# 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.

# Getting ReadyStart 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) (click on Login with Google). 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.The Data

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. 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). 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. 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

| **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

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

|  **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. | ? |

# Enhancements

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 this spreadsheet. 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 a [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>.