Programming Panning and Zooming in Interactive Maps

Andrew W. Woodruff | andy@axismaps.com

Axis Maps, LLC
P.O. Box 421
Hewitt, TX 76643

INTRODUCTION

An essential feature of many web-based, interactive maps is panning and zooming, together referred to as map browsing. The ever-increasing data storage capacity of online mapping systems and the near-ubiquity of web and mobile mapping services like Google Maps mean that panning and zooming are so often required and so often encountered that the likely question facing web cartographers is not *whether* to include panning and zooming, but rather *how* to include it. This tutorial addresses that *how* question from a technical standpoint, offering methods for implementing map browsing capabilities in Adobe Flash.

Harrower and Sheesley (2005) provide a practical framework for evaluating panning and zooming methods that is helpful for deciding what methods to support in an interactive map. They propose several functionality and efficiency criteria and evaluate nine map browsing methods: 1) directly reposition the map, 2) smart scroll bars, 3) rate-based scrolling, 4) keyboard controls, 5) zoom and recenter under mouse click, 6) navigator tabs/ interactive compass, 7) navigator window, 8) specify explicit coordinates or scale, and 9) zoom box. Interactive examples of eight of the nine are available at http://www.cartogrammar.com/blog/map-panning-and-zooming-methods/. Most of these methods are not mutually exclusive, and it is advisable to support multiple types of map browsing in a single map (Roth and Harrower 2008). In this tutorial I focus on three of the most common and efficient methods identified by Harrower and Sheesley (2005): direct manipulation (click-and-drag, or a "slippy map"), zoom box, and navigator window.

Deciding how to implement panning and zooming depends heavily on the design and purpose of the map. If a map does not require much custom design, comprises fairly simple geo-referenced vector data, or needs to be highly extensible or collaborative, then a mash-up approach with mapping APIs such as Google Maps (or services built on top of them) may be the simplest way to incorporate panning and zooming along with other built-in functionality. If the map is only a raster image and otherwise not interactive, software such as Zoomify (http://zoomify.com/) provides an easy solution by automatically generating a panning and zooming interface and breaking the image into tiles that are efficient for online viewing. This tutorial, however, provides methods for manually programming panning and zooming, perhaps best suited to maps with unique, custom designs and interactivity that would be compromised by the constraints of other options. The code provided is meant to facilitate adding panning and zooming to an independent map, requiring little or no modification to the map itself; this approach allows the map to be freely designed and built around its core information and functions rather than the supporting panning and zooming functions. Nonetheless, while a manual method of implementing panning and zooming allows more freedom, the cartographer should bear in mind that not every map may be worth the additional labor and time required.

DECIDING HOW TO IMPLEMENT PANNING AND ZOOMING DEPENDS HEAVILY ON THE DESIGN AND PURPOSE OF THE MAP

The tutorial presents code and instructions for implementing panning and zooming using the Adobe Flash platform, following the precedent of Roth and Ross (2009) in the first digital issue of *Cartographic Perspectives*. Flash affords extensive freedom of both design and interactivity and thus serves interactive cartography very well. It is also currently pervasive on the web both in terms of content and browser support, and although its support is limited on mobile devices, it remains a safe choice of platform for publishing interactive maps intended for desktop or laptop use. The source files for this tutorial are exclusively ActionScript 3, the scripting language for Flash, and they do not require the use of one particular development environment. However, the step-by-step instructions are given only for Adobe Flash CS4 (CS3 is also compatible). Other options include Adobe Flash Builder (formerly named Flex Builder) or a free Flex SDK available at http://opensource.adobe.com/wiki/display/flexsdk/downloads.

The following instructions assume basic familiarity with Flash and ActionScript 3 concepts. For introductory resources and to download a trial version of Flash CS4, visit: http://www.adobe.com/devnet/flash/.

TUTORIAL

Programming Panning and Zooming in Flash CS4

Andrew W. Woodruff | andy@axismaps.com

DOWNLOAD THE SOURCE FILES

The first step is to download the source files referenced in the tutorial. The code library for this tutorial is available at:

http://www.nacis.org/CP/CP66/pan_zoom_tutorial.zip

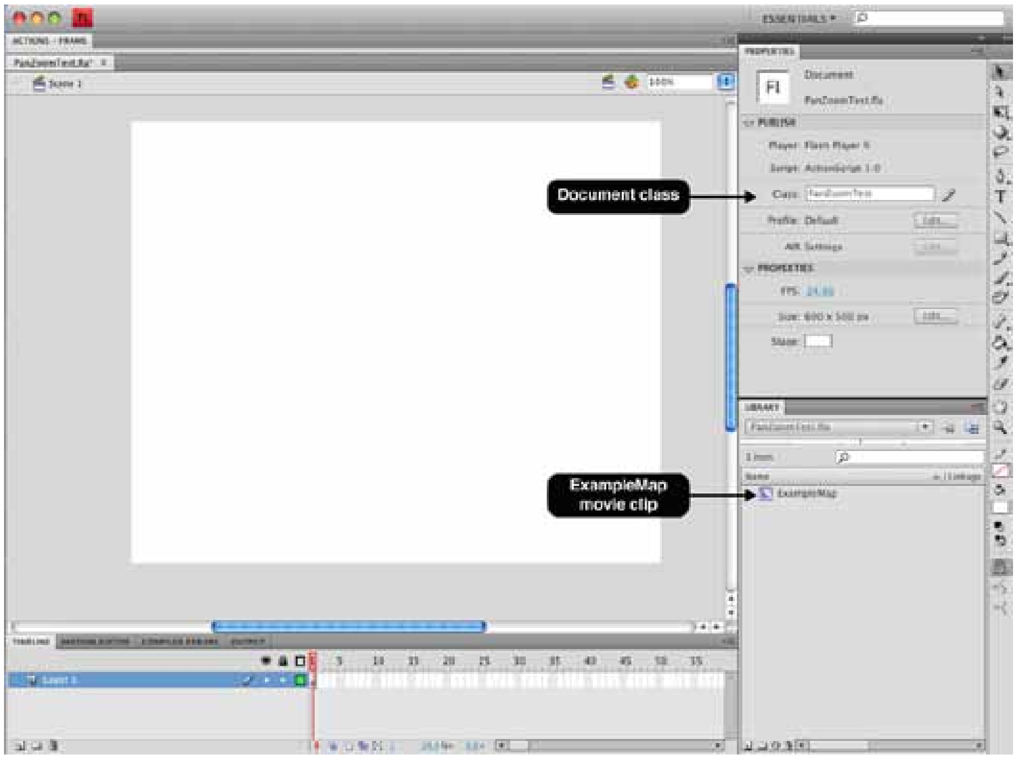
Download and extract the zip file to a folder of your choice. If you examine the extracted files, you should see two files and an **org** folder. Inside the **org/nacis/mapbrowsing** folder are the three main ActionScript classes that control panning and zooming:

1. **PanZoomMap.as:** The main class that creates a map that can be panned and zoomed.
2. **ZoomControls.as:** A set of three simple buttons to control zooming. The primary purpose of this class in the tutorial is to demonstrate how to create controls that interact with the map.
3. **NavigatorWindow.as:** A class that generates a navigator window (i.e., a small interactive overview map) for more advanced control of the panning and zooming methods in **PanZoomMap.as**.

PanZoomTest.fla and PanZoomTest.as are Flash project files used for demonstration in this tutorial.

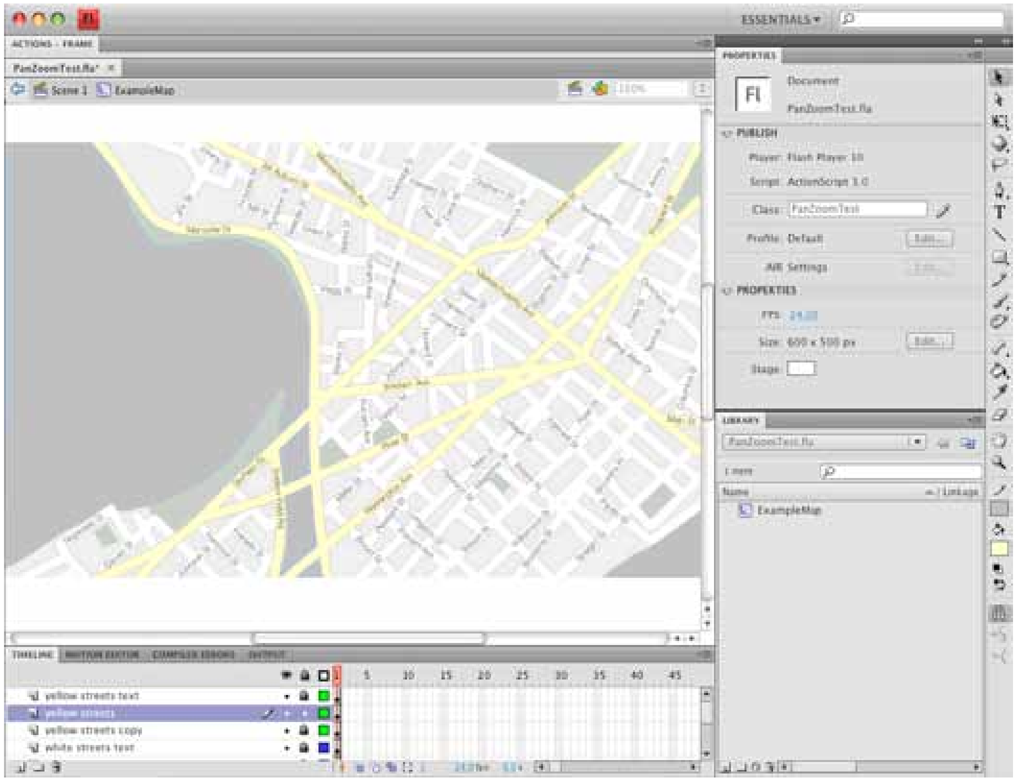
OPEN THE FLASH PROJECT

Begin by opening the FLA file, **PanZoomTest.fla**. This is the main Flash document that contains graphics, code, and other settings. This project file will be used to create a simple panning and zooming interface for an example map. Upon opening the file, you will notice a blank stage and one Movie Clip called `ExampleMap` in the Library panel (Figure 1). Double-click `ExampleMap` to view it; it is simply a vector graphics drawing (Figure 2). This will serve as the source map that will be panned and scaled. A custom map could be inserted here, either by pasting graphics from another program or by drawing directly in the Flash environment, to replace the tutorial example. The map does not need to be a static image; it can contain interactive Flash components or more advanced scripted interactivity.



Document
PanZoomTest.fla
PUBLISH
Player: Flash Player 9
Script: ActionScript 3.0
Class: PanZoomTest
Profile: Default
AIR Settings
PROPERTIES
FPS: 24.00
Size: 600 x 500 px
Stage:
LIBRARY
ExampleMap

Figure 1. PanZoomMap.fla upon first opening the file. ExampleMap is listed in the Library panel on the right side.



Document
PanZoomTest.fla
PUBLISH
Player: Flash Player 10
Script: ActionScript 3.0
Class: PanZoomTest
Profile: Default
AIR Settings
PROPERTIES
FPS: 24.00
Size: 600 x 500 px
Stage:
LIBRARY
ExampleMap

Figure 2. PanZoomTest.fla with the ExampleMap visible.

PREPARE THE SOURCE MAP

Rather than adding the map to the stage here in the authoring mode, it is more efficient to create the `PanZoomMap` instance and add it to the stage using only ActionScript. To do this, the `ExampleMap` library symbol must be exported for use in ActionScript. Right-click `ExampleMap` in the Library panel, select Properties, and check the "Export for ActionScript" box (Figure 3). If you do not see the options shown in Figure 3, click the "Advanced" button in the properties dialog. Note that the Class and Base class text fields become populated when "Export for ActionScript" is selected. `ExampleMap` will become an ActionScript class name that can be used to create a new instance (or multiple instances) of this source map. Both of these can refer to custom classes for advanced control and interactivity of the map, but for the purposes of this tutorial the map will remain simply a set of graphics, so leave the defaults as they are. Click OK on the Properties dialog and OK again on the warning message that appears. Save the FLA file.

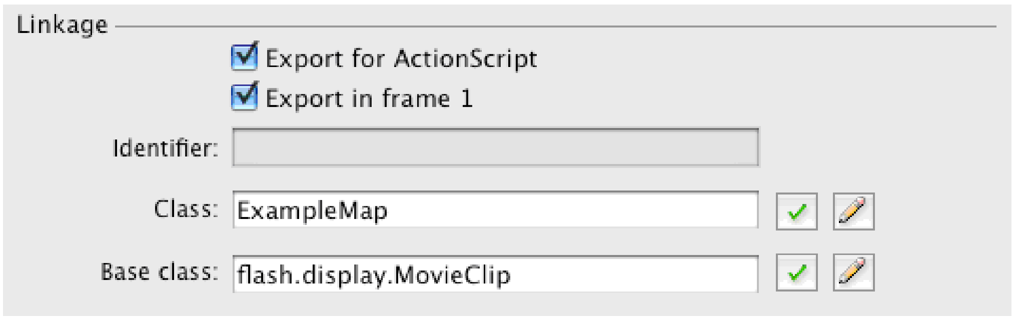

Linkage
[✓] Export for ActionScript
[✓] Export in frame 1
Identifier:
Class: ExampleMap
Base class: flash.display.MovieClip

Figure 3. Exporting the movie clip for ActionScript

OPEN THE DOCUMENT CLASS

The FLA file is now ready to have code attached to it. ActionScript code can be written for the main Flash movie either directly on timeline frames or in a separate document class held in an ActionScript (.as) file. Because it keeps most of the code in a central, well-organized file, a document class is the better choice for projects relying mostly on ActionScript code, such as this tutorial. Click a blank area of the stage to clear any selection, and note that `PanZoomTest` has been defined as the document class in the Properties panel (see Figure 1). Open the class either by clicking the pencil icon next to that name or by directly opening **PanZoomTest.as** from the tutorial files.

CREATE THE MAP INSTANCE

Find the `// Declare variables here` comment in **PanZoomTest.as**. In this spot, instances of `ExampleMap` and the `PanZoomMap` will be created. To do so, insert two lines of code as shown below.

```
 // Declare variables here
private var sourceMap : ExampleMap = new ExampleMap();
private var panZoomMap : PanZoomMap = new PanZoomMap( sourceMap, 500, 310 );
```

The first variable is an instance of `ExampleMap`, being used as our source map for panning and zooming. The second variable is an instance of `PanZoomMap`, placing the source map into a layout that can be panned and zoomed. A new instance of `PanZoomMap` requires three arguments:

1. A reference to the source map. This can be any display object. In this case it is `sourceMap`, the instance of `ExampleMap` created in the previous line.
2. The desired width of the viewable map area on the stage—the "window" through which the map will be viewed. This can be any number. 500 is arbitrarily chosen here.
3. The height of the viewable area. 310 is a height that approximately maintains the original aspect ratio of the source map in this case.

ADD THE MAP ON STAGE

Now find the `// Begin code here` comment in the `PanZoomMap()` constructor function. This code will run as soon as the SWF loads, so it is where the map will actually be added to the stage. The code below adds the `PanZoomMap` instance to the display and positions it a short distance from the top and left edges of the stage.

```
// Begin code here
addChild( panZoomMap );
panZoomMap.x = panZoomMap.y = 15;
```

Save **PanZoomMap.as** now and test **PanZoomMap.fla** by pressing Control-Enter (Windows) or Command-Enter (Mac). The SWF will be created and you should see the source map on the stage, scaled down to fit the specified 500x310 dimensions. By default, `PanZoomMap` establishes a minimum scale equal to the scale required to fit the entire map in the viewable area, and it zooms to that scale at the start. Notice that clicking and dragging the map has no effect. `PanZoomMap` also limits panning so that the left edge of the map cannot be dragged inward past the left edge of the viewable area, and so on. At the minimum scale, this means that no panning at all is possible. To be able to pan, you need to zoom in.

TEST ZOOMING AND PANNING

First, try zooming in and panning explicitly with code. Zooming is accomplished with `PanZoomMap's zoomTo()`, `zoomIn()`, or `zoomOut()` methods, either by calling the functions directly or attaching them to interface controls. The latter two methods increase and decrease the scale of the map by a factor of two. `zoomTo()` allows you to specify an exact scale, where 1 is the scale at which the source map is its original size. To get the current scale of the map, use the zoomAmount property. For a simple test, return to where you left off in **PanZoomTest.as** and add the following line of code to immediately zoom in by a factor of two:

```
panZoomMap.zoomTo( panZoomMap.zoomAmount * 2 );
```

Save and then run the SWF again, and the map will be zoomed in twice as far as before. Now you are able to pan the map by clicking and dragging. The panning is constrained in the manner described above. Similar to the `zoomTo()` method, there is also a `panTo()` method for jumping directly to a point on the map. Pass the `panTo()` method a `Point` object containing x and y coordinates in the coordinate space of the source map (in the case of the `ExampleMap` movie clip, the map's coordinates have their origin at the top left corner and extend to approximately 751 and 456 on the x and y axes, respectively). Insert the code below, save the file, and run the SWF. The map will be centered on a different area.

```
panZoomMap.panTo( new Point( 500,300 ) );
```

ADD USER CONTROL

The above example shows how to change the zoom level to a predetermined value through ActionScript code. Users, however, need to be given control over this value so that they can manipulate the zoom level to meet their own map use needs. Usually this involves creating buttons that invoke the `PanZoomMap` methods. **ZoomControls.as** is provided as an example set of buttons for controlling the map zoom. In **PanZoomTest.as**, erase or comment out the `zoomTo` and `panTo` lines to restore the map to the default behavior. Declare a new instance of `ZoomControls` and then add it to the stage and reposition it, as in lines 15 and 24–26 below.

```
1     package
2     {
3
4     import flash.display.MovieClip;
5     import flash.geom.Point;
6     import org.nacis.mapbrowsing.*;
7
8         public class PanZoomTest extends MovieClip
9         {
10
11        // Declare variables here
12        private var sourceMap : ExampleMap = new ExampleMap();
13     private var panZoomMap : PanZoomMap = new PanZoomMap( sourceMap, 500, 310 );
14     // Declaring a new instance of zoom controls
15     private var zoomControls : ZoomControls = new ZoomControls( panZoomMap );
16
17     public function PanZoomTest()
18     {
19
20           // Begin code here
21           addChild( panZoomMap );
22           panZoomMap.x = panZoomMap.y = 15;
23
24           addChild( zoomControls );
25           zoomControls.x = 15;
26           zoomControls.y = 330;
27         }
28     }
29 }
```

CodeBank 1. PanZoomTest.as with a map instance and zoom controls

When the `ZoomControls` instance is declared in line 14, it must be passed a reference to the `PanZoomMap` that it controls. You can examine **ZoomControls.as** to see how it uses the zooming functions on `PanZoomMap`, or to customize the design and placement of the buttons. When you run the SWF, there will be three simple buttons below the map. The "+" and "-" buttons zoom the map in and out, multiplying and dividing the map scale by two. The third button enables a zoom box mode. In this mode, click-and-drag panning is disabled and the mouse cursor becomes a tool for specifying an area to which to zoom. Click and drag on the map to draw a box, and when the mouse is released, the map will zoom to fit the area within the box.

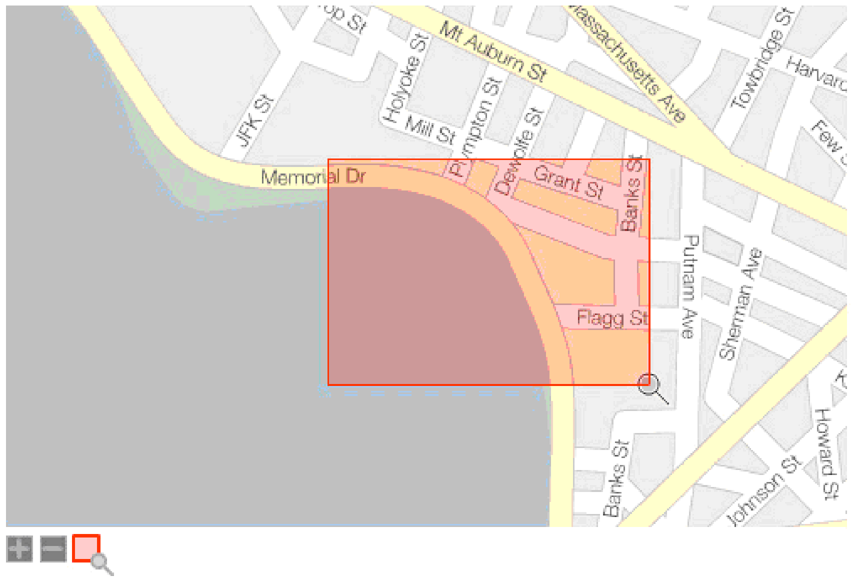


Figure 4. Zooming using the zoom box mode

PanZoomMap contains a zoomBoxMode property that enables or disables this mode. It is disabled by default. Clicking the zoom box button in ZoomControls simply toggles the property. The zoom box works by drawing a box on stage in the global coordinate space, transforming its dimensions and center to map coordinates using the globalToLocal() method, and then calling the zoomTo() and panTo() methods to rescale and recenter the map. Figure 4 illustrates the click and drag zooming implemented by the ZoomControls instance.

ADD A NAVIGATOR WINDOW

A final, more advanced control provided here is the NavigatorWindow class, which creates a small overview map that can be used to see the currently visible map area in context and also to pan the map quickly. Adding a navigator window is accomplished in the same way as adding the zoom control buttons. Create a NavigatorWindow instance, giving it a reference to the PanZoomMap instance and optionally a desired size expressed as a proportion of the full source map size (it defaults to 0.25 if no value is specified), and then add it to the stage (lines 15 and 28–30 below). The final code for this tutorial should look as follows:

```
1    package
2    {
3
4    import flash.display.MovieClip;
5    import flash.geom.Point;
6    import org.nacis.mapbrowsing.*;
7
8    public class PanZoomTest extends MovieClip
9    {
10
11       // Declare variables here
12       private var sourceMap : ExampleMap = new ExampleMap();
13       private var panZoomMap : PanZoomMap = new PanZoomMap( sourceMap, 500, 310 );
14       private var zoomControls : ZoomControls = new ZoomControls( panZoomMap );
15       private var navWindow : NavigatorWindow = new NavigatorWindow( panZoomMap );
16
17       public function PanZoomTest()
18       {
19
20           // Begin code here
21           addChild( panZoomMap );
22           panZoomMap.x = panZoomMap.y = 15;
23
24           zoomControls.x = 15;
25           zoomControls.y = 330;
26           addChild( zoomControls );
27
```

```
            navWindow.x = 365;
            navWindow.y = 330;
            addChild( navWindow );
            }
        }
    }
```

CodeBank 2. PanZoomTest.as with a map instance, zoom controls, and a navigator window.

Run the SWF again to test the navigator window. A red box on the overview map shows the area currently in view, and the box can be dragged to pan the map when possible (you will need to zoom in first). In simplified terms, the indicator box on the `NavigatorWindow` is drawn at the dimensions of the `PanZoomMap's` viewable area and scaled to the inverse of the `PanZoomMap's` zoom level. It is positioned easily—and also allows easy panning—because the coordinates on the `NavigatorWindow` correspond directly to coordinates on the `PanZoomMap` source map.

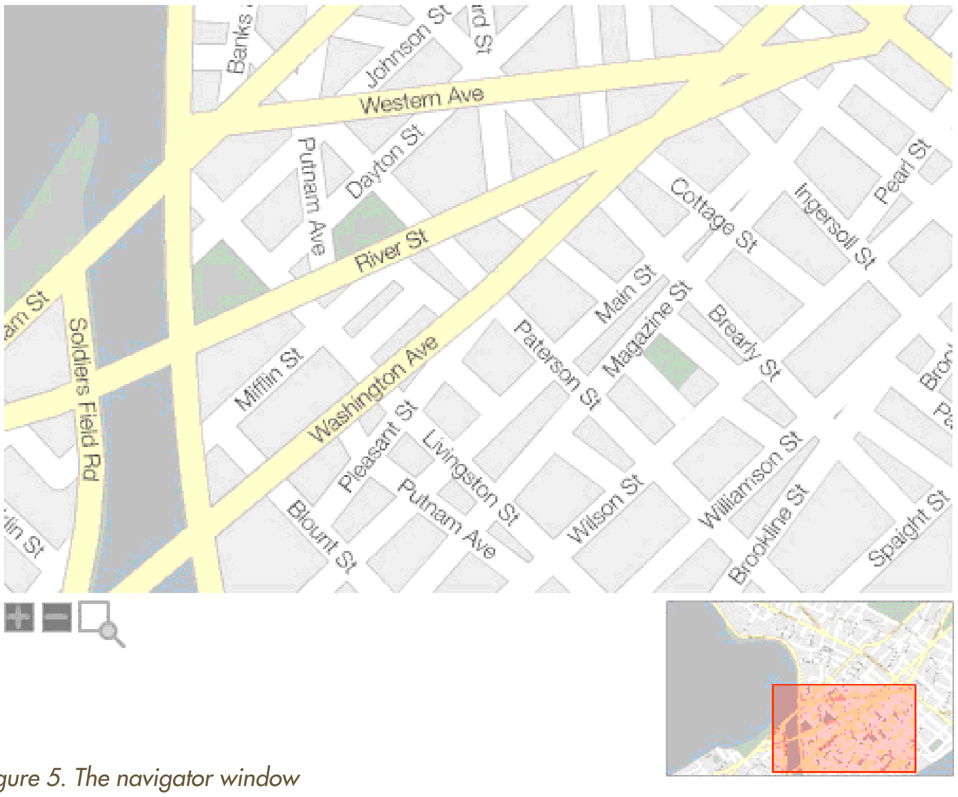


Figure 5. The navigator window

EXTENDING THE PANNING AND ZOOMING FUNCTIONALITY

This tutorial has covered the steps necessary to implement simple panning and zooming using the accompanying code base. Although only a handful of the code's methods specifically have been addressed here, the tutorial is intended to provide a basis for more complex and customized mapping by explaining a few key points. As the tutorial demonstrates, the provided ActionScript classes allow the generation of panning and zooming with only a few lines of code. The classes themselves are written with the expectation of further scrutiny from expert users, who can modify and extend the classes to suit their specific needs. To that end, the files contain explanatory comments within the code in an attempt to be as clear as possible about what exactly is happening in each section of code.

Although the tutorial is written for Flash CS4, the ActionScript classes in the **org/nacis/mapbrowsing** folder have no dependency on any single development environment, and can easily be used in Flex or pure ActionScript projects with Adobe Flex Builder or plain text compiled by an open-source compiler. Additionally, much of the code exhibits general concepts that can serve as a reference for writing the same functionality in other programming languages.

It is my hope that this tutorial, along with recent articles such as those by Roth and Ross (2009) and Peterson (2008), can help establish the new digital *Cartographic Perspectives* as a trusted distributor of programming-based web cartography resources, for the benefit of experienced and novice cartographers alike. With open community contributions, discussion, and updates such a repository can become more than a one-way broadcast of articles, allowing community members to build upon work like this tutorial and share those efforts.

REFERENCES

Harrower, M. and Sheesley, B. 2005. Designing better map interfaces: a framework for panning and zooming. *Transactions in GIS* 9(2):1–16.

Peterson, M. 2008. Choropleth Google Maps. *Cartographic Perspectives* 60:80–83.

Roth, R. E. and Harrower, M. 2008. Addressing map interface usability: learning from the Lakeshore Nature Preserve interactive map. *Cartographic Perspectives* 60:46–66.

Roth, R. E. and Ross, K. 2009. Extending the Google Maps API for event animation mashups. *Cartographic Perspectives* 64:21–40.