Extending the Google Maps API for Event Animation Mashups

Robert E. Roth | reroth@psu.edu
Kevin S. Ross | kevin.ross@psu.edu

GeoVISTA Center, Department of Geography
Penn State University
302 Walker Building
University Park, PA 16801

INTRODUCTION

There is an unfortunate preconception among trained cartographers that mashups—Google-based or otherwise—represent a bastardization of the discipline, with the overcrowded push-pin map becoming the straw man (or straw map) of every Cartography lecture and conference presentation. However, mashups are becoming ubiquitous and today are perhaps the most recognizable map form on the Internet. The cartographic community needs to get serious about mashups, rather than dismiss them as the flavor of the week. We should be the guiding voice, rather than the skeptical outsiders. To be fair, many trained cartographers are doing wonderful things with mashups, but even they need to be more active in extending the API and releasing their source code (for an example of good open source practice published in this journal, see Peterson, 2008). If we do not do this, all of the sound cartographic knowledge generated during the past century will be shredded by a cloud of pushpins.

When asked to participate in this special issue illustrating the benefits of the new open-access, web-delivered, all-digital format of *Cartographic Perspectives*, we felt that a techniques piece approaching this issue was a natural fit. In the following section, we provide a brief overview of mashups broadly and Google Maps mashups particularly. In the third section, we introduce event animation—our focus in this paper—and describe some of the best mashups supporting this technique. In the fourth section, we introduce and describe our contribution: a code library extending the Google Maps API to include event animation. A tutorial containing step-by-step instructions for using the code library is included as an appendix following the main article. We conclude the main body of the paper by offering a few positive words for the new *CP* format and the many possibilities made available through it.

GOOGLE MAPS MASHUPS

A *mashup* is an application, typically served on the web, that integrates elements from multiple sources to form a new service (Shneiderman and Plaisant, 2010). The term mashup originated in the music industry, where there is a longstanding tradition of sampling, remixing, and mashing song tracks. The *Grey Album* in particular (a mashup of the Beatle's *White Album* and Jay-Z's *Black Album*)—and the legal battle that ensued over its distribution—is cited as being particularly influential in placing the term mashup into popular consciousness (McConchie, 2008).

Mashups embody the Web 2.0 philosophy, making use of the Internet as a platform atop which disparate data sources and services can be stitched together in various ways according to user needs (or, equally as common, user whims) (O'Reilly, 2007). Although mashups do not need to be map-based, it is no wonder why map mashups are one of the most common variant of web mashups. Countless novice and expert mapmakers have experienced the 'a-ha' moment when mapping a dataset atop a basemap of familiar features or when charting multiple datasets in relation to one another. Often times, the map mashup is simply the extension of this practice to digital mapmaking. However, a map mashup can be so much more when the mapped geospatial information is combined with sophisticated, geographically-enabled web services. The map component of a mashup helps the user to submit the proper parameters to the web service by providing spatial context and to interpret the results of the web service by displaying the results spatially.

A map mashup can be so much more when the mapped geospatial information is combined with sophisticated, geographically-enabled web services

At the time of writing this piece, the two Google mapping platforms, Google Maps and Google Earth, dominate the map mashup world (Zang et al., 2008); our focus in this paper is on the former. While Google Earth enables a wider range of interactions with a three-dimensional digital globe, it is not natively browser-based, making it difficult to integrate web services (however, a new browser plug-in was recently released, so it will be interesting to see where this leads). A demonstration of the potential of Google Earth mashups for high-end cartography and visualization is provided by Wood and colleagues (2007).

The Google Maps mapping service was released formally in February of 2005. Soon after the release, numerous websites were established that hacked the service to display their own geospatial information atop the Google Maps tiles. In response to these hacks (and likely sensing opportunity), Google released the Google Maps Application Programming Interface (API) to facilitate development of Google Maps mashups (**http://code.google.com/apis/maps/**) (McConchie, 2008). An *API* is a set of code libraries made available by the developers of an application to allow others access to the services provided in the application (Boulos, 2005). Google provides an API in two different programming languages: JavaScript and ActionScript. The other major mapping services since have followed suit with the release of an API, including MapQuest, Microsoft Bing Maps (formerly Microsoft Virtual Earth), OpenLayers/OpenScales, OpenStreetMap, and Yahoo! Maps.

The Google Maps API is free as long as the mashup remains publically available and nonproprietary.

The Google Maps API is free as long as the mashup remains publically available and nonproprietary, in many ways answering the calls for a democratization of Cartography (Rød et al., 2001, Wood, 2003) and resolving the GIS and Society debate (Miller, 2006). Google Maps mashups can be run directly in the browser and do not require the user to download a plug-in when using the JavaScript API. The Google Maps API easily draws KML files, a format that is quickly becoming the PDF of spatial information (Harrower, 2009). There also are a wide array of resources for learning and using the Google Maps API, from the online code documentation (**http://code.google.com/apis/maps/documentation/**) to a large set of technical manuals (e.g., Gibson and Erle, 2006, Brown, 2006). There are now even online tools that assist the generation of mashups, such as GeoCommons Maker! (**http://maker.geocommons.com/**), effectively side-stepping the primary barrier to making mashups: requisite programming skills (Harrower et al., 2008).

EVENT ANIMATION & EVENT ANIMATION MASHUPS

Not all kinds of change are created equally, which means not all kinds of animation (a dynamic representation of change over time) are created equally. Andrienko and colleagues (2003) identify three basic kinds of temporal changes: existential changes (appearance and disappearance), changes of spatial properties (e.g., movement, expansion/contraction, shape change), and changes of thematic properties (i.e., changes in attribute value). *Event animation*, or more broadly event visualization, describes the first kind of temporal change, as an *event* is defined as some type of occurrence that exists only at one specific moment in time (e.g., it appears in one time slice and then disappears for the remainder of the animation) (Chung et al., 2005). An event often has spatial and attribute information associated with it, but these values do not change over time in the majority of event animations. Events are commonly represented as points (e.g., a disease incident, a violent crime, an earthquake); this makes event visualization particularly appropriate for the marker-heavy Google Maps platform.

There are a handful of existing, well-designed mashups that support event animation. A review of these applications is important for understanding what functionality should be included in an event animation code library. Our review includes Trulia Hindsight, AsthMap, SpatialKey, and our own DC Crime Visualization. Interestingly, only our DC Crime Visualization uses the Google Maps API, something that goes against the prevalence of the Google platforms revealed in a study completed by Zang et al. (2008).

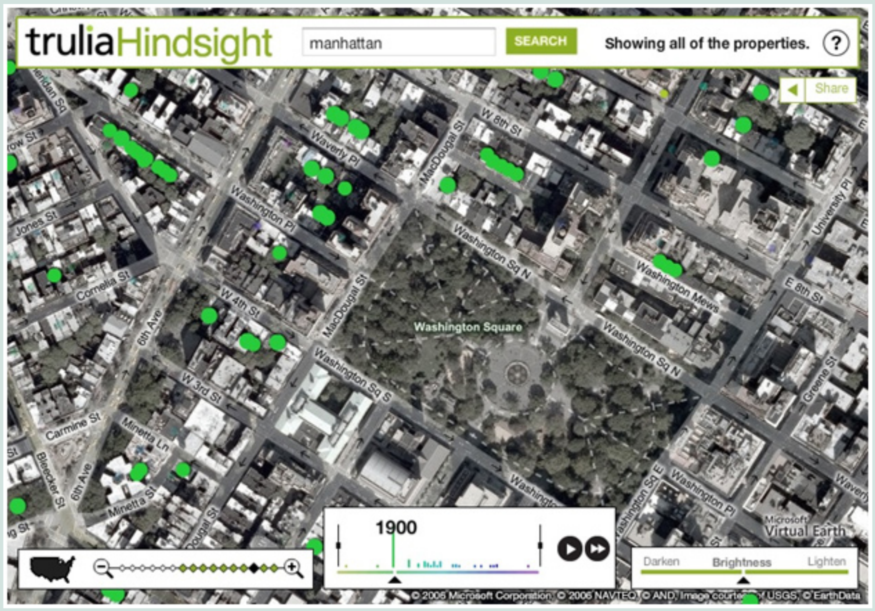

truliaHindsight
manhattan
SEARCH
Showing all of the properties.
Share
Washington Square
1900
Darken
Brightness
Lighten
Microsoft Virtual Earth
© 2006 Microsoft Corporation © 2006 NAVTEQ © AND, Image courtesy of USGS, © EarthData

Figure 1: Hindsight (Trulia) - http://hindsight.trulia.com

Trulia Hindsight: The Hindsight tool (http://hindsight.trulia.com/) developed by Trulia (best known for their real estate search engine) is a mashup that animates housing construction by year, providing prospective buyers a quick overview of the historical development of a neighborhood. Trulia Hindsight uses the Microsoft Virtual Earth platform; only the remotely sensed imagery basemap is made available. Events (construction of a new home) are represented by circles colored according to year built; these circles are not interactive. Basic VCR controls are provided to control the animation as well as a nice temporal legend that doubles as a histogram showing the event frequency for each year. This temporal legend is also interactive, allowing users to jump to a specific year using the slider control and to filter the animation to include only a subset of the years in the animation.

AsthMap: The AsthMap application (demo version with synthetic data available at http://indiemaps.com/asthMap/) developed by the University of Wisconsin-Madison Cartography Laboratory is a mashup for mapping and analyzing asthma exacerbations in space and time (Johnson et al., 2007). The tool was designed for use on an individual level (by doctors and patients) to improve treatment and on an aggregate level (by public health officials) to monitor environmental risk factors. Design of AsthMap was inspired by the Trulia Hindsight tool, although several important improvements were added. Like Trulia Hindsight, the Microsoft Virtual Earth platform

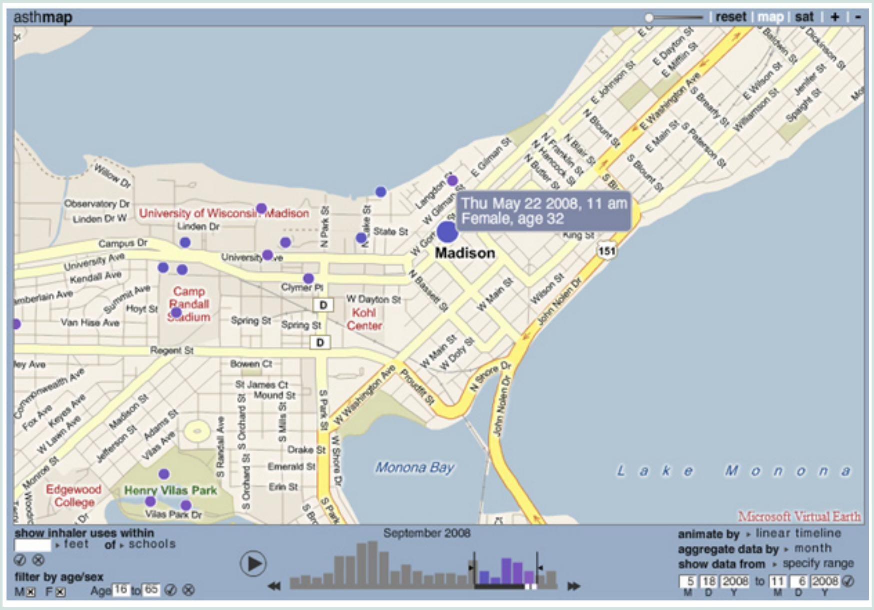


Figure 2: AsthMap (UW-Madison Cartography Laboratory)
http://indiemaps.com/asthMap/

was used, although users are provided the option of using an image or map background. Also like Trulia Hindsight, events (usages of GPS-enabled inhalers) are represented as circles colored according to the date. However, these circles are now interactive, allowing for users to retrieve attribute information about each event. Finally, AsthMap also includes similar VCR controls and an interactive histogram. However, much more powerful filtering (by space, time, sex, and age) and re-expression controls (either linear or composite animation binned by year, month, week, or day) are provided to improve interactive exploration.

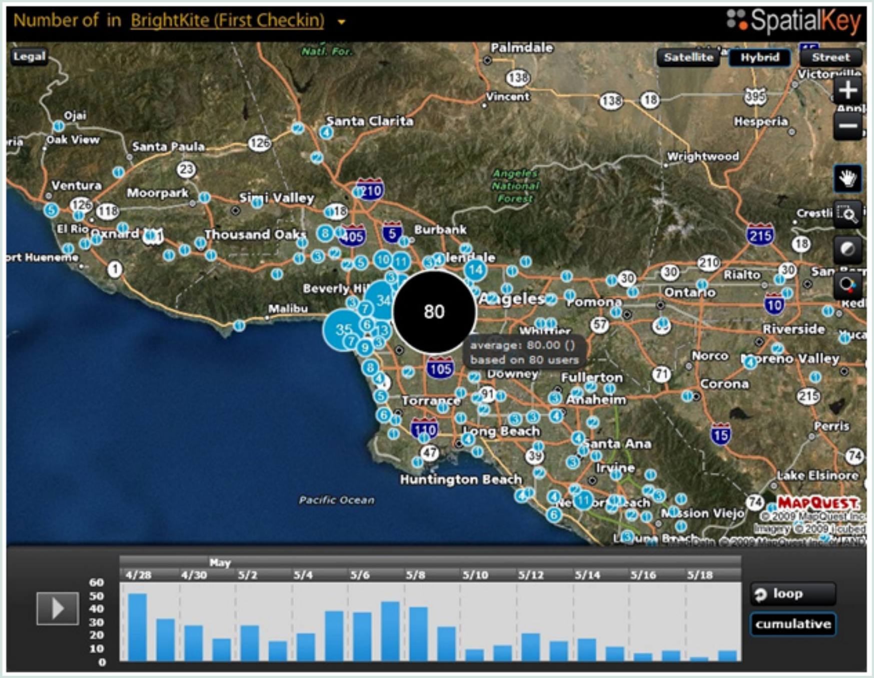
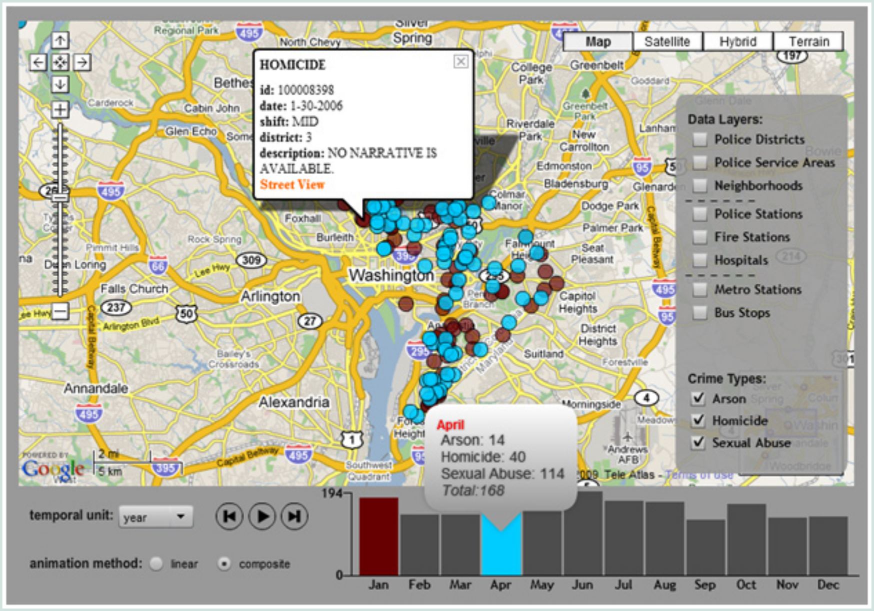


Figure 3: SpatialKey (Universal Mind) - (http://www.spatialkey.com)

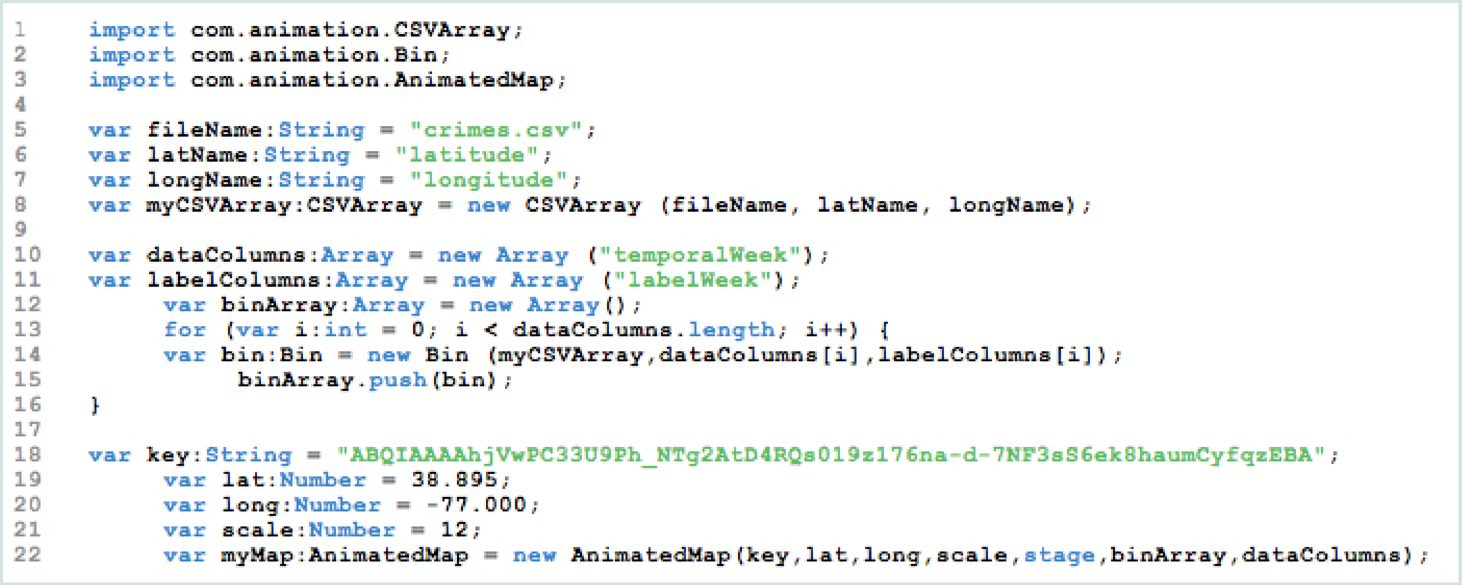
SpatialKey: SpatialKey (http://www.spatialkey.com), developed by Universal Mind, is a collection of mashup templates that provides a suite of visualization techniques for spatiotemporal information, event animation being only one of them (Johnson, 2008). SpatialKey uses the MapQuest platform and provides users with three basemap choices. There are two significant improvements included in the SpatialKey mashup that are not available in Trulia Hindsight or AsthMap. First, the SpatialKey tool allows users to upload their own data, effectively helping them make their own mashups. Once the data is loaded, users are provided with VCR controls and an interactive temporal legend similar to those found in Trulia Hindsight. Second, SpatialKey includes three representation techniques for aggregating individual events by proximity when the display becomes cluttered, a common concern with event visualizations. Aggregation techniques include a graduated circle view (shown in Figure 3), a heat map view (i.e., a kernel density estimation), and a heat grid view (similar to the chorodot map proposed by MacEachren and DiBiase, 1991).


HOMICIDE
id: 100008398
date: 1-30-2006
shift: MID
district: 3
description: NO NARRATIVE IS AVAILABLE.
Street View

April
Arson: 14
Homicide: 40
Sexual Abuse: 114
Total: 168

Figure 4: DC Crime Visualization (GeoVISTA Center) http://www.geovista.psu.edu/DCcrimeviz/

DC Crime Visualization: Drawing from these three examples, we developed an event visualization mashup of violent crimes in the District of Columbia (available at http://www.geovista.psu.edu/DCcrimeViz/) here at the GeoVISTA Center (Ross et al., 2009). The District of Columbia publishes violent crime incidents to their web-accessible data catalog site (http://data.octo.dc.gov/) in near real-time (one business day delay for processing). We have written a script to extract and clean information on select violent crimes (arson, homicide, sexual abuse) for plotting in the mashup. The mashup was developed using the Google Maps API for Flash and provides the basic basemap type toggling, KML layer toggling, and map browsing functionality included in the API. Like its predecessors, the DC Crime Visualization application implements VCR controls and an interactive temporal legend. Users can also change the animation method (linear or composite) and the binning unit (year, month, week, or day). Two new features are included in the mashup. First, we have extended the interactive temporal legend concept to include a temporal comparison feature. Two different time slices can be viewed simultaneously on the map by rolling over a different histogram bar in the temporal legend (shown Figure 4). We have also modified the pop-up info window to provide a link to the Google Maps Street View web service, allowing users to view the context of the crime. The code library described in the following section is based upon our DC Crime Visualization mashup.



```
import com.animation.CSVArray;
import com.animation.Bin;
import com.animation.AnimatedMap;

var fileName:String = "crimes.csv";
var latName:String = "latitude";
var longName:String = "longitude";
var myCSVArray:CSVArray = new CSVArray (fileName, latName, longName);

var dataColumns:Array = new Array ("temporalWeek");
var labelColumns:Array = new Array ("labelWeek");
     var binArray:Array = new Array();
     for (var i:int = 0; i < dataColumns.length; i++) {
     var bin:Bin = new Bin (myCSVArray,dataColumns[i],labelColumns[i]);
          binArray.push(bin);
}

var key:String = "ABQIAAAAhjVwPC33U9Ph_NTg2AtD4RQs019z176na-d-7NF3sS6ek8haumCyfqzEBA";
     var lat:Number = 38.895;
     var long:Number = -77.000;
     var scale:Number = 12;
     var myMap:AnimatedMap = new AnimatedMap(key,lat,long,scale,stage,binArray,dataColumns);
```

Figure 5: The code used in the tutorial example for replicating the functionality in the DC Crime Visualization. Although users of the library will need to change the parameters according to their own mapping context, no further scripting is necessary to make an event animation mashup.

EVENT ANIMATION CODE LIBRARY

When asked to participate in this special issue, we decided to refactor our code from the DC Crime Visualization to make it available as a library of stand-alone, object-oriented classes. We were guided by two primary objectives while completing this process. First, we wanted to make the new code library as easy to use as the Google Maps API itself so that novice programmers can still use it. Only a few lines of code are required for setting up a Google Maps mashup, most of which can be copied directly from their online tutorial. We challenged ourselves to maintain the same level of simplicity, limiting the number of required instantiations and making the parameters as logical as possible. Like Google, we also prepared an accompanying tutorial (see the appendix) that provides step-by-step instructions for setting up an event animation mashup. Figure 5 provides the code used in the tutorial example.

Second, we wanted to make the code library as flexible as possible to allow expert users to extend and revise our initial contribution to suit their needs. Our initial implementation is very basic, and, as demonstrated in the review of extant mashups, many other features can be added. It is our hope that our fellow cartographers will help us complete the library over time.

We have chosen to work with the Google Maps API for Flash because our own expertise matches better with ActionScript and because we feel that the Flash environment offers a wider range of possibilities particularly for animation, but also for interactivity. There are obvious drawbacks to this decision, namely that the development environment is proprietary (although a 30-day free trial of Flash or Flex can be obtained from the Adobe website) and that a browser plug-in is required to view Flash applications. Because of this, we encourage others to replicate our work for the JavaScript API.

Our code library contains six classes, each provided as a separate ActionScript (.as) file: `CSVArray`, `Bin`, `AnimatedMap`, `TemporalControls`, `TemporalLegend`, and `ToolTip`. Coding begins by importing three of the classes that are used in the main application (Figure 5, lines #1-3); the other three classes are called by the `AnimatedMap` class and therefore do not need to be imported.

We then require the user to enter parameters related to the CSV dataset and instantiate the `CSVArray` class (Figure 5, lines #5-8). The CSV file should contain three types of information for each event instance: (1) spatial information (the latitude and longitude of the event), (2) temporal information (one or several columns that include sequence information allowing for binning the events into a set of time intervals), and (3) attribute information (any other information that the developer wants to show in the information window when an event is selected on the map). The `CSVArray` object reads in a CSV file that is located at the URL entered into `fileName` parameter and processes it into a form understandable by ActionScript (an array of generic objects, with each object storing the attribute information for a single event). The `latName` and `longName` are the header names for the columns in the CSV file that contain spatial information. We require that this information is provided as geographic coordinates in units of decimal degrees, as this is what the Google Maps API requires. Also, we currently only support the CSV file format. Extension to other coordinate systems (e.g., UTM) or input formats (e.g., XML) are two examples where the cartographic community can help to build a robust code library for mashups.

We wanted to make the code library as flexible as possible to allow for expert users to extend and revise our initial contribution to suit their needs

After creating a `CSVArray` instance, the `Bin` instances are created (Figure 5, lines #10-16). The `Bin` constructor takes three parameters: the `CSVArray` instance (Figure 5, line #8), the header name of the column containing temporal information (Figure 5, line #10), and the header name of the column containing labels for the bins (Figure 5, line #11). We require that the column containing temporal information passed into the `dataColumns` array to be in a numerical format in order to automate the binning sequence (e.g., we require the user convert "Sunday", "Monday", "Tuesday" to 1, 2, 3 in the CSV file). The labels for each time slice are maintained by passing the original column into the `labelColumns` array. In the Figure 5 example, only a single `Bin` instance is created. However, any number of `Bin` instances can be generated for use in the mashup by simply adding more header names into the `dataColumns` and `labelColumns` arrays.

The `for` loop (lines #13-16) creates a separate `Bin` instance for each of the entries in the **`dataColumns`** array. The product of the `Bin` class is a two-dimensional array of index positions relating to the `CSVArray`; the first dimension stores the order of the bins, with a length equal to the number of time steps (e.g., a length of 7 for a composite week animation), and the second dimension stores the index positions of all events in the `CSVArray` instance that fall within the given bin. Therefore, each `Bin` instance contains the logic needed for a different animation. The set of `Bin` instances are stored in a regular array called `binArray` (Figure 5, lines #12 and #15) for use in the `AnimatedMap` instance. An interface widget is provided by the library in order to toggle among the animations included in the `binArray`.

Following creation of the `binArray`, the `AnimatedMap` class is instantiated (Figure 5, lines #18-22). The `AnimatedMap` class is our adaptation on the `Map` class included in the Google Maps API. Seven parameters are required to instantiate the `AnimatedMap` class: the Google Maps API key (Figure 5, line #18), the center latitude (Figure 5, line #19) and longitude (Figure 5, line #20) for the map, the scale of the map (Figure 5, line #21), a reference to the stage (a reserved word) so that the classes can speak to the specific application, the `binArray` array (Figure 5, line #12), and the `dataColumns` array (Figure 5, line #10).

The `AnimatedMap` class does much of the heavy lifting for the event animation extension. It first creates a `Map` instance using the inputted map parameters, providing logic for resizing the map, formatting markers placed on the map, and interacting with markers placed on the map. The `AnimatedMap` class then calls the `TemporalControls` class, which sets up the simple VCR controls and provides the logic for converting the two-dimensional arrays contained in the `Bin` instances into a working animation. Finally, the `AnimatedMap` class calls the `TemporalLegend` class to draw the histogram and attach the interactivity included in the DC Crime Visualization mashup (including making use of the `ToolTip` class for rolling over the histogram bars).

CONCLUSION

In this paper, we argue that the cartographic community needs to have an active voice in the design and implementation of map-based mashups, providing one small contribution ourselves by extending the Google Maps API for Flash to include event animation. Guidance must come not only in the form of reviews and critiques, but also in the development and public release of code libraries that extend the various map service APIs to include animation, interaction, and representation techniques established by academic and professional cartographers. 21st Century Cartography is as much about scripting as it is about graphic design. If the cartographic community hopes to continue to shape the way that maps are made and used in a positive way (why else would this community exist?), we need to acknowledge this fact and acquire the necessary skill sets to stay current.

We think that the new open-access, web-delivered, all-digital format of Cartographic Perspectives is reflective of the changing nature of Cartography and is a major step in the right direction. The open-access format should ensure that NACIS remains the meeting place for academic cartographers (who have free access to a university library and all its indexed journals) and professional cartographers (who do not have such access). Web-delivery, coupled with the almost unheard of six week review turn-around, means that content can remain current with the rapidly changing field. The all-digital format opens the possibility of publishing multimedia materials along with a traditional textual essay. The code library that we posted to the NACIS website is just one example of utilizing the new digital format—we envision the posting of other materials like geospatial datasets, experimental results, multimedia tutorials, and, most importantly, digital maps as well. In general, we are pleased with the progressive decision to change formats and wish the journal and NACIS success moving forward.

ACKNOWLEDGEMENTS

We wish to thank Craig McCabe for his early input into the DC Crime Visualization mashup, on which much of our coding is based. We also wish to thank Alan MacEachren for supervising development of the DC Crime Visualization and for commenting on early versions of the tutorial. Finally, we wish to thank Tanya Buckingham for putting together this outstanding special issue.

BIBLIOGRAPHY

ANDRIENKO, N., ANDRIENKO, G. & GATALSKY, P. (2003) Exploratory spatio-temporal visualization: an analytical review. Journal of Visual Languages and Computing, 14, 503-541.

BOULOS, M. N. K. (2005) Web GIS in practice III: Creating a simple interactive map of England's Strategic Health Authorities using Google Maps API, Google Earth KML, and MSN Virtual Earth Map Control. International Journal of Health Geographics, 4, 22.

BROWN, M. C. (2006) Hacking Google Maps and Google Earth, Indianapolis, IN, Wiley.

CHUNG, W., CHEN, H., CHABOYA, L. G., O'TOOLE, C. D. & ATABAKHSH, H. (2005) Evaluating event visualization: a usability study of COPLINK spatio-temporal visualizer. International Journal of Human-Computer Studies, 62, 127-157.

GIBSON, R. & ERLE, S. (2006) Google Maps Hacks, Sebastopol, CA, O'Reilly.

HARROWER, M. (2009) Cartography 2.0. http://cartography2.org.

HARROWER, M., HEYMAN, D., SHEESLEY, B. & WOODRUFF, A. (2008) Maker! Mapping the world's data. NACIS 2008. Missoula, MT.

JOHNSON, Z., HARROWER, M., MCGLYNN, E., ROTH, R., SICKLE, D. V. & WOODRUFF, A. (2007) Development of an online visualization tool for the mapping and analysis of asthma exacerbations in space and time. NACIS 2007. St. Louis, MO.

JOHNSON, Z. F. (2008) SpatialKey: Insanely good geovisualization. http://indiemaps.com/blog/2008/08/spatialkey-insanely-good-geovisualization/

MACEACHREN, A. M. & DIBIASE, D. (1991) Animated maps of aggregate data: Conceptual and practical problems. Cartographic and Geographic Information Science, 18, 221-229.

MCCONCHIE, A. L. (2008) Mapping Mashups: Participation, collaboration, and critique on the World Wide Web. PhD Dissertation, Geography. Vancouver, Canada, The University of British Columbia.

MILLER, C. C. (2006) A beast in the field: The Google Maps mashup as GIS/2. Cartographica, 41, 187-199.

O'REILLY, T. (2007) What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. Communications & Strategies.

PETERSON, M. (2008) Choropleth Google Maps. Cartographic Perspectives, 60, 80-83.

RØD, J. K., ORMELING, F. & ELZAKKER, C. V. (2001) An agenda for democratising cartographic visualisation. Norsk Geografisk Tidsskrift-Norwegian Journal of Geography, 55, 38-41.

ROSS, K. S., MCCABE, C. A. & ROTH, R. E. (2009) A near real-time visualization for understanding spatio-temporal patterns of violent crime in the District of Columbia. The Department of Homeland Security Summit. Washington, D.C.

SHNEIDERMAN, B. & PLAISANT, C. (2010) Designing the user interface: Strategies for effective human-computer interaction, Boston, MA, Addison-Wesley.

WOOD, J., DYKES, J., SLINGSBY, A. & CLARKE, K. (2007) Interactive visual exploration of a large spatio-temporal dataset: Reflections on a geovisualization mashup. IEEE Transactions on Visualization and Computer Graphics, 13, 1176-1183.

WOOD, M. (2003) Some personal reflections on change...The past and future of cartography. The Cartographic Journal, 40, 111-115.

ZANG, N., ROSSON, M. B. & NASSER, V. (2008) Mashups: Who? What? Where? Conference on Human Factors in Computing Systems (CHI '08). Florence, Italy, ACM.

TUTORIAL

Extending the Google Maps API for Event Animation Mashups: Tutorial

Robert E. Roth | reroth@psu.edu
Kevin S. Ross | kevin.ross@psu.edu

OVERVIEW

The following tutorial provides step-by-step instructions for using our library extending the Google Maps API for Flash to include event animation. We have written the tutorial in a way that does not require any experience programming with ActionScript3 (AS3) or with the Google Maps API for Flash. For the most part, the code we provide in the following code banks can be copied and pasted directly, only modifying the values of the input parameters according to your particular mapping context. Operational knowledge of AS3 and the API is needed to modify or extend our classes if custom functionality is needed. The Google Maps API for Flash reference is available here:

http://code.google.com/apis/maps/documentation/flash/reference.html

SETTING UP THE GOOGLE MAPS API FOR FLASH

INSTALL FLASH

Flash is a proprietary web authoring environment produced by Adobe as part of the Creative Suite. The Google Maps API works with any development environment using AS3 (Flash CS3 or sooner and all versions of Flex). The most recent version (CS4) is available for a 30-day free trial at:

http://www.adobe.com/downloads/

Figure 1 provides an annotated overview of the various interface panels of the Flash CS4 authoring environment. Depending on the panel configuration left by the last user of the application, you may see a different set of panels (e.g., some present or not present) or these panels may be in different locations on the interface. Flash panels are interactive; they can be dragged around the application or they can be closed completely. If you do not see a panel referenced in this tutorial, you can activate it using the **Window Tab** in the **Top Menu Bar**. Flash CS4 also allows you to tear-away and move all panels to a preferred configuration.

There are three important file types when authoring content in Flash. The project itself is stored in a file with the extension **FLA** - the FLA file saves all project content, settings, and code. In order to convert your Flash project into a file that can be viewed in the Adobe Flash Player, you must first publish it; this will result in the creation of a file with the extension SWF.

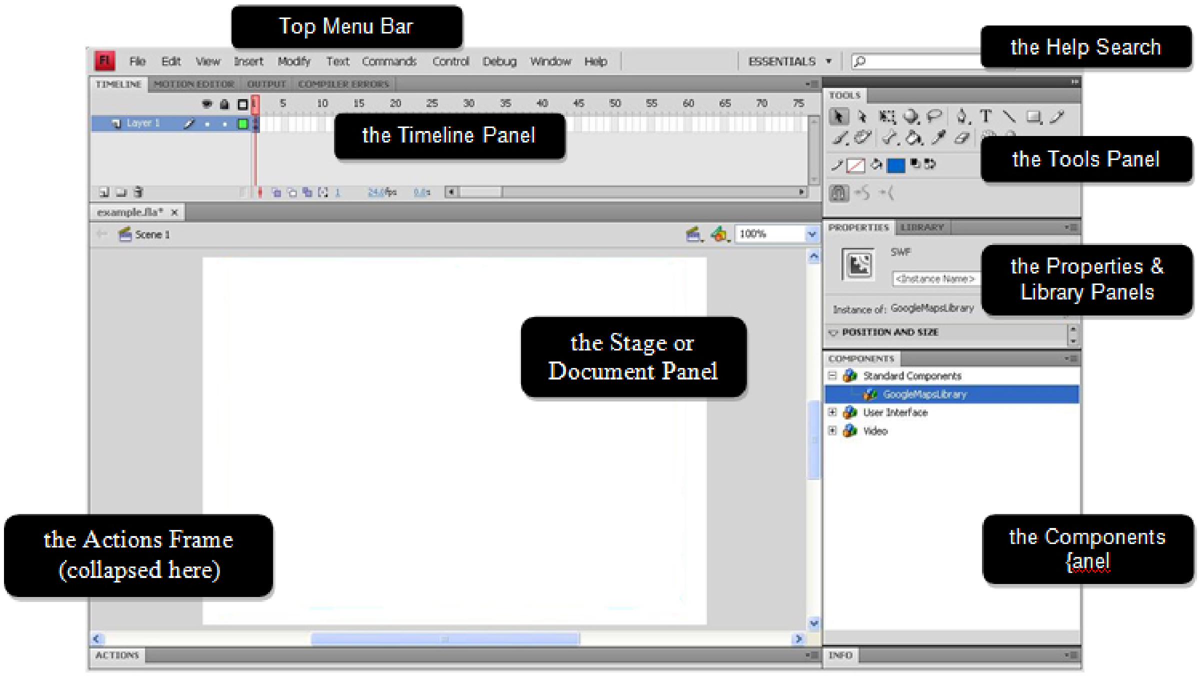


Figure 1 – The Flash authoring environment.

It is important that you save and back-up the FLA file, as this is needed to open the project in Flash CS4 and edit it. It is important that you upload the SWF file to your server space, as this is the format that can be viewed using the Flash Player plug-in. Finally, external classes are saved in files with the extension AS; our code library is provided as a set of AS files.

OBTAIN THE API KEY

Once Flash is installed, it is necessary to add the Google Maps API. Google requires you first to obtain an API key to use the Google Maps API for Flash so that they can ensure you are using the API in accordance with their licensing agreement. You need a Google account to obtain an API key. If you do not already have a Google account (e.g., Gmail), you will need to first sign up for one here:

https://www.google.com/accounts/ManageAccount

After signing into your Google account, you can obtain the API key here:

http://code.google.com/apis/maps/signup.html

Check the box stating you agree with the terms of use and enter the website location (URL) where you will be hosting your mashup (e.g., http://www.personal.psu.ed/rer198/). The API key itself is a long string of letters, numbers, and punctuation symbols. Be sure to record this key in a place where you will not lose it, as your mashup will not work unless you enter the key exactly as given to you.

INSTALL THE GOOGLE MAPS COMPONENT

The next step in setting up the Google Maps API for Flash is installation of the Google Maps component. A component is an AS3 class with related graphics and code designed to provide a user interface quickly. You can think of a component as a pre-programmed interface widget that can be instantiated in your application and then connected to your own data. Google follows this model by providing their mapping service as a custom Flash component.

The custom component must first be installed by copying the **SWC** file (Flash component file) into the Adobe Components Library. The SWC can be obtained at the following link:

http://maps.googleapis.com/maps/flash/release/sdk.zip

There are two SWC files in the zip folder, one for Flex (labeled so) and one for Flash (left unlabeled). Copy the SWC for Flash into the **Configuration\Components** directory. The following is an example for the Windows operating system:

C:\Program Files\Adobe\Adobe Flash CS4\language\Configuration\Components\

Note that your directory may be different depending on the version of Flash you are using, the location you choose when saving and installing Flash, and your operating system.

All installed components are available for use in the **Components Panel** in Flash. If the Components Panel is not currently visible, use the Window Tab at the top of the application to activate it. If you installed the Google Maps component properly, it will show up under the **Standard Components Tab** in the Component Panel. You will need to drag the Google Maps component from the Components Panel to the Library Panel in order to use it (see Figure 2).

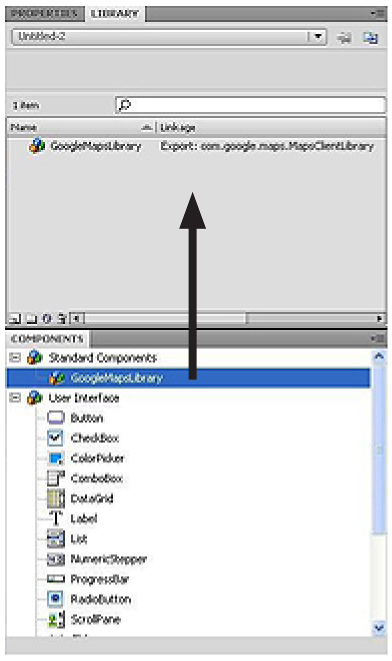
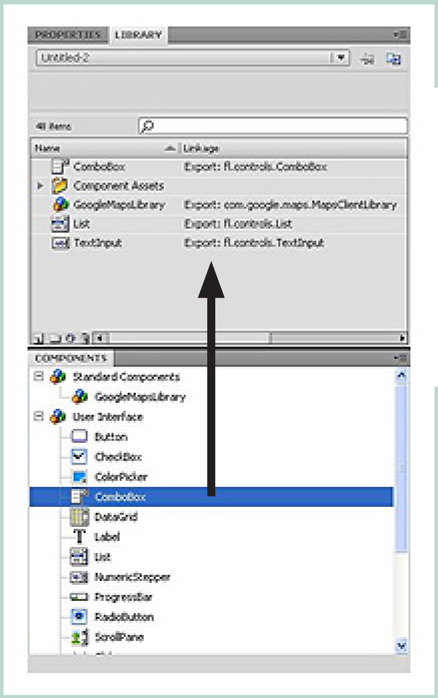


Figure 2 – The Google Maps component in the Components Panel. If the Google Maps component was installed correctly on your machine, it should show up under the Standard Components Tab in the Components Panel. You will need to drag and drop the GoogleMapsLibrary component from the Components Panel into the Library Panel in order to use it.



Name	Linkage
ComboBox	Export: fl.controls.ComboBox
Component Assets	
GoogleMapsLibrary	Export: com.google.maps.MapsClientLibrary
List	Export: fl.controls.List
TextInput	Export: fl.controls.TextInput

Figure 3 – Adding the ComboBox component to the Library. After installing the GoogleMapsLibrary component, it is necessary to add the ComboBox component to the Library. After dragging it from the Component Panel to the Library Panel, you should see the ComboBox in the Library plus several other assets and components used by the ComboBox component.

ADD THE COMBOBOX COMPONENT TO THE LIBRARY

One other component required for your project is the ComboBox component, which will enable toggling among multiple animations. The ComboBox component can be found in the **User Interface** folder in the Components Panel. To add this component to your project library, make sure the Components Panel and the Library Panel are both visible. Locate the ComboBox component in the Components Panel and drag it into the Library Panel. This will allow the component to be instantiated when using the library. Figure 3 shows the result of this action.

ALLOW THE FLASH PLAYER TO COMMUNICATE WITH GOOGLE

The final step is to ensure that your Flash Player plug-in allows a local connection. To adjust this setting, visit:

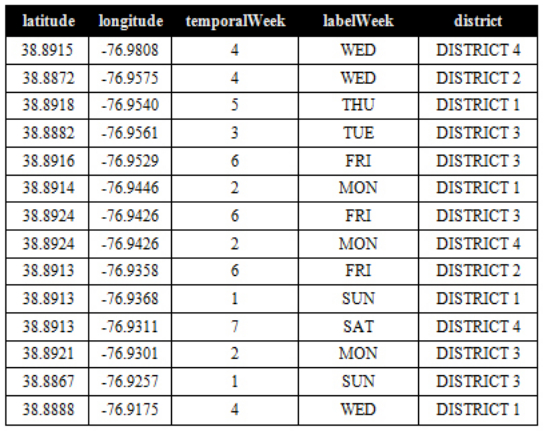
http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04a.html

Select the "Always allow" option. This allows Flash Player to connect to datasets and services that are available online. If you do not do this, you will not be able to test your application locally (i.e., without first uploading it to your webspace).

FORMATTING THE CSV FILE

ADD COLUMN HEADER NAMES

Our current library only supports run-time loading of data in a CSV format. The library requires each column in the CSV file to be formatted in a specific way depending on if it contains spatial, temporal, or attribute data. All columns, regardless of type, require a column header. Therefore, the first step in formatting your data is to add a row to the top of the file that contains a name for the column. This name must be formatted as text and cannot contain commas. Table 1 provides an abbreviated example dataset in Microsoft Excel prior to conversion to the CSV format (File/Save As...).



latitude	longitude	temporalWeek	labelWeek	district
38.8915	-76.9808	4	WED	DISTRICT 4
38.8872	-76.9575	4	WED	DISTRICT 2
38.8918	-76.9540	5	THU	DISTRICT 1
38.8882	-76.9561	3	TUE	DISTRICT 3
38.8916	-76.9529	6	FRI	DISTRICT 3
38.8914	-76.9446	2	MON	DISTRICT 1
38.8924	-76.9426	6	FRI	DISTRICT 3
38.8924	-76.9426	2	MON	DISTRICT 4
38.8913	-76.9358	6	FRI	DISTRICT 2
38.8913	-76.9368	1	SUN	DISTRICT 1
38.8913	-76.9311	7	SAT	DISTRICT 4
38.8921	-76.9301	2	MON	DISTRICT 3
38.8867	-76.9257	1	SUN	DISTRICT 3
38.8888	-76.9175	4	WED	DISTRICT 1

Table 1 – An sample dataset using the required format. All columns in the CSV require a text header in the first row. Columns containing spatial data must use geographic coordinates reported using decimal degrees. Columns containing temporal data must be formatted numerically reflecting the sequence of bins. Columns containing attribute data can be formatted as text or numbers, but cannot contain commas.

FORMAT THE COLUMNS WITH SPATIAL DATA

The Google Maps API requires spatial data to use geographic coordinates using decimal degrees as the unit. It also uses the negative sign prefix (-) to represent latitude coordinates in the southern hemisphere and longitude coordinates in the western hemisphere.

FORMAT THE COLUMNS WITH TEMPORAL DATA

Columns containing temporal data must be formatted as numbers in order for the code library to predict the sequence of time steps (e.g., convert "SUN", "MON", "TUES" to 1, 2, 3). We have set up the code library so that you can still use the original names for labeling the temporal legend as long as it is included in a separate column. In the Table 1 example, the temporalWeek column includes the temporal data for the animation and the labelWeek column includes the associated labels.

All rows with the same number in the temporal column are placed in the same time slice or bin. This flexible format allows for the generation of linear and composite animations using any number of binning units; non-temporal animations can also be created using the numeric conversion as a way to determine the binning order.

FORMAT THE COLUMNS WITH ATTRIBUTE DATA

The code library is written so that any attribute column is displayed in the pop-up information window when clicking the marker. These columns can be text or numerical. The only restriction is that commas cannot be included in any text strings, as the CSV format uses these to discriminate individual records in the data structure.

USING THE EVENT ANIMATION LIBRARY

ACQUIRE THE LIBRARY

The complete library is available as a zip file from the following site:

http://www.nacis.org/CP/CP64/com.zip

Extract the zip file into the same folder that contains your saved Flash project (FLA). You should notice that the files are in the directory **com\animation** – the com folder needs to be in the same folder as your FLA file for the libraries to import properly. The animation folder contains six files:

1. **CSVArray.as:** A class for ingesting a CSV file at run-time and converting it into a usable format

2. **Bin.as:** A class for partitioning the entries in the CSV file according to an attribute, producing a set of bins (i.e., time slices) for the animation

3. **AnimatedMap.as:** A class for loading the Google Maps map service, setting the layout, and instantiating the `TemporalControls` and `TemporalLegend` classes

4. **TemporalControls.as:** A class for drawing and programming the VCR controls and animation ComboBox selection

5. **TemporalLegend.as:** A class for drawing and programming the interactive temporal legend

6. **ToolTip.as:** A class for generating information windows when rolling over the bars in the temporal legend

IMPORT THE CLASSES

The first step in using the code library is importing three of six classes (the remaining three are called directly or indirectly by the `AnimatedMap` class). The `import` function is necessary when using classes outside of the Flash base libraries. You can copy the lines in CodeBank 1 directly into the **Actions Panel**.

```
import com.animation.CSVArray;
import com.animation.Bin;
import com.animation.AnimatedMap;
```

CodeBank 1 – Importing the classes.

INSTANTIATE THE CSVARRAY CLASS

The next step is to instantiate the `CSVArray` class. The `CSVArray` constructor takes three parameters: the name of the CSV file (if it is not in the same location as the deployed SWF, the full path is required), the header name of the column containing the latitude information, and the header name of the column containing the longitude information. The parameters must be passed into the `CSVArray` constructor in this order. You can copy the lines #5-8 from CodeBank 2, although the "___" strings included for the three parameters must be filled in with the appropriate information about your dataset. Multiple CSV files can be loaded into a single application if necessary, each requiring its own `CSVArray` instance (thus repeating lines #5-8).

```
import com.animation.CSVArray;
import com.animation.Bin;
import com.animation.AnimatedMap;

var fileName:String = "___";
var latName:String = "___";
var longName:String = "___";
var myCSVArray:CSVArray = new CSVArray (fileName, latName, longName);
```

CodeBank 2 – Instantiating the CSVArray class.

INSTANTIATE THE BIN CLASS

Once the CSV file is loaded, it can be partitioned to produce the animation by instantiating the `Bin` class. The `Bin` class takes three parameters: the `CSVArray` instance containing the data to be partitioned, the header name of the column used for the partitioning, and the header name of the column used for the X-Axis labels. CodeBank 3 shows the setup required for instantiating two `Bin` instances. Any number of `Bin` instances can be created depending on the number of desired animations. The `dataColumns` array and the `labelColumns` array simply need to be defined as the list of column header names used as the data and corresponding labels for each animation, respectively. The `dataColumns` and `labelColumns` array are associated in that the column entered in the first index position of the `dataColumns` array will be labeled by the

```
import com.animation.CSVArray;
import com.animation.Bin;
import com.animation.AnimatedMap;

var fileName:String = "___";
var latName:String = "___";
var longName:String = "___";
var myCSVArray:CSVArray = new CSVArray (fileName, latName, longName);

var dataColumns:Array = new Array ("___","___");
var labelColumns:Array = new Array ("___","___");
var binArray:Array = new Array();
for (var i:int = 0; i < dataColumns.length; i++) {
      var bin:Bin = new Bin (myCSVArray,dataColumns[i],labelColumns[i]);
      binArray.push(bin);
}
```

CodeBank 3 – Instantiating the Bin class.

column entered in the first index position of the `labelColumns` array, and so on. The column entered at a given index position should be the same in both the `dataColumns` and `labelColumns` array if you do not want to specify a separate label column. Both arrays must have the same length. For lines #10 and #11, you will need to replace the “___” with the appropriate column header names from the CSV file.

After specifying the `dataColumns` and `labelColumns` array, the for loop (lines #13-16) creates the `Bin` instances and stores them in an array called `binArray`. The first `Bin` included in the array will be set as the default. Lines #12-16 can be copied directly into your code.

INSTANTIATE THE ANIMATEDMAP CLASS

The final step in using the library is to instantiate the `AnimatedMap` class, as shown in CodeBank 4. The `AnimatedMap` class takes seven parameters: the Google Maps API key (line #18), the latitude of origin (line #19), the longitude of origin (line #20), the default scale (line #21), a reference to the main stage (the keyword stage should be submitted here in all cases), the `binArray` (line #15), and the `dataColumns` array (line #10). Default values are currently put in for the lat, long, and scale variables; adjust these according to your specific mapping context.

```
1     import com.animation.CSVArray;
2     import com.animation.Bin;
3     import com.animation.AnimatedMap;
4
5     var fileName:String = “___”;
6     var latName:String = “___”;
7     var longName:String = “___”;
8     var myCSVArray:CSVArray = new CSVArray (fileName, latName, longName);
9
10    var dataColumns:Array = new Array (“___”,“___”);
11    var labelColumns:Array = new Array (“___”,“___”);
12    var binArray:Array = new Array();
13    for (var i:int = 0; i < dataColumns.length; i++) {
14           var bin:Bin = new Bin (myCSVArray,dataColumns[i],labelColumns[i]);
15           binArray.push(bin);
16    }
17
18    var key:String = “___”;
19    var lat:Number = 0;
20    var long:Number = 0;
21    var scale:Number = 1;
22    var myMap:AnimatedMap = new AnimatedMap(key,lat,long,scale,stage,binArray, dataColumns);
```

CodeBank 4 – Instantiating the AnimatedMap class.

```
import com.animation.CSVArray;
import com.animation.Bin;
import com.animation.AnimatedMap;

var fileName:String = "crimes.csv";
var latName:String = "latitude";
var longName:String = "longitude";
var myCSVArray:CSVArray = new CSVArray (fileName, latName, longName);

var dataColumns:Array = new Array ("temporalWeek");
var labelColumns:Array = new Array ("labelWeek");
var binArray:Array = new Array();
for (var i:int = 0; i < dataColumns.length; i++) {
      var bin:Bin = new Bin (myCSVArray,dataColumns[i],labelColumns[i]);
      binArray.push(bin);
}

var key:String = "ABQIAAAAhjVwPC33U9Ph_NTg2AtD4RQs019z176na-d-7NF3sS6ek8haumCyfqzEBA";
var lat:Number = 38.895;
var long:Number = -77.000;
var scale:Number = 12;
var myMap:AnimatedMap = new AnimatedMap(key,lat,long,scale,stage,binArray,dataColumns);
```

CodeBank 5 – An example project filling in the parameters using the data from Table 1.

After entering in these parameters, you should be able to publish the file into the SWF format by selecting the **Publish** option under the **File Tab** along Top Menu Bar (or by pressing CTRL+ENTER).

CodeBank 5 shows the filled in code related to the dataset shown in Table 1. Here, only a single animation is created, using the temporalWeek and labelWeek columns. The published SWF file using this code is shown in Figure 4.

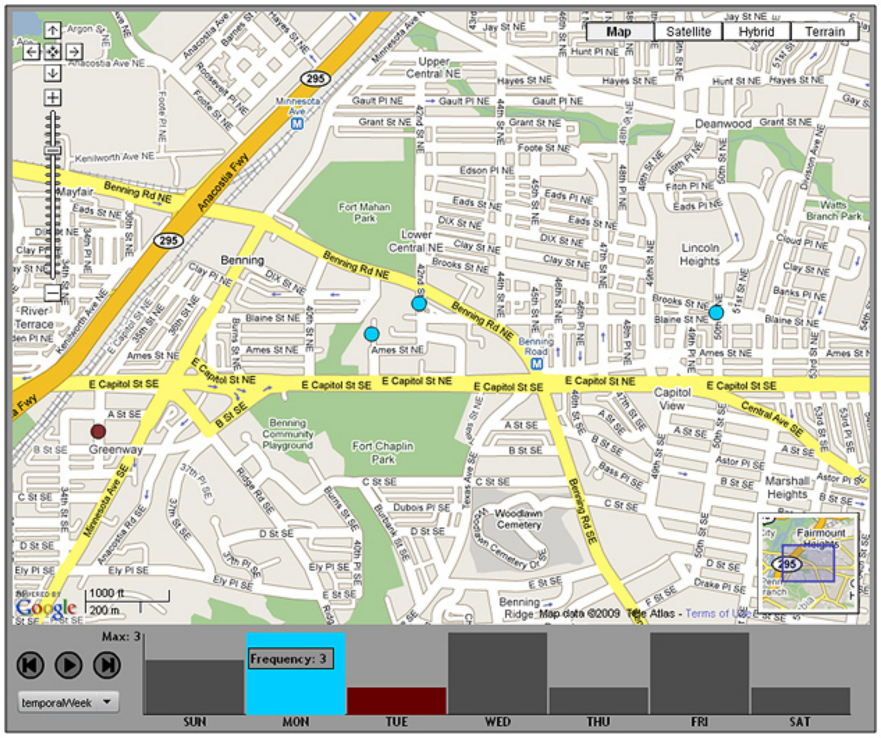


Figure 4 – The interface included by default