Vote 2012 Map Center Documentation

Release 1.4

Justin Myers

August 10, 2012

CONTENTS

This documentation describes how the PBS NewsHour's Vote 2012 Map Center is put together from a technical perspective, primarily for people who are interested in developing new features for it.

Note: Throughout this documentation, we refer to certain file names and file paths. These paths are relative to the Map Center root directory, which is `http://www.pbs.org/newshour/vote2012/map/` or the `frontend/dist` directory of the Map Center's GitHub repository.

CHAPTER

ONE

CONTENTS

1.1 Principles of operation

The Map Center is designed to be a system for rapid (up to near-real-time) generation of browser-based vector choropleth maps related to the 2012 state and federal election cycle.

1.1.1 Compatibility

The Map Center officially supports (i.e., is specifically tested on) the latest versions of Firefox, Chrome and Mobile Safari as well as Internet Explorer 8 and later. More generally, it should be compatible with any relatively modern browser that supports SVG, including Android 3.0; Android 2.x devices are considered partially compatible due to a canvas fallback that does not support events on shapes on the canvas.

There is a particular emphasis on planning for touch events and supporting Mobile Safari on the iPad in particular since an iPad 2 (using HDMI mirroring and eventually feeding into the WETA control room) is used to manipulate the Map Center on the air.

1.1.2 Components

We use jQuery for most of our client-side heavy lifting, including handling AJAX (technically JSONP) requests, *UI* events and animations. jQuery UI is used for specific widgets – primarily dialog boxes. Highcharts handles our occasional charting needs (currently just in the *Electoral College module*), and Dojo gets brought in for its vector graphics library. We use Modernizr to determine whether users are on touch-capable devices.

The core Map Center code handles the rest of the basics–drawing, coloring and clearing map areas (e.g., counties and states), adding city labels where necessary, positioning tooltips on map areas, binding common *UI behaviors* and passing execution off to a specific Map Center *module*.

1.1.3 Execution flow

When the DOM is ready, we load Dojo's graphics library and prepare a surface object on the `#map` element. This surface object is exposed as *nhmc.surface* and is the area on which all map vector graphics will be drawn.

Once that surface is ready, we initialize the rest of the Map Center core in *nhmc.mapCenterInit*. That function is described more fully in *its own documentation*, but overall, it does the following:

- Renders the initial nationwide state map if provided.
- Binds specific map-drawing event handlers.

- Shows the default map view on the page, as defined in the `#map_view` hidden `<input>` element (generally included at the bottom of the sidebar).
- Binds a few outdated UI behaviors (used in initial testing but referring to elements which are currently omitted).
- Binds general UI behaviors for the *tabs* that are included in most modules.

When the initial render of the map is finished, the *drawingComplete* signal fires (which happens any time a map view is finished rendering), which in turn fires (this one time only) the *coreInitialized* signal to pass execution off to whatever *module* is responsible for the page. (All modules ultimately consist of an event handler that is bound to *coreInitialized*.)

1.2 Package reference

All of the core Map Center functionality is contained in "packages" (objects) underneath the global `nhmc` variable. All of these packages (including the root `nhmc` package) are standard JavaScript objects created using a simple namespace function included in the `<head>` of every Map Center page.

Packages:

1.2.1 nhmc (root)

Most of the Map Center's functionality is contained either in its *modules* or in *packages* underneath the root `nhmc` package, but a handful of important objects are contained at this level for lack of a more approprate place to put them.

Graphics objects

nhmc.surface

Object (instance of dojox.gfx.Surface)

Dojo's graphics library requires the creation of a "surface" object, which provides a rectangular drawing area on which vector images may be created. The `nhmc.surface` object always corresponds to that surface for the Map Center, and it is kept to the size of the `#map` element on the page. The Dojo documentation contains more guidance on the functionality of this type of object.

Initialization

nhmc.mapCenterInit

Function

```
nhmc.mapCenterInit()
```

This function executes as soon as *nhmc.surface* is loaded. It prepares the application environment for whatever *module* is ultimately responsible for the page by performing the following duties:

General configuration overrides Many general configuration variables (text sizes, default fill colors, etc.) are located in *nhmc.config*, but sometimes it makes sense to override them. `nhmc.mapCenterInit` begins by searching for a global object named `nhmc_config`; if this object exists, it is merged into `nhmc.config` and overwrites any identically named properties. The logic for this deep copy operation is handled by jQuery.extend.

Pre-drawing nationwide state map The nationwide state map (i.e., the U.S. map that just shows state boundaries) is a special case of *map view* that, if used, must already be rendered. Because of this, `nhmc.mapCenterInit` locates the path data if available, draws the states and adds labels for the especially small ones and then immediately hides all of those objects. This leaves them ready to go in case the user or the *module* selects that map view later on.

Setting initial scale factors The Map Center supports scaling *map views* as necessary to fit the width of the `#map` element that contains the actual *map surface*. At this point in the application, no measurements have been taken of the size of `#map`, so we assume all maps to be rendered at their original size and store the corresponding scale factor of 1 for each possible map name.

Bind to map initialization signal The default map view is set with the value of the `#map_view` hidden `<input>` element, which is typically located at the end of the sidebar markup. We kick off the actual rendering process by calling *nhmc.ctrl.zoomToState* with the initial value of `#map_view`.

Before we do that, however, we listen for the first firing of the *drawingComplete* signal. When this signal fires for the first time, we ensure that the map is scaled properly and then fire the *coreInitialized* signal to pass execution off to the *module* responsible for the rest of the page.

Todo

There's also an event handler bound to `#map_view`'s `change` event that is most likely implemented incorrectly. In practice, it almost certainly never gets used and probably can be removed, but this should be investigated first.

Bind vestigial UI events Toward the beginning of the Map Center's development, we experimented with some UI interactions that would allow for switching between the map view and a large chart of some sort for another view of the information being displayed. (The chart's exact contents would have depended on those of the map, of course.) Because of this, `nhmc.mapCenterInit` still binds UI events that that were related to that interaction and a few other testing elements we were using at that time.

Todo

Consider removing these.

Bind general tab behaviors Finally, `nhmc.mapCenterInit` binds UI behaviors that actually *are* used–namely, those related to the tabs shown next to the page's `<h1>` in most maps (currently all unless they're *embedded*). There are three main behaviors to know about:

- When a tab is selected somehow, assign it the `.view_tab_active` class and remove that class from all other tabs. "Selecting" a tab in this context also includes clicking on an option in that tab's child dropdown menu, which is covered next.
- When a tab contains a button to show or hide a child dropdown menu, ensure that clicking that button actually shows or hides that menu. This possibility is limited to a maximum of two tabs with specific naming conventions:
  - One tab called `#view_tab_more`, with a toggle button called `#view_tab_more_toggle` and a child dropdown menu called `#view_tab_more_menu`, which is most often used for selecting a specific map view. (It's also used in the *Electoral College calculator* for selecting a previous election's results to display.)
  - One tab called `#view_tab_options_more`, with a toggle button called `#view_tab_options_more_toggle` and a child dropdown menu called `#view_tab_options_more_menu`, which is most often used for selecting a more specific data set (for example, a specific ethnic group in a map of ethnic breakdowns).

Event handlers for the options within tabs' dropdown menus are delegated, so it is possible to add and remove these options after the Map Center core has been initialized. This is done for *non-primary election results* to show a list of races to choose from, for example.

- For tabs that contain child dropdown menus, ensure that clicking on the name of the tab toggles the dropdown menu instead of selecting the tab. This was changed after usability testing within the NewsHour (both on mobile and on desktop devices) showed that this was the expected behavior. This is overridden, however, on the *Electoral College calculator*, where tabs also exist without dropdown menus.

Orphans (unused components which should be removed)

nhmc.R

null

Originally used as a reference to the Raphael graphics library; unused since switching to Dojo in September 2011. Currently set to `null`.

Todo

Remove this.

1.2.2 nhmc.charts

This package just contains placeholder objects to be used with the Highcharts charting library. Various charts in the Map Center can use these names for consistency with other modules. Each of these objects should be either `null` or an instance of `Highcharts.Chart`, which is further documented on the Highcharts site.

Chart objects

nhmc.charts.mainChart

Object (instance of Highcharts.Chart) or null

The `nhmc.charts.mainChart` object is not currently used. It used to be in place to support an alternative chart view that could be shown instead of a map in particular cases (see *a longer explanation* with `nhmc.mapCenterInit`), but this isn't currently done.

nhmc.charts.lowerChart

Object (instance of Highcharts.Chart) or null

The `nhmc.charts.lowerChart` object is used for supplemental charts that don't need to be as large as the map itself. Right now this is used in one place: at the bottom of the sidebar in the *Electoral College calculator* to show how many electoral votes each candidate has in a given scenario.

1.2.3 nhmc.cleanup

This package contains a variety of utility functions that remove application elements when they're no longer needed. It also contains a few objects that store references to these elements so that the cleanup functions know what they're looking for.

General-purpose garbage

nhmc.cleanup.futureGarbage

Array

This is a standard JavaScript array that holds jQuery objects that might need removal later. It's meant for use with the convenience function *nhmc.cleanup.clearGarbage*, which just calls `.remove()` on all elements of this array.

nhmc.cleanup.clearGarbage

Function

```
nhmc.cleanup.clearGarbage()
```

This calls `.remove()` on all elements of *nhmc.cleanup.futureGarbage*, which should be jQuery objects referring to elements on the page.

Map graphics

nhmc.cleanup.clearPathColors

Function

```
nhmc.cleanup.clearPathColors()
```

This function simply clears the color of every state and county path drawn on the map; more specifically, it sets the fill color of each of those paths to *nhmc.config.defaultAttributes.fill*.

Dialog boxes

nhmc.cleanup.activeDialogs

Array

Sometimes the Map Center interface makes use of jQuery UI dialog boxes to request or present additional information; for example, the *Electoral College calculator* uses them to allow users to split certain states' electoral votes where that is possible.

`nhmc.cleanup.activeDialogs` is a standard JavaScript array that stores jQuery objects referring to these dialog boxes for easy reference later–primarily by *nhmc.cleanup.closeDialogs*.

nhmc.cleanup.closeDialogs

Function

```
nhmc.cleanup.closeDialogs()
```

This function just closes any dialogs stored in *nhmc.cleanup.activeDialogs* by calling `.dialog("close")` on each object therein.

Event handlers

nhmc.cleanup.clickHandlerTokens

Array

This is a standard JavaScript array that contains zero or more handles returned by dojo.connect. Dojo's event handling for graphics objects (and other items in general, but we only use it for graphics objects) is powerful in that it doesn't require DOM elements to exist as in jQuery–but it introduces a bit more complexity in that binding an event handler returns a handle that must be passed later to dojo.disconnect in order to unbind that handler.

This array specifically is meant to hold handles related to event handlers for click events on map areas. An example of its use from the *Electoral College calculator*:

```
var eventToken = nhmc.geo.usGeo['Nebraska'].statePath.connect(
    'onclick', nhmc.geo.usGeo['Nebraska'].statePath, nebraskaHandler
);
nhmc.cleanup.clickHandlerTokens.push(eventToken);
```

This is done to make it easier to unbind all such handlers at once when needed by using nhmc.cleanup.clearClickHandlers, described below.

nhmc.cleanup.clearClickHandlers

Function

```
nhmc.cleanup.clearClickHandlers()
```

This function simply goes through all handles stored in nhmc.cleanup.clickHandlerTokens and passes them to dojo.disconnect. It's a handy way to unbind all click handlers at the same time when needed.

Charts

nhmc.cleanup.clearCharts

Function

```
nhmc.cleanup.clearCharts()
```

This function checks for the existence of the chart objects described in *nhmc.charts*; if any of those objects are found, they are destroyed using the Highcharts Chart.destroy method.

Everything

nhmc.cleanup.clearMap

Function

```
nhmc.cleanup.clearMap()
```

This calls all of the above cleanup functions and removes all tooltips using *nhmc.tooltips.destroy* and *nhmc.tooltips.unbindHover*. It's a nuclear option for getting everything you can off of the map.

1.2.4 nhmc.config

This package contains a variety of options related to the general appearance and operation of the Map Center. When a Map Center page is loaded, *nhmc.mapCenterInit* checks for the presence of a global object called `nhmc_config`; if that object exists, its values are merged into this package before initialization proceeds, allowing for individual pages to override these options as necessary.

Rendering options

nhmc.config.defaultDimensions

Object

This contains the dimensions of the original SVG images on which the Map Center views are based. Several *scaling* calculations take these dimensions into account.

> **Caution:** These dimensions should not be changed unless the source vector imagery on which the Map Center views are based also have changed. If you're attempting to change the size of the map, do so by setting the size of the `#map` element in CSS.

nhmc.config.countySteps

Object

The Map Center renders maps containing counties gradually to avoid locking up users' browsers and giving them the dreaded "browser unresponsive" error. To do this, we take the set of counties to be drawn and divide it into several subsets of counties so the browser can catch its breath between subsets. (For more, see *nhmc.ctrl.zoomToState*.)

This object sets how many subsets should be used. `nhmc.config.countySteps.state` is used for statewide county maps, and `nhmc.config.countySteps.national` is used for the nationwide county map. Their values have been reached through some trial and error in performance testing, and we don't recommend changing them.

Color options

nhmc.config.styleColors

Object

This object contains an assortment of named colors for use in various *modules*. It's not used much, but it's there in case it comes in handy.

nhmc.config.defaultAttributes

Object

This object contains the default fill and stroke colors for state and county paths.

City labels

nhmc.config.cityLabels

Object

This object contains options for the city labels on statewide county maps.

City labels contain three components:

Label The label is a text element containing the city's name. It has three configuration options in `nhmc.config.cityLabels`:

- `labelNudge` is used to tweak the alignment of the label and the marker (described below). It is represented as a number of pixels the label should be moved downward.
- `labelOpacity` sets the label's opacity and may be set to any value from 0 to 1 inclusive; a `labelOpacity` of 1 makes all labels completely opaque.
- `labelSize` sets the size of the label text in pixels.

Marker The marker is a black circle placed at the city's location on the map. It has two configuration options in `nhmc.config.cityLabels`:

- `markerOpacity` sets the marker's opacity and may be set to any value from 0 to 1 inclusive; a `markerOpacity` of 1 makes all markers completely opaque.
- `markerRadius` sets the radius of the marker in pixels.

Background The background is an optional white box that sits behind the label and marker to better distinguish them from the map behind them. This isn't currently used anywhere in the Map Center, but it has three configuration options in `nhmc.config.cityLabels`:

- `backgroundOpacity` sets the background's opacity and may be set to any value from 0 to 1 inclusive; a `backgroundOpacity` of 1 makes all city label backgrounds completely opaque.
- `backgroundPadding` sets the amount of extra space on each side of the label/marker combination that should be backed by the background element. This property is an array in standard CSS order; that is, the four values in this array are used for the top, right, bottom and left of the city label, in that order.

  City labels typically are drawn such that the label is placed to the right of the marker. When this is impractical (either because such a placement would leave the label cut off by the right edge of the map surface or because this label placement would overlap with the label text of another city), though, the label is placed to the left of the marker instead. Because of this, "right" and left" have special meanings in the `backgroundPadding` property: "right" always refers to the end of the background farthest from the marker, and "left" always refers to the end of the background closest to the marker.
- `hideBackgrounds` should be set to `true` if backgrounds should not be drawn or `false` if they should be drawn.

Identifiers

`lib/map_center/usps_fips.js` contains a few useful sets of identifiers that get used frequently.

U.S. Postal Service state abbreviations

Two objects contain mappings to and from USPS state abbreviations:

- `nhmc.config.stateToUSPS` has full state names as keys and abbreviations as values.
- `nhmc.config.USPSToState` has abbreviations as keys and names as values.

Some states did not report Republican presidential primary results by county in 2012; these states have additional abbreviations listed for maps with one area for the entire state. For example, the single-area map of Maine corresponds to a state name of `Maine (statewide)` and an abbreviation of `ME_EMPTY`.

FIPS 6-4 county codes

Two objects contain mappings to and from five-digit FIPS county codes:

- `nhmc.config.countyToFIPS` has full state names as keys and objects as values. Those objects in turn have full county (or county-equivalent) names as keys and FIPS county codes as values. For example:

  ```
  nhmc.config.countyToFIPS["Virginia"]["Arlington County"] === "51013"
  ```

  To accommodate the single-area statewide maps described *above*, there are also sub-objects for each state that has such a map, using the pseudo-FIPS code consisting of the state's two-digit FIPS code followed by `000` for the "county" comprising the entire state. For example:

  ```
  nhmc.config.countyToFIPS["Maine (statewide)"]["Maine"] === "23000"
  ```

- `nhmc.config.FIPSToCounty` has FIPS county codes as keys and arrays as values. Each array contains two elements: The first is the full state name, and the second is the full county name. For example:

  ```
  nhmc.config.FIPSToCounty["51013"] === ["Virginia", "Arlington County"]
  nhmc.config.FIPSToCounty["23000"] === ["Maine (statewide)", "Maine"]
  ```

Miscellaneous

nhmc.config.hashParams

Object

This object controls the format of values used in a page's fragment identifier to let users link to specific map views within particular *modules* or other initial states of module code. (For example, it could be used to share users' predictions of Electoral College outcomes.) *nhmc.ctrl.hashParams* handles the setting and getting of these values, but `nhmc.config.hashParams` controls the characters used to format them within the fragment identifier.

1.2.5 nhmc.ctrl

This package contains functions mostly for manipulating, rendering and switching among various map views, though it also contains other utility functions that can be useful to various *modules*.

Scaling

nhmc.ctrl.scaleStateMap

Function

`nhmc.ctrl.scaleStateMap(state, factor)`

This function scales the map view specified by `state` by the scale factor specified by `factor`. Scaling the map view consists of resizing each state or county path in the map view and then resizing each city label in the map view.

Note: Despite the name `state` in the argument and the function name, that term in this context is used in a general sense to refer to the name of any map view, including `us_all` for the *nationwide state map* and `us_counties` for the *nationwide county map*.

---

nhmc.ctrl.scaleSurface

Function

`nhmc.ctrl.scaleSurface(factor)`

This function sets the height and width of the *map surface* to the default image size scaled in both dimensions by `factor`; for example, a 768x486px image scaled by 0.5 would be 384x243px.

nhmc.ctrl.scaleCurrentMap

Function

`nhmc.ctrl.scaleCurrentMap(factor)`

This function first *scales the map surface* and then *scales the current map view*, both by `factor`.

Colors

nhmc.ctrl.setStateColors

Function

`nhmc.ctrl.setStateColors(states, color)`

This function only has an effect on the *nationwide state map* and sets the fill color of each specified state. It accepts two arguments:

- `states` is an array of full state names as found in *nhmc.config.stateToUSPS*.
- `color` is a string consisting of either a hexadecimal color code or a color name found in *nhmc.config.styleColors*.

nhmc.ctrl.setCountyColors

Function

`nhmc.ctrl.setCountyColors(state, counties, color)`

This function affects both *nationwide* and *statewide* county maps. It accepts three arguments:

- `state` is a full state name specifying the state containing the named counties.
- `counties` is an array of full county names within the state as found in *nhmc.config.countyToFIPS*.
- `color` is a string consisting of either a hexadecimal color code or a color name found in *nhmc.config.styleColors*.

Map rendering

nhmc.ctrl.addCity

Function

`nhmc.ctrl.addCity(cityName, cityData, existingCities)`

This function adds a *city label* to a map view (currently limited to *statewide county maps*). It takes three arguments:

- `cityName` is a string with the name of the city.
- `cityData` is an array with two elements:
  - The first element is an x-y coordinate pair expressed as a two-element array.
  - The second element is a city rank that can be used to distinguish cities by size or importance. This element is a number, and lower values represent more important/larger cities. (By convention, zero is used for a state capital.) This value currently is not used to vary a city's visual representation, but it *is* used in *nhmc.ctrl.zoomToState* to ensure that more important cities are drawn first before proceeding to other cities.
- `existingCities` is an array of city labels that have already been drawn on the current map view. This is used to ensure city labels don't inadvertently overlap.

nhmc.ctrl.zoomToState

Function

`nhmc.ctrl.zoomToState(state)`

This function clears the current map view (by calling *nhmc.cleanup.clearMap*) and renders the one named by the `state` argument.

If *tooltips* are already bound on the page before this function is called, they are *reinitialized* when the map view has finished rendering.

See Also:

For more information on the map views themselves and the objects they comprise, see *nhmc.geo*.

Miscellaneous

nhmc.ctrl.hashParams

Function

`nhmc.ctrl.hashParams([newParams[, destroyCurrent]])`

Some *modules* make use of the URL fragment identifier to direct users to a particular starting map view or other initial state. To streamline this process, the `nhmc.ctrl.hashParams` function returns an object that contains all values stored in the fragment identifier. Where possible, items are coerced to Number objects; Boolean values are also accepted in the form of the values "true" or "false", and keys without values are treated as `true`.

Assuming the default values for the properties of *nhmc.config.hashParams* and the fragment identifier `foo=bar|spam=false|magic|xyzzy=42`, calling `nhmc.ctrl.hashParams()` would return the following object:

```
{
    "foo": "bar",
    "spam": false,
    "baz": true,
    "xyzzy": 42
}
```

This function also may be used to *set* any or all values in the fragment identifier, and it accepts up to two arguments to that effect:

- `newParams` is an object containing keys and values in the same fashion as the object this function returns. This adds key-value pairs to the fragment identifier, overwriting any keys that already exist (using jQuery.extend).
- `destroyCurrent` is a Boolean (defaulting to `false`) that, if `true`, replaces all key-value pairs in the fragment identifier with those specified in `newParams` instead of simply adding/overwriting them as described above.

Manipulation

These functions are used for programmatically manipulating states and counties, which consists of clicking them or triggering *tooltips* related to them.

Lower-level functions

nhmc.ctrl.highlightArea *Function*

```
nhmc.ctrl.highlightArea(area, pageX, pageY)
```

This function highlights the `area` object, which is a state or county object from *nhmc.geo*, renders its *tooltip* if any exists and positions the tooltip at the coordinates `(pageX, pageY)` if those coordinates are specified. If those coordinates are not specified, the tooltip is positioned at the left edge of the map with its top edge one-third of the way down the map.

nhmc.ctrl.clickArea *Function*

```
nhmc.ctrl.clickArea(area)
```

This function triggers a click event on the `area` object, which is a state or county object from *nhmc.geo*.

> **Warning:** Because this uses the actual DOM node for `area`, this is not supported for browsers that do not support either SVG or VML rendering, such as Android 2.x.

Higher-level functions

nhmc.ctrl.dehighlightAreas *Function*

```
nhmc.ctrl.dehighlightAreas()
```

This function removes any highlights and tooltips created by *nhmc.ctrl.highlightArea*.

nhmc.ctrl.highlightCounty *Function*

```
nhmc.ctrl.highlightCounty(stateName, countyName, pageX, pageY)
```

This function highlights the county with the name `countyName` in the state with the name `stateName` and positions the tooltip (if applicable) at `(pageX, pageY)` if specified. For more details, see *nhmc.ctrl.highlightArea*.

nhmc.ctrl.clickCounty *Function*

```
nhmc.ctrl.clickCounty(stateName, countyName)
```

This function triggers a click event on the county with the name `countyName` in the state with the name `stateName`. This only makes sense for maps that show counties, such as the *nationwide county map* or the *statewide county maps*.

Warning: Because this uses the actual DOM node for the county, this is not supported for browsers that do not support either SVG or VML rendering, such as Android 2.x.

nhmc.ctrl.highlightState *Function*

```
nhmc.ctrl.highlightState(stateName, pageX, pageY)
```

This function highlights the state with the name `stateName` and positions the tooltip (if applicable) at `(pageX, pageY)` if specified. For more details, see *nhmc.ctrl.highlightArea*.

nhmc.ctrl.clickState *Function*

```
nhmc.ctrl.clickState(stateName)
```

This function triggers a click event on the state with the name `stateName`. This only makes sense for map views with state boundaries, which currently only includes the *nationwide state map*.

Warning: Because this uses the actual DOM node for the state, this is not supported for browsers that do not support either SVG or VML rendering, such as Android 2.x.

Orphans

nhmc.ctrl.renderLegend

Function

```
nhmc.ctrl.renderLegend(title, items)
```

This function was intended to simplify rendering of map legends by accepting two parameters:

- `title`: A string to be used as the title of the legend.
- `items`: An array of items to be used as the entries of the legend. Each item should be an array containing two elements:
  - The text of the entry.
  - A background color for that entry.

This function hasn't actually been used, though, since individual *modules* often render the legend in different ways.

Todo

Remove this.

1.2.6 nhmc.geo

This package contains the source data and the actual graphics objects for map views, including their constituent state or county path objects and any applicable *city labels*.

The contents of this package are primarily used by *nhmc.ctrl.zoomToState*, which switches between map views.

State and county path objects contain `nhmcData` attributes with identifying information that might be useful for event handlers interacting with those paths; more details about those objects' properties are provided below.

There are three general types of map view currently in use:

- The *nationwide county map* is a map containing every county and county-equivalent in the 50 United States and the District of Columbia. "County-equivalent" areas include boroughs in Alaska; parishes in Louisiana; independent cities in Maryland, Missouri, Nevada and Virginia; the District of Columbia; and census areas in Alaska.
- The *nationwide state map* is a map containing all 50 states and the District of Columbia.
- The *statewide county maps* are maps of each of the 50 states and the District of Columbia. Each state map (and the DC map) contains all of the county-equivalents in that state and can contain *city labels* for important metropolitan areas in that state.

Todo

The entire map view system should be refactored to work in a common way to avoid some of the complications resulting from having three different kinds of map. Part of a prototype implementation of this exists but still needs to be incorporated into the current version of the Map Center.

Nationwide county map

The nationwide county map consists of one basic type of object: county paths.

This map is identified by the view name (also referred to in some contexts as the "state" name) `us_counties`.

Each county and county-equivalent in this map is identified by its five-digit FIPS code as found in *nhmc.config.FIPSToCounty*. For example, the raw path string corresponding to Arlington County, Va., which has a FIPS code of `51013`, is located at `nhmc.geo.allCounties["51013"]`.

Important: If you intend to use this map view in a Map Center page, you must include `lib/maps/all_counties.js` in the document `<head>`. In theory, any other JavaScript file with the appropriate structure could be used, though, which could be useful for making modifications to this view for the purposes of a specific *module*. For example, a different version of this map view is included in the map of 2008 general election results to reflect the fact that Alaska results were reported by state House district instead of by county; that version of `all_counties.js` is available at `lib/map_center/modules/static_maps_data/08general-all_counties.js`.

nhmc.geo.allCounties

Object (values are strings)

This object has county FIPS codes as keys and raw SVG path strings as values. These strings theoretically could be included as the `d` attribute of an SVG `path` element and properly represent the counties' desired shape. (In fact, these path strings were taken directly from such attributes; all current Map Center views existed first as standard SVG images.)

nhmc.geo.countyGeo

Object (values are instances of dojox.gfx.path.Path)

This object has keys of county FIPS codes and values of Dojo path objects corresponding to the actual image components representing the counties. It is generated directly from the path strings found in *nhmc.geo.allCounties*.

Each county path object also includes an `nhmcData` object to help event handlers identify the county when needed. `nhmc.geo.countyGeo[foo].nhmcData` contains one property, `county_fips`, with a value of the county's FIPS code as found in *nhmc.config.FIPSToCounty*.

Nationwide state map

The nationwide state map consists of two basic types of object: state paths and labels for some smaller states.

This map is initially rendered directly by *nhmc.mapCenterInit* and is identified by the view name (also referred to in some contexts as the "state" name) `us_all`.

The objects making up the nationwide state map live in `nhmc.geo.usGeo` under the key corresponding to the state's full name as found in *nhmc.config.stateToUSPS*, represented as `foo` in the headings below. For example, the raw SVG path string for North Carolina is located at `nhmc.geo.usGeo["North Carolina"].state`.

Important: If you intend to use this map view in a Map Center page, you should include `lib/maps/states_only.js` in the document `<head>`. In theory, any other JavaScript file with the appropriate structure could be used, though, which could be useful for making modifications to this view for the purposes of a specific *module*.

---

nhmc.geo.usGeo[foo].state

String

This property is the raw SVG path string for the state `foo`; that is, this string theoretically could be included as the `d` attribute of an SVG `path` element and properly represent the state's desired shape. (In fact, these path strings were taken directly from such attributes; all current Map Center views existed first as standard SVG images.)

nhmc.geo.usGeo[foo].statePath

Object (instance of dojox.gfx.path.Path)

This property is a Dojo path object corresponding to the actual image component representing the state `foo`. It is generated directly from the path string found in *nhmc.geo.usGeo[foo].state*.

This object also includes an `nhmcData` object to help event handlers identify the state when needed. `nhmc.geo.usGeo[foo].statePath.nhmcData` contains one property, `state`, with a value of the state's full name as found in *nhmc.config.stateToUSPS*.

nhmc.geo.usGeo[foo].label

Object (instance of dojox.gfx.shape.Text)

This property is only used for a handful of East Coast states and is a Dojo text object corresponding to a label for that state.

Statewide county maps

Statewide county maps are an interesting case because there are so many of them; all maps other than the two mentioned above are considered statewide county maps.

Each statewide county map consists of two basic types of object: county paths and city labels.

Each map is identified by a view name (also referred to in some contexts as the "state" name) corresponding to the all-lowercase version of its state abbreviation as found in *nhmc.config.USPSToState*.

The objects making up each statewide county map live in `nhmc.geo.usGeo` under the key corresponding to the state's full name as found in *nhmc.config.stateToUSPS*, represented as `foo` in the headings below. For example, the Dojo group object corresponding to the city label for Jefferson City, Mo., is located at `nhmc.geo.usGeo["Missouri"].cityPaths["Jefferson City"]`.

In order to keep users from having to download all map views' path data at once whenever loading a Map Center page, *nhmc.ctrl.zoomToState* makes AJAX (technically JSONP) requests to obtain the data for a specific statewide county map the first time it is needed for a given pageview. The JSONP file for each map view is located at `http://s3.amazonaws.com/newshourroot/nhmc_geo_json/VIEW_NAME.json`, where `VIEW_NAME` is the view name mentioned earlier in this section.

Cities

nhmc.geo.usGeo[foo].cities *Object (values are arrays)*

This object has city names as keys and city location arrays as values. The format of those arrays is described more in detail in the documentation for *nhmc.ctrl.addCity*.

nhmc.geo.usGeo[foo].cityPaths *Object (values are instances of dojox.gfx.shape.Container)*

This object has city names as keys and Dojo group objects as values. Each of those groups contains the components of the labels for the cities defined in *nhmc.geo.usGeo[foo].cities*, which provides the source data used to generate this object.

Counties

nhmc.geo.usGeo[foo].counties *Object (values are strings)*

This object has full county names (as found in *nhmc.config.CountyToFIPS[foo]* as keys and raw SVG path strings as values. These strings theoretically could be included as the `d` attribute of an SVG `path` element and properly represent the counties' desired shape. (In fact, these path strings were taken directly from such attributes; all current Map Center views existed first as standard SVG images.)

For example, the path string corresponding to Dane County, Wis., is located at `nhmc.geo.usGeo["Wisconsin"].counties["Dane County"]`.

nhmc.geo.usGeo[foo].countyPaths *Object (values are instances of dojox.gfx.path.Path)*

This object has keys of full county names and values of Dojo path objects corresponding to the actual image components representing the counties. It is generated directly from the path strings found in *nhmc.geo.usGeo[foo].counties*.

Each county path object also includes an `nhmcData` object to help event handlers identify the county when needed. `nhmc.geo.usGeo[foo].counties[bar].nhmcData` contains one property, `county`, with a value of the county's full name as found in *nhmc.config.CountyToFIPS[foo]*.

1.2.7 nhmc.tooltips

This package is meant to provide a standard way to handle informational tooltips when users hover over (on desktop devices) or touch (on mobile devices) a state or county in the Map Center.

When a user performs either of those actions, the following things occur:

- The tooltip element `#tooltip` is rendered and added to the page by *nhmc.tooltips.render*.
- The tooltip element is positioned by *nhmc.tooltips.position*–normally to an area close to where the user touched or hovered over the state or county, but this can be overridden if desired.
- If the user is on a mobile device, the rest of the map no longer responds to tooltip-related events and waits for the user to manually close the tooltip. (The *module* developer's implementation of *nhmc.tooltips.render* should call *nhmc.tooltips.addClose* when rendering the tooltip element in order to provide the user with this opportunity.) Once the tooltip is closed, the aforementioned flow is able to start again.

If the user is on a desktop device, the tooltip position will update continously as the cursor moves within the same state or county, and the tooltip element will be removed when the cursor is no longer over that area.

Initialization

nhmc.tooltips.init

Function

`nhmc.tooltips.init()`

This function performs any initialization steps required by the tooltip system. This currently includes only calling *nhmc.tooltips.bindHover*.

Event handling

nhmc.tooltips.hoverHandlerTokens

Array

This is a standard JavaScript array that contains zero or more handles returned by dojo.connect. Dojo's event handling for graphics objects (and other items in general, but we only use it for graphics objects) is powerful in that it doesn't require DOM elements to exist as in jQuery–but it introduces a bit more complexity in that binding an event handler returns a handle that must be passed later to dojo.disconnect in order to unbind that handler.

The handles stored in this array correspond only to the event handlers required by the tooltip system. This array should be treated as an implementation detail of *nhmc.tooltips.bindHover* and *nhmc.tooltips.unbindHover* and probably should not be accessed by any other parts of the Map Center.

nhmc.tooltips.bindHover

Function

`nhmc.tooltips.bindHover()`

This function binds all event handlers needed for the interactions listed *above* to occur. This includes all checking for mobile devices in order to bind these event handlers correctly; of course, any *module*-specific handling of such devices (such as different positioning, different content or providing a close button) will have to perform those checks elsewhere as well.

nhmc.tooltips.unbindHover

Function

`nhmc.tooltips.unbindHover()`

This function shuts down the tooltip system by unbinding all event handlers bound by *nhmc.tooltips.bindHover*.

Positioning

nhmc.tooltips.position

Function

`nhmc.tooltips.position(e)`

This function is an event handler that sets the position of the tooltip element. By default, this function places the tooltip close to the user's mouse cursor (or touch location), but offset in the *y*-direction by *nhmc.tooltips.xOffset* and in the *x*-direction by *nhmc.tooltips.yOffset*. (Yes, this makes no sense. It's the same convention used by CSS Globe's tutorial, and it doesn't make sense there either.) If the tooltip would extend below the bottom of the document or to the right of the document's right edge, it is moved to the opposite side of the user's cursor (or touch location) in order to prevent this.

In certain cases, a *module* developer might want to override this position; for example, we do this for broadcast versions of just about all of our maps so the tooltip is always on the left side of the document. To do this, simply assign a new function to `nhmc.tooltips.position`. For example, since we set the position of `#tooltip` in this case using broadcast-specific stylesheets, we simply assign a function with an empty body:

```
nhmc.tooltips.position = function(e) {};
```

nhmc.tooltips.xOffset

Number

This property determines how many pixels away from the mouse cursor (or touch location) the tooltip should be positioned in the *y*-direction.

nhmc.tooltips.yOffset

Number

This property determines how many pixels away from the mouse cursor (or touch location) the tooltip should be positioned in the *x*-direction.

Rendering

nhmc.tooltips.render

Function

`nhmc.tooltips.render()`

This function creates the tooltip element `#tooltip`, appends it to the `<body>` and renders its content.

Important: This function is meant to be overridden by *module* developers. The value of `this` is the Dojo path object being hovered over (or touched). If you need to access information about that area, access the name or FIPS code with `this.nhmcData`; more information about the properties of that object are available in the *nhmc.geo* documentation.

Closing

nhmc.tooltips.addClose

Function

`nhmc.tooltips.addClose()`

This function is meant to be called from within *nhmc.tooltips.render* for touch devices as determined by Modernizr.touch. It provides a close button for users to remove the currently showing tooltip and rebind the tooltip system's event handlers.

nhmc.tooltips.destroy

Function

`nhmc.tooltips.destroy()`

This function removes the tooltip element.

1.3 Signal reference

Signals are implemented as jQuery custom event types triggered (and handled) on the `document` object. For example, firing the drawingComplete signal is accomplished with the following:

```
$(document).trigger('drawingComplete');
```

1.3.1 coreInitialized

This signal fires from *nhmc.mapCenterInit* when the Map Center core is fully initialized and the first (default) map view has finished being drawn.

1.3.2 drawingComplete

This signal fires any time a map view has finished being drawn.

1.3.3 mapDataRetrieved

This signal fires whenever the *non-primary election results* module has finished retrieving data from the server. No handlers for it currently exist, and no other modules currently fire it.

Todo

Evaluate this signal's usefulness and decide whether to keep it (and implement it elsewhere) or remove it.

1.4 Module reference

Individual Map Center pages often make use of additional functionality that isn't included in the core *nhmc* package; without such additional functionality, all that would appear is a blank map! The Map Center implements these extras in the form of "modules", which are simply JavaScript functions that are executed when the *coreInitialized* signal fires.

The general organization of a module consists of:

- Declaring which core *packages* are in use by calling the namespace function.
- Defining an handler for the `coreInitialized` signal. This signal should only fire once, but existing modules tend to bind to it using `$(document).one()` for additional safety.

  The body of that handler generally includes:

  - Any module-specific data to display or global-ish (within the scope of the handler) objects for intermediate storage of the module's state.
  - Configuration options.
  - Utility functions to handle different calculation or rendering tasks that will be needed.
  - Binding of *UI* event handlers.
  - Initialization of the module.

A simple stub module might look something like this:

```
namespace("nhmc");
namespace("nhmc.charts");
namespace("nhmc.cleanup");
namespace("nhmc.config");
namespace("nhmc.ctrl");
namespace("nhmc.geo");
namespace("nhmc.tooltips");
$(document).one('coreInitialized', function() {
    // Module body goes here.
});
```

Modules:

1.4.1 Electoral College results and calculator

The Electoral College calculator was the Map Center's first module, and it contains the Electoral College results of all presidential elections since 1964. Users can view the results of all of those elections and make predictions of the 2012 election.

This module lives in `lib/map_center/modules/electoral_college.js`.

Operation

This module has two different but related functionalities: a list of previous electoral results from elections since 1968 and a calculator allowing users to predict the electoral results of the 2012 election.

Historical results

The *tab* available to the user contains a dropdown list of all election years for which we have election results; selecting any of those options colors each state according to its electoral votes that year and shows the total number of electoral votes won by each candidate according to the number of electoral votes cast by each state in that year (including split votes and faithless electors where applicable).

Note: Because of the fact that the same module runs both this functionality and that of the calculator described in the next section, the module determines which functionality to use based on the presence of *tabs*. If any tab exists and contains a dropdown list (such as the one used for past years' results), it shows historical results; otherwise, it acts as a calculator instead.

Calculator

This allows users to predict electoral results for the 2012 election. Each state has a click handler bound to it that lets users change the party winning that state's electoral votes. How this works depends on the state:

- Maine and Nebraska use the Congressional District Method to allocate their electoral votes. In practice, then, this means at least three electoral votes in each of those states will go to one candidate, but any remaining electoral votes in that state could be assigned to any other candidate.

  To let users predict scenarios in which either or both of these states can split votes, clicking one of these states brings up a *dialog box* that lets users assign these votes.

- All other states and the District of Columbia award all of their electoral votes to the winner of the statewide popular vote, so clicking one of those areas toggles the winner in the following order: Republican, Democratic, tossup, no votes.

Changing any state's vote updates the vote totals in the sidebar, including marking one candidate or the other as winning if that candidate has obtained a majority of electoral votes.

Architecture

This module has a lot of moving parts, so they've been integrated into a small publish/subscribe system.

The pub/sub system lives in the `ecMap` object in the module source. That object has the following methods:

- `on` binds subscriber functions to published events. It takes the same arguments as jQuery.bind.

  The only event type is `change`, which fires whenever the map's status is changed. To subscribe to this event:

  ```
  ecMap.on("change", function(event, status) {
      // Handler body goes here. status is a status object, described below.
  });
  ```

- `off` unbinds subscriber functions. It takes the same arguments as jQuery.unbind.
- `set` takes one argument: a new status object (described below) with which to replace the map's entire internal status object.
- `reset` takes no arguments and removes all currently set electoral votes.
- `get` takes no arguments and returns a copy of the map's current status object.
- `modifyVotes` takes one argument: an object of changes to state electoral votes in order to modify totals without having to know the map's entire current state.

  For example, if you just wanted to add five Democratic electoral votes for Nebraska:

  ```
  ecMap.modifyVotes({
      "Nebraska": {
          dem: 5,
          rep: 0,
          toss: 0
      }
  });
  ```

The status objects that are ultimately used all throughout this system have the following properties:

- `stateVotes` is an object that contains the state-by-state electoral vote breakdown. The keys are full state names, and the values are objects with the keys `dem`, `rep` and `toss` and integer values with the electoral votes received by each political party in that state.
- `totals` is simply an object with the keys `dem`, `rep` and `toss` and integer values with the total electoral votes received by each political party.

Data structure

The data for this module is stored in two objects toward the top of the module source:

- `electoralVotes` is an object with election years as keys and objects as values. The object for each year contains:
  - `states` is an object with the number of electoral votes each state had in that election. The keys are full state names, and the values are numbers of votes.
  - `republican`, `democratic` and `tossup` are arrays containing the names of states whose full electoral votes went to each of those respective parties in that election.

    Note: Throughout this module, `tossup` is used to refer both to predictions that keep states in play for either party and to third-party candidates receiving Electoral College votes such as George Wallace in 1968.

  - States that did not award their full electoral votes to any party in that election are not listed in the `republican`, `democratic` and `tossup` arrays; instead, they have their own objects in the election data (using their full state names as keys) containing their electoral vote breakdowns by party. For example, since Nebraska split its vote in 2008:

```
electoralVotes["2008"]["Nebraska"] = {
    "republican": 4,
    "democratic": 1,
    "tossup": 0
}
```

Instances of faithless electors are also represented this way. For example, since an elector from the District of Columbia left her 2000 ballot blank in protest:

```
electoralVotes["2000"]["District of Columbia"] = {
    "republican": 0,
    "democratic": 2,
    "tossup": 0
}
```

- `candidateNames` is an object with election years as keys and objects as values. The object for each year contains the names of the parties' candidates participating in that election, using the same `democratic`, `republican` and `tossup` (where applicable) key names as before and string values with each candidate's last name. For example, for the 2012 election:

```
candidateNames["2012"] = {
    "democratic": "Obama",
    "republican": "Romney"
}
```

1.4.2 Incumbent governors' election results

This module allows users to view the results of the elections that put each state's incumbent governor in office.

This module lives in `lib/map_center/modules/governors.js`.

Note: This currently only includes data from Wisconsin due to deadline pressures involved with that state's recall election. The other states will be added later.

Todo

Add support for viewing a *nationwide state map* with incumbent governors' party affiliations and support for clicking a state to view that state's election results.

Operation

The one *tab* available in this module allows users to select a state in order to view the county-level results of the most recent election that elected that state's incumbent governor (i.e., not including recall efforts that governor survived).

The *sidebar* in this module shows the statewide vote breakdown in the race, and the *tooltips* for each county shows the countywide vote breakdown in the race.

Configuration options

All configuration options live in the `config` object toward the top of the module source and may be overridden by defining a global object named `nhmcGovernorsConfig`, which will be merged into `config` using jQuery.extend.

- `bigCandidates` is an integer defining the number of candidates in the sidebar that will have large legend entries including their photo and large vote percents as opposed to smaller, more compact legend entries.
- `partyColors` is an object with party abbreviation letters (`D`, `R` or `O`) as keys and hexadecimal color codes as values.
- `candidateImages` is an object with candidate names as keys and headshot image URLs as values.
- `flyoutsEnabled` is a Boolean value stating whether tooltips should be rendered as *flyouts*, which is only done for broadcast.
- `strokeHighlight` is a hexadecimal color code used when config.flyoutsEnabled is true; when that option is enabled, this color is used to outline the county targeted by the tooltip.
- `tooltipsEnabled` is a Boolean value stating whether tooltips with countywide vote totals should be enabled at all.

Data structure

The data driving this module is located in an object called `resultsData` toward the top of the module source. It has capitalized state abbreviations as found in *nhmc.config.USPSToState* as keys and objects as values. In each state's object:

- `areas` is an object with *county FIPS codes* as keys and arrays as values. Each array contains one two-element array per candidate: The first element is the candidate ID listed in `candidates`, and the second element is the number of votes that candidate received. The county's array is sorted by the number of votes received in descending order; that is, the candidate with the most votes in that county is listed first.
- `breakdown` is an array identical in format to the county arrays in `areas` but with results for the entire state.
- `candidates` is an object with candidate IDs as keys and candidate names as values. This allows the data in `areas`, `breakdown` and `parties` to use the more compact candidate IDs instead of candidate names to save space.

  Note: All configuration options use candidates' names, not their IDs. IDs should not be used with any other modules; they are arbitrarily assigned to candidates.

- `electionYear` is a string with the year the election described took place.
- `parties` is an object with candidate IDs as keys and party abbreviation letters (corresponding to the keys in config.partyColors) as values.

Fragment identifier

This module makes use of one key-value pair in the fragment identifier via *nhmc.ctrl.hashParams*:

- `map_view`, if provided, should be set to the view name of the state to display first. The fragment identifier will update as the map view is changed.

1.4.3 Republican presidential primary results

This module allows users to view the results of 2012 Republican presidential primaries and caucuses by state.

This module lives in `lib/map_center/modules/live.js`.

Todo

Add support for viewing a *nationwide state map* of state winners such as the one currently provided by the *static maps module* that would also allow users to click a state to see the results of that state's primary or caucuses.

Operation

The one *tab* available in this module allows users to select a state in order to view the county-level results of that state's 2012 Republican presidential primary or caucus. (Some states did not report county-level caucus results and only reported statewide results; for those states, statewide maps as described in *nhmc.config* are used instead.)

The *sidebar* in this module shows the statewide vote breakdown in the race, and the *tooltips* for each county shows the countywide vote breakdown in the race.

If config.autoRefresh is `true`, the map will request and display updated results every config.autoRefreshDelay milliseconds. This does not require a full page refresh.

Results in this module are provided by the Associated Press' AP Election Services.

Configuration options

All configuration options live in the `config` object toward the top of the module source and may be overridden by defining a global object named `nhmc_live_config`, which will be merged into `config` using jQuery.extend.

- `autoRefresh` is a Boolean value stating whether the map should periodically request and display updated results. This is useful on election night to keep users from having to manually refresh the page to view updated results.
- `autoRefreshDelay` is a number stating the number of milliseconds to wait between requests for updated results. This is ignored if config.autoRefresh is `false`.
- `bigCandidates` is an integer defining the number of candidates in the sidebar that will have large legend entries including their photo and large vote percents as opposed to smaller, more compact legend entries.
- `blankMap` is a Boolean value stating whether to force the map to have all counties uncolored. This was used in one broadcast segment.
- `candidateColors` is an object with candidate names as keys and hexadecimal color codes as values for colors to represent them on the map.
- `candidateImages` is an object with candidate names as keys and headshot image URLs as values.
- `condenseCandidates` is a Boolean value stating whether to restrict the list of candidates to those named in `showCandidates` and group all others under an "Other" entry.
- `flyouts` is a value stating whether tooltips should be rendered as *flyouts*, which is only done for broadcast. Originally it was intended to be a string value setting a position for the flyout (or `null` if flyouts should not be used), but this value currently is only checked for truthiness. Positioning is entirely handled in CSS.
- `flyoutWidth` is a number stating the number of pixels wide to render tooltips if they are rendered as *flyouts*. (This was here in order to support certain animations.) Positioning and formatting are entirely handled in CSS now, so this option should be considered deprecated.
- `showCandidates` is an array of candidate names to show when config.condenseCandidates is true.
- `strokeHighlight` is a hexadecimal color code used when *flyouts* are enabled; when this is the case, this color is used to outline the county targeted by the tooltip.
- `tooltipsEnabled` is a Boolean value stating whether tooltips with countywide vote totals should be enabled at all.

Data structure

The results data for each state lives in a static JSONP file at `http://www.pbs.org/newshour/vote2012/map/live_data/V` where `VIEW_NAME` is the state's abbreviation as found in *nhmc.config.stateToUSPS*, lowercased. The file's callback name is the same abbreviation, but the name there is uppercased; this was done to avoid conflicts with the lowercase abbreviation `in` for the state of Indiana, which happens to be a reserved word in JavaScript.

Each state's results object contains the following properties:

- `areas` is an object with area IDs as keys and objects with area results as values.

  Note: For the purposes of this module, an "area ID" is generally a *county FIPS code*; for non-county areas (such as states), the full name of the area is used instead.

  Each area's results object contains the following properties:

  - `data` is an array that contains one two-element array per candidate: The first element is the candidate ID listed in `candidates`, and the second element is the number of votes that candidate received. The area's array is sorted by the number of votes received in descending order; that is, the candidate with the most votes in that area is listed first.
  - `precincts` is an array containing two elements:
    - The first is the number of precincts reporting in the area.
    - The second is the total number of precincts in the area.

    These may be used to calculate the percent of precincts reporting.

    > **Warning:** Rounding errors become significant once more than 99 percent of precincts are reporting; module developers should ensure they do not erroneously report 100 percent of precincts reporting before that is actually the case. (This is primarily an issue when rounding to the nearest integer.)

- `breakdown` is an array identical in format to the county `data` objects in `areas` but with results for the entire state.
- `candidates` is an object with candidate IDs as keys and candidate names as values. This allows the data in `areas`, `breakdown`, `colors`, `images` and `winners` to use the more compact candidate IDs instead of candidate names to save space.

  Note: All configuration options use candidates' names, not their IDs. IDs should not be used with any other modules; AP Election Services arbitrarily assigns them to candidates and may use different IDs from state to state for the same candidate.

- `colors` is an object with candidate IDs as keys and hexadecimal color codes as values. This is deprecated due to the introduction of the config.candidateColors option.
- `images` is an object with candidate IDs as keys and headshot image URLs as values. This is deprecated due to the introduction of the config.candidateImages option.
- `lastUpdated` is an array containing five elements describing the date and time (in the Eastern time zone, including Daylight Saving Time if in effect) when this results file was last updated.
- `precincts` is an array containing two elements:
  - The first is the number of precincts reporting statewide.
  - The second is the total number of precincts statewide.

These may be used to calculate the percent of precincts reporting.

> **Warning:** Rounding errors become significant once more than 99 percent of precincts are reporting; module developers should ensure they do not erroneously report 100 percent of precincts reporting before that is actually the case. (This is primarily an issue when rounding to the nearest integer.)

- `test` is a Boolean value stating whether AP Election Services marked any of the data used to generate this results file as being test data not intended for publication.
- `winners` is an object with area IDs as keys. For each area ID, the value is either the candidate ID for the winner projected by AP or `null` if no such projection has yet occurred.

Fragment identifier

This module makes use of one key-value pair in the fragment identifier via *nhmc.ctrl.hashParams*:

- `map_view`, if provided, should be set to the view name of the state to display first. The fragment identifier will update as the map view is changed.

1.4.4 Non-primary election results

This module allows users to view the results of elections other than Republican presidential primaries.

This module lives in `lib/map_center/modules/other_votes.js`.

Operation

Two *tabs* are available to the user; one allows the user to select a state for which to view results, and the other allows the user to select a specific race in that state.

The *sidebar* in this module shows the statewide vote breakdown in the race, and the *tooltips* for each county shows the countywide vote breakdown in the race.

If config.autoRefresh is `true`, the map will request and display updated results every config.autoRefreshDelay milliseconds. This does not require a full page refresh.

Results in this module are provided by the Associated Press' AP Election Services.

Configuration options

All configuration options live in the `config` object toward the top of the module source and may be overridden by defining a global object named `nhmc_live_config`, which will be merged into `config` using jQuery.extend.

- `autoRefresh` is a Boolean value stating whether the map should periodically request and display updated results. This is useful on election night to keep users from having to manually refresh the page to view updated results.
- `autoRefreshDelay` is a number stating the number of milliseconds to wait between requests for updated results. This is ignored if config.autoRefresh is `false`.
- `bigCandidates` is an integer defining the number of candidates in the sidebar that will have large legend entries including their photo and large vote percents as opposed to smaller, more compact legend entries.
- `blankMap` is a Boolean value stating whether to force the map to have all counties uncolored. This was used in one broadcast segment that used *another module*, and this functionality still exists in this one since it was based on that other module.

- `candidateColors` is an object with candidate names as keys and hexadecimal color codes as values for colors to represent them on the map.
- `candidateImages` is an object with candidate names as keys and headshot image URLs as values.
- `condenseCandidates` is a Boolean value stating whether to restrict the list of candidates to those named in `showCandidates` and group all others under an "Other" entry.
- `defaultRaceNames` is an empty object that is completely unused. It has no mention anywhere in the module source except in the initial definition of the `config` object.

Todo

Remove this.

- `flyoutsEnabled` is a Boolean value stating whether tooltips should be rendered as *flyouts*, which is only done for broadcast.
- `friendlyRaceNames` is an object with race names as listed in the *state results file* as keys and race names as the *module* developer would prefer them to be displayed as values. These should be short in order to fit within the space available in the *race selection tab*.
- `randomColors` is an array of hexadecimal color codes from which candidates' colors will be selected for counties they're winning and their sidebar legend entries. A candidate's color is determined first by the candidate-specific color listed in config.candidateColors if one exists or by the first unused color in `config.randomColors` otherwise.
- `showCandidates` is an array of candidate names to show when config.condenseCandidates is true.
- `showRaces` is an array of names of races that are allowed to be displayed. If this is empty (as it is by default), all available races may be displayed. The names in this array should be the race names as provided in the *state results file*, *not* the alternative race names potentially defined in config.friendlyRaceNames.
- `strokeHighlight` is a hexadecimal color code used when config.flyoutsEnabled is true; when that option is enabled, this color is used to outline the county targeted by the tooltip.
- `tooltipsEnabled` is a Boolean value stating whether tooltips with countywide vote totals should be enabled at all.

Data structure

The results data for each state lives in a static JSONP file at `http://www.pbs.org/newshour/vote2012/map/live_data/V` where `VIEW_NAME` is the state's abbreviation as found in *nhmc.config.stateToUSPS*, lowercased. The file's callback name is the same abbreviation, but the name there is uppercased; this was done to avoid conflicts with the lowercase abbreviation `in` for the state of Indiana, which happens to be a reserved word in JavaScript.

Each state's results object contains the following properties:

- `candidates` is an object with candidate IDs as keys and candidate names as values. This allows the data in `areas`, `breakdown`, `colors`, `images` and `winners` to use the more compact candidate IDs instead of candidate names to save space.

Note: All configuration options use candidates' names, not their IDs. IDs should not be used with any other modules; AP Election Services arbitrarily assigns them to candidates and may use different IDs from state to state for the same candidate.

- `lastUpdated` is an array containing five elements describing the date and time (in the Eastern time zone, including Daylight Saving Time if in effect) when this results file was last updated.

- `raceNames` is an object with race IDs as keys and race names as values. This allows the data in `racees` to use the more compact race IDs instead of race names to save space.

  Note: All configuration options use races' names, not their IDs. IDs should not be used with any other modules; AP Election Services arbitrarily assigns them to races and may use different IDs from state to state for races for the same (i.e., federal) position.

- `races` is an object with race IDs as keys and race results objects as values. Each race results object has the following properties:
  - `areas` is an object with area IDs as keys and objects with area results as values.

    Note: For the purposes of this module, an "area ID" is generally a *county FIPS code*; for non-county areas (such as states), the full name of the area is used instead.

    Each area's results object contains the following properties:

    * `data` is an array that contains one two-element array per candidate: The first element is the candidate ID listed in `candidates`, and the second element is the number of votes that candidate received. The area's array is sorted by the number of votes received in descending order; that is, the candidate with the most votes in that area is listed first.
    * `precincts` is an array containing two elements:
      · The first is the number of precincts reporting in the area.
      · The second is the total number of precincts in the area.

      These may be used to calculate the percent of precincts reporting.

      Warning: Rounding errors become significant once more than 99 percent of precincts are reporting; module developers should ensure they do not erroneously report 100 percent of precincts reporting before that is actually the case. (This is primarily an issue when rounding to the nearest integer.)

  - `breakdown` is an array identical in format to the county `data` objects in `areas` but with results for the entire state.
  - `precincts` is an array containing two elements:
    * The first is the number of precincts reporting statewide.
    * The second is the total number of precincts statewide.

    These may be used to calculate the percent of precincts reporting.

    Warning: Rounding errors become significant once more than 99 percent of precincts are reporting; module developers should ensure they do not erroneously report 100 percent of precincts reporting before that is actually the case. (This is primarily an issue when rounding to the nearest integer.)

  - `winners` is an object with area IDs as keys. For each area ID, the value is either the candidate ID for the winner projected by AP or `null` if no such projection has yet occurred.
- `test` is a Boolean value stating whether AP Election Services marked any of the data used to generate this results file as being test data not intended for publication.

Todo

Add some support for specifying the location of race-specific results files to allow for viewing archived results from past races in states that have held other races since then.

Fragment identifier

This module makes use of one key-value pair in the fragment identifier via *nhmc.ctrl.hashParams*:

- `map_view`, if provided, should be set to the view name of the state to display first. The fragment identifier will update as the map view is changed.

1.4.5 General-purpose static data mapping

This module drives all data-driven choropleth maps not covered under any of the other Map Center *modules*. It is designed to display category-based and continuous value-based data for any of the available *map types*, and it includes a variety of useful *hooks and overrides* in order to be more useful for creating variations on these types of maps relatively quickly.

This module lives in `lib/map_center/modules/static_maps.js`, and the current data sets used with it live in `lib/map_center/modules/static_maps_data/` by convention.

Operation

A page making use of this module must include one additional JavaScript file containing the data to display. As described above, this file should live in `lib/map_center/modules/static_maps_data/`. That file should define one global variable, `nhmcStatic`, with a format described *below*.

Depending on whether the page is intended to show one or multiple data sets, the user may have one or two *tabs* available; the one that is always available allows the user to select a specific *map view*, and the one that is available in the case of multiple data sets allows the user to select a specific data set to view.

In either case, the states or counties on the map are colored according to the data provided. The *sidebar* in this module shows the meaning of each color used on the map, and the *tooltips* for each state or county shows the specific data value that caused that state or county to receive the color it has.

Configuration options

There are three major configuration options that exist outside the *data file itself*, all changeable via global variables:

- `nhmcStaticBreakFormatter` is a function used for rendering the *sidebar* entries for each color shown on the map when the data set shown uses *continuous values*. It accepts six arguments:
  - `thisBreak` is the break value for this particular sidebar entry.
  - `prevBreak` is either `null` or the break value for the sidebar entry before this one.
  - `isLastBreak` lets the formatter function know this is the last break value to be included in the sidebar.
  - `breakPrefix` is a string meant to be displayed before a break value. This often is used for currency signs.
  - `breakSuffix` is a string meant to be displayed after a break value. This often is used for units (points, percent, etc.).
  - `breakDecimals` is an integer requesting that a break value be rounded to a specific number of decimal places; for example, small dollar amounts might have `breakDecimals` set to `2` to properly show a number of cents.

By default, this function renders the sidebar as a list of ranges from one break to the next. This function will execute every time the map view is changed.

- `nhmcStaticFlyouts` is an object that determines whether *tooltips* should be rendered as *flyouts*. No specific properties need to be defined in this object for flyouts to be enabled; the object just needs to exist. By convention, though, to enable flyouts, the following value is used:

```
var nhmcStaticFlyouts = {
    enabled: true
};
```

  Other properties that get merged with those provided in this object are:

  - `corner` was intended to be a string value setting a position for the flyout (or `null` if flyouts should not be used), but now positioning is entirely handled in CSS, so this option should be considered deprecated.
  - `strokeHighlight` is a hexadecimal color code for the color used to outline the state or county targeted by the tooltip.
  - `width` is a number stating the number of pixels wide to render the flyout. (This was here in order to support certain animations.) Positioning and formatting are entirely handled in CSS now, so this option should be considered deprecated.

- `nhmcStaticTooltipFormatter` is a function used for rendering the content of the *tooltips* corresponding to each state or county on the map. It accepts five arguments:

  - `thisFIPS` is a *county FIPS code* if the tooltip should show data for a county or an empty string if it should show data for a state.
  - `thisState` is a full state name for the area for which the tooltip should show data. This is included whether the area is a county or a state (since counties *do* exist within states); therefore, `nhmcStaticTooltipFormatter` should check the values of the `thisFIPS` and `thisCounty` arguments to determine the type of area being touched or hovered over.
  - `thisCounty` is a full county name if the tooltip should show data for a county or an empty string if it should show data for a state.
  - `countyOnly` is a Boolean value that, if true, requests that the tooltip show just the name of the county being touched or hovered over as opposed to the name of the county *and* the name of that county's state (e.g., `Arlington County` or `Arlington County, Virginia` if `countyOnly` is `true` or `false` respectively).
  - `currentData` is the full data set being displayed (as described in *Data structure* below).

  This function will execute every time a tooltip is created, but not while that tooltip simply updates its position within the same state or county.

One additional configuration option exists:

- `nhmcStaticDataIndex` is an integer with the same interpretation as the *data_index* parameter in the *fragment identifier*. It will be ignored if the *data_index* fragment parameter is set.

Data structure

Two types of data sets can be displayed in a map using this module:

- *Category-based* data sets place states and counties in any of a number of categories and color areas that are in the same category with the same color.
- *Continuous value-based* data set assign a value to each state and/or county and color those areas based on which of a number of ranges of values happen to include the areas' values.

As described in *Operation* above, any page using this module must include a JavaScript file (or `<script>` block) that defines a variable named `nhmcStatic`. For a page that uses only one data set, `nhmcStatic` should be an object with the properties listed in either *Categories* or *Continuous values* below; for a page that uses multiple data sets, `nhmcStatic` should be an array of such objects.

Note: In both of the description sections below, the term "area name" refers to *FIPS codes* for counties and full state names for states.

Categories

A data set object for category-based data must contain the following three properties:

- `categories` is an array of strings containing the names of the categories to use in the data set, listed in the order they should be displayed in the sidebar.
- `colors` is an object with category names (as listed in `categories`) as keys and hexadecimal color codes as values. These colors will be used to fill areas that fall in each category and will be shown next to each category's name in the sidebar.
- `areaLists` is an object with category names (as listed in `categories`) as keys and arrays of area names as values. Each array should contain the names of all states and the FIPS codes of all counties that fall in that category. Any states or counties that are unlisted in all area lists will remain their default fill color as defined in *nhmc.config.defaultAttributes*.

See Also:

For an example of a category-based data set, check out the Patchwork Nation categories map. The page source is located at `patchwork_types.html`, and the data set source is located at `lib/map_center/modules/static_maps_data/patchwork_types.js`.

Continuous values

A data set object for continuous value-based data must contain the following three properties:

- `breaks` is an array of numbers that represent the greatest value in each group of areas that should be colored identically. The last value in `breaks` should be at least as great as the greatest value in the data set and may be greater. (The maximum possible value is a good default, such as 100 for percentages.)

  For example, if you are mapping a data set with values ranging from 0 to 100 and want to split it up into five equally spaced groups of 20 each:

  ```
  "breaks": [20, 40, 60, 80, 100]
  ```

  This will define groups of:

  - 0 to 20 (rendered by default in the sidebar as `20`)
  - 20 to 40
  - 40 to 60
  - 60 to 80
  - 80 to 100 (rendered by default in the sidebar as `>80`)

- `colors` is an array of hexadecimal color codes in the same order as in `breaks`. Continuing the previous example:

```
"colors": ["#000000", "#ffffff", "#ff0000", "#00ff00", "#0000ff"]
```

would assign the color `#ff0000` to the `40 to 60` group.

- `areas` is an object with area names as keys and data points (numbers that fall in the ranges defined in `breaks`) as values. Continuing the previous example:

```
"areas": {
    "Missouri": 75
}
```

would fill the state of Missouri with `#00ff00` (since it falls in the `60 to 80` range).

It also may contain any combination of the following five properties, including none of them or all of them:

- `decimalPlaces` is a number of decimal places to which the breaks and data values should be rounded when they're displayed. This defaults to zero, which rounds everything to the nearest integer.
- `prefix` is a string (defaulting to the empty string) that is prepended to the breaks and data values when they're displayed. This is primarily used for currency indicators such as the `$` before dollar amounts.
- `suffix` is a string (defaulting to the empty string) that is appended to the breaks and data values when they're displayed. This is primarily used for units, including percents.
- `seriesName` is a string (defaulting to `undefined`) which, if defined, is used to populate any element with the class `static_map_name`.
- `sidebarTitle` is a string (defaulting to `undefined`) which, if defined, is used to populate the element with the ID `sidebar_title`.

See Also:

For an example of a continuous value-based data set, check out the unemployment rate map. The page source is located at `unemployment.html`, and the data set source is located at `lib/map_center/modules/static_maps_data/unemployment.js`.

Fragment identifier

This module makes use of two key-value pairs in the fragment identifier via *nhmc.ctrl.hashParams*:

- `data_index`, if provided, should be set to the index of the data set to display first. This only makes sense if *nhmcStatic* is defined as an array of objects.
- `map_view`, if provided, should be set to the name of the map view to display first. The fragment identifier will update as the map view is changed.

1.5 User interface

The Map Center includes a variety of user interface items that aim to keep the user experience relatively consistent across pages and across modules. Some of these are missing in *embedded maps*, but other pages should include these as needed.

We use jQuery UI's Overcast theme as a base for items that make use of jQuery UI; right now, that just includes *dialog boxes*.

1.5.1 Broadcast maps

We keep broadcast-ready versions of all of our maps that omit several of the page features that normally surround Map Center pages, such as site navigation and featured politics stories. The main markup parts of these maps (everything inside `#view_info` and `#content_area`) should almost always be the same, though. Usually there are some additional styles applied to change text sizes and/or reposition elements to make them more readable on air.

1.5.2 Dialog boxes

These are standard jQuery UI dialog boxes. When creating these, be sure to add them to *nhmc.cleanup.futureGarbage* and *nhmc.cleanup.activeDialogs* so the appropriate cleanup routines know about them for later.

1.5.3 Map page selection

Toward the top of each Map Center page is a `#nav_outermost` element that contains all of the containers needed for the user to select the page they want to view. This is handled outside of the Map Center core by the standalone JavaScript file `lib/map_center/navigation.js`, which takes care of all of the event bindings and populates the navigation's `#nav_container` with an option for every page listed in that JS file's `navObjects` array. Each object in `navObjects` should contain four properties:

- `featured` is a Boolean value that, if `true`, will color the navigation option differently as a way to make it stand out. This should only be `true` for one item at a time.
- `title` is a string with the text of the navigation option.
- `href` is a string with the filename of the page to which the navigation option should link. (This will automatically have `-broadcast` inserted before the file extension when the navigation detects that the user is viewing a *broadcast map*.)
- `id` is a string that should be short and is added to the navigation option's ID in order to give each one a unique identifier for CSS purposes. Each option will have the ID `nav_option_ID`, where `ID` is the value of this property.

This navigation uses the Roto library, which works great but raises errors from time to time; we keep it collapsed by default, but it doesn't expect to ever be hidden. Errors raised by this library can be safely ignored.

1.5.4 Sidebars

All pages (so far) have extra information (most commonly map legends) in the `#sidebar` element. This element contains three important child elements:

- `#sidebar_title` is a header with the title to be displayed in the sidebar.
- `#legend` is the main content container for the sidebar.
- `#map_view` is a hidden `<input>` that keeps track of the current *map view* that is being displayed. This could be anywhere in theory, but this seemed like a reasonable place to put it.

1.5.5 Tabs

Tabs live in the `#view_tabs` element and are each `<div>` elements with the class `view_tab`. The tab marked as active on the page also should have the class `view_tab_active`; this will be moved as necessary by the event bindings in *nhmc.mapCenterInit*.

Each tab at least should contain an `<a>` element with the class `view_tab_option` containing the tab's text. It is up to the *module* developer to add module-specific handlers for these options, including parsing their text or `href` attributes as needed.

Tabs with the classes `view_tab_more` and `view_tab_options_more`, if provided, can include child dropdown menus, which are implemented as `<ul>` elements (called `#view_tab_more_menu` and `#view_tab_options_more_menu`, respectively) that contain more of these `a.view_tab_option` elements; when these are clicked, their text and `href` attributes replace those of the tab's top-level `.view_tab_option` element (i.e., above the `<ul>`). For an example of both of these tabs, see `ethnicity.html`; for an example of tabs without dropdown menus, see `calc.html`.

1.5.6 Tooltips

These informational boxes appear next to the user's cursor (if on a desktop device) or touch location (if on a mobile device) in certain *modules* to show details about the area touched or hovered over.

See Also:

See the *tooltip package documentation* for details.

Flyouts

Flyouts are a special case of *tooltip* intended specifically for *broadcast maps*. They are positioned on the left side of the screen and tend to be vertically centered. These allow for the tooltip information to be presented without actually covering up the part of the map with which the people on air are interacting.

When flyouts are rendered, they should add a stroke to the area being touched or hovered over; when they are destroyed, they should remove that stroke.

1.6 Embeddable maps

There is some support for embedding certain Map Center modules (currently *static maps*, *past primary results* for which tabulation is complete and the *Electoral College calculator*) in other Web pages, such as those of PBS member stations.

Embeds are handled through two main files:

- A basic PHP script serves markup for a particular map to the user depending on its query string arguments.
- A JavaScript file provides support for creating this query string, calculating the height of the embed and generating the `<iframe>` element that will hold the embedded map.

1.6.1 Server side

The server side of this is a PHP script (located at `embed/embed.php`) that just writes slightly different versions of its content depending on the query string arguments provided. It otherwise contains most of the same elements as the normal map pages (except for less useful ones such as the *navigation* and *tabs*) and is designed to be responsive using the styles in `embed/embed.css`.

The script checks that the arguments fall into the set of allowed values to avoid weird exploits, but this means adding a new module's functionality to this takes quite a bit of hand-holding. Ask if you need help.

1.6.2 Client side

`embed/nhmc_embed.js` defines the `embedNHMC` function, which embed codes call to create the actual `<iframe>` element for the map.

It calculates the height of the element (using lots of our trial and error) in the `calcHeight` function, which should be tweaked as necessary. (Being able to change this and affect all embeds is probably the main reason Map Center embeds are structured this way, honestly.)

It also generates an ID unique to the embed's contents in case the `<iframe>` element needs to be styled in any way. This is currently unused, but it could be a useful support tool later on for sites that embed this.

`embedNHMC` appends the embedded map by default, but it can be made to just return the `<iframe>` markup, which can be useful for previews such as in `embed/index.html`. (See the source of `embed/nhmc_embed.js` for details.)