SAS® GLOBAL FORUM 2020 #SASGF

Paper 4260-2020

# Build an HTML5 App Using SAS®

Allan Bowe

## Abstract

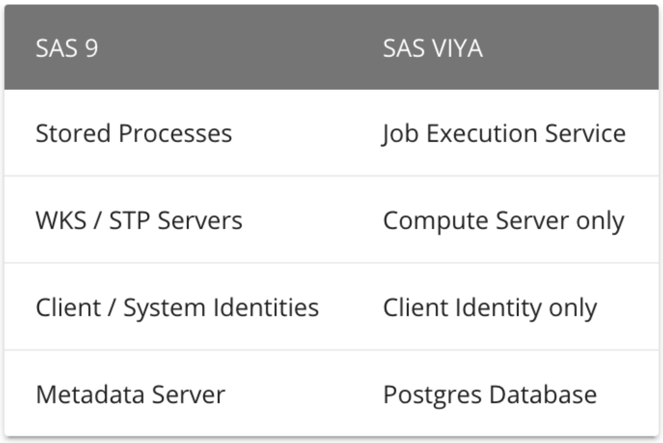
With SAS® you have the Power to Know, HTML5 provides the "Power to Show"! A workshop to kickstart the development of new web apps on SAS Platforms.

## Introduction

Building HTML5 apps is now easier than ever, thanks to the growth of open source tooling designed to work specifically with SAS. This paper will walk through some examples of building web apps, and provide a bunch of tips & tricks along the way.

## SAS 9 or SAS Viya?

From a "building web apps" perspective there are a few high level differences to be aware of between the two server types:



| SAS 9 | SAS VIYA |
|---|---|
| Stored Processes | Job Execution Service |
| WKS / STP Servers | Compute Server only |
| Client / System Identities | Client Identity only |
| Metadata Server | Postgres Database |

Figure 1 - comparison of SAS 9 and SAS Viya.

Both server types stream content to the automatic _webout fileref and can make use of the _debug parameter. SAS 9 can run regular SAS code plus metadata enabled functions / procs. SAS Viya can run regular SAS code plus CAS enabled procs.

For the rest of this paper, the term "service" refers to either a Stored Process or Job Execution Service.

## SPACE9 INVADERS

One approach for building web apps on SAS is to stream web content directly from the service itself. An example of this can be seen by running the following code snippet (SAS9, internet access needed):

```
filename playme url "https://sasjs.io/game";
%inc playme;
```

The above will create a Stored Process in the My Folder tree. Open the log and copy the url into a browser and you will find a SAS-streamed space-invaders game.

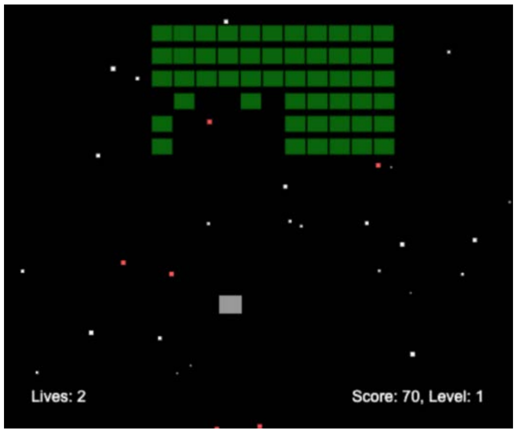


Figure 2 - Space Invaders on SAS® (credit to dwmkerr)

The main advantage of the streaming approach is that you really do not need anything except services to serve your content. The main disadvantage is that all your content must come directly from services.

This causes a number of challenges:

- Line length limits. Some JavaScript (JS) variables / strings are longer than 32767 characters.
- Responsiveness. Calling a service has overhead (~400ms in SAS 9, ~3s in Viya).
- Complexity. It's harder (more expensive) to maintain apps that are built this way.

If your use case involves compiled JS, minified CSS, images, frameworks, and all the fantastic tooling available in the Node Package Manager (NPM) then - read on.

## CORE SERVICES

Registering services in SAS can be done in a jiffy using the MacroCore library (https://github.com/macropeople/macrocore). This is a set of SAS macros divided into the following categories:

- macro functions (can be used in open code)
- macro procedures (generate and execute base code)

- metadata macros (require metadata)
- viya macros (require Viya)

To compile these macros, simply run the following code:

```
filename mc url "https://raw.githubusercontent.com/macropeople/macrocore/master/mc_all.sas";
%inc mc;
```

If you do not have internet access from your SAS session, simply open the location above in your browser and copy/compile the macros into your editor.  Check the README for more options.

## SAS 9

Creating services in SAS9 is now very straightforward - simply make your code available with a fileref, and run a macro to register it.  The following code will create the two services for our seed apps:

```
%let appLoc=/SOME/META/FOLDER/myApp; /**/
filename ft15f001 temp;
parmcards4;
  proc sql;
  create table areas as select distinct area
    from sashelp.springs;
  %webout(OPEN)
  %webout(OBJ,areas)
  %webout(CLOSE)
;;;;
%mm_createwebservice(path=&appLoc/common, name=appInit
  , code=ft15f001 ,replace=YES)
parmcards4;
  proc sql;
  create table springs as
    select * from sashelp.springs
    where area in (select area from areas);
  %webout(OPEN)
  %webout(OBJ,springs)
  %webout(CLOSE)
;;;;
%mm_createwebservice(path=&appLoc/common, name=getData
  , code=ft15f001 ,replace=YES)
```

## SAS VIYA

Creating services in Viya requires a couple of one-time steps to set up the API credentials. This code is also available here: https://sasjs.io/quickstart/services/#viya

### Step 1 - Get API Credentials

You need to be an administrator for this part, as you will need to query the consul token. Run the following in a SAS Viya session (SASStudio or SASStudioV):

```
%let client=new%sysfunc(ranuni(0),hex16.);
%let secret=MySecret;
%mv_getapptoken(client_id=&client,client_secret=&secret)
```

### Step 2 - Get Access Token

The macro in step 1 will print a URL to the log. Open this in a browser to retrieve an authorisation code. This code is used in the first macro below (code= parameter).

```
 %mv_getrefreshtoken(client_id=&client
    ,client_secret=&secret
    ,code=5qAdLPYiiw)
 %mv_getaccesstoken(client_id=&client,client_secret=&secret)
```

This access token will expire. Keep a copy of your CLIENT, SECRET, and REFRESH_TOKEN to regenerate it.

### Step 3 - Make Viya Services

This step is now the same as SAS9 except a slight change in the macro prefix (mv vs mm).

```
%let appLoc=/Public/myapp;
filename ft15f001 temp;
parmcards4;
  proc sql;
  create table areas as select distinct area from sashelp.springs;
  %webout(OPEN)
  %webout(OBJ,areas)
  %webout(CLOSE)
;;;;
%mv_createwebservice(path=&appLoc/common, name=appInit
  , code=ft15f001 ,replace=YES)
parmcards4;
  proc sql;
  create table springs as
```

```
    select * from sashelp.springs
    where area in (select area from areas);
  %webout(OPEN)
  %webout(OBJ,springs)
  %webout(CLOSE)
;;;;
%mv_createwebservice(path=&appLoc/common, name=getData
  , code=ft15f001 ,replace=YES)
```

Right - that's the backend sorted.

## Speedy Seed Apps

Now, whether you be a JS ninja or first time noob, you'll save time in building your first web app if you can take an example of something that's been done before, right?

A seed app is a "quick start" code repository with some example components and a minimal framework to get you started. Examples exist for Angular, React, and also a Minimal version for the JS die-hards out there.

Below follow the steps for setting up the React version. You will need to have GIT, Node JS, and ideally a copy of Visual Studio on your system. This can all be done on a locked-down windows machine - here is a guide for those who are "administratively challenged": https://sasjs.io/windows/

Once set up, and using your favourite shell, we can execute:

```
git clone https://github.com/macropeople/react-seed-app
cd react-seed-app
npm install
```

The last command will install all of the third party libraries as detailed in the `package.json` file. In the meantime let's find the config file and tell JS where SAS is located. Search the project for "`new SASjs`" or check directly in `src/context/sasContext.tsx`. We find the following parameters:

- serverUrl - this is the hostname of your SAS Web server
- port - if your normally see a port in your url, enter it here
- appLoc - The location of the app, as used when building the services above
- serverType - should be either SASVIYA or SAS9
- debug - setting this to `true` will enable debug mode.

As the seed apps are pre-built, that's the only frontend part we need to configure. We can run `npm start` to launch a local version of the app, or `npm run build` to create a production version ready for deployment.

## Ready, Set, Launch!

So we've registered the services and we've configured our app locally. Time to check it out! Wait - "how do I get it onto the Web Server" I hear you ask? Yep, that's a typical challenge. The server names will have been shown to you in the log when you created the services (a macro to extract is also available). Your web content needs to go in the htdocs (static content) folder which is typically located here:

- **SAS 9:** `[SASCONFIG]/Lev1/Web/WebServer/htdocs`
- **SAS Viya:** `/var/www/html`

You have a number of options for getting your frontend into this location:

- rsync command
- map a network drive and copy (or robocopy) it across
- push the code elsewhere (a git repo?) and the pull it back using a cron job or service

If you really have no way to get to that location, there are other options. You could use a different server, and whitelist the IP in SAS Environment Manager. You can even use a command such as `sasjs` to create a streaming version of the app. More details on that follow in the next section.

## SASjs - the Framework

SASjs is an open-source framework for building apps on SAS®. It provides everything you need to build apps in a quick, repeatable, and maintainable fashion. The components are:

- macrocore - a set of SAS macros designed for SAS application development
- SASjs - the adapter that handles SAS-JS communication
- sasjs-cli - a client tool used for project instantiation and building of SAS files

Each of these components can be used individually, or with seed apps, as your use case permits.

### SASjs - the Adapter

The adapter is made up of a javascript function and a SAS macro. The SAS macro is automatically created when building apps using `mX_createwebservice` and the JS function is made available when running `npm install sasjs` or by including the following script tag (with latest version):

```
  <script src="https://cdn.jsdelivr.net/npm/sasjs@1.9.0/index.js" crossorigin="anonymous"
     integrity="sha384-ppDMKTgsjRnY/9ORpxSHFFFo/0yv70I/IYjyWClq1dXOxYkFWaw+lCWU6BSjMKMP">
  </script>
```

A call to the adapter looks like this:

```
   const data: any = {
      table1: [{ pi: 3.14159, magic: 1.618, answer: "42" }]
    };
    adapter.request("common/getData", data).then((res: any) => {
       /* do stuff with the `res` object */
    });
```

On the SAS side we have a single macro with the following functions:

- `%webout(FETCH)` - converts the tables sent by JS into SAS datasets
- (all your SAS code goes here)
- `%webout(OPEN)` - prepares the JSON to be returned to SAS
- `%webout(OBJ,dataset1)` - send dataset in Object format (easy to work with)
- `%webout(ARR,dataset2)` - sends dataset in Array format (faster for large files)
- `%webout(CLOSE)` - closes off the JSON and sends some additional data.

For more details on the adapter itself, see the README file.

## SASjs-CLI

The CLI (or, client) tool is a command-line facility that can be used to create, and then build, the SAS part of your web app project. The tool is installed as follows:

```
npm i -g sasjs-cli
```

This will install the repo in the global node_modules folder and add the folder to your PATH - making the sasjs command available.

From now on, if you are starting a new web app project, simply run the following command:

```
sasjs create myNewApp
```

This will create a new folder (called myNewApp) and initialise an NPM project within it, including `macrocore` as a dependency. It will also create a specific folder structure, that should be retained to enable the build process. These folders are:

- build - containing the serviceinit / serviceterm / buildinit programs
- db - containing the datalines and ddl for your libraries
- macros - all macros used in your services
- services - to contain all the services, divided into subfolders.

A few files are worth mentioning specifically:

- serviceinit.sas - this will be added to the start of each service (after the macros)
- serviceterm.sas - added to the end of each service
- buildinit.sas - contains the build configuration (eg appLoc and serverType)
- appInit.sas - this should always be the first service that your app calls on startup.

Notice also that every service has a specific header structure. This is to enable doxygen documentation. An example of doxygen documentation can be found here: https://core.macropeople.com/files.html

Each header also lists dependencies. This is a list of all the macros used by the service, macro, or program. This list is used by the next function of the cli tool - the build step.

```
sasjs build
```

It's a short command. But it packs a punch! This command will first create a folder in your project called sasbuild. It will then create one file for each service, containing:

- All the dependent macros (located in the macros folder or macrocore library)
- The serviceinit.sas program
- The code for the service itself
- The serviceterm.sas program

At this point we have a nice collection of self contained service definitions. But SASjs doesn't stop there. Next up we concatenate all of these files, stripping out comments, including the buildinit.sas program and wrapping with the service creation macros.

This will give us a single deploywebservices.sas program that can be executed in SAS Studio to create all of our services!

For many that should be enough to start building automated pipelines and enterprise grade web applications. For others - there is still the basic challenge of getting access to that blasted web server.

May the SASjs be with you:

```
   sasjs stream
```

This command will take build|dist/index.html file to create a deploystream.sas program in sasbuild. This is executed to create a new service in the root of your deployed project that streams the HTML and associated JS and CSS content. Which means, you are no longer betrothen to the gods of SAS Web Servers for building Amazing Things.

### MacroCore

These macros are automatically added to your services as precode by the sasjs-cli when they are listed as dependencies in the header section.

There are utilities for creating folders, listing users, performing recursive joins, mapping libraries, you name it. The library can be found here:
https://github.com/macropeople/macrocore

## CONCLUSION

The joy of building open source web applications is the speed at which you can deliver business value in a scalable and secure fashion. Using GIT, VSCode, NPM and SASjs you can Experience Your New Possible.

For up to date information on building web apps with SAS, be sure to revisit https://sasjs.io

## Recommended Resources

- https://sasjs.io - information on building web apps with SAS
- https://github.com/macropeople/sasjs - adapter for SAS-JS communication
- https://github.com/macropeople/sasjs-cli - CLI tool for SAS projects
- https://github.com/macropeople/macrocore - Macro Library for Application Development
- https://github.com/macropeople/react-seed-app - Seed app using React
- https://github.com/macropeople/angular-seed-app - Seed app using Angular
- https://github.com/macropeople/minimal-seed-app - Seed app with minimal code
- https://js.cards - quiz game to test your JS skills!
- https://sasensei.com/flash/45 - A-Z of building web apps with SAS
- https://sasensei.com/questions/filter?tags_any=[13] - quiz on building SAS web apps
- https://datacontroller.io - Example of a commercial HTML5 SAS web app
- https://www.youtube.com/watch?v=vSNBea_M8yU - Build a Viya web app in 5 mins

## Contact Information

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

Allan Bowe
allan@bowe.io
https://www.linkedin.com/in/allanbowe