Paper 010-30

Sexy SAS/IntrNet®: A Macromedia Flash front-end for SAS® Web Applications

John Leveille, d-Wise Technologies Inc., Raleigh, NC
Joey Leveille, SAS Institute Inc., Cary, NC

ABSTRACT

Many of us remember the “eureka” effect we experienced the first time we saw a SAS/IntrNet application come to life and connect SAS Software to the web browser. For some of us that was as many as seven years ago. Since that time, the landscape of the Internet has changed drastically. Web applications are everywhere and the capabilities of these applications, as well as the look and feel of their associated web sites, has steadily improved.

The ubiquitous Flash® player from Macromedia® has emerged as a leading technology for developing next generation web sites and application user interfaces. With Flash you can leap beyond the limitations of the web browser and have a fast, dynamic, customizable user interface at your disposal.

This paper demonstrates how you can utilize Flash functionality in order to create a dynamic, platform-independent web application that is configured and driven by SAS Software.

INTRODUCTION

For many years since its initial release, SAS/IntrNet has been used by SAS programmers to generate dynamic reports and web pages. The vast majority of SAS/IntrNet applications use SAS to generate HTML, though the product can produce a variety of output forms including graphics, XML, and PDF. As the concepts of the Internet penetrated every aspect of our businesses and organizations, these SAS/IntrNet reports and applications were carried along for the ride. Of course, with the acceptance of these new information sources came raised expectations from the information consumer. Coworkers and customers that were once comfortable with a stack of green bar reports are now expecting dynamic report delivery and demanding full-fledged applications that allow them to input data and perform on-line processing and analysis.

The SAS DATA Step and the SAS/IntrNet Application Dispatcher are a powerful software combination you can use to answer this need. In the majority of cases, the output from a SAS/IntrNet program is an HTML page. HTML-based applications are both exciting and powerful, yet, as many web developers are aware, they suffer from certain interface limitations, and are subject, quite often, to the whim of web browser behavior.

WHAT IS FLASH?

Macromedia’s Flash is a vector-based animation environment that enables the creation of highly dynamic and interactive multimedia experiences. These experiences can be delivered via the web or as a stand-alone application. Currently, Flash is the world’s most pervasive software platform, reaching 97% of Internet-enabled desktops worldwide, as well as many popular devices. Since its inception, Macromedia has continued to add functionality to the development environment that has allowed it to grow in its richness and power. Flash originally began as a means to provide web-based animation in a package that was smaller and more functional than animated GIF’s. It is easily accessible to visual artists and designers with limited technical skill sets. With each successive release, Macromedia has added more scripting functionality and modularity. This has culminated in the most recent version, Flash MX 2004 with ActionScript v2.0. This release adds true object-oriented programming capabilities and increased robustness and accessibility for programmers. Flash’s combination of technical functionality and multimedia capabilities can now provide a toolset from which to create incredibly rich and immersive experiences that are not only interactively stunning, but also very dynamic and easily interoperable with external environments.

PROGRAMMING APPLICATIONS USING SAS/INTRNET

The typical SAS/IntrNet application consists of a series of SAS programs, some HTML files, some JavaScript, and some image files. You write your HTML pages using hyperlinks that call through the SAS/IntrNet Application Dispatcher into your SAS programs on the server. Here is an example of a hyperlink that you might find in the source for a SAS/IntrNet HTML page.

```
<a href =
"http://myserver/scripts/broker.exe?_service=default&_program=sugi30.myprogram.sas&act
ion=start">Click here to start the application</a>
```

When the application user clicks this link in the web browser, all of the parameters in the query string (the part following the question mark in the hyperlink) are sent to SAS and available as global macro variables. Inside the source code for the program "myprogram.sas" you can submit the statement

```
%put The action is &action;
```

and this will produce the following message in the SAS log

```
The action is start
```

It is also quite common in SAS/IntrNet applications to use an HTML form to post the name/value pairs. In this case the result is still the same -- you receive the data as global macro variables inside your SAS program each time the user submits an HTML form.

From this point you can run DATA steps, procedures, ODS, and any other non-visual component of SAS to respond to the user's request. Ultimately, your SAS program must produce some kind of content that the web browser can display. This can be images, a PDF file, an RTF file, but it is most often a combination of HTML and JavaScript that displays the next page in your application. The HTML and JavaScript you produce can contain hyperlinks back to other SAS programs that are part of your application. Figure 1 depicts a typical SAS/IntrNet application flow.

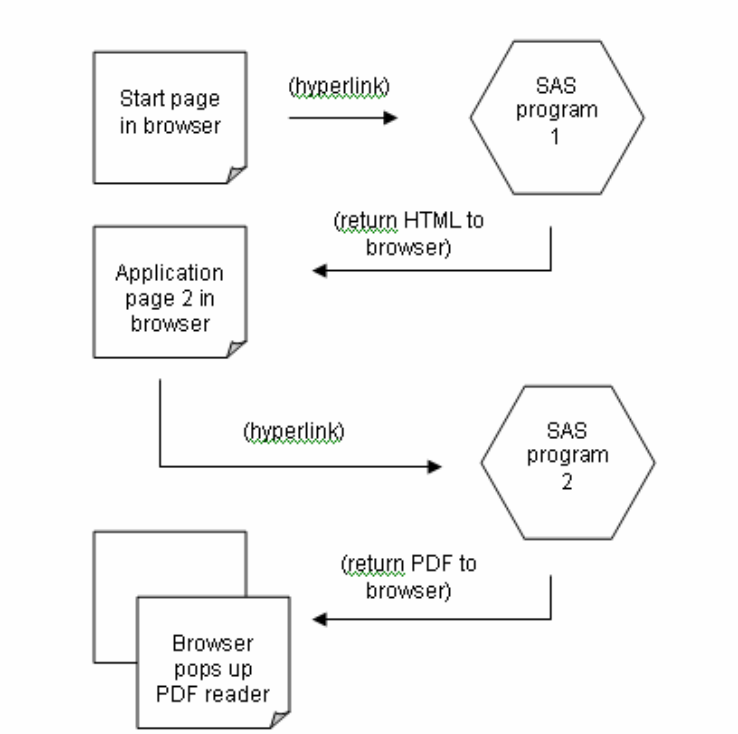

Figure 1

This typical model for SAS/IntrNet programming is quite effective in a lot of circumstances. However, the fact that the SAS program is dynamically generating the HTML and JavaScript code is a common source of errors. Many of the SAS/IntrNet programs that you write will appear as a confusing weave of SAS, HTML, and JavaScript code. A missing comma, quote, or parenthesis will produce a runtime error in your JavaScript that can be very difficult to debug. In addition to the propensity for errors, this model of application development almost forces the SAS programmer, HTML programmer, and JavaScript programmer to be the same person. This could be a problem for you because you may not have all of these skills. The next section describes how you can replace the HTML and JavaScript pieces of the application with a Flash movie, and allow for a different application model that still uses SAS/IntrNet.

REPLACING HTML AND JAVASCRIPT WITH FLASH

Before replacing the visual layer of your application with Flash, it is important to note that a blended approach is also possible. Flash can be embedded in a portion of an HTML page and can even interact with JavaScript in the web page through ActionScript function calls. This blended approach is beyond the scope of this paper, but it is an alternative technique that can be useful in many situations.

The first step in replacing the visual layer of an application with Flash is to draw each screen of the application as a Flash movie clip or layer within the root movie clip. The Flash development environment provides a drag and drop visual designer as well as a component library with many pre-built elements. The library has all of the typical elements you encounter on an HTML form such as: input fields, check boxes, drop lists, etc. Figure 2 shows the Flash MX development environment and visual designer.

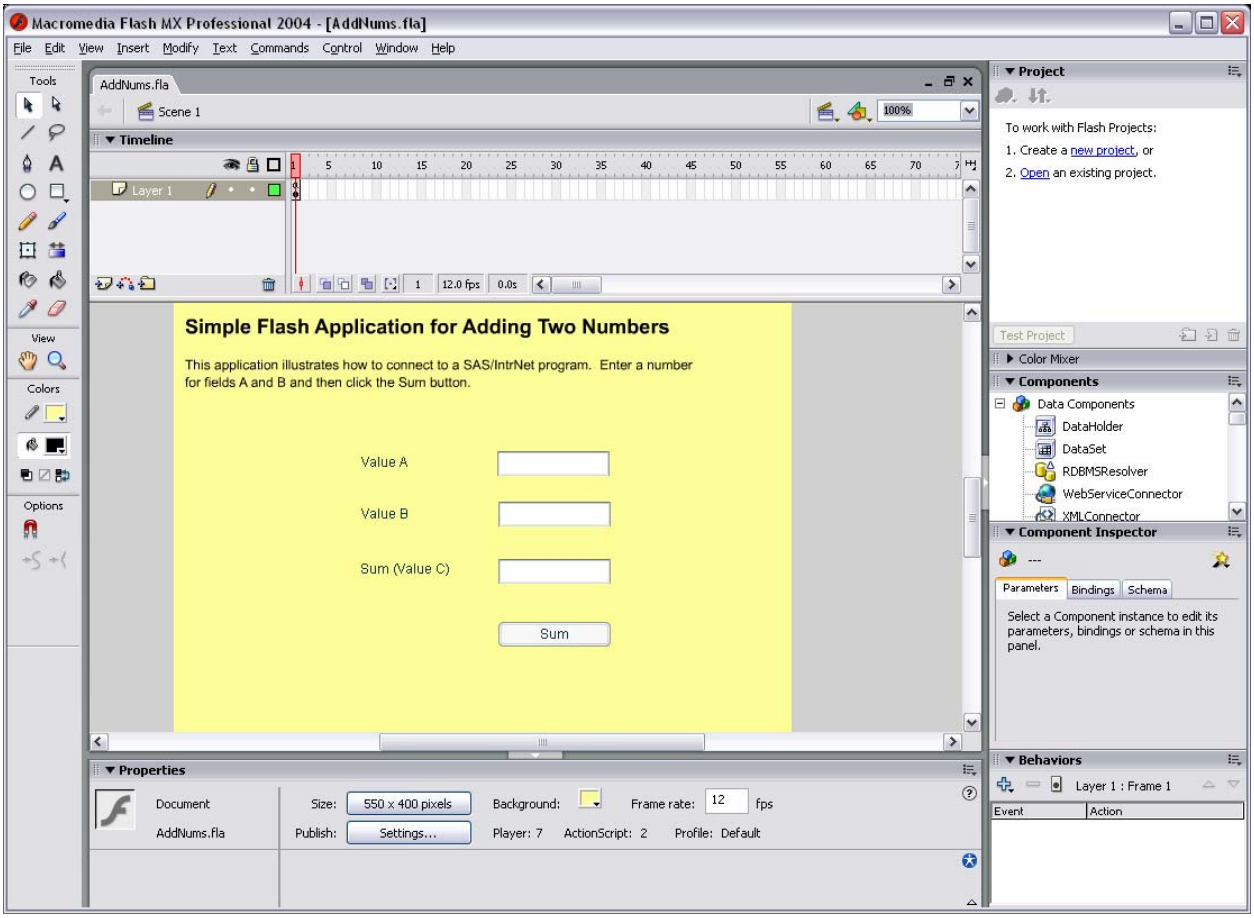

Figure 2

The window inside of Flash where you draw the application screen is called the stage. Each Flash document that you open has a stage.

The next step is to add event code to your Flash movie. Flash is an event-based programming environment much like the web browser. Each button and text field in your Flash movie will fire events such as onPress and onRelease which are analogous to the onClick events that are so familiar to the JavaScript programmer. For that matter, the ActionScript programming language has a syntax that is very much like JavaScript. Suppose you have an HTML-based application where the user clicks a button and it fills a text field with today's date. The same functionality is possible within the ActionScript for a Flash movie, and the resulting code is nearly identical. One big difference between creating a visual interface in Flash and one in HTML is that the Flash movie is able to communicate with the web server without performing a page transition. Web programmers are familiar with the paradigm that in order to post information to the web server your application has to transition to another web page. It can, of course, redisplay the same page, but never the less, a browser page transition occurs with each form post. Since Flash does not suffer from this limitation you may decide to program the flow of your application a bit differently. The resulting Flash application can often be much simpler than the HTML-based version.

Once you have completed the visual design and ActionScript code for the Flash movie that will become the visual component of your application, you need to export the movie as a `.swf` file. This single file is a compressed, compiled version of your application. It contains all of the images, screens, ActionScript, sounds, etc. that make up the visual component of your application. You can create the `.swf` file by selecting `File->Export->Export Movie`. To play the movie you can simply open the `.swf` file in your web browser. Most browsers support this by invoking the Flash player inside the browser and playing the movie. The Flash player can also be invoked as a stand-alone application.

In order to deliver the Flash application to the end user you will probably want to serve it up in an HTML page. This makes it easy for users to navigate to the application from other pages on your web site. It also makes it easy for you to provide supporting content around your application. It only takes a single HTML tag to embed the flash player and your movie into an HTML page. Unfortunately, that single tag is different depending on the type of browser that is used to view the web page. Some browsers honor the <OBJECT> tag while others honor the <EMBED> tag. The best approach is to insert both tags into your HTML page so that a single page works in as many browsers as possible. The code below shows how to embed the `addnums.swf` Flash movie in an HTML page. Simply insert this code anywhere inside the body of an HTML page.

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"

codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#versio
n=7,0,0,0"
        width="550"
        height="400"
        id="AddNums">

<param name="allowScriptAccess" value="sameDomain" />
<param name="movie" value="AddNums.swf" />
<param name="quality" value="high" />
<param name="bgcolor" value="#ffff99" />

<embed src="AddNums.swf"
       quality="high"
       bgcolor="#ffff99"
       width="550"
       height="400"
       name="AddNums"
       allowScriptAccess="sameDomain"
       type="application/x-shockwave-flash"
       pluginspage="http://www.macromedia.com/go/getflashplayer" />

</object>
```

Another way to combine standard web pages with a Flash application is to create hyperlinks from the web pages directly to the `.swf` file on your web server. If the user clicks a link to the `.swf` file the browser will load and play the Flash movie. This behavior is the same as when you open the `.swf` file from the `File->Open` dialog in the web browser.

CONNECTING FLASH TO SAS

Events within the Flash movie trigger the execution of ActionScript code that makes up the visual component of your application. Sometimes you will find that the Flash programming environment provides all the tools you need to program a piece of business logic. However, more complex pieces of business logic will require you to call out of Flash and into SAS where a SAS program will handle the particular task or series of tasks that you want to perform.

As part of a long list of built-in functions and objects, the Flash environment provides a mechanism for calling HTTP URLs. The URL objects and functions can be used to connect the ActionScript in your Flash movie to your SAS/IntrNet programs in a way that is equivalent to what the browser does each time you click a hyperlink. The Flash elements we will discuss are

- `loadVariables()`
- `loadVariablesNum()`
- `LoadVars`
- `XML`

FUNCTIONS: loadVariables() AND loadVariablesNum()

The global ActionScript functions `loadVariables` and `loadVariablesNum` have been part of the Flash environment since Flash Player version 4 (the most recent version is Flash Player 7). These functions allow you to call a URL and load the data returned from that URL into variables inside of your Flash movie. The `loadVariables`

function allows you to direct the variables into a specific movie clip. The `loadVariablesNum` function allows you to direct the variables into a level within the current movie clip. A level is similar to a layer within a drawing such as you would find in a graphics program like Adobe Photoshop ®. The `LoadVars` object described next provides a more robust programming interface for connecting to SAS so we will not go into more depth on `loadVariables` and `loadVariablesNum`. You can read more about these functions in any ActionScript reference manual.

OBJECT: LoadVars

The `LoadVars` object is available in the Flash environment starting in Flash Player 5. It serves as a collection of arbitrary name/value pairs that can be sent to and received from a URL. `LoadVars` has methods for setting HTTP headers which is something that the `loadVariables` and `loadVariablesNum` functions do not provide. You can read and set name/value pairs on a `LoadVars` object using the names of the name/value pairs as object properties. For example, one of the name/value pairs that you are required to pass to a SAS/IntrNet URL is named `_program`. To set the value of `_program` on a new `LoadVars` object, use the following ActionScript

```
var sendvars = new LoadVars();
sendvars._program = “myprogs.program1.sas”;
```

Continue your program by setting properties to fill out the same `LoadVars` object with all of the data that needs to be sent to your SAS/IntrNet program. Set any other SAS/IntrNet special name/value pairs as well as any name/value pairs that are part of your application.

The next step is to call a method on the `LoadVars` object that will communicate with the web server by invoking a URL. Setting properties on `LoadVars` does not perform any communication over the web. You have to call one of these methods: `load`, `send`, or `sendAndLoad` in order to communicate with a web server program like the SAS/IntrNet Application Broker. The load method is convenient if you want to call a URL and read the output without providing any input parameters. The send method is convenient if you want to provide some input data and invoke a URL, but you don’t care about reading any response data from the URL back into your ActionScript program.

The `sendAndLoad` method is the most flexible way to send and receive name/value pair data between Flash and SAS/IntrNet. It provides for sending input data and receiving output data while relying on the `LoadVars` object to do all of the preparation of data you are sending and parsing of data you are receiving. It uses one LoadVars object for input and another `LoadVars` object for output. The `sendAndLoad` method is asynchronous so Flash will allow your ActionScript program to continue right away after you call `sendAndLoad`. It uses a “callback” function so that your program can take some specified action after the SAS/IntrNet program sends its response. Since this method requires a callback function it is a little more difficult to program than some of the other options for external communication. The following ActionScript shows the preparation of a simple `sendAndLoad` call to SAS/IntrNet.

```
//set up a LoadVars object to send over to SAS
var brokerurl = "http://localhost/scripts/broker.exe";
var service = "default";
var program = "addnums.addnums.sas";
var debug = "0";

var sendVars = new LoadVars();
sendVars._service = service;
sendVars._program = program;
sendVars._debug = debug;
sendVars.a = ValueA.text;
sendVars.b = ValueB.text;

var receiveVars = new LoadVars();
receiveVars.onLoad = printsum;

sendVars.sendAndLoad(brokerurl, receiveVars, "GET");
```

Notice that there are two instances of `LoadVars` in this program. They are named `sendVars` and `receiveVars` indicating the purpose of each object. The first several lines of this program set name/value pair data into the `sendVars` object. The `_service` variable is passed to the broker to indicate the Application Dispatcher service that should receive the request. This program selects the service named “default”. The details for this service were previously configured and stored in the `broker.cfg` file. The `_program` variable instructs the Application Server to

invoke the addnums.sas program located in the addnums program library. After setting the SAS/IntrNet reserved parameters, the program sets variables a and b that are part of the input to addnums.sas.

The second-to-last line of this code shows how to set the callback function. There is a function named printsum defined somewhere in this same ActionScript program. Here, the name of that function is supplied as the callback for the receiveVars object. When the sendAndLoad method receives a response from SAS it will fire the onLoad event which will call the printsum function.

Finally, the sendAndLoad method is called on the sendVars object to begin the communication sequence. The first argument to this method is the URL that Flash should invoke. In this case it is the path to the SAS/IntrNet Application Broker. It is important to note that there are security restrictions on the web server domain names that you can call using this method. Please refer to an ActionScript reference manual for the specific details. The second argument to the sendAndLoad method is the LoadVars object that you want to receive the response from the SAS program. The third parameter to this method is the type of HTTP method that Flash should use to send the input data to the server. The two valid values for this argument are GET and POST. Please refer to an HTML reference manual that includes documentation on HTML form processing to learn about the differences between GET and POST. Since SAS/IntrNet supports both GET and POST methods, either one will work when following the techniques outlined in this paper.

The sendAndLoad method expects that the SAS program will return data in a specific format. Though the typical SAS/IntrNet program produces HTML, the sendAndLoad method requires URL output to take the format described by the MIME type application/x-www-form-urlencoded. Simply put, this means that the output of the SAS program must be a string of name/value pairs. Each name and value is joined by an equals sign while each pair is joined by an ampersand. Here is an example urlencoded string:

```
state=NC&author1=John+Leveille&author2=Joey+Leveille
```

In addition to these rules, special characters such as spaces and other non alphanumerics are encoded using a specific set of rules. Fortunately, SAS/IntrNet provides the urlencode function so that you can easily encode any values that might contain special characters. A helpful hint that the authors discovered is to start and end the urlencoded string with an ampersand when producing output intended for Flash. Though the leading and trailing ampersands are not typically part of a urlencoded string, Flash is tolerant of them and they help to ensure that all values are parsed properly when received by Flash.

The program code below is from the SAS/IntrNet program addnums.sas.

```
data _null_;
  a = input(symget('a'), best.);
  b = input(symget('b'), best.);
  c = a + b;

  file _webout;
  put 'Content-type: application/x-www-form-urlencoded';
  put ;
  put '&c=' c +(-1) '&'; /*trailing ampersand to prevent parsing problem*/
run;
```

This program receives two numbers as input parameters a and b. It adds the two number and returns the sum as parameter name c. If the program is passed a=2 and b=3 the resulting output sent to Flash will be

```
Content-type: application/x-www-form-urlencoded

&c=5&
```

As described above, when Flash receives this response from the server, it parses the name/value pairs and copies them into the receiveVars object that was passed as the second argument to sendAndLoad. Next, Flash fires the onLoad event and calls the printsum method. The contents of the receiveVars method is available within the printsum method using the special object reference this. Below is the printsum method from the addnums Flash application. It shows how the Flash program reads out the returned value of the parameter c and shows it in a text field.

```
printsum = function()
{
        ValueC.text = this.c;
}
```

OBJECT: XML

The Flash environment provides an object type called `XML`. This object is very similar to the `LoadVars` object. While `LoadVars` operates strictly on a name/value pair data structure, the `XML` object operates on XML documents. It has many of the same methods as the `LoadVars` object including `load`, `send`, `sendAndLoad`, etc. It also contains a variety of methods for parsing the XML that is returned from calling a URL or reading a file. If you use the `XML` object to parse XML data you will also use the `XMLNode` class. If you want to use this object to connect with a SAS/IntrNet program then your SAS/IntrNet program must produce XML output. The SAS Output Delivery System (ODS) is a convenient way to produce XML output from SAS and should interoperate with the Flash XML object in a straight-forward manner though the authors have never experimented with this particular combination.

EXAMPLE: LIFE SIMULATION

The authors of this paper have developed an example Flash front-end to a SAS/IntrNet application that illustrates some of the animation features of Flash. This example performs the simulation known as “The Game of Life” originated by John Conway. The program displays a grid of dots in which each dot simulates a life form. Each dot can be shown (indicating that it is “alive”) or hidden (indicating that it is “dead”). Initially, the user can set a pattern of life forms and then start the simulation. The simulation progresses due to a short list of very simple rules. Each cell on the grid is evaluated by counting its neighboring life forms. If a cell is occupied by a life form but there are too many neighbors then that life form will die of overcrowding. If there are too few neighbors then the life form will die of loneliness. At moderate population levels the life form will survive or a new life form will grow where none existed before.

The construction of this application starts with the visual design of the simulation grid shown in Figure 3 and the life form movie clip (named Cell) in Figure 4.

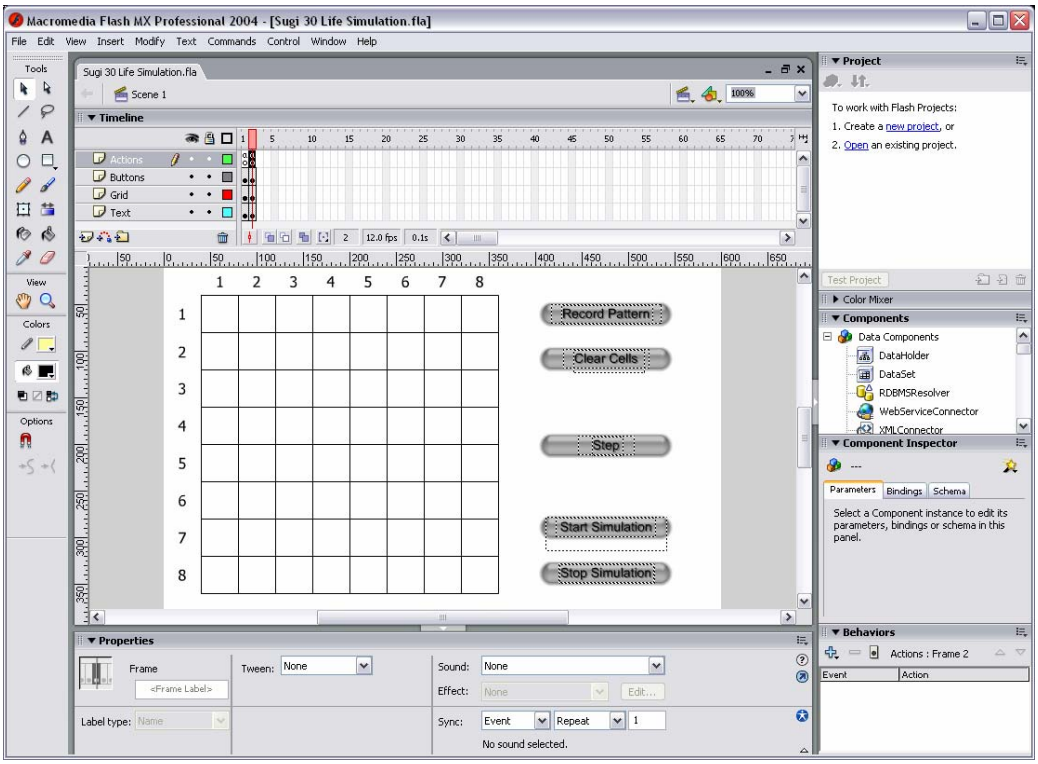

Figure 3

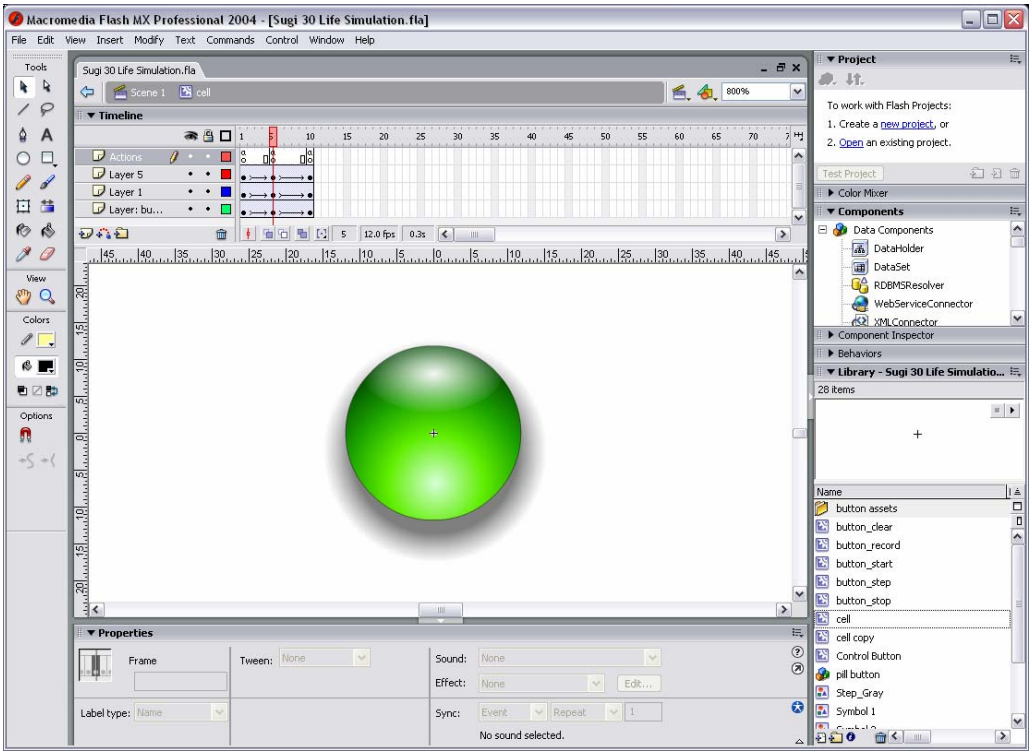

Figure 4

The timeline of the root Movie (shown just above the grid in Figure 3) has only two frames. The first frame displays the stage with three of the 5 buttons disabled and gray. The second frame (this is the frame currently displayed in Figure 4) shows all of the buttons enabled. ActionScript code is attached to these two frames to alternate the movie between frames 1 and 2 at the appropriate times. For example, once the user selects the "Record Pattern" button, data is sent to SAS and then it is time to enable all of the buttons on the user interface. So the ActionScript behind the "Record Pattern" button communicates with SAS and then moves the play head for the root movie clip to frame 2. The user now sees all 5 buttons enabled. The timeline for the life form (the green dot) consists of 10 frames. On frames 1 and 10 the dot is faded so that it is completely invisible while on frame 5 (the frame that is currently displayed in Figure 5) the dot is fully visible. Between frames 1 and 5 and also between frames 5 and 10 the timeline shows a thin arrow connecting the frames. This represents the "tweening" that is assigned for these intermediate frames. Tweening is a feature of Flash. By placing tweening on the frames the developer is telling Flash to animate the intermediate frames and gradually change from the first frame to the last. In the case of the life form, the tweening causes the life form to fade in from invisible to visible as the movie is played from frames 1 through 5. Then the life form fades back to invisible when the movie is played from frames 5 through 10. Next, a small amount of ActionScript is added and the behavior is set so that each time the user clicks on it, the green dot fades in or out.

The next step in developing this simulation is writing the SAS programs that will record the pattern of life forms and step through the simulation. Since all of the communication between Flash and SAS is done with a collection of name/value pair data, the development of the SAS programs can move forward in parallel with the development of the visual components provided that the specifics of how the data will be passed has been worked out in advance. There are two SAS programs that make up the life simulation SAS component. They are `lifeinit.sas` and `life.sas`. The purpose of `lifeinit.sas` is to collect the current state of all of the life forms on the grid. This program is invoked when the user clicks on the "Record Pattern" button. This program uses the Application Dispatcher server-side session mechanism to save the state of the life forms into a data set named `SAVE.LIFE`. Here is a partial listing of this program:

```
%macro lifeinit(rows=);

/* rows supplied is interpreted as the number of rows and cols */
%local lenstmt assign;
%let rc=%sysfunc(appsrv_session(create));

%do i = 1 %to &rows;
```

```
    %let lenstmt = &lenstmt. c_&i._1 - c_&i._&rows;
    %do j = 1 %to &rows;
     %let assign = &assign. c_&i._&j.=max(input(symget("c_&i._&j."), best.),0)%str(;);
    %end;
%end;

data save.life;
  length &lenstmt 8;
  &assign;
run;

%mend lifeinit;

%lifeinit(rows=&rows)
```

SAS Macro code dynamically builds a length statement and series of assignment statements to populate the SAVE.LIFE data set. This program expects the Flash movie to pass in `rows=<number>` and `c_1_1=<1 or 0>`, `c_1_2=<1 or 0>`, etc. At the end of this program (code not shown) SAS code writes the Application Server `_sessionid` back to the Flash movie as a response using the reserved fileref `_webout`. The Flash movie will store the `_sessionid` so that it can find the saved data when it connects to run the simulation.

The simulation is accomplished through repeated calls from Flash to the SAS program `life.sas`. This program uses the data set SAVE.LIFE as the current state of the life forms. It makes one pass over the grid of cells and evaluates the fate of each life form. It stores the next state of each cell back into the data set using a value of 1 to represent that the life form is alive or a 0 to represent that the life form is dead. Here is a partial listing of this program:

```
data save.life;
  length i j rows 8;
  drop i j rows neighborcount
          left right top bottom;
  set save.life;

  &arraystmt;
  array tmpcell[&rows, &rows] _temporary_;

  rows = input(symget('rows'), best.);
  do i = 1 to rows;
      do j = 1 to rows;
          /*determine if this spot lives or dies*/
          neighborcount = 0;

          /*find neighbor coordinates*/
          left = mod(j-1,&rows);  if left = 0 then left = &rows;
          right = mod(j+1, &rows);  if right = 0 then right = &rows;
          top = mod(i-1, &rows);  if top = 0 then top = &rows;
          bottom = mod(i+1, &rows);  if bottom = 0 then bottom=&rows;

          /*get neighbor count*/
          neighborcount + cell[i,left];
          neighborcount + cell[i,right];
          neighborcount + cell[top, j];
          neighborcount + cell[bottom, j];
          neighborcount + cell[top, left];
          neighborcount + cell[top, right];
          neighborcount + cell[bottom, left];
          neighborcount + cell[bottom, right];
```

Here SAS DATA step code loads the current state from the SAVE.LIFE data set using the `set` statement. The columns of the data set are assigned to an array named `cell` using the dynamic code resolved from `&arraystmt`. The cell array holds the current state of the system as the program loops and examines each cell. In addition, a temporary array named `tmpcell` is allocated to hold the next state of the system as it is being calculated. The

neighbor count is calculated by assessing the cells in all eight directions from the current cell. The grid wraps around on itself to create a system that is finite but unbounded.

The next part of the DATA step checks the `neighborcount` for the current cell and assigns the fate of the cell.

```
      /*is the current cell empty?*/
      if (cell[i, j] = 0) then do;
          /*does the cell have exactly 3 neighbors?*/
          if (neighborcount = 3) then tmpcell[i, j] = 1; /*birth*/
          else tmpcell[i, j] = cell[i, j]; /*stasis*/
      end;
      else do;
          if (neighborcount < 2) then tmpcell[i, j] = 0; /*die of loneliness*/
          else if (neighborcount > 3) then tmpcell[i, j] = 0; /*die of overcrowding*/
          else tmpcell[i, j] = cell[i, j]; /*stasis*/
      end;
 end;
```

After the program loops over all of the cells it saves the `tmpcell` data back into the `cell` array which stores them back into the single-row data set to await the next call. Finally, the DATA step returns the state of the simulation back to the Flash movie by writing the cell data as a urlencoded string of name/value pairs such as: `&c_1_1=1&c_1_2=0`… The result is an attractive, animated simulation of The Game of Life powered by SAS.

CONCLUSION

The SAS/IntrNet Application Dispatcher enables the power of your SAS application to come to life in the user's web browser. By generating dynamic web pages including HTML, graphics, and browser scripting you can create new and exciting modes of information delivery. Unfortunately, it is often a challenge and a considerable amount of effort to create an application that is powerful, aesthetically pleasing, and easy to maintain using these traditional elements of web application development.

Macromedia Flash provides an environment that makes it easy to add the "eye-candy" to web-based business applications. With its nearly ubiquitous deployment along side virtually every web browser, it maintains the high level of compatibility that developers are accustomed to with HTML-based applications while conveniently extracting and isolating the visual components of an application from the back-end SAS programs. And, as this paper illustrates, it is not very difficult to connect these technologies and open a whole new realm of application development.

REFERENCES

Macromedia, Inc. (2003), macromedia FLASHMX 2004, ActionScript Reference Guide, First Edition, San Francisco, CA: Macromedia, Inc.

Lee Stemkoski. "Conway's Game of Life." 2004. http://www.geocities.com/CapeCanaveral/Hangar/7773/life.html.

Macromedia, Inc. "Flash MX 2004." 2005. http://www.macromedia.com/software/flash/.

TRADEMARK CITATION

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

John Leveille
d-Wise Technologies, Inc.
3115 Belspring Ln
Raleigh, NC 27612
Voice: 919.880.9068
Fax: 919.510.8391
jleveille@d-wise.com
http://www.d-wise.com

Joey Leveille
SAS Institute, Inc.
100 SAS Campus Dr
Cary, NC 27513
Voice: 919.531.3125
Fax: 919.677.4444
Joey.leveille@sas.com
http://www.sas.com