Using the ArcGIS API for JavaScript::Deanne Lundin::GEOG863 (Fall 2011)

Task:

One of Esri's sample servers provides a map service depicting hurricane tracks observed between 1851 and 2007 (http://sampleserver3.arcgisonline.com/ArcGIS/rest/services/Hurricanes/NOAA_Tracks_1851_2007/MapServer). Using this map service:

- Create a map that shows hurricanes of Category 4 or 5 strength.
- Override default symbology with your own using a light green to dark green color ramp to indicate the strength of the hurricane.
  Beyond the basics:
  - List the hurricanes in a sidebar

Resources included: GEOG863 text, Esri's JavaScript API, O'Reilly's JavaScript: The Definitive Guide and Javascript Pocket Reference, Dojo Toolkit (dojotoolkit.org), w3schools.com (JavaScript, CSS, HTML, XML), stackoverflow.com, Mozilla Developer Network (developer.mozilla.org), and Rich's AsktheEngineer Tutoring Service

Workflow and Methods

For me this exercise was learning-intensive—I'm a novice programmer. That shows most in my comprehension of all a task will require. In GEOG485 I was introduced to the concept of pseudocode, which I find tremendously effective in sorting through problems. Everything still feels new to me but I want to keep adding experience with new tools and resources, and this project certainly achieved that.

With many examples before me, the task of filtering the feature layer to display only hurricanes of H4 and H5 was something I could see how to do. Changing the default symbology proved more challenging in Dojo than I'd expected. Using the renderer and its subclass, UniqueValueRenderer, isn't difficult once you've done it but setting a default symbol only to override it later with addValue() wasn't intuitive. Scrupulous attention to everything the text showed me took me through the basic process of using a map service to load a basemap and a new feature layer and allowing for IE's idiosyncrasies. I used Dojo.layout to build a few samples. But these were truly minor challenges once I cracked open the sidebar task.

As I began looking through examples (including those in the text) I realized the nature of my data was fundamentally different. Trying to query multiple tracks from hundreds of hurricanes was not going to be possible by simply selecting H4/H5 from the category field since any hurricane varies in intensity along its path. Each of those variations meant data associated with a single observation. And while each hurricane did have a unique ID associating all its data, that ID could not be directly queried for H4/H5 values. And I didn't want a grid that contained every H4/H5 track for a hurricane, either—I just wanted a single hurricane name associated with the category of its highest intensity which needed to be either H4 or H5. I began to think I was going to need some kind of array.

Solving the Datagrid Problem

It was already clear the problem was going to take quite a bit more than code-on-the-fly and test. GEOG485 introduced me to the practice of pseudocode—problem-solving in plain English. It had made possible a number of challenging coding tasks that looked impossibly thorny at first.

My first draft was no more than description of how the grid should look but getting it down in writing revealed the nature of what I needed to do. I wanted the hurricane name, its category, its wind speed, and its date. But there were going to be multiple observations for each hurricane; many hurricanes never achieved the categories I needed; and each recorded observation meant a corresponding track (Object ID) and date for the observation. Looking at a meteorologist's table of hurricane data I realized it was just a matter of identifying the maximum wind speed each hurricane achieved (I toyed briefly with the idea of including an intersection with its minimum pressure but knew it would exponentially increase the complexity of the already beyond-me task). That maximum wind speed would represent a single record (and a single date). Once I had a collection of wind speeds for each BTID (unique storm ID) I could query that collection for the H4/H5 values, which would also be associated with the wind speeds. It was a simple matter of reorganizing the data so that it could be queried.

What I wound up with was this: Use a SQL query to retrieve only the observations which were H4 or H5 category. Pass all this data one observation at a time to a function that would note the highest wind speeds for each storm track in an array. Pass that array to the datagrid for display on the web page.

Well, a simple matter for any intermediate engineer. Not for a novice. This was going to take me into as-yet uncharted territory of getting JavaScript and Dojo to play nicely together. I had to understand how I could use these resources to create arrays, walk through them, and inspect the data for the values I wanted. (I'd already done some of this in Python, but not in JavaScript.) I spent several days in a whirlwind of tracking down how-to sources for both JavaScript and Dojo, looking at classes, methods, properties, and examples using them. I had the dim idea from my brief experience with Python that I was probably going to want to make a function myself and after dwelling in tutorials and API documentation that suspicion became a certainty. Even with all these resources, the O'Reilly books (which I thankfully already had) were invaluable, teaching me more about Ajax and jQuery, for example, as well as the task-specific information I needed. But I was in danger of whiteout from information overload so I used a LifeLine: I called an engineer I knew and asked for some serious tutoring. "You need JavaScript Objects," he said, and it sounded like "You need pi," but we started there and spent several days of intensive code work and learning until we got it.

One of the more challenging aspects was looking through the examples and sorting out what the anonymous functions do and how they scan through the data in the FeatureLayer. I think code might be easier to understand if programmers would use clearly named functions which they'd pass (e.g. to layer.queryFeatures()) instead of coding anonymous functions right in the parameter list.

In the first working script I saw that the date would not be sortable (unless I gave up and got rid of everything except the year. It was tempting.) Once it was working beautifully (in Firefox) IE kept freezing and eventually crashing, so I couldn't debug to see what was wrong. I did however have a good clue in that everything was loading (including the datagrid headers) but the datagrid was not populating; this turned out to be lack of support for the date formatting method "toISOString" (although the newest IE finally does support it). I couldn't see how to re-format the date for sortability except by setting up a series of variables, getting the year, concatenating a leading zero to the month and day strings, and using an "if" statement.

As a result of this process, the script is heavily commented throughout partly because I was learning most of it as I went and needed to keep track of what was actually happening in the script. Primarily though I find it the best way to take notes I can refer to later. Because I'd started this habit with the only other programming course I've had (GEOG485) I was able to remind myself of what I'd learned (but hadn't used since).

I’m coming to the conclusion that one primary characteristic of a mashup is its collaborative nature—not just bringing together different datasets but using many resources for working through a problem. Maybe that’s one reason they’re so appealing. There’s something profoundly social about them.

This application demonstrates that principle: its map layers are provided by Esri’s map services, which themselves draw in data from disparate sources; it uses the Dojo class “datagrid” to display tabular data in the side-bar. But it may be the process of learning how to code a web application that is most social. It might be a sad commentary on my life that I spent Thanksgiving with my best friend (and his hideous cold) working through JavaScript and Dojo in our separate corners of the room (because a virus is also social) and we actually had a great time. OK, it's sad. But in a happy kind of way.