HERE Maps API - How to create a tabbed Infobubble

This article explains how to use CSS styling to modify an `Infobubble`. Initially a simple change of size and color scheme is discussed, the article then moves on to more advanced topics such as how to create a usable, tabbed `Infobubble` through combining CSS styling with DOM event handling.

Introduction

The `Infobubble` is a map component which is used to associate text with a particular point on the map. When it is displayed, it appears as a speech bubble, with its tail pointing towards a specified location. The location of the content box of the `Infobubble` will shift around the anchor point to best display the content on the map canvas. In other words, the `Infobubble` content will move either above/below, or left/right of the anchor point as the map is scanned.

A standard Nokia Map `Infobubble` is black, with white text as shown below:

Simple CSS Styling

The `InfoBubble` has several `options` associated with it. The most useful of these is the `defaultWidth`. By defining the `defaultWidth`, the content within the `Infobubble` is wrapped beyond a fixed size. This would be useful when displaying a Map with an `Infobubble` on mobile devices for example. The content of the `Infobubble` can either be plain text or HTML. If the `Infobubble` contains HTML content, and this HTML describes elements which have a greater width than the `defaultWidth` , the width of the `InfoBubble` is stretched to be able to display the specified content. To define a `defaultWidth` use the code below:

```
var infoBubbles = new nokia.maps.map.component.InfoBubbles();
infoBubbles.options.defaultWidth = 400;
```

For variety and personal preference, you may wish to alter the default color scheme. All of the `Infobubble` content is held within a `<div class='nm_bubble_content'>` element, so it is possible to restyle the content by applying a CSS style to the class.

```
.nm_bubble_content{
   color:black;
   background:white;
   border: 1px solid black;
   padding:0px;
}
```

The text has been altered to black and the background is white. A single pixel border has been added to the `InfoBubble` to stop the text from melting into the background of the map. Note that the tail of the `Infobubble` cannot be styled, and will always be black.

(C) Copyright Nokia 2014. All rights reserved.

Tabbed Infobubbles

If an `InfoBubble` contains a lot of information, and the size of the map is restricted, it may be that not all of content of the `InfoBubble` can be displayed at once without scrolling. This can be seen in the examples shown above, where the bottom of the `Infobubble` is "below the fold". In this case, and especially if the `Infobubble` contains information which can be split thematically into parts, it may be preferable to display the content in a tabbed control. Although there is currently no instance of an actual tabbed `Infobubble` within the standard map components, it is possible to create a `TabbedInfobubble` component through CSS styling, and the creation of a single DOM event

This has been done by extending the current `Infobubble` control into a tabbed infobubble control. All the code is hidden in a separate library, the library is set up as follows:

```
function setUpTabbedInfoBubbles(map){
 extend(TabbedInfoBubbles, nokia.maps.map.component.InfoBubbles);
 infoBubbles = new TabbedInfoBubbles();
 infoBubbles.options.defaultWidth = 400;
 map.components.add(infoBubbles);
}
```

Where `extend()` has been defined as:

```
function extend(B, A) {
 function I() {}
 I.prototype = A.prototype;
 B.prototype = new I();
 B.prototype.constructor = B;
}
```

Commentary

Styling Tabbed Infobubbles

Styling must be added to each of the HTML elements so that the desired effect of a tabbed dialog is achieved.

Structure of an HTML Tabbed Dialog

A tabbed dialog consists of two parts :

- A *list* of tab headers which are displayed in a row. All entries in the list are displayed.
- The tabbed content area, where only the content of the currently selected tab is visible.

An outline of the HTML can be seen below:

```
<div>
    <ul>
        <li>Header of the first tab</li>
        <li>Header of the second tab</li>
        <li>Header of the third tab</li>
```


(C) Copyright Nokia 2014. All rights reserved.

```
...etc
    </ul>
    <div> Content of the first tab</div>
    <div> Content of the second tab</div>
    <div> Content of the third tab</div>
 ...etc
</div>
```

CSS of the tab headers

Since the tab headers are presented as an un-ordered list, they are contained within a `<ul>` element. This must be styled to shift the position of the `<ul>` element such that it lies just above the main `InfoBubble`. The offset required must be judged by eye, and the magic number used here is 39 pixels, which places the `<ul>` above the close button **[**X**]** on the right-hand side. The `list-style-type: none` removes any bullet points preceding the header text on each tab; the remaining styles space the tabs and align look with the "black-on-white" `InfoBubble` defined above.

```
ul.nm_tabnav { /* general settings */
  position: relative;
  top: -39px;
  color:black;
  text-align: left;
  margin: 0px -1em 1em 0px;
  list-style-type: none;
  padding: 3px 10px 0.1em 10px;
  font-size:small;
}
```

Each tab header is held within a `<li>` element. We want to make sure that the tab headers display in a **single** row, and therefore the `<li>` needs to be styled `display: inline`. The `background` and `border` matches the `InfoBubble` styling, so that the text of tab header is not displayed over an invisible background. The colour and style of the text may be altered as desired, but there needs to be some way to distinguish between the currently selected tab and the others. Therefore the currently selected tab is given its own styling, as well as a slightly altered `bottom-border` so that the tag merges into the main `InfoBubble`.

```
ul.nm_tabnav li {
  display: inline;
  background:white;
  border: 0.1em solid black;
  position: relative;
  top: 1px;
  padding: 3px 1em;
  /* You can add your own styling here */
  color:blue;
}
ul.nm_tabnav li.nm_tab_current { /* do not change */
  border-bottom: 0.2em solid white; /* set border COLOR as desired */
  font-weight:bold;
  color:red;
}
```

CSS of the tabbed content area

The CSS of the tabbed content area is very simple. The first `<div>` element is made visible by applying the `display: block` styling. The second and subsequent `<div>` elements are made invisible using the `display: none` styling. The CSS styling here uses the plus syntax make all `<div>` elements preceded by a `<div>` element invisible. `display: none` means that not only is the element invisible, but that it does not take up any space on the screen.

(C) Copyright Nokia 2014. All rights reserved.

```
.nm_bubble_content > div{
    display: block;
}
.nm_bubble_content > div + div{
    display: none;
}
```

Enabling the tabbed dialog to respond to DOM Events

The HERE Maps API already contains library functions for attaching a function to a DOM event. The following order of initialisation needs to occur when the `Infobubble` is attached to the `mapDisplay`

- The `Page` object needs to be initialised to query if DOM events are supported.
- An `EventTarget` initialised and fed an existing DOM object to wire up the event handler
- Finally the function holding the dialog tabbing code can be added as an event listener using the `addListener()` method.

It should be noted that the only way to identify the correct `<div>` holding the `InfoBubble` text is through its style class. This can be done using the `document.getElementsByClassName()` method, however not all browsers support this. In particular older versions of Internet Explorer lack native `document.getElementsByClassName()` support. In order to overcome this, a standard `getElementsByClassName()` shim must be added to ensure the largest range of cross-browser support. The `<div class='nm_bubble_content'>` is injected into the DOM when the map is intialised, so it is necessary to wait until the map has been created before wiring up the tabbed dialog DOM event.

```
this.openBubble = function(content, coordinate){
 divLength = document.getElementsByClassName("nm_bubble_content").length;
 nokia.maps.map.component.InfoBubbles.prototype.openBubble.call(this, content,
coordinate) ;
 wireUp(1);
 wireUp(0);
}
```

```
function wireUp(index){
     var Page = nokia.maps.dom.Page;
 var EventTarget = nokia.maps.dom.EventTarget;
 // This element will only exist once the map has been displayed.
 var infoBubbleDisplay = document.getElementsByClassName("nm_bubble_content")[index];


 // Query Page support for the node.
 Page(infoBubbleDisplay);

 // Attach EventTarget interface to the document to allow normalized events at the node.
 EventTarget(infoBubbleDisplay);

 // Add a listener for the click event to the node and show an alert if clicked.
 infoBubbleDisplay.addListener("click", clickFunction , false);
}
```

The "click" DOM event will be fired whenever the user clicks **anywhere** within the `InfoBubble`. An initial check restricts the code to only update the `InfoBubble` if a tab has been clicked. The code itself comes in two parts: the first part traverses the DOM until the first `<div>` of the tabbed content is discovered, the second part iterates thorough each of the `<li>` elements in turn and associates the `nm_tab_current` class with the `<li>` that was clicked and displays its associated content, whereas the `nm_tab` class is associated to all the other tabs, and their content `<div>` elements are hidden, by setting the style to `display:none`.

```
var clickFunction =  function (evt) {
```


(C) Copyright Nokia 2014. All rights reserved.

```
 if(evt.target.className == "nm_tab"){
  var offset = 0;
  var tabs = evt.target.parentNode.children;
  var tabContent = this.children;

  while (offset < tabContent.length) {
   if (tabContent[offset].nodeName== "DIV" ) {
     break;
   }
   offset++;
  }
  // Loop through all the LI elements and set the clicked on to current,
  // At the same time ensure only the nth DIV associated with the Nth Tab
  // is visible - all other are set to display:none.

  for (var i = 0, len = tabs.length; i < len; i++){
   if ( tabs[i] == evt.target){
    tabs[i].className = 'nm_tab_current';
    tabContent[i+ offset].style.display ='block';
   } else {
    tabs[i].className = 'nm_tab';
    tabContent[i + offset].style.display ='none';
   }
  }
 }
};
```

Adding content to the tabbed dialog control

The `InfoBubble` map component accepts HTML as one of it input parameters, all that remains is to create a function that creates the HTML in the required format. A function can be written which takes an `Array` of tab headers and an `Array` of tab contents, along with an optional title for the `TabbedInfoBubble`, and create the necessary HTML for the tabbed dialog control.

```
this.addTabbedBubble = function(tabs, content, title, coordinate){
 this.openBubble(tabbedContent(tabs, content, title), coordinate) ;
}
```

```
var tabbedContent = function (tabs, content, title){
 var myHTMLcontent = "<ul class=\"nm_tabnav\">";
 for (var i = 0; i < tabs.length; i++){
   if (i==0){
     myHTMLcontent = myHTMLcontent + "<li class=\"nm_tab_current\">"+ tabs[i] + "</li>";
   } else {
     myHTMLcontent = myHTMLcontent + "<li class=\"nm_tab\">"+ tabs[i] + "</li>";
   }
 }
 myHTMLcontent = myHTMLcontent + "</ul>" + title;
 for (var i = 0; i < content.length; i++){
  if (i==0){
     myHTMLcontent = myHTMLcontent + "<div>"+ content[i] + "</div>";
   } else {
     myHTMLcontent = myHTMLcontent + "<div>"+ content[i] + "</div>";
   }
  }
 return myHTMLcontent;
}
```


(C) Copyright Nokia 2014. All rights reserved.

This function can be used to create a tabbed `Infobubble` using the syntax:

```
infoBubbles.addTabbedBubble(tabs, content, title, coordinate) ;
```

The result of this can be seen in the following example:

Battle of Saratoga

Using tabbed Infobubbles with KML

Using KML helps to separate the actual content from the way it is displayed on screen. The HTML which is displayed in the `Infobubble` is defined as a `<BalloonStyle>` and the content taken from the `<description>` and `<name>` elements. It is a simple matter to change the styling by associating an appropriate `<BalloonStyle>` using the `<styleUrl>` element.

```
<Style id='tabbed'>
 <BalloonStyle><text><![CDATA[<ul class="nm_tabnav">
   <li class="nm_tab_current">TAB ONE</li>
   <li class="nm_tab">TAB TWO</li>
   </ul><h2>$[name]</h2><br/>$[description]]]></text>
 </BalloonStyle>
</Style>

<Style id='untabbed'>

<BalloonStyle><text><![CDATA[<h2>$[name]</h2><div>$[description]</div>]]></text></BalloonStyle>
</Style>


<Placemark id="1">
   <styleUrl>#tabbed</styleUrl>
   <name>TITLE TEXT</name>
   <description><![CDATA[<div>TAB ONE TEXT</div>
   <div>TAB TWO TEXT</div>]]></description>
    <Point><coordinates>23.771,38.857,0</coordinates></Point>
</Placemark>
```

When displaying the KML data, the required order of events is a follows: firstly the map is initialised, secondly the tabbed `InfoBubble` is wired up and finally the KML is loaded. An example of the use of a tabbed `InfoBubble` showing loaded KML data can be found in the Decisive Battles Example

(C) Copyright Nokia 2014. All rights reserved.

Summary

Although there is currently no instance of an actual tabbed `Infobubble` within the standard map components, it is possible to create a usable tabbed dialog by overriding the standard `Infobubble` component and adding the necessary CSS styling, and through the creation of a single DOM event. When such a tabbed dialog is created, it is also possible to use it to display KML generated data.

(C) Copyright Nokia 2014. All rights reserved.