Overview:

Viztool is a framework that aims to help a user easily visualize any type of data from any source in one location. A user runs Viztool as a Flask server then sends data to the hub component of Viztool. The hub processes incoming data updates and then with WebSockets (socket.io) sends data to the frontend to render the visualizations of the data in the form of various graphic objects or tables to an endpoint <host name>:<port>. By allowing the user implement their visualizations in JavaScript libraries, Viztool is highly customizable and extensible. Currently, the demos provided use c3.js and d3.js to create these visualizations. The entry point to the Viztool system is a static JSON configuration file which outlines the devices that will send data, the events and data these devices will send, and in which graphs (as well as their format) the user wants the data to be displayed.

Installation Instructions:

To install the Viztool system onto your own machine, follow these steps:

1. Download and extract the contents of the Viztool zip folder
2. Ensure Python version 2.7
3. Install dependencies with the command “pip install -r requirements.txt” (strongly recommend virtual environment)

Running Viztool on “localhost”:

To run this as a local server simply run the ‘hub.py’ script with ‘HOST’ set to “localhost”. This starts the Flask server. A configuration JSON file must be placed in the ‘config’ folder (see below more information about the config file) and then once the user visits localhost:<PORT>/<urlExtension> the config file with that urlExtension will be registered in the hub. At this point a user can send data to the hub (see below) and visualize the data in graphs or tables at localhost:<PORT>/<urlExtension>

Setting up Viztool to Run on Central Server with Public Endpoint:

Viztool can also be run as a central server (standalone eventlet WSGI server) that data sources on other machines can push data to. Likewise, the visualizations would be viewable to all by going to that endpoint. For example, we have Viztool running on “vacancy.cs.umd.edu:8000/temperature”. On this page you can view an IoT dashboard monitoring several different temperatures.

Point of Entry:

The point of entry to Viztool is the static JSON config file. Config files must be named with a .json extension and placed in the ‘config’ directory. The config file defines the devices which will be sending data (in the form of events) and what graphs are to be displayed on the browser and in which format. See examples like ‘temperature.json’, ‘stocks.json’, and ‘status\_page.json.’ The schema for the config is also defined in ‘configValidator.py.’ The graph section of the config contains an object called ‘json\_format’ this object defines the graphic object in a very similar way to the parameter of the c3.generate function here: <http://c3js.org/gettingstarted.html#generate>

The sources section of the graph object is very important. This is what specifies which data fields (from which devices/events) will update the graph object.

Sending Data Event from a Device to Viztool Hub:

All data updates sent to the hub must be in the form of JSON and contain a “header” object with “device”, “event”, “urlExtension” (note this is a list) fields, and and optional “token” field. The name of these fields must match exactly with what was defined in the hub otherwise the updates will be disregarded. The optional “token” field is just a minor security measure that allows a user to ensure not just anyone can update the data visualization, only those with the proper token defined by the admin in the config file. After the header object is a “data” object. The fields in this must match the data fields listed in the sources section of the config. See “thermostat1.py” or “scraper.py,” among other scripts in the ‘devices’ directory for examples.

Extending the Frontend to Support Custom Visualizations:

An important distinction that must be made clear is that c3.js and d3.js are not parts of Viztool but rather help us in creating the visualization demos. For the walkthrough and in these provided demos they are essentially part of Viztool; however, upon final delivery Viztool will aim to be more modular following the c3 ‘generate’ and ‘load’ model, but allowing the user to actually visualize with his own more customizable JavaScript code.

The interfaces for visual extension and customization are the data receiving interfaces (socketio event handlers) themselves. In this case, they serve as the first-level handlers for data visualization. The user could include .js files which contain his own visualizations in the header of this template page and then call his “generate” and “reload” functions (the second-level handlers) within those first-level handler. This is also why we could easily support the entire c3.js.