

Map Making for the Web with ArcGIS using Flex App Builder

Bryan Goodrich

This project aimed to demonstrate how to make a basic web application using Flex App Builder. Included will be one geoprocessing task for buffering a point layer. Also, the steps to installing a custom widget with Flex Viewer are demonstrated. The result is a working website with customized geoprocessing task and widget.

The purpose of this web application was to present the steps and problems one may face when deploying a web map to a Flex framework. During the initial steps of developing this website, it was discovered that the Flex App Builder would be required. A prerequisite to using this software is to have Adobe Air installed. Admin rights are required. Once Flex and App Builder are available on the system, one can use ArcGIS to deploy their map and geoprocessing services and utilize them through Flex.

The Map Document

The map document for this project contained 2 spatial layers and 1 tool layer. The spatial layers were the Sacramento Regional Transit light rail line feature and light rail station houses point feature. Both of these are available from the Sacramento County or SACOG websites. The point feature would be used in the tool layer for buffering a specified distance.

The Tool Layer

The geoprocessing task was straight-forward. Two directories adjacent to the map document were created: scratch and schema. Within the schema directory a file geodatabase was created with a generic point feature class. The model properties were set to use the scratch folder for

the workspace and scratch space. The output from the tool was set to use this space with the variable declaration

```
%scratchworkspace%\BufferedPoints.shp
```

Two parameters were defined for this tool: the input feature and the buffer distance. The data type for the distance was an aerial unit. The data type for the input was a feature set. It's schema was set to the generic point feature within the geodatabase in the schema folder.

The tool was tested, worked, and loaded into ArcMap as a tool layer. From here, the MXD was right-clicked, and "publish" was selected. Using the default options, this produced a map service, a geoprocessing service, and a KML service.

Setting up the Services

Once the services are started on the server, viewable in ArcCatalog, one needs to know the REST URLs to those services. These are accessible at

```
(1) http://<server-name>:8399/rest/services
```

When working with the REST services and App Builder, it proved useful to keep open a window to

(2) `http://<server-name>:8399/rest/admin`

This URL takes the developer to a login screen that gives them access to a number of options. The important one is caching management. This lets you set up how often, if ever, the server clears the cache, or lets you clear the cache manually. This is necessary if you update (restart) your services and try to load them into App Builder because the REST services will not be updated from the cache. The active ones should be viewable in (1).

The other URL that needs to be known is the Flex Viewer. This is usually deployed directly off the root as

(3) `http://<server-name>:8099/flexviewer/`

Using App Builder

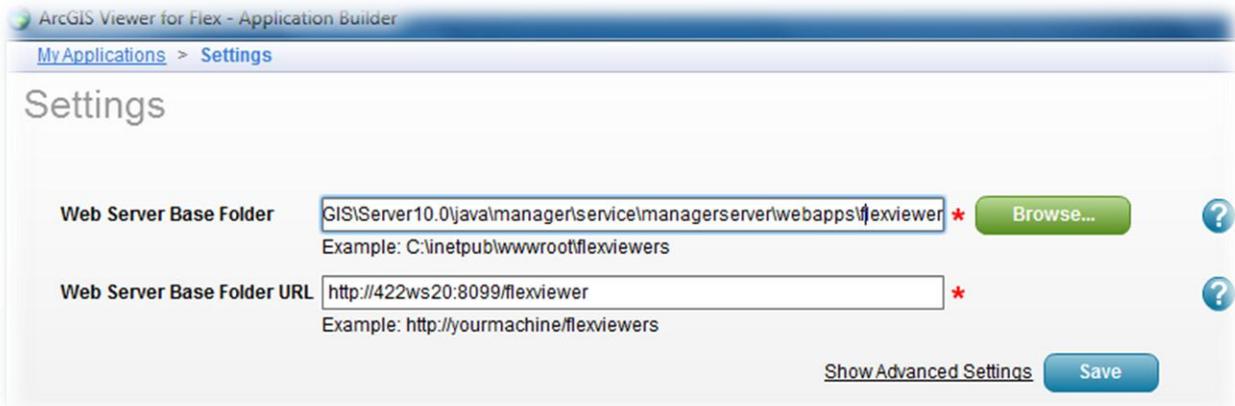
App Builder starts up requiring two locations to work. The first is (3) indicated above. This tells App Builder where to deploy its applications you develop. Each app, whatever it is named, will be appended to (3).

The other location required is the path to the flexviewer folder in your ArcGIS server. For this project, it is deployed as a Java app within the java

folder within the services folder: `C:\Program Files`

`(x86)\ArcGIS\Server10.0\java\`

`manager\service\managerserver\webapps\flexviewer`

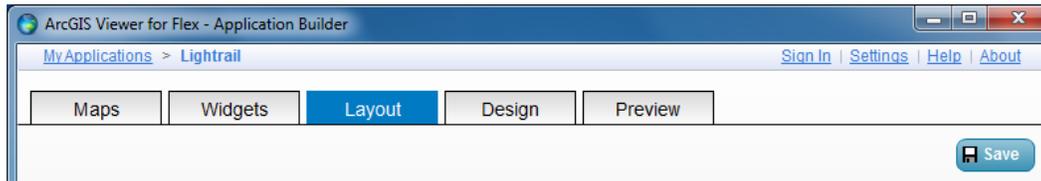


Custom Widgets

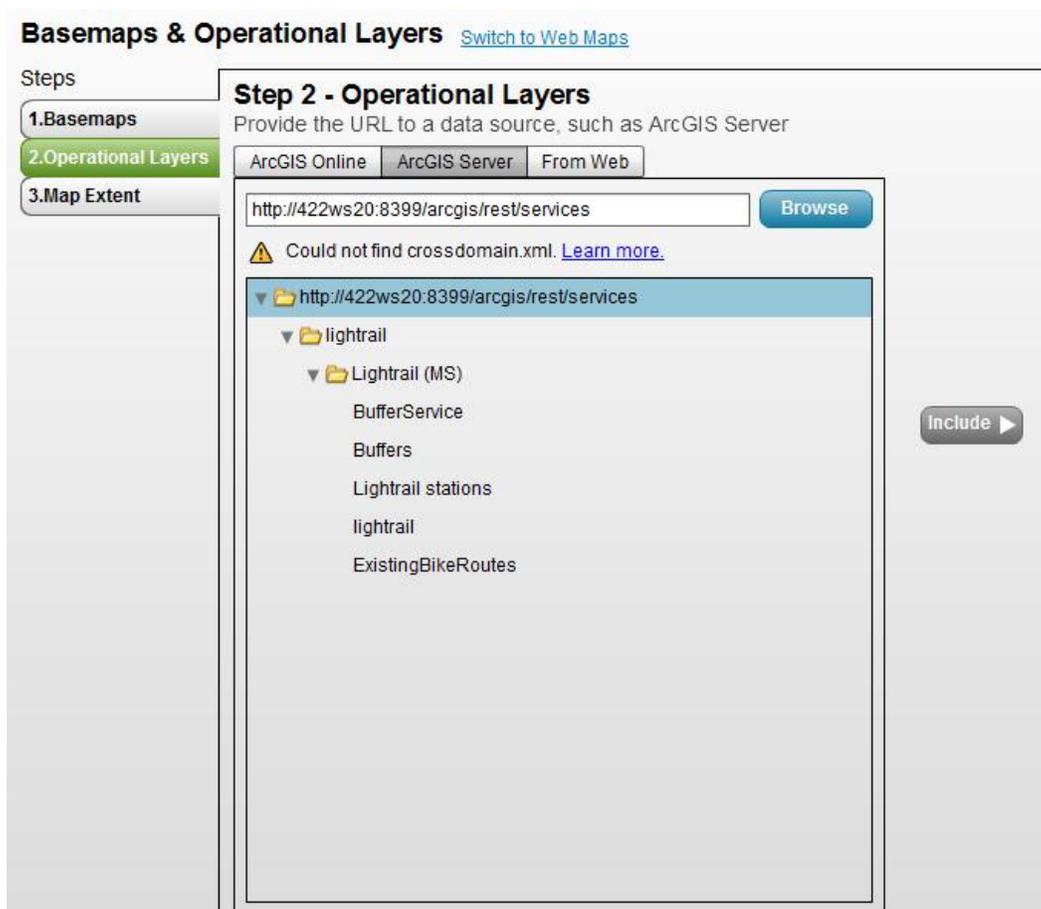
The custom widgets can be installed from the settings page displayed above. By clicking "Show Advanced Settings" the option to install custom widgets is available. When you download a widget online, it comes as a zipped archive you can extract anywhere. Included is a Readme with instructions. They tell you which folder contained in your extract that you need to point App Builder toward to install the custom app. From there, it installs and shows up in your library of widgets shown below.

Setting up the Application

Once App Builder is ready to develop the application, there are a series of tabs along the top of the program.



The “Maps” tab is where you can set up the base maps, the operational layers, and the extent at which your viewer begins. For this project, the default base maps were selected with the check box for ESRI base maps to be available. This gives the user the ability to select their choice of basic ESRI base maps in the application. Alternatively, the developer could select options from ArcGIS Online and WMS services (KML only).



The important step is accessing the map services provided by the deployed map. The operational layers lets the developer select from ArcGIS Online and from WMS services (KML only). The ArcGIS Server option lets the user paste the URL to the REST services in (1) to load the available layers to include. The button "include" puts the selected layers into a list on the right-hand side of the tab screen. From there, the user can adjust properties (e.g., transparency of the layer) or order them in the table-of-contents. After that, the extent can be set for this application.

Using Widgets

Widgets make a great way to leverage advanced functionality in a natural way with Flex. Luckily, the Flex developer community provides some of these for free. One used for this project was the Heat Map tool. The properties of the tool are designed for its own data. The XML configuration file needs to be edited, which is accessible from the Widgets screen.

Maps

Widgets

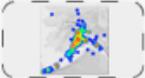
Layout

Design

Preview

Configure 1

Widget label

Widget icon 

Open at start

[Learn more about Heat Map widget](#)

Widget Settings

```
<configuration>
  <layers>
    <layer>
      <name>Lightrail Station Heatmap</name>
      <url>http://422ws20:8399/arcgis/rest/services/lightrail/Lightrail/MapServer/2</url>
      <outFields>ACTIVE,LINE,NAME,STATUS,STREET_NUM,STREET_NAM,STREET_TYP,SITE_CITY,SITE_ZIP</outFields>
      <useproxy>>false</useproxy>
      <useamf>>true</useamf>
      <defaultquery>1=1</defaultquery>
      <querytemplate>ACTIVE = REPLACE_STR</querytemplate>
      <types>'Yes','No'</types>
      <type_descs>ACTIVE = Yes,ACTIVE = No</type_descs>
    </layer>
  </layers>
  <ui>
    <widget_dimensions>
      <width>378</width>
      <height>206</height>
    </widget_dimensions>
    <icons>
      <base_url>widgets/HeatMap/assets/</base_url>
      <main_icons>rainbow.png</main_icons>
    </icons>
  </ui>
</configuration>
```

OK

Cancel

The widget comes with some basic properties that can be adjusted, such as the label, the icon, and whether or not to display it on application start up. There are many options to tweak within the XML, but only the first block within the "layer" node needs to be adjusted to work with the data. Four lines need to be adjusted.

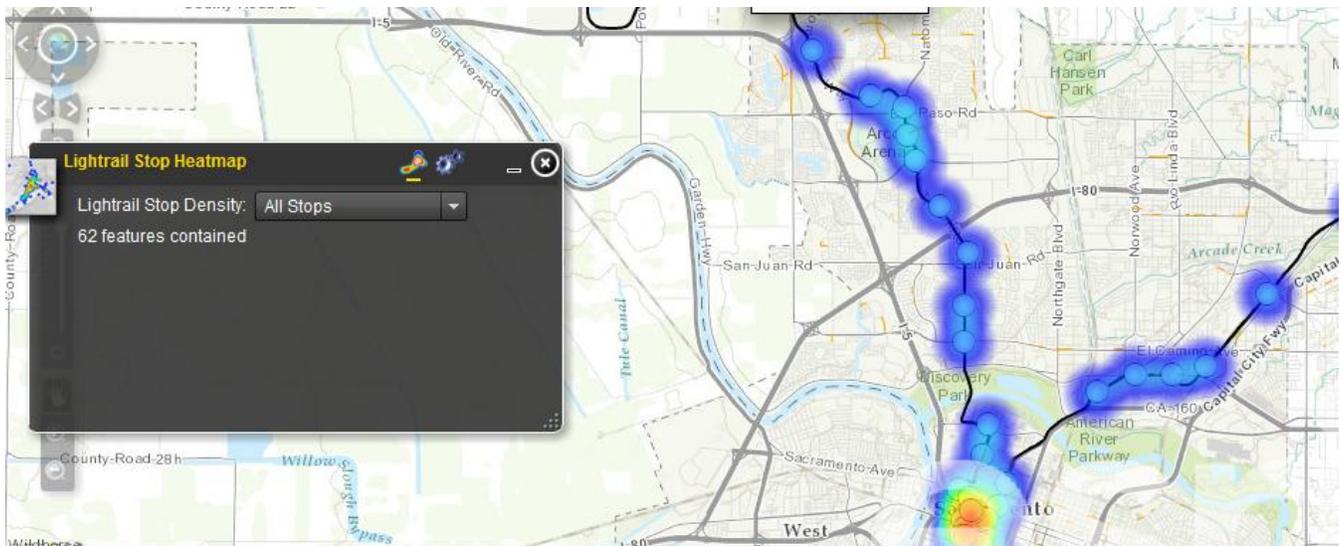
First, the "name" tag specifies how it shows up in the window on the app. The "url" tag needs to be set to the specific REST service URL for the point layer to be operated on. The "outFields" tag is a comma delimited list of field names from that REST service. Use its URL page to see what fields are available. At least the one to be used in the query needs to be present.

The next 3 tags to be altered involve the querying used in the widget. The "querytemplate" tag needs to have the REST service field in the layer that is to be queried. In this project, it was the ACTIVE field that specifies yes or no just in case that light rail station is an actively used station. The REPLACE_STR is a variable that is replaced by a value when one of the options to be discussed below are selected.

The "types" tag is a comma delimited list of options to be used in place of REPLACE_STR. The data type for this query is important. The

default earthquake data used with this widget uses numeric values with inequalities. This project is using strings and equality. Double-quotes *do not work*. The strings 'yes','no' must be single-quotes. This list corresponds to what the user will see in the widget window of the app. These are set by the "type_descs" tag which is a comma separated list of the expressions that shows in the widget dropdown list.

Once an option is selected, the corresponding "type" list value is put into the REPLACE_STR variable to complete the query. The widget will update the heat map to only operate on the points that satisfy the query.



Geoprocessing Widget

To deploy the geoprocessing service requires the geoprocessing widget.

This widget appears to be relatively new to App Builder (like, Summer 2012). It works in a similar way to deploying operational layers. Required is the REST geoprocessing service URL. This is loaded in the widget options and it will populate the input and outputs recognized by the service.

The screenshot shows the 'Configure 5' dialog for a Geoprocessing Widget in the ArcGIS Viewer for Flex - Application Builder. The 'Widgets' tab is selected. The widget label is 'Buffer Tool'. The widget icon is a gear icon, and a 'Browse' button is available. The 'Open at start' toggle is set to 'Off'. A link to 'Learn more about Geoprocessing widget' is provided. The 'Widget Settings' section shows the 'Task URL' as 'http://422ws20:8399/arcgis/rest/services/lightrail/Lightrail/GPService/BufferService' and a 'Load' button. Below this, there is a table with columns for Name, Label, Type, Visible, Required, and Action. The table contains two rows of input/output information.

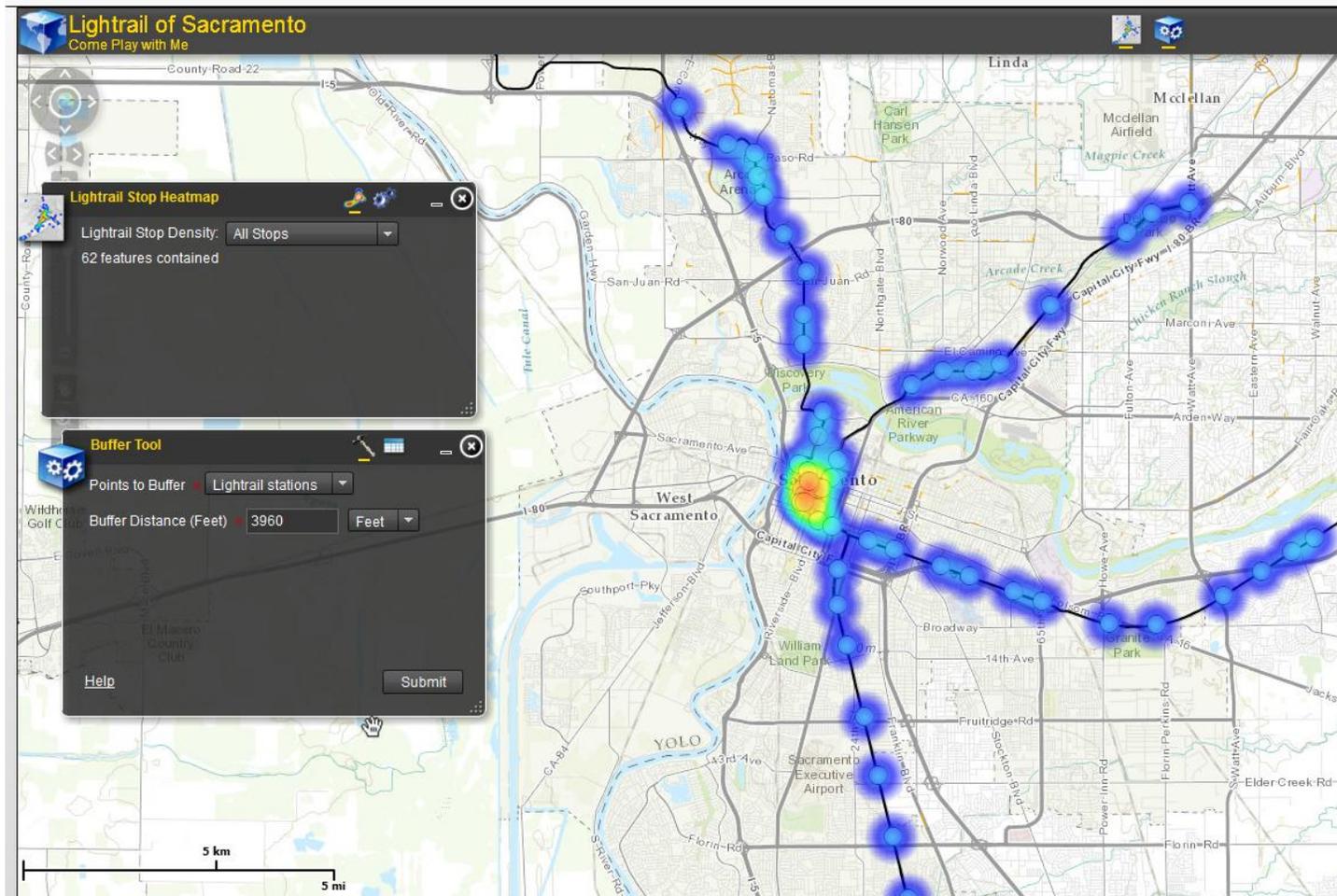
Name	Label	Type	Visible	Required	Action
Points_to_Buffer	Points to Buffer	featurerecordset	true	true	
Buffer_Distance__Feet_	Buffer Distance (Feet)	linearunit	true	true	

Like the operational layers, the inputs and outputs can have options set for them, depending on their data type. For this buffering tool, the option to let the user select points to buffer or use a feature layer can and should be set appropriately. Since this project aimed to buffer the light rail stations, the option was set to use a feature layer for the input `Points_to_Buffer`. Other basic widget options (title and emblem) are available.

Running the Application

The other tabs in the App Builder can be used to customize what is included on the display (Layout) or how things are displayed (Design), such as fonts and colors. However, if one wants to see their website in action, they need only click on the Preview tab.

Included in the preview are the name of the application and the URL to the application (as a hyperlink). The preview is fully functional as-if it were loaded in a web browser. Since it is entirely developed out of Flex, the experience from the preview to different browsers should be relatively consistent.



For this project, the Buffer Tool runs with a default buffer distance set. When it completes, buffers are displayed as blue circles (not shown). The output circles were set using a defined layer symbology that was

output and used in the options in the Model for the output layer. These buffers do not show up in the table-of-contents, however. Instead, they appear as long as the tool window is not destroyed.

Complications and Conclusions

This project did not attempt to accomplish grand complex geoprocessing tasks nor attempt to leverage various possible WMS KML services out there on the web. The aim was to keep it simple and make it work using a new emerging technology that can be used in the future to improve the deployment of ArcGIS web applications and make for a more consistent and functional interface.

Besides complications with the server, understanding how a tool layer works with a server differently than a published toolbox, and generally learning how to use a new application, the major difficulty was in getting the geoprocessing service to work. It took a lot of toying around, and this project began with a much more complicated model, but due to isolating where the problem in getting the tool to work, the model was reduced to a simple buffering tool.

Nevertheless, ESRI help pages suggested that the problem was probably with file management. I originally had my map document, layers,

and tools, all within a folder in my local user folder. This folder is only accessible to that user. This choice was made because the server was running in a multi-user environment. To secure the data and such, it was put into a secure location.

Unfortunately, this may have been why the processes were not working. The help page explicitly said "do not put your content in a local user folder." Once this folder was moved to `C:\temp\mapserver`, everything worked fine.

In conclusion, it seems that after a bit of toying around with App Builder, as long as the prerequisite REST endpoint URLs and file paths are known, it is an easy substitute for the other available web management front-ends open to the ArcGIS web developer.