# Ad Component Example: Twitter Geo Search

This example goes through how to create a simple Flite ad component. We create a Twitter Geo Search component that allows a user to find tweets made on a certain topic near a particular location. You can download the accompanying code examples in Flite's Github code example repository.

A user of this component will be able to enter a latitude, longitude, and radius. The component will then use the Twitter Search API to retrieve all tweets in the area defined by the user that have a search string predefined by the advertiser running the component. The Twitter Search API was chosen for this example because it is a popular publicly available API, and no authentication is required to use it.

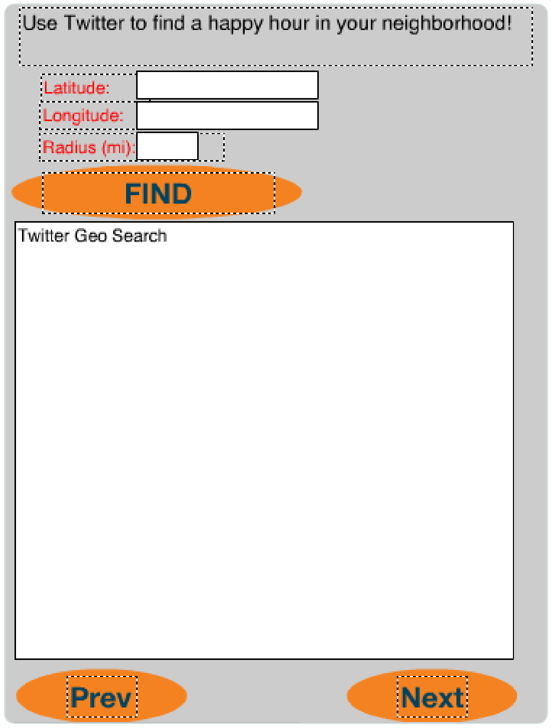
Changing this component to use a geocoding API to allow users to enter an address instead of lat/long coordinates would, of course, make this component more useful. We leave it as an exercise for the reader to add this improvement.

## STEP 1: BUILDING THE BASIC FUNCTIONALITY

Let's start by creating the component's basic functionality in Flash, and inserting the resulting SWF into Component Studio. This basic functionality includes creating a basic layout, and allowing the component to be used as intended for a specific search string.

**Note**: To see the final result of Step 1, simply download Component-Example-1.swf and skip to step 10 below.

1. **Set up your stage.** Download Component-Example-1.fla from Flite's code examples repository on Github, and open it in Adobe Flash CS5. This file contains the component layout, as shown in the image below. It contains the following elements:
    a. A Dynamic Text area called IntroText.
    b. Three Input Text areas called InputLat, InputLon, and InputRadius, along with their corresponding labels – LatLabel, LonLabel, and RadiusLabel.
    c. A button MovieClip called FindButton that the user will press upon entering the inputs.
    d. A large Dynamic Text area called FeedText where the tweets will appear.
    e. Two buttons called PrevButton and NextButton that will be used to page through the available tweets.



2. Start an ActionScript 2.0 document. In the Actions panel of your Flash workspace, enter:

```
#include “Component–Example–1.as”
```

Open a blank document in your favorite text editor, and save it as Component-Example-1.as. Make sure you save it in the same folder as your Component-Example-1.fla file.

3. **Initialize the Flite Ad API.** In your new ActionScript document, write the following code.

```
import mx.utils.Delegate

System.security.allowDomain(“*”);
var api :Object;
function apiInit (inAPI:Object):Void
{
      api = inAPI;
}
```

This will initialize the API and make sure your code works when inserted into a Flite component. See the section on creating your SWF file in the Flite Developer Center for more details on how this works.

4. **Add global variables.** Add some useful global variables at the top level of your ActionScript document.

```
var search_str:String = “happy hour”;
var tweets_per_page:Number = 5;
var total_tweets:Number = 100;
var mytweets :Object;
var myoffset = 0;
```

   - `search_str` hardcodes “happy hour” as the Twitter search string
   - `tweets_per_page` sets the maximum number of tweets displayed to 5
   - `total_tweets` sets the number of tweets to be downloaded
   - `mytweets` will contain the JSON object containing Twitter’s response
   - `myoffset` will specify the first tweet that is displayed

5. **Invoke the Twitter Search API.** Invoke the Twitter API by building the JSONP query string and sending it to the appropriate URL using the Flite Ad API’s `json.fetch` function. Add this code at the bottom of your `apiInit` function.

```
FindButton.onRelease = Delegate.create(this, function () {
// Build JSON query URL & make the query
json_url = “http://search.twitter.com/search.json?q=” +
     escape(search_str) + “&geocode=” +
           InputLat.text + “,” + InputLon.text + “,” +
           InputRadius.text + “mi&rpp=” + total_tweets;
api.util.json.fetch(json_url, fetch_load, fetch_error, false);
});
```

For more on Twitter’s Search API, click here. For more on Flite’s `json.fetch` API, click here.

6. **Write the success and failure callbacks.** First write the `fetch_load` function, which is invoked when the JSONP request is successful and returns a response. This function sets the `mytweets` variable to contain the JSON object generated from the Twitter response, and calls the `load_tweets` function to display the tweets.

```
function fetch_load (obj):Void
{
      mytweets = obj;
      load_tweets ();
}
```

Next, write the `fetch_error` function, which is invoked when the JSONP request to the Twitter Search API is unsuccessful. This function will output a simple error message.

```
function fetch_error (error):Void
{
FeedText.htmlText = “Twitter Geo Search failed. Please check” +
  “your Internet connection or contact Flite.”;
}
```

7. **Write a function to display the tweets.** Write the `load_tweets` function, which will display tweets in the FeedText text area. The `tweets_per_page`, `total_tweets`, and `myoffset` variables control which tweets are displayed from the `mytweets` object.

```
function load_tweets ():Void
{
      // First, display the search parameters
      FeedText.htmlText = “<b>Search string:</b> “ +
                          search_str + “\n”;
      FeedText.htmlText += “<b>Lat:</b> “ + InputLat.text +
                           “, <b>Lon:</b> “ + InputLon.text +
                           “, <b>Radius (miles):</b> “ +
                           InputRadius.text + “mi\n\n”;

      // Now display the Tweets based on the offset,
      // tweets per page, and total tweets values
      for (i = myoffset; i < (tweets_per_page + myoffset); i++)
      {
         // If we’ve reached total number of tweets, break the loop
         if (i == total_tweets) { break; }

         // Add the user’s name in bold to the print string
         s = “<b>” + mytweets.results[i].from_user + “:</b> “;
         user_tweet = mytweets.results[i].text;

         // Break the Tweet down into words, and add each one to
         // the print string. For words starting in “#”, “@” or
         // “http”, underline them and hyperlink appropriately.
         var tweet_words:Array = user_tweet.split(“ “);
         for (j=0; j<tweet_words.length; j++)
         {
               item = tweet_words[j];
               if (item.charAt(0) == ‘#’)
               {
                  s += “<u><a href=’” +
                       “http://twitter.com/#!/search/%23” +
                       item.substr(1,item.length) +
                       “’ target=’_blank’>” + item + “</a></u> “;
               }
               else if (item.charAt(0) == ‘@’)
```

```
                  {
                     s += "<u><a href='" +
                          "http://twitter.com/#!/search/%40" +
                          item.substr(1,item.length) +
                          "' target='_blank'>" + item + "</a></u> ";
                  }
                  else if (item.substr(0,4) == 'http')
                  {s += "<u><a href='" + item + "' target='_blank'>" +
                        item + "</a></u> ";
                  }
                  else { s += item + " "; }
               }
               // Print each Tweet on a new line
               FeedText.htmlText += s + "\n"
         }
   }
```

8. Add callbacks for the Previous and Next buttons, which will increment or decrement, respectively, the `myoffset` variable that determines which tweets are displayed, and then reload the tweets. Place this code at the bottom of your `apiInit` function.

```
// When user clicks Prev button, decrease offset and reload Tweets
PrevButton.onRelease = Delegate.create(this, function () {
      if (myoffset > 0) {
            myoffset = myoffset - tweets_per_page;
            load_tweets();
      }
});

// When user clicks Next button, increase offset and reload Tweets
NextButton.onRelease = Delegate.create(this, function () {
      if (myoffset < total_tweets - tweets_per_page) {
            myoffset = myoffset + tweets_per_page;
            load_tweets ();
      }
});
```

9. **Publish a SWF file.** You are done with the basic functionality of the Twitter Geo Search component! Save your ActionScript and FLA files, and publish a SWF file by choosing **Publish** from the **File** menu in Adobe Flash CS5.
10. **Create a new component and load your SWF into it.**
    a. Point your browser to http://www.flite.com.
    b. Click the **Login** link at the top and log into your Flite account.
    c. In the upper left corner of the console, expand the **Make New Item** menu and click **Component**. The **Make a Component** page loads.
    d. In the **Flash SWF** field, upload the SWF you just created.

e. In the **Component Name** field, enter a name for your component: “Component-Example-1”.
f. When your component loads in Flite’s Component Studio, change the **Width** to 400 and the **Height** to 600. Our component has been built to fit in a window of this size.
g. Click the **Publish Changes** button at the top to publish your component.

You have just created and published a new component in your Flite account. Now let’s test this component’s functionality by finding happy hour tweets within two miles of Flite’s headquarters in San Francisco:

- Enter 37.7861 in the **Latitude** textbox.
- Enter -122.4025 in the **Longitude** textbox.
- Enter 2 for the **Radius**.
- Click the **Find** button.

The FeedText area will display five tweets containing the phrase “happy hour” that were sent from within 2 miles of Flite’s headquarters. This is a surprisingly effective way of finding happy hours in your neighborhood. Click the **Next** button to display more tweets, and the **Prev** button to go back to ones you’ve already seen.

This is a simple example, and there are a number of things that can be done to improve this component, including designing a more attractive interface and giving users the ability to enter an address rather than a GPS location. We leave these improvements as exercises for the reader.

## STEP 2: ADDING METRICS TRACKING

In this section, we add some metrics tracking functionality to our Twitter Geo Search component. You will learn how to use the Flite Ad API to track hovers, button clicks, and clickthrough links. We start this section where we left off with the previous one.

**Note:** To see the final result of Step 2, simply download Component-Example-2.swf and skip to step 7 below.

1. **Set up your workspace.** Let's use the work from Step 1 as the base, and build on that here.
    a. In Adobe Flash CS5, open the Component-Example-1.fla file from the previous section and save it as Component-Example-2.fla. Change the `#include` statement to read
    ```
            #include "Component-Example-2.as"
    ```
    b. In your favorite text editor, open Component-Example-1.as and save it as Component-Example-2.as.
    c. In Flite's Component Studio, open the component you built in the previous section, and create a copy.
2. **Add button click tracking.** Every time a user clicks the Find, Previous, or Next buttons, the event should be tracked by Flite's system. In order to do this, you must use the createUserEvent and trackEvent functions from the Flite Ad API.

    In the `FindButton.onRelease` callback, add the following lines:
    ```
    var userEvent = api.logging.createUserEvent("ButtonClick", "Find");
    api.logging.trackEvent(userEvent);
    ```
    In the `PrevButton.onRelease` callback, add the following lines:
    ```
    var userEvent = api.logging.createUserEvent("ButtonClick", "Prev");
    api.logging.trackEvent(userEvent);
    ```
    In the `NextButton.onRelease` callback, add the following lines:
    ```
    var userEvent = api.logging.createUserEvent("ButtonClick", "Next");
    api.logging.trackEvent(userEvent);
    ```
    In each case, we have created a "ButtonClick" event with a payload corresponding to the button's name. This will uniquely identify which button the user clicked.
3. **Add text box click tracking.** It may also be useful to track when each input text box was clicked. Let us track these as "FieldClick" events with the name of the text box as the payload. To do this, add the following code at the end of the `apiInit` function.

```
InputLat.onSetFocus = Delegate.create(this, function () {
      var userEvent = api.logging.createUserEvent(“FieldClick”,
                      “Latitude”);
      api.logging.trackEvent(userEvent);
});
InputLon.onSetFocus = Delegate.create(this, function () {
      var userEvent = api.logging.createUserEvent(“FieldClick”,
                      “Longitude”);
      api.logging.trackEvent(userEvent);
});
InputRadius.onSetFocus = Delegate.create(this, function () {
      var userEvent = api.logging.createUserEvent(“FieldClick”,
                      “Radius”);
      api.logging.trackEvent(userEvent);
});
```

4. **Add mouse hover tracking.** Let us also track when a user hovers over the Find button. To do so, add the following code at the end of the `apiInit` function.

```
FindButton.onRollOver = Delegate.create(this, function () {
      var userEvent = api.logging.createUserEvent(
            api.logging.constants.labels.HOVER, “FindButton”);
      api.logging.trackEvent(userEvent);
});
```

   Note that we pass in a constant here instead of a string. This is because a number of events have designated constants associated with them in the Ad API, and the mouse hover or rollover is one of those events.

5. **Add clickthrough tracking.** Finally, let’s create a clickthrough for our call-to-action text at the top of the component.  To do this, add the following code to the end of the `apiInit` function.

```
IntroTextMC.onRelease = Delegate.create(this, function () {
      api.link.openPage(“http://www.flite.com”);
});
```

   Notice that we use a different function that we did in the previous event tracking additions. The Ad API’s openPage function opens the specified URL and also tracks a clickthrough event, so there is no need to explicitly call the `trackEvent` function in this case. Several other functions in the Ad API automatically track events. See here for a list of such functions.

6. **Save and publish.** We are done adding event tracking to this component. Save your AS and FLA files, and use Adobe Flash CS5 to publish to a SWF.
7. **Add new SWF to your component.** In Flite's Component Studio, click the **Upload** button in the bottom right, and upload the new SWF file. Click the **Publish Changes** button at the top.

You have added some event tracking to your component. There is currently no way to test that you added metrics correctly that is straightforward and fast. We suggest the following two methods:

- **Use your browser's developer tools.** When Flite components track an event, they do so via an HTTP POST request to the http://t.widgetserver.com/t/image.gif URL, with the event type and payload in a query string. You can use your browser's developer tools to see when these requests are being made, and see what events are being tracked.
- **Add component to an ad and wait a day.** The events tracked by a Flite ad take a day to appear in Flite reports. You can add your component to an ad, open the ad's homepage, and interact with the component in different ways to trigger different events. Then the following day you can take a look at the reports to see if the events were tracked properly.

For more on event tracking with the Flite platform, see here. For more on the functions available in the Flite Ad API, see here.

## STEP 3: PARAMETERIZING YOUR COMPONENT

In this section, we add some configuration parameters to our Twitter Geo Search component. This allows ad developers to adapt this component to their own specific use case. We group our parameters into three categories:

- **Source:** Parameters that control the core functionality and content of the component.
- **Layout:** Parameters that control how the various elements of the component are laid out.
- **Appearance:** Parameters that control what the component and its various elements look like.

We start this section where we left off with the previous one.

**Note:** To see the final result, start by performing steps 2, 4 and 6 below to add the config parameters in Component Studio. Then simply download Component-Example-3.swf and skip to step 8.

1. **Set up your workspace.** Let's use the work from Step 2 as the base, and build on that here.
    a. In Adobe Flash CS5, open the Component-Example-2.fla file from the previous section and save it as Component-Example-3.fla. Change the `#include` statement to read
    `#include "Component-Example-3.as"`
    b. In your favorite text editor, open Component-Example-2.as and save it as Component-Example-3.as.
    c. In Flite's Component Studio, open the component you built in the previous section, and create a copy.
2. **Add Source parameters in Component Studio.** Let's begin with the Source parameters. The table below lists the parameters you should add, along with the label, type, and default value for each one.

| Parameter | Label | Type | Default Value |
|---|---|---|---|
| SearchStr | Twitter Search String | Text Field | @Flite |
| IntroText | Intro Text | Text Area | Search for all @Flite tweets in your area |
| IntroClickthru | Intro Text Clickthrough | Text Field | http://www.flite.com |
| TweetsPerPage | Total Tweets Per Page | Choice List (1, 2, 3, 4, 5, 6, 7) | 4 |
| TotalTweets | Total Number of Tweets | Choice List (25, 50, 75, 100, 125, 150) | 100 |
| DisplayRadius | Display Radius? | Checkbox | Leave blank |

We highlight four different types of parameters here – the Text Field, Text Area, Choice List, and Checkbox.

3. **Add Source parameters to your ActionScript code.** Add the following lines just below the `api = inAPI;` line at the top of your `apiInit` function.

```
search_str = api.config.component.SearchStr;
tweets_per_page = Number(api.config.component.TweetsPerPage);
total_tweets = Number(api.config.component.TotalTweets);
if (!api.config.component.DisplayRadius)
{
       InputRadius._visible = false;
       RadiusLabel._visible = false;
       InputRadius.text = 25;
}
IntroTextMC.IntroText.text = api.config.component.IntroText;
```

Let's go over the parameters one by one to see what each one does and insert any additional code where necessary.

**SearchStr:** We had our Twitter search string hardcoded as a global variable with

```
    var search_str = “happy hour”;
```

This now allows advertisers to use any Twitter search string they like with this component.

**TweetsPerPage:** Sets the total number of tweets displayed per page in the FeedText area. We had this hardcoded to 5, and this makes it variable.

**TotalTweets:** The total number of tweets loaded via the Twitter Search API. We had this hardcoded to 100, but now can change it.

**DisplayRadius:** This controls whether the “Radius” input textbox is shown or not. If it is not shown, then the radius is always automatically set to 25 miles.

**IntroText:** The call-to-action text that appears at the top of the component. Since we are allowing customers to change the search string, it makes sense to allow them to change this intro text as well.

**IntroClickthru:** The clickthrough URL for the introductory text. Previously we had it hardcoded to http://www.flite.com. To change this, replace

```
    api.link.openPage(“http://www.flite.com”);
```

with

```
    api.link.openPage(api.config.component.IntroClickthru);
```

4. **Add Layout parameters in Component Studio.** These parameters control the position of each element within the component, allowing a user to completely change the component’s layout. An element’s position is measured as the position of the element’s top left corner with respect to the ad’s top left corner. The table below summarizes these parameters.

| Parameter | Label | Type | Default Value |
|---|---|---|---|
| IntroTextPosition | Intro Text Position | Text Field | 20,20 |
| LatLabelPosition | Latitude Label Position | Text Field | 25,40 |
| LatitudeBoxPosition | Latitude Box Position | Text Field | 90,40 |
| LonLabelPosition | Longitude Label Position | Text Field | 25,60 |
| LongitudeBoxPosition | Longitude Box Position | Text Field | 90,60 |
| RadiusLabelPosition | Radius Label Position | Text Field | 25,80 |
| RadiusBoxPosition | Radius Box Position | Text Field | 90,80 |
| ButtonPosition | Find Button Position | Text Field | 5,110 |
| TweetBoxPosition | Tweet Box Position | Text Field | 10,150 |
| PrevButtonPosition | Previous Button Position | Text Field | 5,460 |
| NextButtonPosition | Next Button Position | Text Field | 240,460 |

All of these parameters are represented as text fields. The syntax is “x,y”. Let’s go over how to add these in ActionScript.

5. **Add Layout parameters in your ActionScript code.** For each parameter, we need to parse it into the X and Y positions, and then move the corresponding element accordingly. To make this happen, add the following code underneath the code we added in step 3.

```
// Layout
introtext_pos=api.config.component.IntroTextPosition.split(‘,’);
IntroTextMC._x = introtext_pos[0];
IntroTextMC._y = introtext_pos[1];

latlabel_pos = api.config.component.LatLabelPosition.split(‘,’);
LatLabel._x = latlabel_pos[0];
LatLabel._y = latlabel_pos[1];

latbox_pos = api.config.component.LatitudeBoxPosition.split(‘,’);
InputLat._x = latbox_pos[0];
InputLat._y = latbox_pos[1];

lonlabel_pos = api.config.component.LonLabelPosition.split(‘,’);
LonLabel._x = lonlabel_pos[0];
LonLabel._y = lonlabel_pos[1];

lonbox_pos=api.config.component.LongitudeBoxPosition.split(‘,’);
InputLon._x = lonbox_pos[0];
InputLon._y = lonbox_pos[1];

radlabel_pos=api.config.component.RadiusLabelPosition.split(‘,’);
RadiusLabel._x = radlabel_pos[0];
RadiusLabel._y = radlabel_pos[1];

radbox_pos = api.config.component.RadiusBoxPosition.split(‘,’);
InputRadius._x = radbox_pos[0];
InputRadius._y = radbox_pos[1];

button_pos = api.config.component.ButtonPosition.split(‘,’);
FindButton._x = button_pos[0];
FindButton._y = button_pos[1];

feed_pos = api.config.component.TweetBoxPosition.split(‘,’);
FeedText._x = feed_pos[0];
FeedText._y = feed_pos[1];

button_pos = api.config.component.PrevButtonPosition.split(‘,’);
PrevButton._x = button_pos[0];
PrevButton._y = button_pos[1];

button_pos = api.config.component.NextButtonPosition.split(‘,’);
NextButton._x = button_pos[0];
NextButton._y = button_pos[1];
```

6. **Add Appearance parameters in Component Studio.** These parameters control what the various elements in your component look like. For this example, we focus on the intro/call-to-action text and on the Find button. We leave parameterizing the rest of the elements as an exercise for the reader. Add parameters as described in the table below.

| Parameter | Label | Type | Default Value |
|---|---|---|---|
| IntroTextFont | Intro Text Font | Choice List | Lucinda Grande |
| IntroTextSize | Intro Text Size | Text Field | 12 |
| IntroTextColor | Intro Text Color | Color Picker | #ff0080 |
| ButtonText | Find Button Text | Text Field | Find Now! |
| Button Color | Find Button Color | Color Picker | #f0f022 |
| ButtonTextFont | Find Button Text Font | Choice List | Georgia,Utopia,_serif |
| ButtonTextSize | Find Button Text Size | Text Field | 15 |
| ButtonTextColor | FindButton Text Color | Color Picker | #ff0080 |

We've added another parameter type into the mix here – the Color Picker. This allows you to pick a color from a spectrum, or one of several preset colors.

For the two font choice lists, use the following option/value combinations:

| Option | Value |
|---|---|
| Lucinda Grande | Lucida Grande,Verdana,_sans |
| Helvetica, Arial, sans-serif | Helvetica,Arial,_sans |
| Georgia, Utopia, serif | Georgia,Utopia,_serif |
| Times New Roman, Times, serif | Times New Roman,Times,_serif |
| Courier New, Courier, mono | Courier New,Courier,_typewriter |

The option is how the choice will appear in the dropdown. The value is the value that will be passed to the Ad API component variable.

7. **Add Appearance parameters in your ActionScript code.** Let's start with the intro text. We can set its font, size and color all at once using the `TextFormat` type. Add the following code just above where you set the intro text in Step 3. It is important for this code to be above where the text is set, because the new text format will only be applied to new text.

```
tf = new TextFormat(api.config.component.IntroTextFont,
      Number(api.config.component.IntroTextSize),
      Number("0x" + api.config.component.IntroTextColor.substr(1)));
IntroTextMC.IntroText.setNewTextFormat(tf);
```

   Now let's format the appearance of the Find button. We format the text font, size and color as above, plus set the button color and the button text itself.

```
var button_color:Color = new Color(FindButton.FindOval);
button_color.setRGB(Number("0x" +
      api.config.component.ButtonColor.substr(1)));
tf = new TextFormat(api.config.component.ButtonTextFont,
      Number(api.config.component.ButtonTextSize),
      Number("0x" + api.config.component.ButtonTextColor.substr(1)));
FindButton.FindText.setNewTextFormat(tf);
FindButton.FindText.text = api.config.component.ButtonText;
```

8. **Group your parameters.** We are done adding parameters. Now let's divide them into three tabs in Component Studio.
   a. Click **Add Group** to add a new group, and use the group's control to change it to a Tab Set.
   b. Click the icon in the upper right of the tab set block three times to create three new tabs.
   c. Rename the three tabs by clicking each tab's name, typing in a new name, and clicking **OK** to save. Name the first tab "Source", the second one "Layout", and the third one "Appearance".
   d. Drag and drop all the settings from step 2 into the Source tab, the settings from step 4 into the Layout tab, and the settings from step 6 into the Appearance tab.
9. **Save and publish.** We are done parameterizing this component. Save your AS and FLA files, and use Adobe Flash CS5 to publish to a SWF.
10. **Add new SWF to your component.** In Flite's Component Studio, click the **Upload** button in the bottom right, and upload the new SWF file. Click the **Publish Changes** button at the top.

You can test the new parameters by clicking the icon in the control bar at the top to view a preview of the component. You will see how the component will appear to customers when they try to edit it. Navigate through the Source, Layout, and Appearance tabs, and change the various parameters. The changes will be reflected immediately in the component preview, so you can make sure that the parameters work as expected.

This is a very simple parameterization that we are using as an example, and would not pass Flite's certification process. Some of the things that need to be changed or added are:

- **Parameter ease of use:** Parameters should be easy to understand so customers could use them without any external help, and we have not worked on that at all beyond giving the parameters straightforward labels. Some things we can do include setting the width and height for text areas, setting the maximum number of characters for text fields, adding help text where necessary, and so on.
- **Additional parameters:** There are a lot more things that should be parameterized in this component . For instance, the text and appearance of the Latitude/Longitude/Radius labels; the appearance of the tweet text; the size of the Latitude/Longitude/Radius text boxes; and the dimensions of the tweets text area should all have parameters associated with them.
- **Additional categorization:** As the number of parameters grows, additional grouping is needed beyond our three-tab paradigm. This is particularly true in the Appearance tab, where many components will often have dozens of parameters. Component Studio allows you to further categorize settings into sections and disclosures.

We leave it as an exercise for the reader to work on some of the improvements listed above.