list using either a for loop or a for each loop. Both the geojson file for the features and the state data file have the same order, alphabetically by state name so the state at index 1, Alabama, is the same in each file. A for loop here from 1 to length of the FeatureCollection.Features (the loop variable number has been renamed index here) will allow us to use an index that can both traverse the state features in the map and the state info in the data file.



The first thing we want to do in the loop is to set up two local variables for the state feature in the map and the state data in the data file. As explained above, the same index can be used for both parallel lists here because they are both ordered alphabetically by state name.



Now, we can enable the InfoBox property for each state in the map, so that we can pop up an infobox with the data for each state when the user clicks on a state. To do this, we need one of the **Any Component Blocks** which is at the bottom left of the component blocks since we do not have separate components for each state.



Now use select blocks to get the data that you want from the stateData list. You should look at the [spreadsheet of CDC state data](https://drive.google.com/open?id=1JbW50ohaUMmZl3h3fo4ntlxW5g8P8NCnuSoBbeCg3J8) to choose which columns you want to display and count which column it is. You can use the Any Component blocks under Polygon to put these in the Title and Description of the InfoBox for each state. For example, the following code pulls out column **1** for the state name, column **2** for its population, and column **5** for the number of deaths by firearms in 2016:



In summary,

|  **Event Handlers** | **Algorithms** |
| --- | --- |
| Screen1.Initialize | Call File1.ReadFrom filename “//StateDataCDC.csv” .  |
| File1.GotText | -Set the data variable to [list from csv table](http://appinventor.mit.edu/explore/ai2/support/blocks/lists.html#listfromcsvtable) using the text from the file. This will read the comma separated values (csv) from the file into your list. This will be a **list of lists** for all the rows in the spreadsheet.-Remove the first list item in the data list (this is the column headers).-For index from 1 to length of the list FeatureCollectionStates.Features -Set up two local variables, **stateFeature**, which selects a list item from FeatureCollectionStates.Features at index and **stateData** which selects a list item from the global data variable (which has the data from the state data file) at index.  -Use an Any Component block (bottom left) for Polygons to set Polygon.EnableInfobox of each stateFeature to true. - Use an Any Component block for Polygons to set the Polygon.Title and the Polygon.Description of the component stateFeature to items in the stateData list using a select block and the right column number for the index (use the number of each column in the data [spreadsheet](https://drive.google.com/open?id=1JbW50ohaUMmZl3h3fo4ntlxW5g8P8NCnuSoBbeCg3J8), for example the state name is in column 1 so select index 1).  |

# Testing the App

|  **Inputs** | **Expected Outputs** | **Actual Outputs** |
| --- | --- | --- |
| Click on each state | An infobox with the state’s data should pop up. | ? |

# Enhancement Projects

Your instructor may ask you to do some or all of the following challenging enhancements:

1. **Data Visualization with Colors:** Create a map visualization with 3 color shades for states to show the differences in one of the data columns in the data [spreadsheet](https://drive.google.com/open?id=1JbW50ohaUMmZl3h3fo4ntlxW5g8P8NCnuSoBbeCg3J8). For example, here the states that had less than 10 death rate by firearms for every 10K people in 2016 are shown in light blue, the states that had between 10 and 20 deaths in medium blue, and the states that had the greater than 20 death rate by firearms in dark blue. To create this color scheme, add an if block and use the blue mutator to make it into a three-way choice: if/else-if/else block and set up the 3 ranges and use the Any Component block for set **Polygon.FillColor**.

**Error Checking:** You may run into errors with the data in spreadsheets. Often we have to clean data or check for special conditions before we use it. Some of the values are empty in the spreadsheet which may cause errors. You should first save the data in a local variable and check that it is not empty text to avoid errors. For example, if the data in the spreadsheet contained commas “,” in numbers such as “1,203” this would cause problems with comparing them as numbers so the commas would need to be deleted.

.

1. **Procedure with Parameters:** Our app would be even more useful if it let the user decide which data to present. Add 3 buttons to choose between different columns of data in the spreadsheet, for example Uninsured Rate, Firearms Death Rate, Overdose Rate. Refactor your code to add a **procedure with parameters** for the **column number** and the **label** of that column and have button click event handlers call this procedure. This procedure will update the map data (and color code it if you did extension 1) using a loop through the states like in File.GotText (except for the first two lines of code that set up the global data variable) but using your parameter variables. In fact, you can refactor your code so that File.GotText also calls this procedure with some default variables after setting up the global data variable in the first two lines of code. Note that if you choose data columns where the data is not in the same ranges (for example population), you may need more parameters to adjust the limit values where the color shades change in your if statements. You can also change the spreadsheet to include other data, see <https://www.cdc.gov/nchs/pressroom/stats_of_the_states.htm>. Note that this procedure with parameters meets the requirements of the AP Performance task. Thank you to Mobile CSP teacher Jocelyn Humphries from John Jay High School in NY for this awesome coding idea!
2. **WebViewer:** Note the last column in the data [spreadsheet](https://drive.google.com/open?id=1JbW50ohaUMmZl3h3fo4ntlxW5g8P8NCnuSoBbeCg3J8), contains a url for more information about the state on the CDC site. Use a webviewer to display a url that joins the base url <https://www.cdc.gov> with the url in the last column (#8) when each state is clicked by using the **Map.FeatureClick** event handler. To get the URL data, you can find the index of the state feature that is clicked by using an [Index in List](http://appinventor.mit.edu/explore/ai2/support/blocks/lists.html#indexinlist) block with the feature that is clicked and the list FeatureCollectionStates.Features. Once you have this index, you can use it to select that state’s data from the global data list. Remember that this is a list of lists, so once you find the correct list of data for that state, you will need to use select again to find the URL data which is at index 8.
3. **Weather API (Optional, requires registering for a free API key): Weather API:** [APIs](https://en.wikipedia.org/wiki/Application_programming_interface) can be used to read in real-time current data, for example the current weather report for a clicked state. Read about the OpenWeatherMap API here: <https://openweathermap.org/current>. Try clicking on this example: <https://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b6907d289e10d714a6e88b30761fae22> to get the current weather data in JSON format for London. OpenWeatherMap requires a registration key called appid. To get this free key, your instructor should follow the directions at <https://openweathermap.org/appid> and then tell you the key, for example appid=8bb5e8bedfe6fe3f1a44e0a2c04b6540.

We need to build this url for each clicked state and pull out the main weather description. To make an API request, you will need a **Connectivity/**[**Web**](http://ai2.appinventor.mit.edu/reference/components/connectivity.html#Web) component (this is different than the WebViewer component).

* Use a **Map.FeatureClick** event handler and set the **Web.url** to the API url like **http://api.openweathermap.org/data/2.5/weather?q=*state*&appid=*yourAppId*** using a join to put in the state name which is the title of the clicked feature (using an Any Feature Component) and your appid (the API key) given by your instructor (you can try the Mobile CSP one appid=8bb5e8bedfe6fe3f1a44e0a2c04b6540 but it may be blocked if too many people are using it).
* Then, call **Web1.get**. This will fetch that webpage and then go to the event-handler **When Web1.GotText**.
* In the GotText event handler, you will need to parse the result to find the weather main description, for example “clouds” below: {"coord":{"lon":-78.39,"lat":43.1},"weather":[{"id":804**,"main":"Clouds"**,"description":"overcast clouds","icon":"04n"}.

The [List/**Lookup in pairs**](http://appinventor.mit.edu/explore/ai2/support/blocks/lists.html#lookupinpairs)block can pull out the weather key and then the main key in the result text. The following code will pull out this part of the JSON data returned from this API which you can then display in a label:



If you’re curious about other APIs, here’s a list of different public APIs that you can use in apps: <https://github.com/toddmotto/public-apis>.

# Summary

Review the following new blocks in this lesson:

| File readFrom |  |
| --- | --- |
| List of Lists from csv table |  |
| Any Component Blocks |  |
| Look up in pairs |  |

# Vocabulary Review

Review the following vocabulary in this lesson:

* CSV files
* JSON
* GeoJSON
* API
* List of Lists

**