Embedding the Postmile Query Tool

Table of Contents

Table of Figures

List of Tables

Embedding the Postmile Query Tool

Introduction

The Postmile Query Tool is an interactive map interface that enables a user to query the Postmile Web Service. The user can:

- Validate that a postmile specification identifies a recognized postmile;
- Determine the geo-coordinates associated with a postmile;
- Determine the State odometer corresponding to a postmile;
- Determine the recognized postmile that is geographically closest to a given set of geo-coordinates;
- Validate a pair of postmiles representing a segment of a roadway; and
- Determine the route segment whose endpoints are closest to a pair of geo-coordinates.

Validated postmiles and route segments are visually displayed on a map, and the user can specify geo-coordinates by dragging markers on the map.

This style of interaction with the Postmile Web Service can be a useful adjunct in a web application that requires capture of postmile, state odometer or geo-coordinate data. However, as a standalone website, using the Postmile Query Tool is somewhat awkward, as there is no easy way to pass information back and forth between the tool and the web application, apart from manual cut-and-paste. It would be much better for the user if the tool were *integrated* with the application.

This document describes how the Postmile Query Tool can be integrated into any web application built on any platform.

Integration

There are three aspects to integration:

1. The application must be able to invoke the tool programmatically, and show it on a page of the application;
2. The application should be able to initialize the tool (with, for example, a specific postmile to be validated, a specific map region to show, etc.) and configure it (specify which mode the tool should start out in);
3. The application should be able to obtain information from the tool (what postmile has been validated, what geo-coordinates the user specified, etc.)

Invoking the Postmile Query Tool

To place the Postmile Query Tool on your web page, you must do the following:

1. Load the Postmile Query Tool API using a script tag:

```
<script src="https://postmile.dot.ca.gov/PMQT/scripts/PMQTEmbed.js"
type="text/javascript" ></script>
```

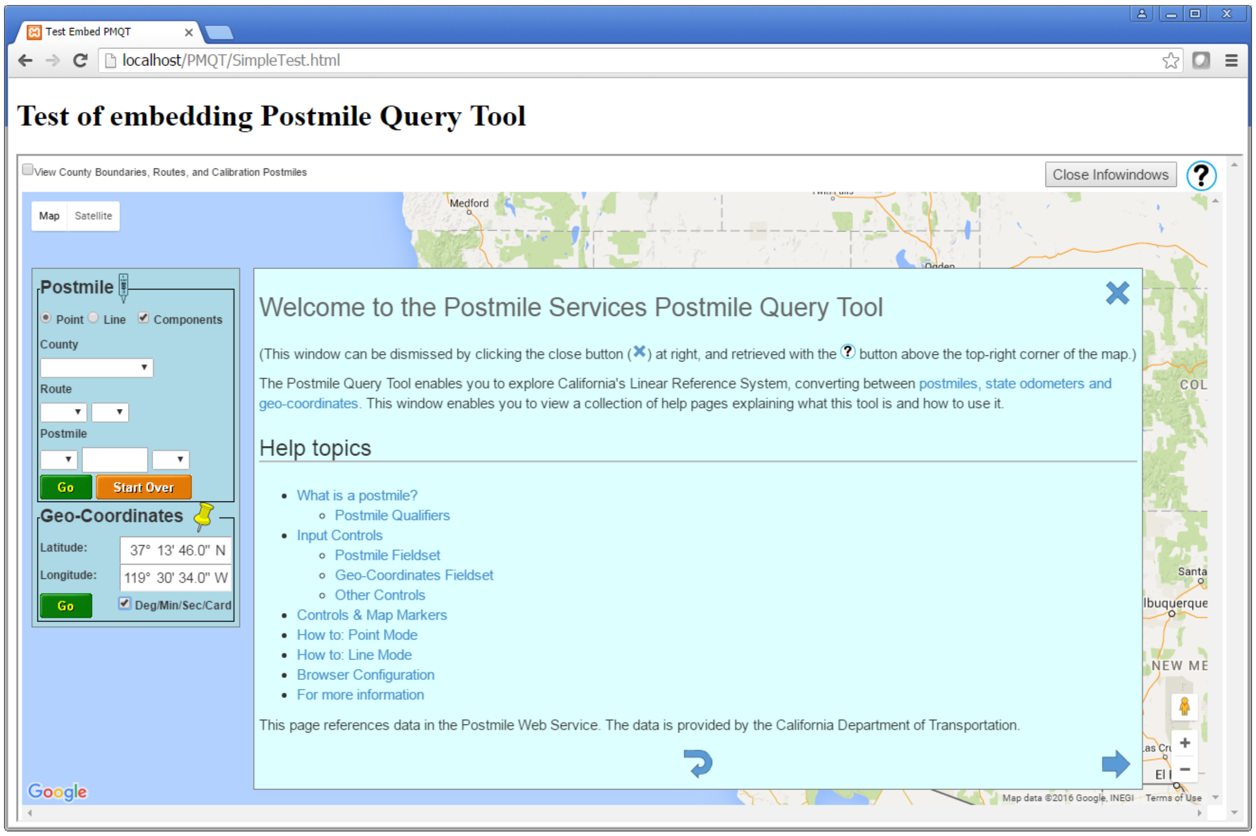
Figure 1. <script> tag to invoke the Postmile Query Tool.

2. Include a div with an id to hold the tool in your html layout;
   (e.g., `<div id="postmileTool"></div>`)
3. Invoke the javascript function `ct.PMQT.addPMQT`, passing it the name of the div that should hold the tool. For example, you can invoke it as part of the onload attribute for the body of your page: `<body onload="ct.PMQT.addPMQT('postmileTool')">`

N.B.: The ct.PMQT.addPMQT function should only be called **ONCE** for any html page.

This puts the Postmile Query Tool on your page in the div that you specified. (N.B.: the page should be an HTML5 page, declared using the <!DOCTYPE html> tag.)

For example, the page shown in Figure 2 would be created by the HTML code in Figure 3.



Test of embedding Postmile Query Tool

Welcome to the Postmile Services Postmile Query Tool

(This window can be dismissed by clicking the close button (✖) at right, and retrieved with the ? button above the top-right corner of the map.)

The Postmile Query Tool enables you to explore California's Linear Reference System, converting between postmiles, state odometers and geo-coordinates. This window enables you to view a collection of help pages explaining what this tool is and how to use it.

Help topics

- What is a postmile?
  - Postmile Qualifiers
- Input Controls
  - Postmile Fieldset
  - Geo-Coordinates Fieldset
  - Other Controls
- Controls & Map Markers
- How to: Point Mode
- How to: Line Mode
- Browser Configuration
- For more information

This page references data in the Postmile Web Service. The data is provided by the California Department of Transportation.

Figure 2. Page created by HTML shown in Figure 3.

The Postmile Query Tool is incorporated into your page in an *iframe* element. This isolates it from your page, so there are no name collisions between the names and ids you use for your html elements, css classes and Javascript functions, and those used by the tool.

That's all there is to putting the tool on your page, providing your application is hosted by Caltrans. If it isn't, see the subsection below entitled "Servers, Protocols and License keys."

However, to enable communication between your application and the tool, and to configure the tool, you must supply additional arguments to the `ct.PMQT.addPMQT` function. These are addressed in the sections entitled "Getting data into and out of the tool" and "Configuring the Postmile Query Tool".

```
<!DOCTYPE html>
<html>
    <head>
        <title>Test Embed PMQT</title>
        <script
          type="text/javascript"
          src="https://postmile.dot.ca.gov/PMQT/scripts/PMQTEmbed.js">
        </script>
    </head>
    <body onload="ct.PMQT.addPMQT('Map')">
        <h1>Test of embedding Postmile Query Tool</h1>
        <div id="Map">
        </div>
    </body>
</html>
```

Figure 3. HTML code to generate the page shown in Figure 2

Servers, Protocols and License keys

There are four servers hosting the Postmile Query Tool:

- https://postmile.dot.ca.gov – the public-facing unrestricted production version;
- http://postmile-internal.dot.ca.gov – the internal production version on the Caltrans intranet;
- http://postmile-dev.dot.ca.gov – the internal intranet version intended for development of the Postmile Query Tool;
- http://postmile-test.dot.ca.gov – the internal intranet version intended for User Acceptance Testing of updated versions.

Please note that: if you access the Embeddable Postmile Query Tool from the "postmile" server (as shown in the example), then you must use the "https" (secure transport layer) protocol in the url of the <script> tag. Also, this will access the GoogleMap Javascript API using the "external" version of the Caltrans license for the API, which is intended for sites that are outside the firewall and have unrestricted access.

If you access it from any of the other hosts, you must specify the "http" protocol.

The postmile-internal site uses the "internal" version of the Caltrans license for the API, which is intended for "internal" applications: sites that can only be accessed from inside the firewall, or that require login credentials to access.

The postmile-dev and postmile-test sites use a free, private development key for the API, which is intended only for development and test.

GoogleMap API LIcense

The Postmile Query Tool uses the GoogleMap Javascript API to display the map. If your application is served by a California Department of Transportation server (i.e., the domain matches *.dot.ca.gov and the scheme/protocol is http or https) then the Postmile Query Tool will use Caltrans' own license key for the GoogleMap Javascript API (as noted above in "Servers, Protocols and License keys").

We would like to have made the Postmile Query Tool available for other hosts, by enabling them to submit their own API license key (or client ID). However, Google does not provide a hacker-proof way to verify that the key presented to us is valid for the host wishing to embed the tool. (The Google license also apparently explicitly prohibits this sort of iframe-based embedding.) Therefore, the next section, which describes how a non-Caltrans host could embed the Postmile Query Tool by providing its own license key, is not currently valid (and won't be, unless Google changes its policy).

Embedding for non-Caltrans hosts (*not valid or supported*)

If your application is served by a host from any other domain, then you must do the following:

1. Obtain your own license key for the GoogleMap Javascript API;
2. Provide your key to the Postmile Query Tool;
3. White list the Postmile Query Tool as an acceptable referrer for your key.
4. White list your application host as an acceptable referrer for your key.

Getting a GoogleMap API key

You can obtain a license key for the GoogleMap Javascript API from Google by going to https://developers.google.com/maps/documentation/javascript/ and clicking the "Get a key" button.

Google will walk you through the steps of creating a project, activating the API, and generating a key.

Providing the key to the Postmile Query Tool.

You provide your own license key for the API by including it as a parameter in the URL for the <script> tag that loads the Postmile Query Tool:

```
<script>src="https://postmile.dot.ca.gov/PMQT/scripts/PMQTEmbed.js?key
=<your API key>" type="test/javascript" ></script>
```

Figure 4. <script> tag to invoke the Postmile Query Tool with an API key.

White-Listing the Postmile Query Tool

If you restrict the use of your license key by setting a "white list" of acceptable referrers (recommended), then to use that key with the Postmile Query Tool you must add "*postmile.dot.ca.gov/*" (the host from which you are accessing the Embeddable Postmile Query Tool) to your list of acceptable referrers. (It is important that the domain that hosts your application is also listed as an acceptable referrer, even though your application may not directly invoke the Google Map API, because this is how the Postmile Query Tool server will ensure that your key isn't being used illegally by another host.)

(*End of invalid, unsupported section.*)

Getting data into and out of the tool

The principal way that your application and the Postmile Query Tool will communicate is through html (hidden) input elements. You create an input element on your page that corresponds to each datum that you wish to share with the tool. You give each such element a unique id, then you tell the Postmile Query Tool what these ids are. If any of these input elements have values, those values will be used to initialize the corresponding elements within the tool. Each time a user interaction with the tool changes the values of any of the corresponding elements, the new values will be communicated back to your application page by updating the values of your input elements. If these elements are part of a form, then you can communicate the values back to your web application by submitting the form.

You tell the Postmile Query Tool the id of the input element for each datum by creating and configuring an instance of the class `ct.PMQT.InputControls`. You then pass this instance as the second argument in the call to the `ct.PMQT.addPMQT function`. This class has a property for each datum that can be communicated between your application and the Postmile Query Tool. These are listed in Table 1.

For example, if your application wanted to initialize and/or track the geo-coordinates in decimal format, you would create input elements with ids like “latitude” and “longitude”, then do the following:

```
<script>
  var controls = new ct.PMQT.InputControls();
  controls.latDecimal = "latitude";
  controls.lngDecimal = "longitude";
</script>
<input id="latitude" hidden="hidden" />
<input id="longitude" hidden="hidden" />
```

Figure 5. Javascript to create and initialize a ct.PMQT.InputControls instance.

Then your call to add the tool to the page would look like:

```
ct.PMQT.addPMQT('postmileTool', controls).
```

Table 1. Properties of ctPMQT.InputControls

Property	Description
	Postmiles (Text format)
postmileSpec	Text specification of the postmile for the Point, or the first postmile of the Line (to be) validated
postmileSpec2	Text specification of the second postmile of the Line (to be) validated
	Postmiles (Components)
postmileRoute	Route number of the route for the postmile(s) (to be) validated
postmileRouteSuffix	Route suffix of the route for the postmile(s) (to be) validated
postmileCounty	County code of the county for the postmile of the Point or the first postmile of the Line (to be) validated
postmileCounty2	County code of the county for the second postmile of the Line (to be) validated
postmilePrefix	Postmile Prefix of the postmile of the Point or the first postmile of the Line (to be) validated
postmilePrefix2	Postmile Prefix of the second postmile of the Line (to be) validated
postmileValue	Postmile Value of the postmile of the Point or the first postmile of the Line (to be) validated
postmileValue2	Postmile Value of the second postmile of the Line (to be) validated
postmileSuffix	Postmile Suffix of the postmile of the Point or the first postmile of the Line (to be) validated
postmileSuffix2	Postmile Suffix of the second postmile of the Line (to be) validated

Property	Description
Geo-coordinates (Decimal format)	
latDecimal	Latitude (in decimal format) of the geo-coordinates pushpin marker (Point mode) or the first geo-coordinates pushpin marker (Line mode)
latDecimal2	Latitude (in decimal format) of the second geo-coordinates pushpin marker (Line mode)
lngDecimal	Longitude (in decimal format) of the geo-coordinates pushpin marker (Point mode) or the first geo-coordinates pushpin marker (Line mode)
lngDecimal2	Longitude (in decimal format) of the second geo-coordinates pushpin marker (Line mode)
Geo-coordinates (Deg/Min/Sec/Card format)	
latDeg	Degree component of the latitude of the geo-coordinates pushpin marker (Point mode) or the first geo-coordinates pushpin marker (Line mode)
latDeg2	Degree component of the latitude of the second geo-coordinates pushpin marker (Line mode)
lngDeg	Degree component of the longitude of the geo-coordinates pushpin marker (Point mode) or the first geo-coordinates pushpin marker (Line mode)
lngDeg2	Degree component of the longitude of the second geo-coordinates pushpin marker (Line mode)
latMin	Minutes component of the latitude of the geo-coordinates pushpin marker (Point mode) or the first geo-coordinates pushpin marker (Line mode)
latMin2	Minutes component of the latitude of the second geo-coordinates pushpin marker (Line mode)
lngMin	Minutes component of the longitude of the geo-coordinates pushpin marker (Point mode) or the first geo-coordinates pushpin marker (Line mode)
lngMin2	Minutes component of the longitude of the second geo-coordinates pushpin marker (Line mode)
latSec	Seconds component of the latitude of the geo-coordinates pushpin marker (Point mode) or the first geo-coordinates pushpin marker (Line mode)
latSec2	Seconds component of the latitude of the second geo-coordinates pushpin marker (Line mode)
lngSec	Seconds component of the longitude of the geo-coordinates pushpin marker (Point mode) or the first geo-coordinates pushpin marker (Line mode)
lngSec2	Seconds component of the longitude of the second geo-coordinates pushpin marker (Line mode)
latDir	Direction component (as compass cardinal point) of the latitude of the geo-coordinates pushpin marker (Point mode) or the first geo-coordinates pushpin marker (Line mode)
latDir2	Direction component (as compass cardinal point) of the latitude of the second geo-coordinates pushpin marker (Line mode)
lngDir	Direction component (as compass cardinal point) of the longitude of the geo-coordinates pushpin marker (Point mode) or the first geo-coordinates pushpin marker (Line mode)
lngDir2	Direction component (as compass cardinal point) of the longitude of the second geo-coordinates pushpin marker (Line mode)

Property	Description
	State Odometers*
stateOdometer	State Odometer of the postmile for the Point or the first postmile of the Line validated
stateOdometer2	State Odometer of the second postmile for the Line validated

*currently, state odometers are outputs only: the Postmile Query Tool does not currently validate postmiles starting from a state odometer.

Initializing the Postmile Query Tool

If your web application generates the input control html elements with initial values, then these values will be used to initialize the corresponding values in the Postmile Query Tool.

This raises an issue: some input elements specify values for the same variable redundantly, and some values must be consistent with one another, or have components that must be self-consistent. For example, specifying a decimal latitude of latDecimal = 37.229444 is the same as specifying latDeg = 37, latMin = 13, latSec = 46.0 and latDir = ‘N’. What should the Postmile Query Tool do if the elements for the two different ways of expressing the latitude specify two different latitudes?

As another example, the two postmiles for a Line validation should be mutually consistent (they should have the same route and route suffix, if any, and have county codes for counties that both contain the specified route. Thus, postmileSpec = “BUT 6 0.372” and postmileSpec2 = “COL 16 4.023” are mutually inconsistent (and self-inconsistent):

1. The BUT (Butte) and COL (Colusa) counties have no routes in common, and the two specifications indicate different routes (6 vs 16) in any case;
2. Route 6 does not pass through Butte county.

Syntactic inconsistency (two expressions for the same variable that don’t evaluate to the same value) is handled by following some precedence rules. For example, the decimal format expressions of geo-coordinates overrule the degree/minute/second/cardinal point expressions. Thus, the latter will populate the geo-coordinates only if they have values and the decimal inputs do not.

Semantic inconsistency (selections for components of a specification being geographically inconsistent) are handled in the same way they would be if the user interactively typed them into the tools’ input controls. That is to say, inconsistent selections are disallowed and filtered out.

The precedence rules are:

1. Postmile text specifications overrule postmile individual component selections;
2. Decimal geo-coordinates overrule deg/min/sec/card format coordinates.

When initializing the two postmiles for a line, the postmiles are entered in the order postmile 1 followed by postmile 2.

Configuring the Postmile Query Tool

Your application may want to exert some control over the appearance and functionality of the Postmile Query Tool, to customize it for the role it plays in your application. For this purpose, the `ct.PMQT.addPMQT` function accepts a third argument. The value passed as the third argument, if any, should be an instance of the class `ct.PMQT.ControlsConfig`. Table 2 lists the properties of this class that you can choose to set, their default values, and what they mean.

Table 2. Properties of the ct.PMQT.ControlsConfig class that you can modify.

Property	Default	Description
includePointMode	true	Whether the user should be able to select Point Mode. If this value is false, no Point vs Line radio buttons will be displayed.
includeLineMode	true	Whether the user should be able to select Line Mode If this value is false, no Point vs Line radio buttons will be displayed.
mode	“Point”	The initial mode. Must be “Point”, “Line” or null. This value is moot when only one mode is enabled. If null, the tool will use cookies to start out in the same mode in which it was last used within the browser.
includePostmileComponents	true	Whether the user can select to enter/view postmiles using the component drop-down lists. If this value is false, the “Components” check box will not be displayed.
includePostmileTextSpec	true	Whether the user can select to enter/view postmiles as text specifications. If this value is false, the “Components” check box wil not be displayed.
postmileFormat	“Components”	The initial format in which postmiles are displayed. Must be “Components”, “Text” or null. This value is moot when only one format is enabled. If null, the tool will use cookies to start out in the same format in which it was last used within the browser.
includeGeoDecimalFormat	true	Whether the user can select to enter/view geo-coordinates in decimal format. If this value is false, the “Deg/Min/Sec/Card” checkbox will not be displayed
includeGeoDegMinSecCard	true	Whether the user can select to enter/view geo-coordinates in degrees/minutes/seconds/cardinal Compass point format. If this value is false, the “Deg/Min/Sec/Card” checkbox will not be displayed
geoFormat	"Deg/Min/Sec/Card"	The initial format for displaying geo-coordinates. Must be “Deg/Min/Sec/Card”, “Decimal” or null. This value is moot if only one format is enabled. If null, the tool will use cookies to start out in the same format in which it was last used within the browser.
includeViewCountyBoundaries	true	Whether the user can select to view the county boundaries, routes and calibration

Property	Default	Description
		postmiles. If this value is false, the check box for this functionality will not be displayed.
viewCountyBoundaries	false	Whether the county boundaties view should be set initially. This value is moot if the functionality is not enabled.
includeHelp	true	Whether the user should be able to view the help pages. If this value is false, the help button is not displayed, and the initial help page is not displayed.
showHelp	false	Whether the first help screen should be displayed initially (moot when Help is not enabled).
validatePostmiles	false	Whether the tool should attempt to validate the initial values for postmiles.
validateGeoCoordinates	false	Whether the tool should attempt to validate the initial geo-coordinates (i.e., determine the postmiles closest to the geo-coordinates)
mapCenterLatitude	null	Latitude of the location on which to center the map display. Moot if postmiles or geo-coordinates are validated, as the validation process will determine the map center (unless the validation fails). If null a default location in the center of the state is used.
mapCenterLongitude	null	Longitude of the location on which to center the map display. Moot if postmiles or geo-coordinates are validated, as the validation process will determine the map center (unless the validation fails). If null a default location in the center of the state is used.
mapZoom	null	An integer between 0 (show entire world) and 19 (2^19 times closer) determining the initial zoom level for the map. If null a zoom level that shows the entire state is used.

If you do not provide an instance of the `ct.PMQT.ControlsConfig` class, or if you create an instance and pass in it unmodified, then the default values shown in the table will apply. The following code shows how you would arrange that the tool starts out in line mode (but preserves the option to select point mode) and shows geo-coordinates only in decimal format (making the degrees/minutes/seconds/cardinal compass point format unavailable):

```
<script>
  var configuration = new ct.PMQT.ControlsConfig();
  configuration.mode = "Line";
  configuration.includeGeoDegMinSecCard = false;
</script>
```

Figure 6. Javascript to create and initialize a ct.PMQT.ControlsConfig instance.

You would then invoke the tool with the call:

```
ct.PMQT.addPMQT('postmileTool', controls, configuration)
```

Customizing the Help System

The Postmile Query Tool comes with a built-in system of help pages that discuss what a postmile is, what controls the tool has and how to use them, and other topics helpful to the user. If you customize the tool by configuring its controls, then some of these help pages will contain information that is not relevant, and possibly incorrect for your customized version of the tool. Also, these pages are perforce not designed to be relevant to the specific application in which you are embedding the tool.

For these reasons, you will likely want to customize the help pages also.

The tool does not have a mechanism for modifying the content of the help pages for a specific context. Instead, the way to customize the help system is to turn it off completely within the tool (by setting the configuration parameter includeHelp = false), and to set up your own application-specific help system in your embedding application.

You are, of course, free to accomplish this in any way that fits your needs. However, the Postmile Query Tool exposes its own simple help system as a possible starting point.

Help System Files

The Postmile Query Tool help system consists of just three files:

https://postmile.dot.ca.gov/PMQT/css/helpPages.css -- a simple Cascading Style Sheet for help pages
https://postmile.dot.ca.gov/PMQT/scripts/navHelp.js -- a Javascript file for navigating help pages
https://postmile.dot.ca.fov/PMQT/helpPages.html -- an html page containing the help page information

helpPages.html

The "help system" information is contained entirely within one html file with the following structure:

```
<!DOCTYPE html>
<html>
  <head> … </head>
  <body>
    <div id="HelpFrame" style="display:inline-block;" >
      <div> … </div> ← (for "close button")
          ↓ (one "HelpPage" div for each help page, each with a unique id)
      <div class="HelpPage" id="…" style="display:none"> … </div>
      <div class="HelpPage" id="…" style="display:none"> … </div>
      <div class="HelpPage" id="…" style="display:none"> … </div>
      <div class="HelpPage" id="…" style="display:none"> … </div>
      <div class="HelpPage" id="…" style="display:none"> … </div>
     …
     <table>…</table> ← (for "prev", "back" and "next" buttons)
    </div>
  </body>
</html>
```

Figure 7. Structure of the Help System HTML.

In other words, the entire content of the help system is in one div with the id "**HelpFrame**." This div contains a div at the beginning to hold the "close" button, and a table at the end that holds the other navigation buttons. Between these two elements, there is one div for each help page. Each help page div has a unique id. Each page must also have the class= "**HelpPage**."

The sequence of the help page div's within the help frame div is the order in which the pages are visited using the "Next" button.

If you need to insert a hyperlink from one page to another (such as in a table of contents or a footnote), you insert a link like the following:

```
<a href="#" title="View topic: How to: Line Mode"
   onclick="showHelpPage('HowToLineMode')">How to: Line Mode</a>
```

The key factor is to set the "onclick" attribute of the link to a call to the **showHelpPage** function, passing the id of the div to be displayed.

navHelp.js

The navHelp.js Javascript file can be used as is, simply by referencing it within your help page html file using an external <script> tag. It defines only one global variable and seven functions.

The global variable is named **topicStack**. This is an array implementing a stack that tracks what pages the user has visited, in order to implement the "back" button. Each visited page is represented in the stack by a string giving the id of the page. Each "page" is a div with a unique id.

The functions are:

Table 3. Functions defined in navHelp.js

Function	Description
hideHelp()	Makes all help page divs and the help frame invisible by changing their style.display property.
showHelpPage(pageId)	Makes the help frame and the help page div with the indicated id visible.
showNextPage()	Implements the "Next" button by hiding the currently visible page, and making the next one visible.
showPrevPage()	Implements the "Previous" button by hiding the currently visible page, and making the previous one visible.
showReturnPage()	Implements the "Back" button by hiding the currently visible page, and making the last visited page visible.
nodeListIndexOf(nodeList, testfn)	A helper function that determines the index within an html nodelist of the element that satifies the testfn.
visiblePage(div)	A test function that determines whether the div element of a help page is currently visible.
pageHasId(div, pageId)	A test function that determines whether the div element of a help page has the specified id.

Most of the functions are used to implement the navigation buttons within the help system. If you use the Postmile Query Tool helpPages.html as the starting point for creating your own set of pages, then the only function you need to use is `showHelpPage`. You use this both to open the help system on its first page (your “show help” button), and to implement hyperlinks between the pages.

However, if you use this file It will be important not define functions or variables that collide with these names.

helpPages.css

The helpPages.css file can be used as is, by referencing it using a <link> tag within your help page file.

This file defines six classes that determine the appearance of the help navigation buttons, and it provides two classes used to determine the appearance of the help frame: HelpFrameOverlay and HelpFrameSequential. The Postmile Query Tool normally uses the first of these two classes to overlay the help frame “window” over the map area, but switches to the alternate when the window’s width is too narrow. You may wish to create a completely different class to define the appearance and location of your help frame div, in your own .css file.

You should avoid name collisions with the following class names:

1. HelpPage (identifies divs as help pages)
2. HelpFrameOverlay
3. HelpFrameSequential
4. HelpNavButton (defines defaults for the appearance of all the help buttons)
5. HelpButton (defines the appearance of the “show help” button)
6. CloseButton (defines the appearance of the “close” button)
7. NextButton (defines the appearance of the “next” button)
8. PrevButton (defines the appearance of the “previous” button)
9. BackButton (defines the appearance of the “back” button)

Nota Bene: If you re-use any of the Postmile Query Tool help pages in your application-specific help system, be aware of the following caveat: these pages reference images using relative pathnames (such as `src="images/Help_icon_100x100.png"`. For your pages to work, you will have to substitute absolute pathames, as in
`src="https://postmile.dot.ca.gov/PMQT/images/Help_icon_100x100.png"`.

To invoke your help system, you can include a button like this:

```
<button class="HelpNavButton HelpButton"
        onclick="showHelpPage(<id of your first page>)"
        title="Open Help Window"></button>
```