Fast Dashboards: Producing real-time dashboard using SAS® Event Stream Processing.

From Machine Time to Human Time

Fast is a relative term. In the connected, always-on, real-time world computers can continuously send information much faster than humans can understand, categorize or visualize it. Much of what we use computers for is to slow down that information data stream in order to aggregate, transform, analyze and store it for use at a future time and date.

Event stream processing is a technology used to help with this task. It is an incredibly fast, in-memory based technology designed to examine streaming data at machine time speeds: hundred, thousands and millions of events per second. At the same time, this technology puts that data steam into a better, understandable context for ultimate consumption by humans.

The term dashboard has become the catch-all word to describe the grouping of tables of data and data visualization, in a quick, conveyable format such as a web page.

Thus the term "fast dashboards" is being used in several ways. One, to demonstrate the ease of connecting the dashboards to the streaming data. Two, to capture the time it takes for dashboard development. Three, to depict the rate of data updates to dashboard visualizations.

The purpose of this paper is to examine several dashboard technology options and how to connect them to SAS Event Stream Processing models. In the process, I will cover some of the issues and trade-offs when using these technologies together.

All of the dashboard technologies are web enabled and utilize the components of HTML5, in particular JavaScript, albeit, at very different levels of implementation.

I will cover 3 different dashboard technologies:

- SAE ESP Stream Viewer
- Freeboard
- D3 JavaScript

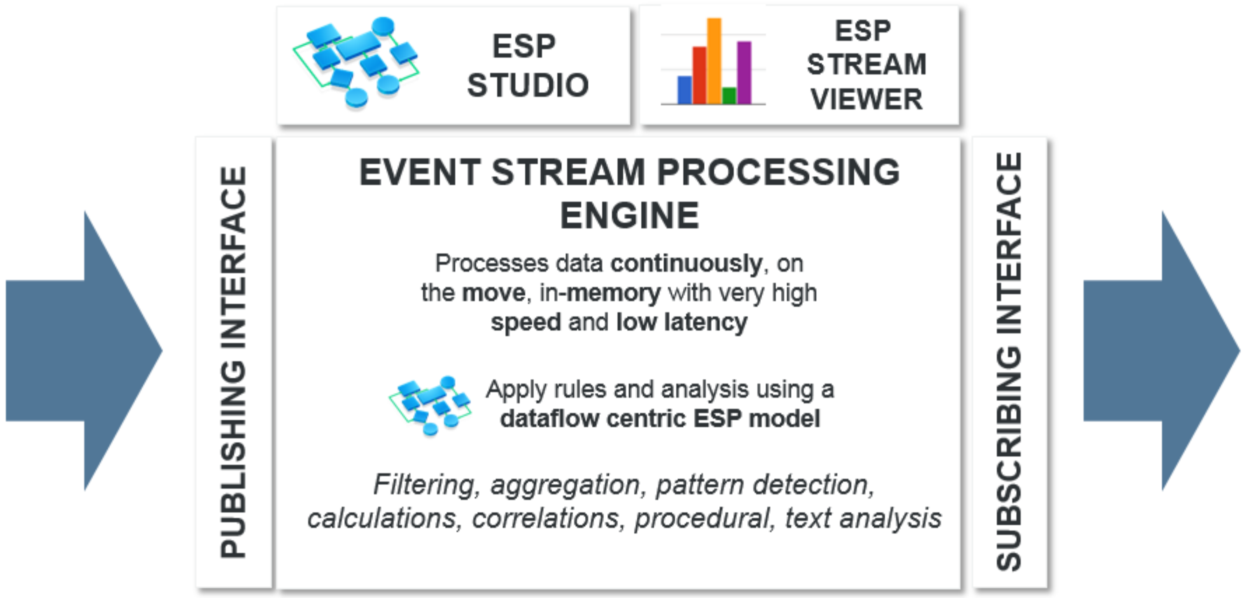
SAS® Event Stream Processing

SAS Event Stream Processing (ESP) is an in-memory analytics engine designed to work on real-time streaming data.

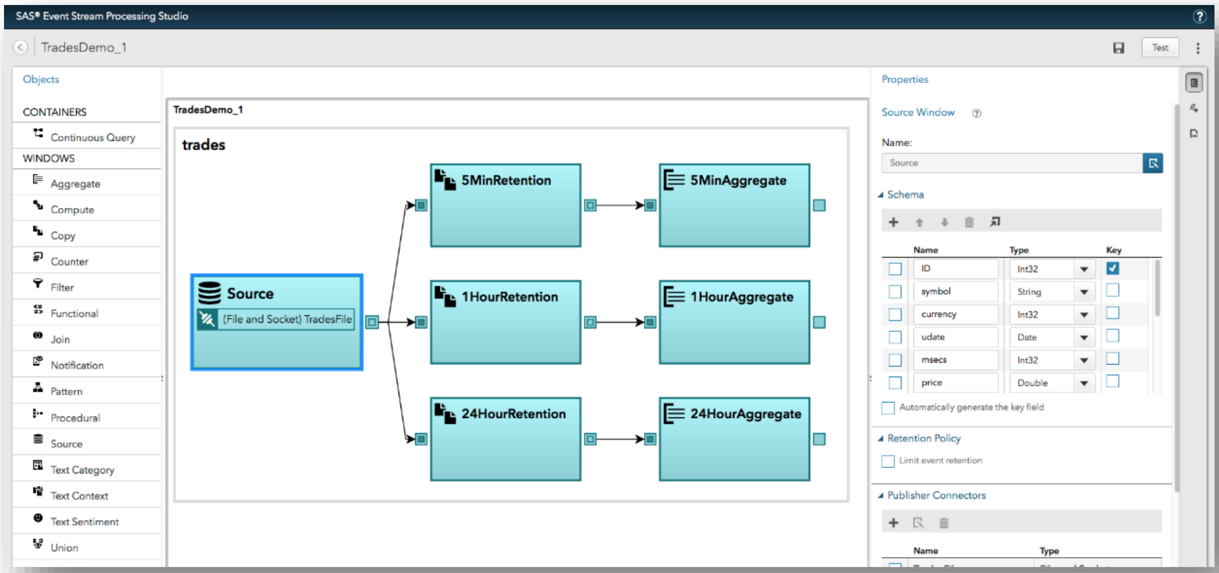
ESP is lightweight and scales well. Thus, it can be run on small low powered IoT edge devices to high capacity data center and expansive cloud configuration. More importantly, it can link the data streams between those devices and provide analysis at the different levels of devices.

ESP is made up of many components, but for purpose of this explanation the focus is on ESP Studio, ESP Stream Viewer and the ESP Engine.

A functional diagram of SAS Event Stream Processing.


ESP STUDIO
ESP STREAM VIEWER
EVENT STREAM PROCESSING ENGINE
Processes data continuously, on the move, in-memory with very high speed and low latency
Apply rules and analysis using a dataflow centric ESP model
Filtering, aggregation, pattern detection, calculations, correlations, procedural, text analysis
PUBLISHING INTERFACE
SUBSCRIBING INTERFACE

ESP Studio is a visual development environment that is used to create and test ESP models.



Name: Source

Schema

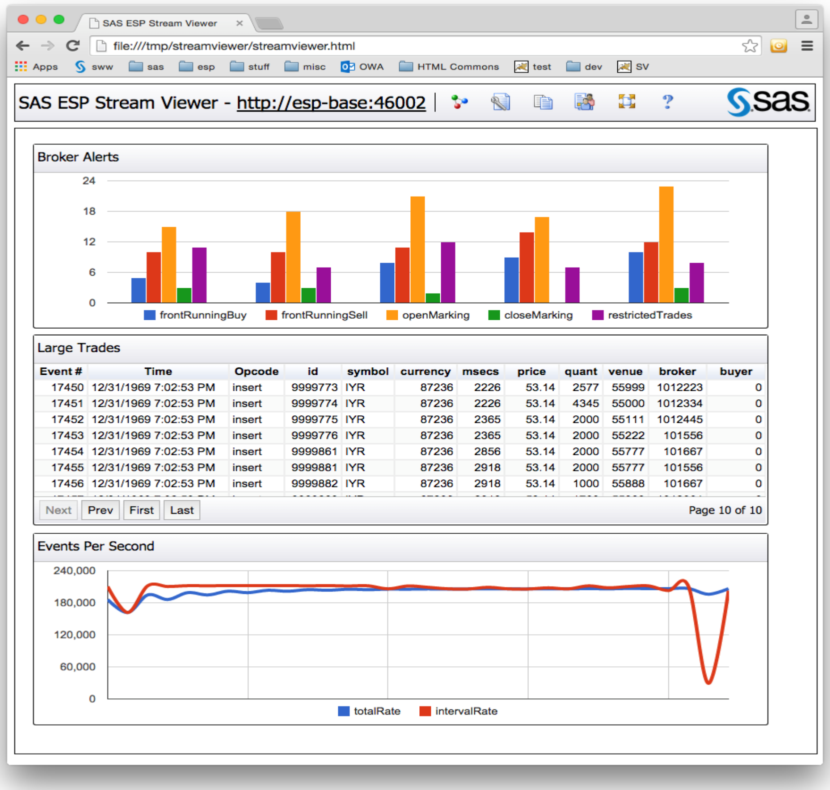
	Name	Type	Key
☐	ID	Int32	✓
☐	symbol	String	☐
☐	currency	Int32	☐
☐	udate	Date	☐
☐	msecs	Int32	☐
☐	price	Double	☐

☐ Automatically generate the key field

Retention Policy

☐ Limit event retention

ESP Stream Viewer is the out-of-the-box dashboard tool for ESP.



Large Trades

Event #	Time	Opcode	id	symbol	currency	msecs	price	quant	venue	broker	buyer
17450	12/31/1969 7:02:53 PM	insert	9999773	IYR	87236	2226	53.14	2577	55999	1012223	0
17451	12/31/1969 7:02:53 PM	insert	9999774	IYR	87236	2226	53.14	4345	55000	1012334	0
17452	12/31/1969 7:02:53 PM	insert	9999775	IYR	87236	2365	53.14	2000	55111	1012445	0
17453	12/31/1969 7:02:53 PM	insert	9999776	IYR	87236	2365	53.14	2000	55222	101556	0
17454	12/31/1969 7:02:53 PM	insert	9999861	IYR	87236	2856	53.14	2000	55777	101667	0
17455	12/31/1969 7:02:53 PM	insert	9999881	IYR	87236	2918	53.14	2000	55777	101556	0
17456	12/31/1969 7:02:53 PM	insert	9999882	IYR	87236	2918	53.14	1000	55888	101667	0

Next Prev First Last Page 10 of 10

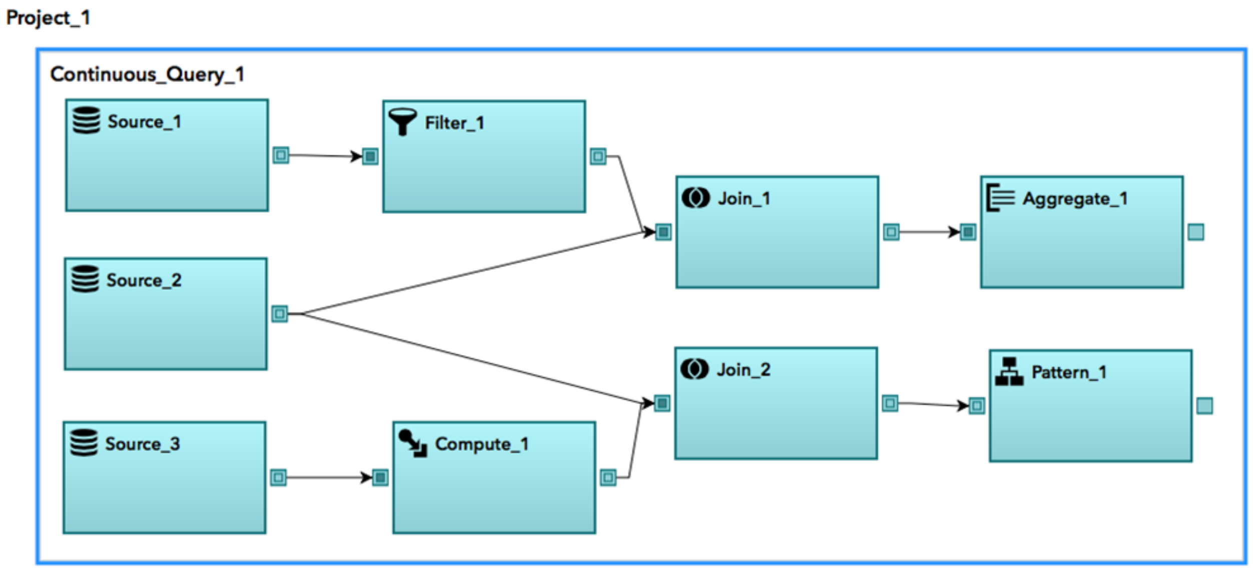
ESP Engine is the computing container used to house ESP analytic models.

Three Layers of ESP models: Project, Continuous Query, Windows

ESP models as they relate to producing dashboards are made up of 3 layers: Projects, Continuous Queries and Windows.

The functionality of ESP analytical models are group together into Projects. Projects are made up of one or more Continuous Queries. A Continuous Query is the flow of data, broken-down into functional steps called Windows. ESP's Windows provide data definition, analysis and some SQL like functionality for manipulating the flow of data through the Continuous Query and Project. Each Window provides a snapshot of data. We can make a connection to any Window in a Project/Continuous Query.

Diagram: Example of ESP Project for receiving and manipulating the data stream. Notice in the diagram the Project houses the Continuous Query and the Continuous Query contains linked Windows.



Connecting to the ESP data

There are many methods for connecting to ESP and its data stream. I will focus ESP's REST interface. The REST interface uses ESP's Publish and Subscribe interfaces for moving data in and out of ESP. All the dashboard technologies being covered can use REST based technologies to receive data.

REST - Representational State Transfer

REST (Representational State Transfer) is a web based architectural design style for creating scalable and stateless, distributed systems of communication. Roy Fielding introduced REST in his 2000 doctoral dissertation.[1,2]

- REST messages use the web standard URI's/URL's and HTTP methods.
- REST uses JSON or XML data formats to represent data objects and attributes.
- HTTP methods (for example, GET, POST, PUT, and DELETE) are mapped to CRUD (create, retrieve, update, delete) operations.

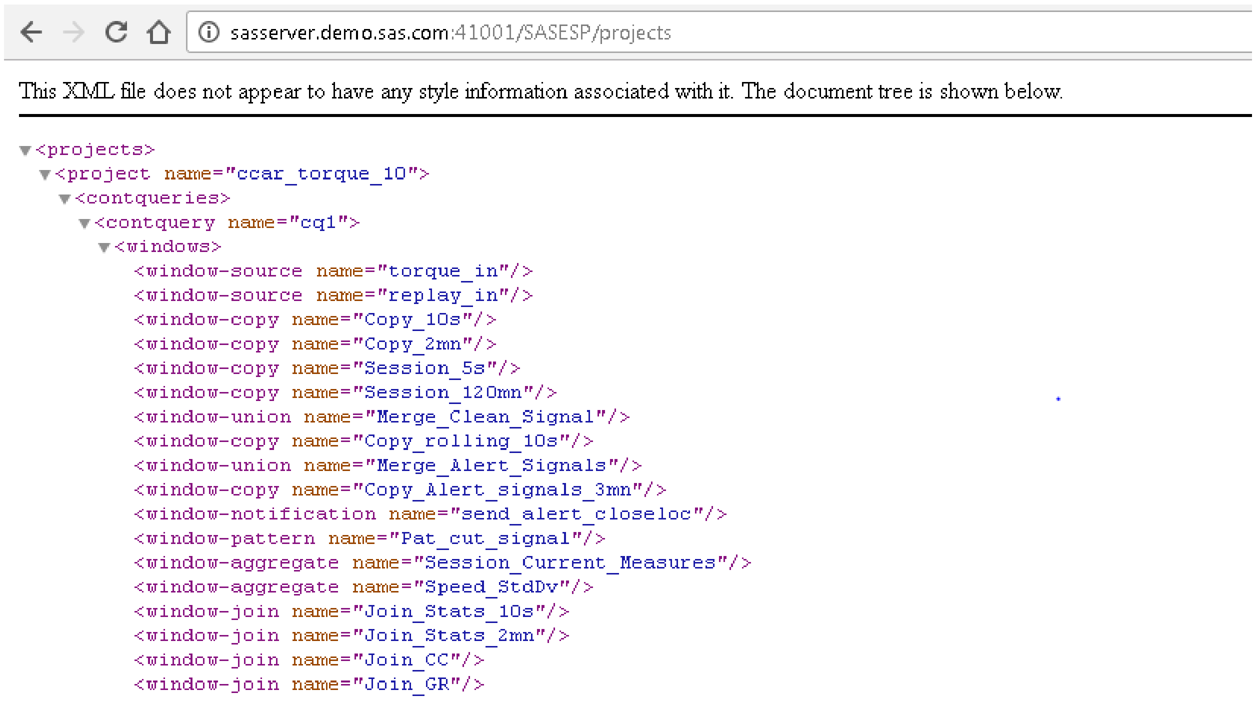
Rest is used in ESP to manage projects as well as to inject and retrieve data. Because of REST's adherence to HTTP methods, a web browser can be used for testing ESP's REST interface. Browsers use the GET http method (GET=Read) when using the URL address bar to pull information. The URI format for ESP's base REST address is http://espServer:port/SASESP

ESP Project Query

To pull an ESP project using a web browser, the URI used in the ULR address bar is:

http://espServer:port/SASESP/projects

Below is an example of pulling project information from ESP. Notice the Project information is returned as XML. In addition, the ESP Model Windows are nested inside the Continuous Query and Project XML.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<projects>
  ▼<project name="ccar_torque_10">
    ▼<contqueries>
      ▼<contquery name="cq1">
        ▼<windows>
            <window-source name="torque_in"/>
            <window-source name="replay_in"/>
            <window-copy name="Copy_10s"/>
            <window-copy name="Copy_2mn"/>
            <window-copy name="Session_5s"/>
            <window-copy name="Session_120mn"/>
            <window-union name="Merge_Clean_Signal"/>
            <window-copy name="Copy_rolling_10s"/>
            <window-union name="Merge_Alert_Signals"/>
            <window-copy name="Copy_Alert_signals_3mn"/>
            <window-notification name="send_alert_closeloc"/>
            <window-pattern name="Pat_cut_signal"/>
            <window-aggregate name="Session_Current_Measures"/>
            <window-aggregate name="Speed_StdDv"/>
            <window-join name="Join_Stats_10s"/>
            <window-join name="Join_Stats_2mn"/>
            <window-join name="Join_CC"/>
            <window-join name="Join_GR"/>
```

ESP Event Query

To pull ESP events using a web browser, the base URI used in the ULR address bar is:

http://espServer:port/SASESP/events

Also, the Project, Continuous Query and Windows need to be specified.

http://espServer:port/SASESP/events/project/continuousquery/window

To further control what event data is returned, parameters can be passed to further filter, sort and limit the amount of data returned. Ampersands (&) are used to separate parameters. A single question mark (?) is used between the query and parameters.

http://espServer:port/SASESP/events/project/continuousquery/window?filterField=filterValue&sortBy=sortField&limit=1000

- Filter – Use a filter to determine what events are returned.

  Examples:

  filter=in(Email,fahrzeug1@sas.com) or Email=fahrzeug1@sas.com

  /events/project/query/largeTrades?filter=in(broker,1012112,1012223)

  /events/project/query/largeTrades?filter=in(brokerName,'Joe','Lisa')

  /events/project/query/largeTrades?filter=and(gt(quant,50000),gt(price,1000))

- SortBy - The field on which to sort the results. The default sort direction is descending.

  sortBy=sortField

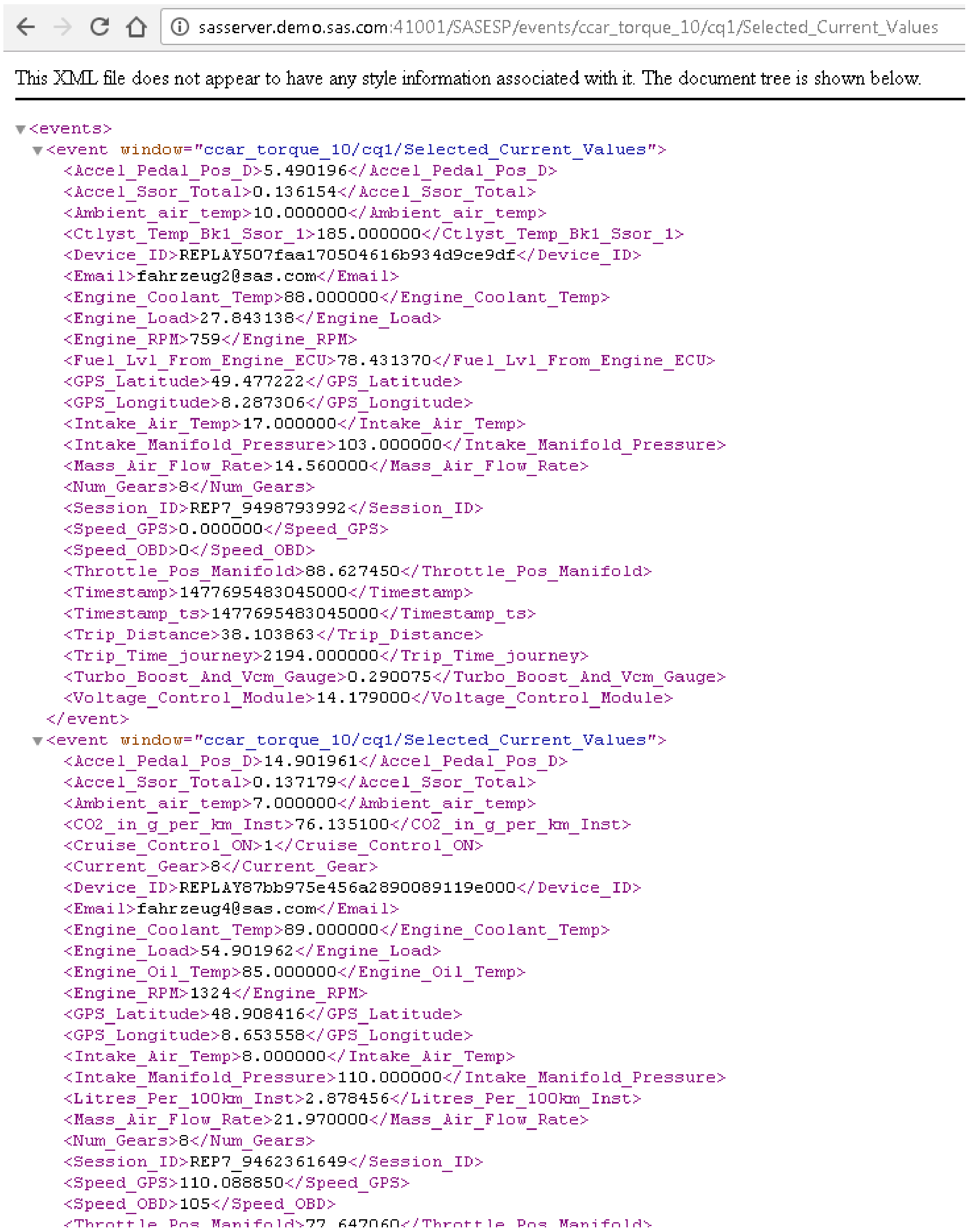
  To sort in ascending order, use sortBy=sortField:ascending

- Limit - The maximum number of events to return.

  limit=number_of_everts

  limit =100

Below is an example of events returned when running an ESP REST event query.


sasserver.demo.sas.com:41001/SASESP/events/ccar_torque_10/cq1/Selected_Current_Values

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<events>
  ▼<event window="ccar_torque_10/cq1/Selected_Current_Values">
     <Accel_Pedal_Pos_D>5.490196</Accel_Pedal_Pos_D>
     <Accel_Ssor_Total>0.136154</Accel_Ssor_Total>
     <Ambient_air_temp>10.000000</Ambient_air_temp>
     <Ctlyst_Temp_Bk1_Ssor_1>185.000000</Ctlyst_Temp_Bk1_Ssor_1>
     <Device_ID>REPLAY507faa170504616b934d9ce9df</Device_ID>
     <Email>fahrzeug2@sas.com</Email>
     <Engine_Coolant_Temp>88.000000</Engine_Coolant_Temp>
     <Engine_Load>27.843138</Engine_Load>
     <Engine_RPM>759</Engine_RPM>
     <Fuel_Lvl_From_Engine_ECU>78.431370</Fuel_Lvl_From_Engine_ECU>
     <GPS_Latitude>49.477222</GPS_Latitude>
     <GPS_Longitude>8.287306</GPS_Longitude>
     <Intake_Air_Temp>17.000000</Intake_Air_Temp>
     <Intake_Manifold_Pressure>103.000000</Intake_Manifold_Pressure>
     <Mass_Air_Flow_Rate>14.560000</Mass_Air_Flow_Rate>
     <Num_Gears>8</Num_Gears>
     <Session_ID>REP7_9498793992</Session_ID>
     <Speed_GPS>0.000000</Speed_GPS>
     <Speed_OBD>0</Speed_OBD>
     <Throttle_Pos_Manifold>88.627450</Throttle_Pos_Manifold>
     <Timestamp>1477695483045000</Timestamp>
     <Timestamp_ts>1477695483045000</Timestamp_ts>
     <Trip_Distance>38.103863</Trip_Distance>
     <Trip_Time_journey>2194.000000</Trip_Time_journey>
     <Turbo_Boost_And_Vcm_Gauge>0.290075</Turbo_Boost_And_Vcm_Gauge>
     <Voltage_Control_Module>14.179000</Voltage_Control_Module>
   </event>
  ▼<event window="ccar_torque_10/cq1/Selected_Current_Values">
     <Accel_Pedal_Pos_D>14.901961</Accel_Pedal_Pos_D>
     <Accel_Ssor_Total>0.137179</Accel_Ssor_Total>
     <Ambient_air_temp>7.000000</Ambient_air_temp>
     <CO2_in_g_per_km_Inst>76.135100</CO2_in_g_per_km_Inst>
     <Cruise_Control_ON>1</Cruise_Control_ON>
     <Current_Gear>8</Current_Gear>
     <Device_ID>REPLAY87bb975e456a2890089119e000</Device_ID>
     <Email>fahrzeug4@sas.com</Email>
     <Engine_Coolant_Temp>89.000000</Engine_Coolant_Temp>
     <Engine_Load>54.901962</Engine_Load>
     <Engine_Oil_Temp>85.000000</Engine_Oil_Temp>
     <Engine_RPM>1324</Engine_RPM>
     <GPS_Latitude>48.908416</GPS_Latitude>
     <GPS_Longitude>8.653558</GPS_Longitude>
     <Intake_Air_Temp>8.000000</Intake_Air_Temp>
     <Intake_Manifold_Pressure>110.000000</Intake_Manifold_Pressure>
     <Litres_Per_100km_Inst>2.878456</Litres_Per_100km_Inst>
     <Mass_Air_Flow_Rate>21.970000</Mass_Air_Flow_Rate>
     <Num_Gears>8</Num_Gears>
     <Session_ID>REP7_9462361649</Session_ID>
     <Speed_GPS>110.088850</Speed_GPS>
     <Speed_OBD>105</Speed_OBD>
     <Throttle_Pos_Manifold>77.647060</Throttle_Pos_Manifold>
```

Documentation for ESP REST interface can be found here:

http://go.documentation.sas.com/#/?cdcId=espcdc&cdcVersion=4.2&docsetId=espxmllayer&docsetTarget=p111ycfjon4sran1a72zunszhq5x.htm&locale=en#n03qjqoiyn8z70n17w3mp6ljh6w0
]

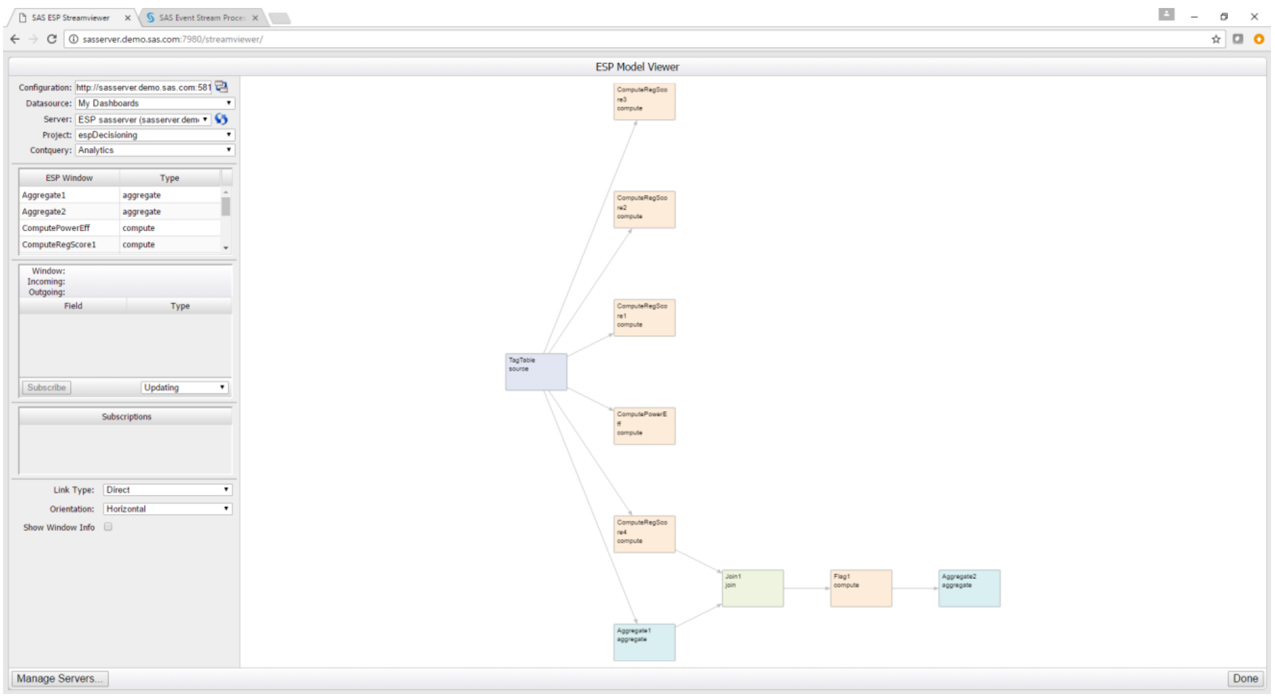
Dashboard Tools and Technologies

Each of the dashboard tools and technologies are different with their own use case. The table below is subjective and will be explained further in this section.

TECHNOLOGY/TOOL	SPEED OF DEVELOPMENT	CUSTOMIZABLE	MOBILE FRIENDLY	SPEED OF UPDATES
SAS ESP STREAMVIEWER	Rapid	Low - Medium	No	Less than 1 second
FREEBOARD	Rapid - Normal	Medium - High	Yes	1 second
D3 JAVASCRIPT	Slow	Extremely High	Yes	Less than 1 second

ESP Stream Viewer

The ESP Stream Viewer comes with ESP toolset and provides out-of-the-box event streams visualization for ESP model designers. The ESP Stream Viewer is very tightly integrated with ESP's Rest API. Thus, selecting which part of the ESP model and what Windows to visualize is made very easy. However, this tool is less customizable and would generally not be considered mobile friendly.



Configuration: http://sasserver.demo.sas.com:581
Datasource: My Dashboards
Server: ESP sasserver (sasserver.dem
Project: espDecisioning
Contquery: Analytics

ESP Window	Type
Aggregate1	aggregate
Aggregate2	aggregate
ComputePowerEff	compute
ComputeRegScore1	compute

Window:
Incoming:
Outgoing:

Field	Type

Updating

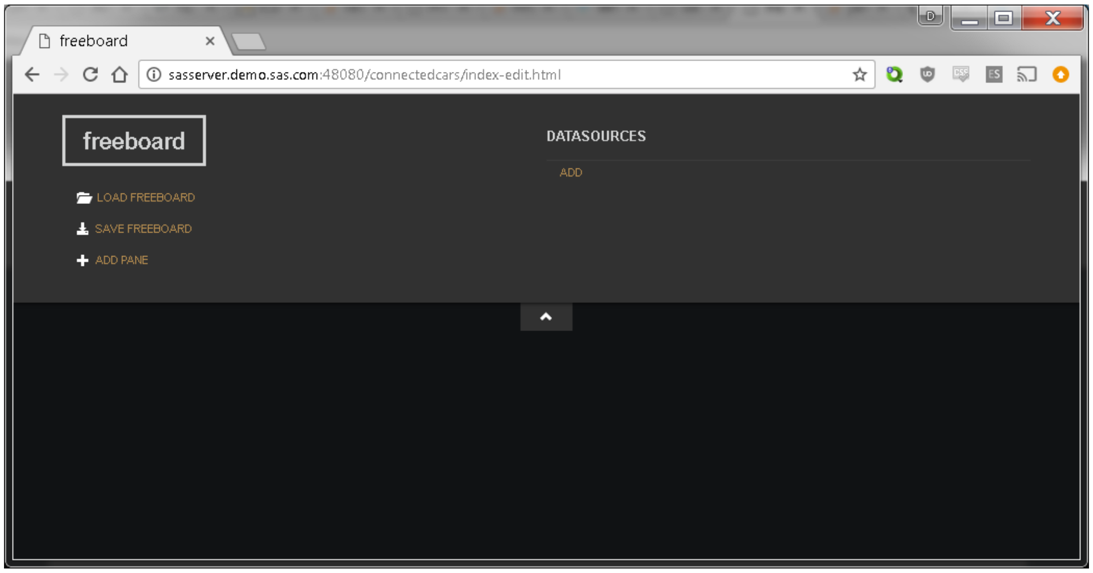
Link Type: Direct
Orientation: Horizontal
Show Window Info ☐

Freeboard

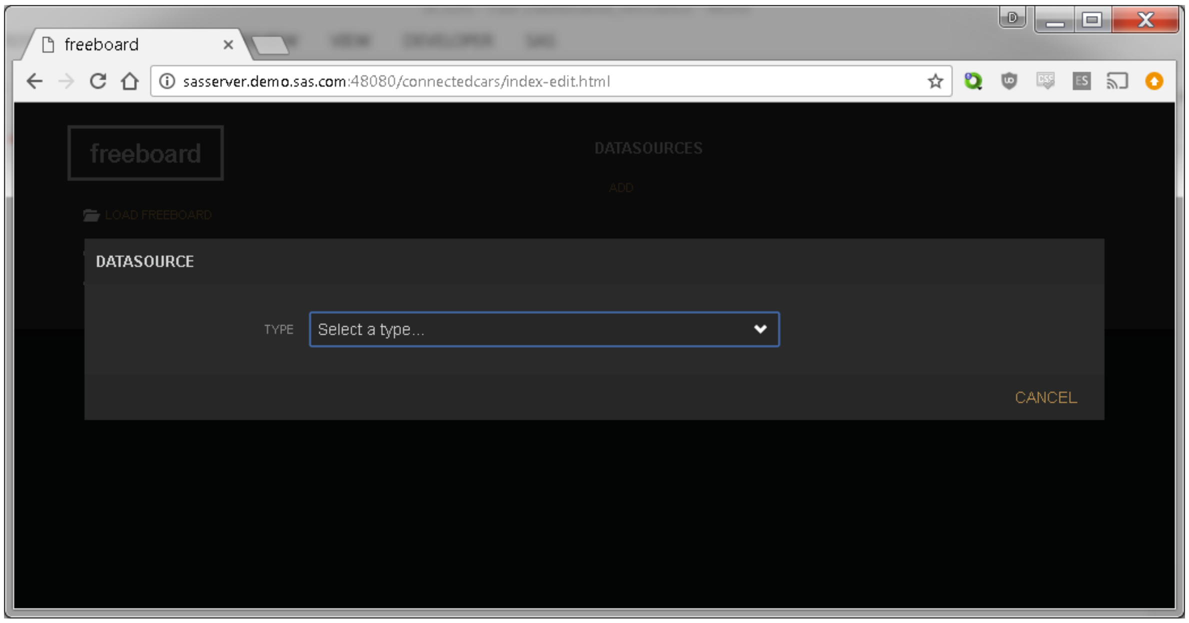
Freeboard is an Open Source dashboard that easily connects to ESP's REST services. It has many of the same visualizations found in ESP Stream Viewer and has a more managed approach to the layout of visualization widgets. Due to the more managed layout of dashboard, Freeboard can be configured for mobile and a variety of screen sizes.

It only takes a few quick steps to hook Freeboard to ESP's rest services.

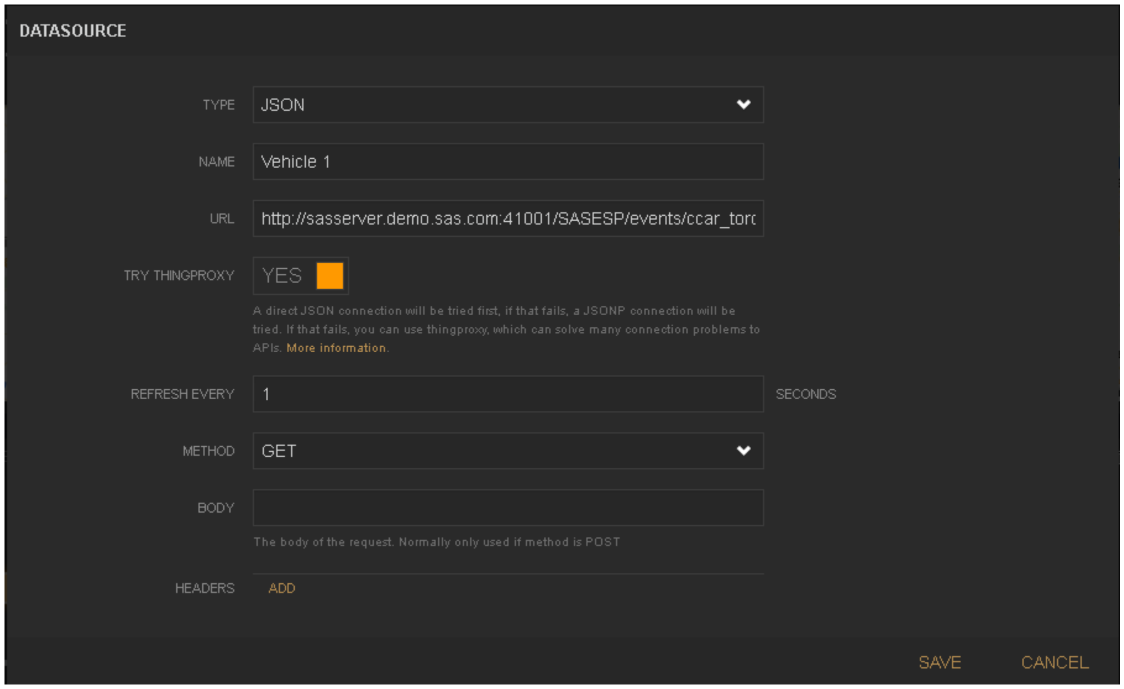
1. The first step is to put Freeboard in edit mode by calling the index-edit.html page from browser window.
2. At this screen, you can add a REST data source by clicking ADD under DATASOURCES.



3. This will bring up the DATASOURCE window. From the TYPE dropdown menu choose JSON



4. The URL is where we add the HTTP URL of the ESP's REST API. The URL follows ESP REST API convention of: http://espServer.example.com:port/SASESP/events/project/continuousquery/window to point to an ESP Window as a data source. Notice the various HTTP methods under the METHOD dropdown. Leave the default as GET method. Refresh defaults to 5 but Freeboard does quite well with 1 second refreshes.



DATASOURCE

TYPE: JSON

NAME: Vehicle 1

URL: http://sasserver.demo.sas.com:41001/SASESP/events/ccar_torc

TRY THINGPROXY: YES

A direct JSON connection will be tried first, if that fails, a JSONP connection will be tried. If that fails, you can use thingproxy, which can solve many connection problems to APIs. More information.

REFRESH EVERY: 1 SECONDS

METHOD: GET

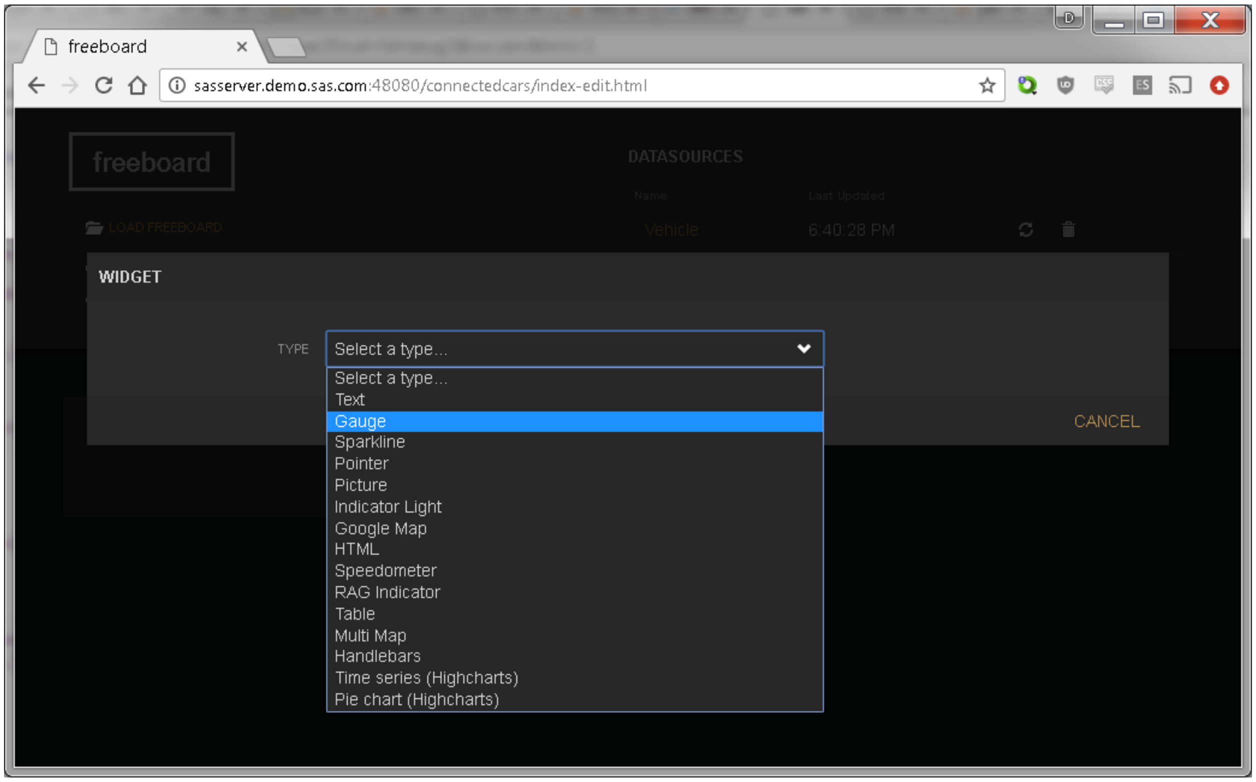
BODY:

The body of the request. Normally only used if method is POST

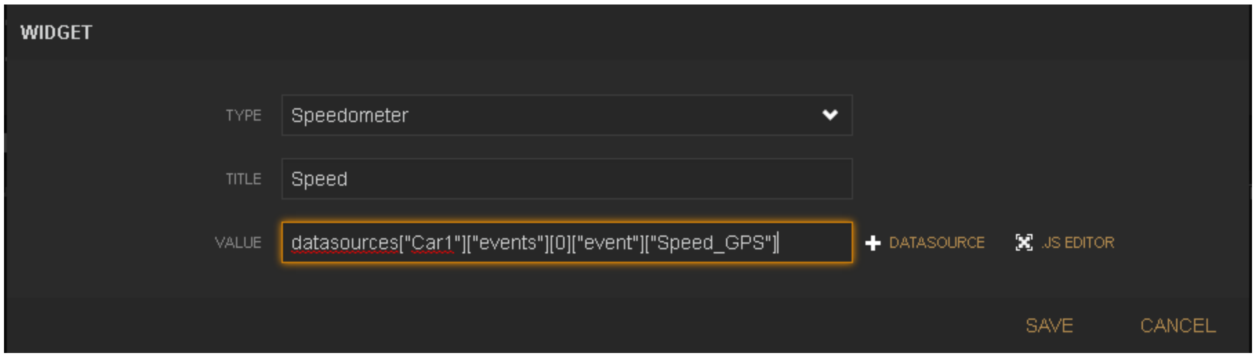
HEADERS: ADD

SAVE CANCEL

5. Next step is to add a PANE and WIDGET to the Freeboard Dashboard. This is accomplished by clinking ADD PANE and then clicking the plus (+) in the newly created PANE. This will bring up a WIDGET menu where many types of visualizations widgets can be selected



6. The last step is to add a DATASOURCE to WIDGET. The drop for VALUE will guide the user to the field values defined in the ESP REST event query. Below the field Speed_GPS was selected as the value to use for the Speedometer WIDGET.

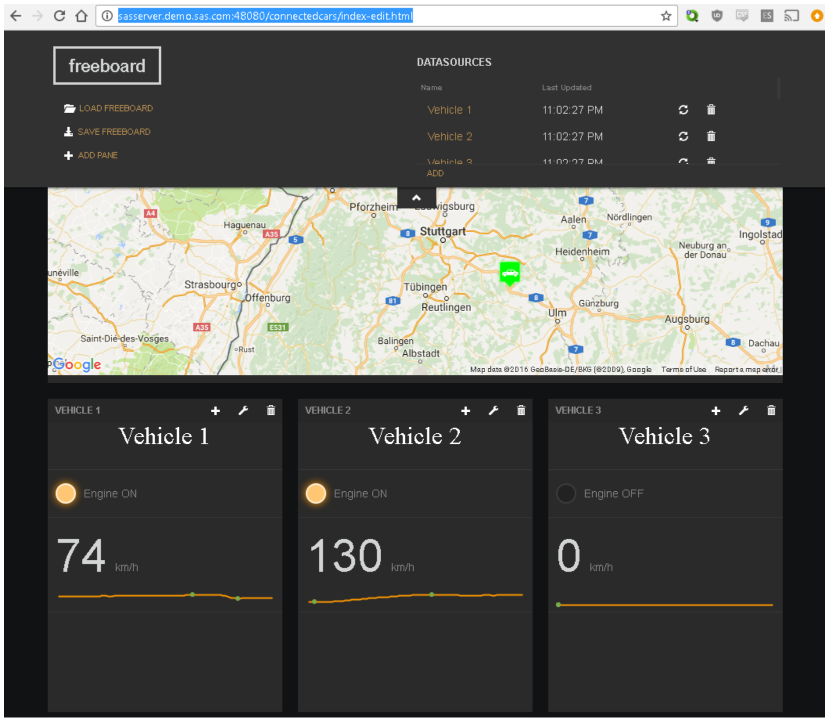

WIDGET

TYPE Speedometer

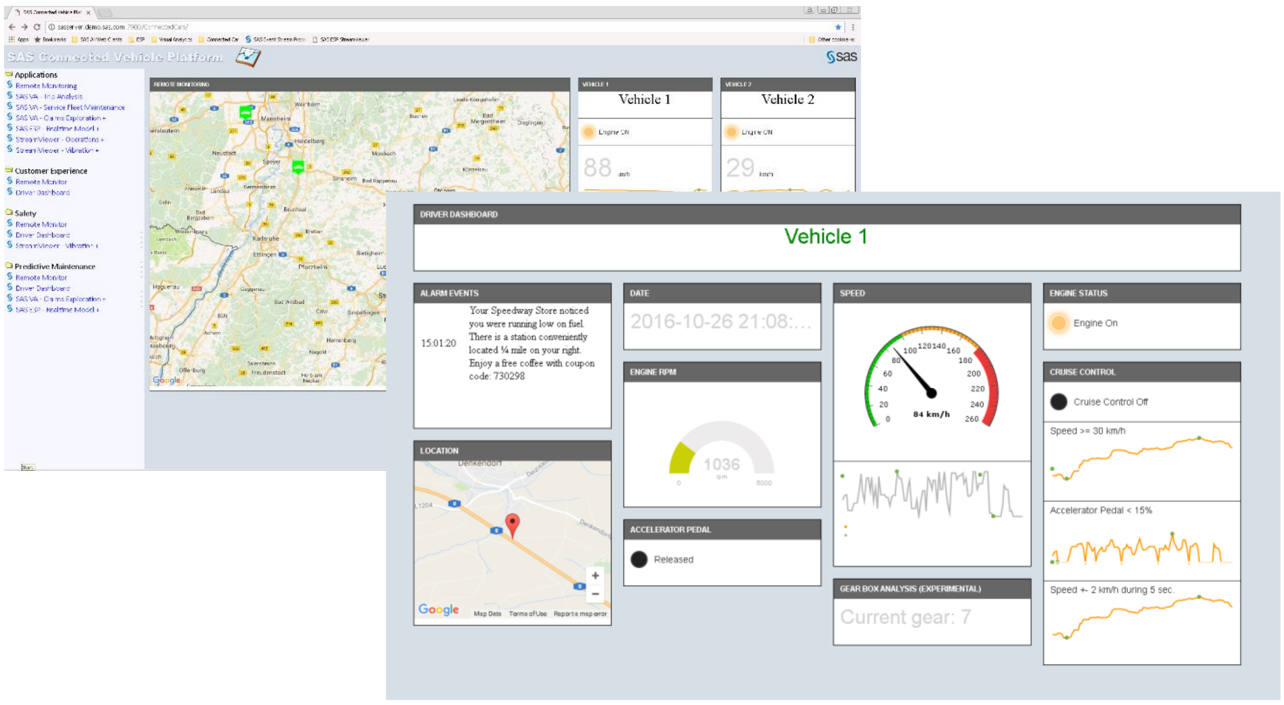
TITLE Speed

VALUE datasources["Car1"]["events"][0]["event"]["Speed_GPS"]

The default dark color scheme of Freeboard is faster when creating dashboards. This is due to the fact that many of the add-ons and plug-ins will only work well with the dark color scheme.



The lighter theme took much longer to develop due to color combination inconsistency.

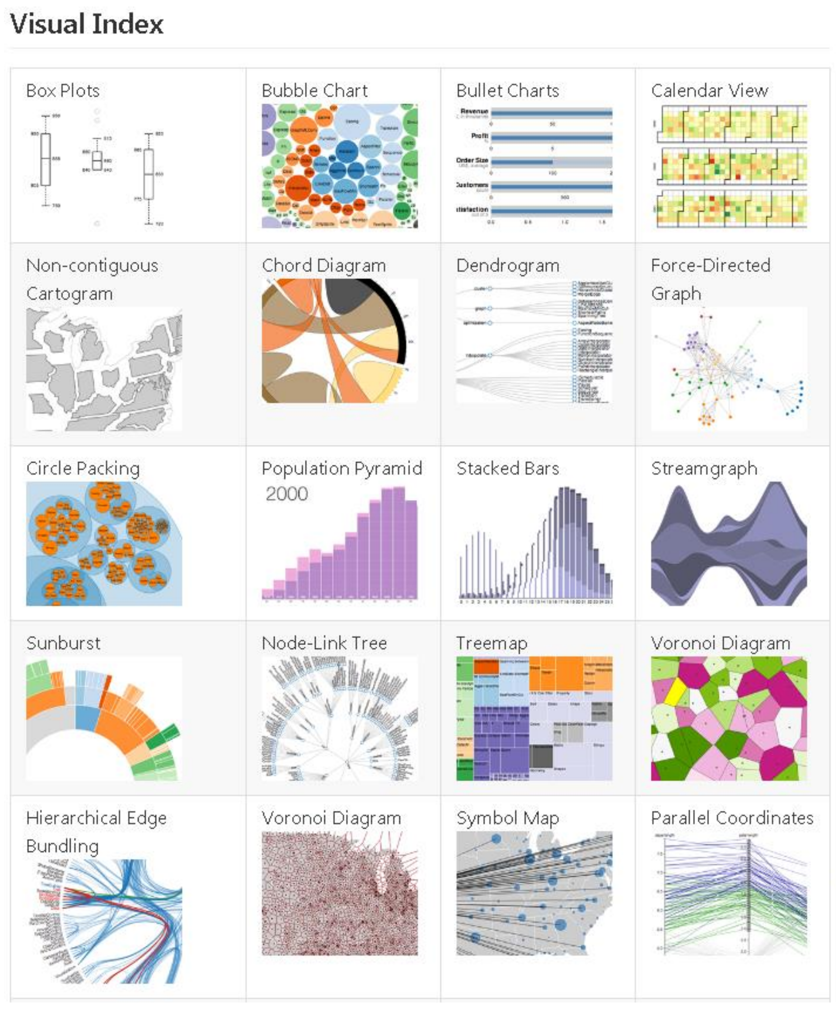


15:01:20 Your Speedway Store noticed you were running low on fuel. There is a station conveniently located ¼ mile on your right. Enjoy a free coffee with coupon code: 730298

D3 JavaScript

D3 JavaScript is a visualization programming language based on and written in JavaScript. Because D3 is a programming tool the development time for producing a dashboard is much longer. However there is no denying that visualizations and dashboards created with D3 are very compelling. D3 is also lightning fast due to fast loading, light-weight library written in web native JavaScript.

https://github.com/d3/d3/wiki/Gallery



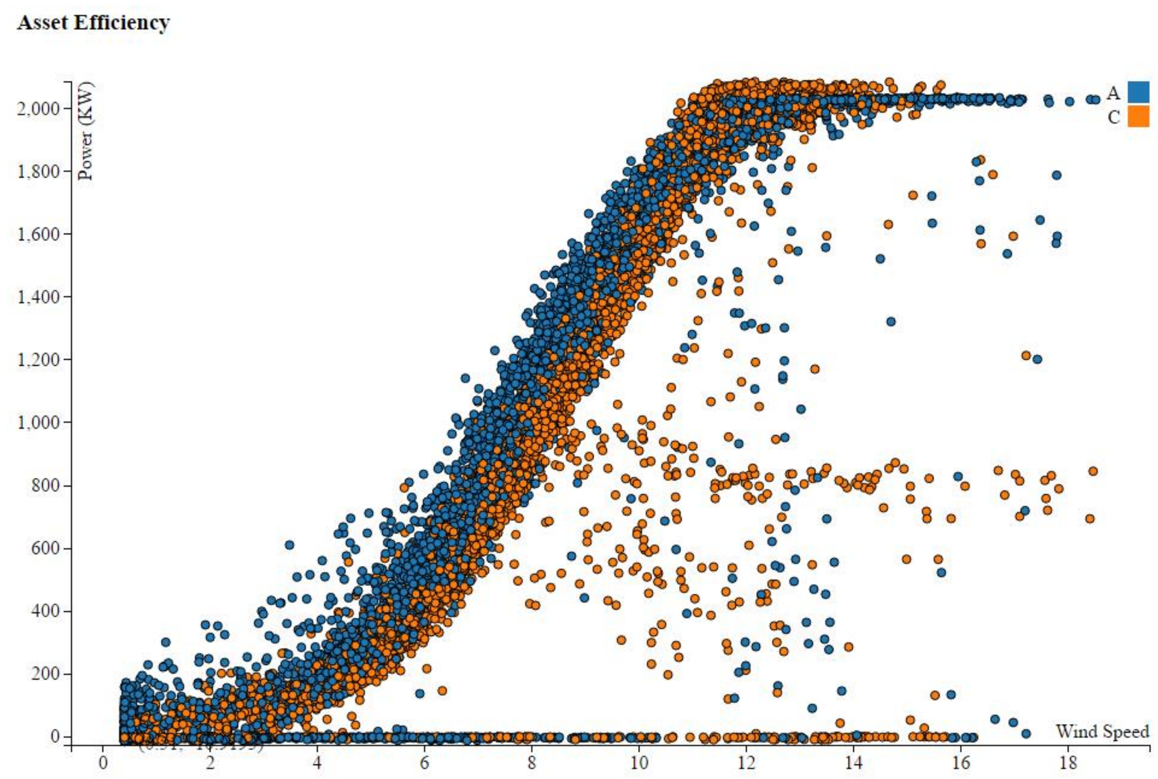
D3 has easy to use connectors to make REST calls to ESP. Below is an example of D3 function requesting JSON formatted data. However, because returned data is all character, measures need to be changed to numeric values. This done with JavaScript in D3 style.

```
d3.json("http://sasserver.demo.sas.com:41001/SASESP/events/Prj_Turbine_AC/Cont_Query_AC/Window_Source_AC?limit=10000", function(error, ac_data) {
            ac_event = ac_data.events;

          // returned data is all character therefore measures need to be changed to numeric values
            ac_event.forEach(function(d) {
                      d.event.wind_speed = +d.event.wind_speed;
                      d.event.obs_kw = +d.event.obs_kw;

  });
```

D3 is extremely fast able to product this 10,000 point scatter plot in under a second.



This is the full code to the graph on the previous page. Notice the all the elements like margins and axis can be finely controlled. Each part of the graph is broken-down into logically components in the D3 language.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="">
    <link href="images/favicon.ico" type="image/vnd.microsoft.icon" rel="shortcut icon">

    <title>Asset Efficiency</title>

    <style>
                        .axis path,
                        .axis line {
                          fill: none;
                          stroke: #000;
                          shape-rendering: crispEdges;
                        }

                        .dot {
                          stroke: #000;
                        }

                        .tooltip {
                          position: absolute;
                          width: 200px;
                          height: 28px;
                          pointer-events: none;
                        }

    </style>
  </head>
  <body>
     <h3>Asset Efficiency</h3>
       <div id="chart1" ></div>

<!-- script src="http://d3js.org/d3.v3.js"></script -->
<script type="text/javascript" src="js/d3.min.js"></script>

 <script>

// setup chart size
```

```
var margin = {top: 20, right: 20, bottom: 30, left: 45},
    width = 960 - margin.left - margin.right,
    height = 600 - margin.top - margin.bottom;

/*
 * Value - returns the value to encode for a given data object.
 * Scale - maps value to a visual display encoding, such as a pixel position.
 * Map   - maps from data value to display value
 * Axis  - sets up axis
 */

// setup x
var xValue = function(d) { return d.event.wind_speed;}, // data -> value
    xScale = d3.scale.linear().range([0, width]), // value -> display
    xMap = function(d) { return xScale(xValue(d));}, // data -> display
    xAxis = d3.svg.axis().scale(xScale).orient("bottom");

// setup y
var yValue = function(d) { return d.event.obs_kw;}, // data -> value
    yScale = d3.scale.linear().range([height, 0]), // value -> display
    yMap = function(d) { return yScale(yValue(d));}, // data -> display
    yAxis = d3.svg.axis().scale(yScale).orient("left");

// setup fill color
var cValue = function(d) { return d.event.turbine;},
    color = d3.scale.category10();

// add the graph canvas to the body of the webpage
var svg = d3.select("#chart1").append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
    .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

// add the tooltip area to the webpage
var tooltip = d3.select("#chart1").append("div")
    .attr("class", "tooltip")
    .style("opacity", 0);


d3.json("http://sasserver.demo.sas.com:41001/SASESP/events/Prj_Turbine_AC/Cont_Query_AC/Windo
w_Source_AC?limit=10000", function(error, ac_data) {
  ac_event = ac_data.events;

// returned data is all character. change measures to numeric values
  ac_event.forEach(function(d) {
     d.event.wind_speed = +d.event.wind_speed;
              d.event.obs_kw = +d.event.obs_kw;
```

```
//    console.log(d);
   });


  // don't want dots overlapping axis, so add in buffer to data domain
  xScale.domain([d3.min(ac_event, xValue)-1, d3.max(ac_event, xValue)+1]);
  yScale.domain([d3.min(ac_event, yValue)-10, d3.max(ac_event, yValue)+1]);

  // draw x-axis
  svg.append("g")
      .attr("class", "x axis")
      .attr("transform", "translate(0," + height + ")")
      .call(xAxis)
    .append("text")
      .attr("class", "label")
      .attr("x", width)
      .attr("y", -6)
      .style("text-anchor", "end")
      .text("Wind Speed");

  // draw y-axis
  svg.append("g")
      .attr("class", "y axis")
      .call(yAxis)
            .append("text")
      .attr("class", "label")
      .attr("transform", "rotate(-90)")
      .attr("y", 6)
      .attr("dy", ".71em")
      .style("text-anchor", "end")
      .text("Power (KW)");

  // draw dots
  svg.selectAll(".dot")
      .data(ac_event)
      .enter().append("circle")
      .attr("class", "dot")
      .attr("r", 3.5)
      .attr("cx", xMap)
      .attr("cy", yMap)
      .style("fill", function(d) { return color(cValue(d));})
      .on("mouseover", function(d) {
          tooltip.transition()
               .duration(200)
               .style("opacity", .9);
          tooltip.html(d.event.turbine + "<br/> (" + xValue(d)
                 + ", " + yValue(d) + ")")
```

```
              .style("left", (d3.event.pageX + 5) + "px")
              .style("top", (d3.event.pageY - 28) + "px");
      })
      .on("mouseout", function(d) {
         tooltip.transition()
              .duration(500)
              .style("opacity", 0);
      });

  // draw legend
  var legend = svg.selectAll(".legend")
      .data(color.domain())
      .enter().append("g")
      .attr("class", "legend")
      .attr("transform", function(d, i) { return "translate(0," + i * 20 + ")"; });

  // draw legend colored rectangles
  legend.append("rect")
      .attr("x", width - 18)
      .attr("width", 18)
      .attr("height", 18)
      .style("fill", color);

  // draw legend text
  legend.append("text")
      .attr("x", width - 24)
      .attr("y", 9)
      .attr("dy", ".35em")
      .style("text-anchor", "end")
      .text(function(d) { return d;})
});

   </script>
  </body>
</html>
```

References

1 - Architectural Styles and the Design of Network-based Software Architectures

http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

2- Using HTTP Methods for RESTful Services

http://www.restapitutorial.com/lessons/httpmethods.html

FREEBOARD
https://github.com/Freeboard/freeboard

D3
https://d3js.org/