Paper SAS638-2017

How's your Sports ESP? Using SAS® Event Stream Processing with SAS® Visual Analytics to Analyze Sports Data

John Davis, SAS Institute Inc.

ABSTRACT

In today's instant information society, we want to know the most up-to-date information about everything, including what is happening with our favorite sports teams. In this paper, we explore some of the readily available sources of live sports data and look at how SAS® technologies, including SAS® Event Stream Processing and SAS® Visual Analytics, can be used to collect, store, process, and analyze the streamed data. A bibliography of sports data websites that are used in this paper is included, with emphasis on the free sources.

INTRODUCTION

Vast amounts of live data are being produced today: by automobiles, refrigerators, medical monitors, space based telescopes, ESPN and Twitter feeds, to mention a few. Much of this data is not only captured, it is also collected, and recently surfaced through various live data feeds for analysis and interpretation. The task for technologists today is to determine how to use these data feeds to make better sense of the world by looking for meaning and trends in this firehose stream of data.

Figure 1. Real-Time Data Sources

Increasingly, these live data sources are finding their way to the internet, and are accessible to the public via a variety of different application programming interfaces (APIs). Using these APIs, we can access data in real or near-real time. The challenge becomes finding a way to process this volume of data so that meaningful and actionable information can be extracted.

Figure 2. Analysis of Live Data Streams (Sports, Weather, Stocks)

In this paper I will explore one solution to this problem by applying a set of software tools now available from SAS Institute. It involves integrating several software offerings from SAS Institute with open source application development tools. To make the process fun, the project applies the software and techniques to live streaming data from a sports data website. The techniques used in this paper can be applied to any other data that is available from a website via an HTTP request.

STREAMING DATA AND THE INTERNET

A variety of techniques have been used during the lifetime of the Internet to serve up data for consumption by client applications. Generally these techniques fall into either Push or Pull technology. As the names imply, they differ by whether the data is pushed out to the client through a persistent

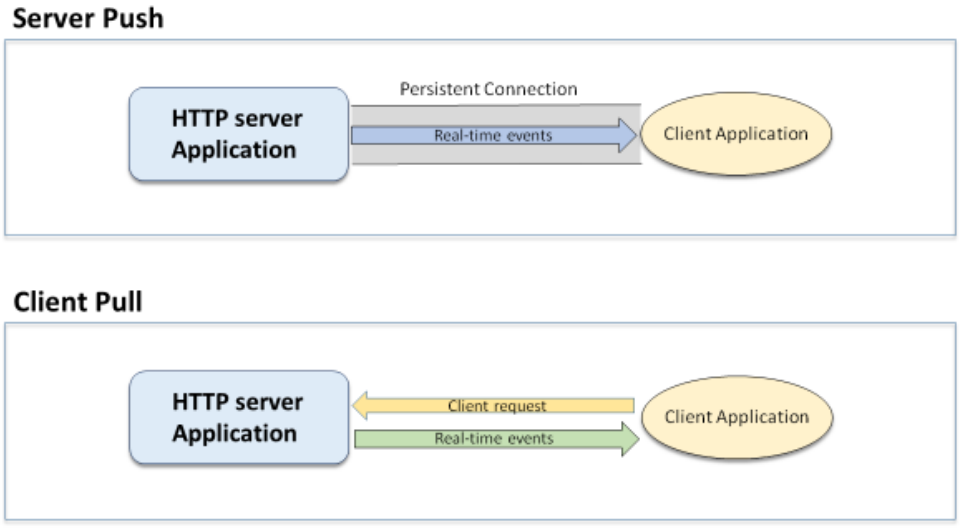

Figure 3. Push vs Pull Network Communications

connection between the client and server, or the data is pulled from the server by a client-initiated request. The push technique can also be simulated with multiple pull requests. Data is pulled down on a regular basis by generating repeated requests from the client application to the server. In this project, we will use the pull technique to stream data.

The data streamed from these sources is typically formatted in one of two ways, Extensible Markup Language (XML) or JavaScript Object Notation (JSON). Most streaming data websites today support both formats for data. While both formats will transfer the same data, each has pros and cons. Generally, XML is more easily read and understood by humans, while JSON is considered the "fat-free" format, requiring less overhead to transfer the same amount of data.

Figure 4. Extensible Markup Language (XML) vs JavaScript Object Notation (JSON)

Different tools are used to parse the XML and JSON formats, so typically an application is written to process one format or the other, not a mix of both formats.

When using the pull technology to initiate transfer of a data stream, an application is needed to format and submit a request for the data from the source website. This application can be written in a variety of programming languages depending on the type of application requesting the data and what the disposition of the data will be when it is received. In the case of a web application, JavaScript or the .Net framework can accomplish this task. If the data is being processed for storage or used outside of a website, then other options are available for requesting and processing the data. Command line interfaces like curl and programming languages like Python or Ruby can be used to issue requests for data, as well as receive and process the data stream. These tools can fully process the data stream, or serve as an intermediate step, passing the data stream along to other applications for further processing. The latter is the technique that will be described in this paper.

PROJECT DETAILS

As a member of the Customer Experience Testing team at SAS Institute, I am always looking for interesting projects that will allow me to use our software like our customers do to test the functionality and usability of the SAS software and documentation. Having an interest in sports, and being tasked with testing SAS Event Stream Processing software, I decided to attempt to find a source of real-time, live data from sports events and devise a way to use Event Stream Processing to process this data. SAS Event Stream Processing has a component which allows data to be streamed into a SAS® LASR™

Analytic Server, from where it can be consumed by SAS Visual Analytics applications.

STREAMING SPORTS DATA

The first challenge was to locate a source for the data. I wanted access to a variety of sports. The data needed to be very detailed so as to provide both large volumes for robust testing as well as interesting data for generating reports. To fully test the capabilities of SAS Event Stream Processing, the data needed to be available in real-time or near real-time, in a streaming format. Many websites are available that meet some of those requirements, but few that meet all three. After completing a search of the web I found a website that met these criteria. The website provides data on over 40 sports around the world at both the professional and collegiate level. The data is streamed live when the event is being played, and can be streamed from their data archives afterwards. This one solution allows access to both live and archived data. I decided to focus on data from the college level for NCAA basketball games.

SAS EVENT STREAM PROCESSING

The central component of the project is the SAS Event Stream Processing software. Event Stream Processing collects streaming data from source feeds, can perform real-time analysis on the data to identify important events, and send out notifications if the events require action. It also allows storage of the streaming data in a variety of formats, in raw or aggregated form.

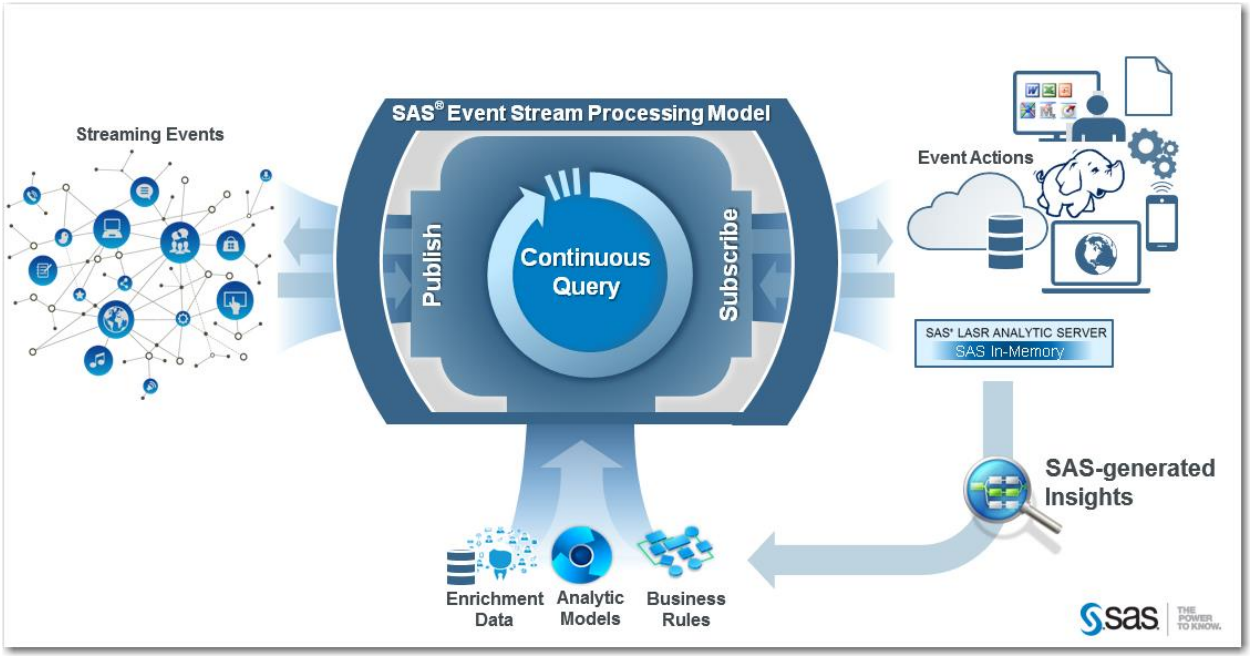

Figure 5. SAS Event Stream Processing

SAS Event Stream Processing projects run inside of a server engine. The engine manages interactions with the incoming events, and pushes results out to subscriber processes.

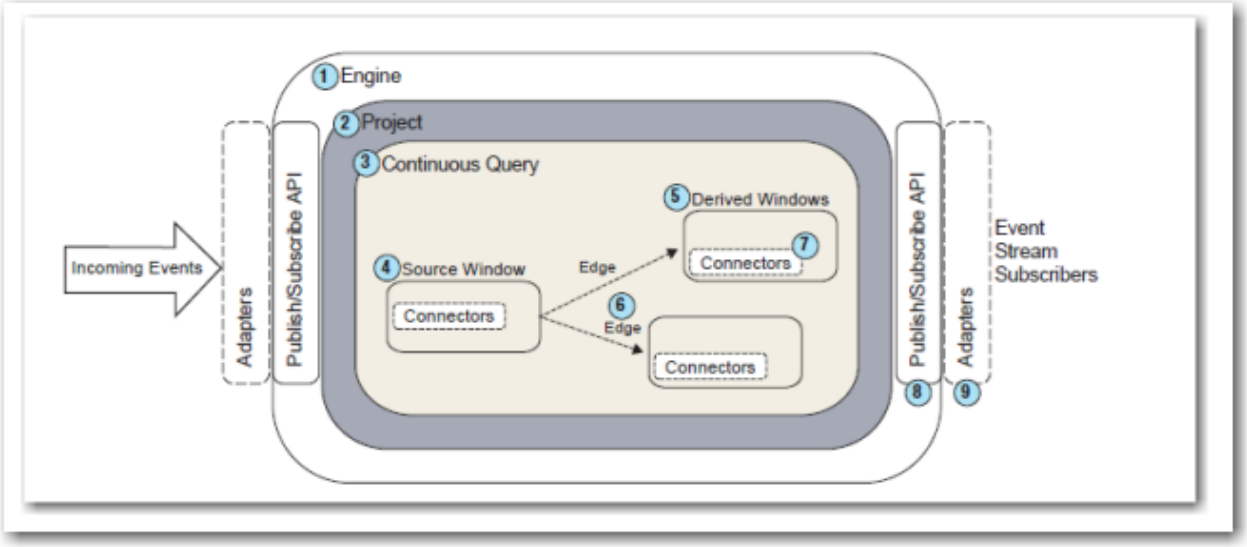

Figure 6. Event Stream Processing Components

Running inside the engine are the processes that receive, process, and transform the incoming events. These processes are called “Continuous Queries” and are organized into groups by a collection object called a “Project”. The incoming stream of events is injected into the continuous query using one of the SAS developed “adapters” or by a custom application which uses the SAS provided Publish/Subscribe application programing interface (API). Output from the continuous query can be passed to subscribing applications through the same set of adapters.

DATA FLOW

Applying this architecture to the specific problem we are tackling here, data is streamed from the website into a custom adapter written in Python. The adapter manages interactions between the website and the continuous query running in the Event Stream Processing engine. Additional windows in the continuous query will modify the data and send alerts on specific events as they occur. Then the data is passed to an Event Stream Processing adapter which loads the data into a SAS LASR Analytic server table. The LASR server is an in-memory data store that is read by SAS Visual Analytics to analyze and report on the data. In addition, another Event Stream Processing adapter archives the data to SAS datasets

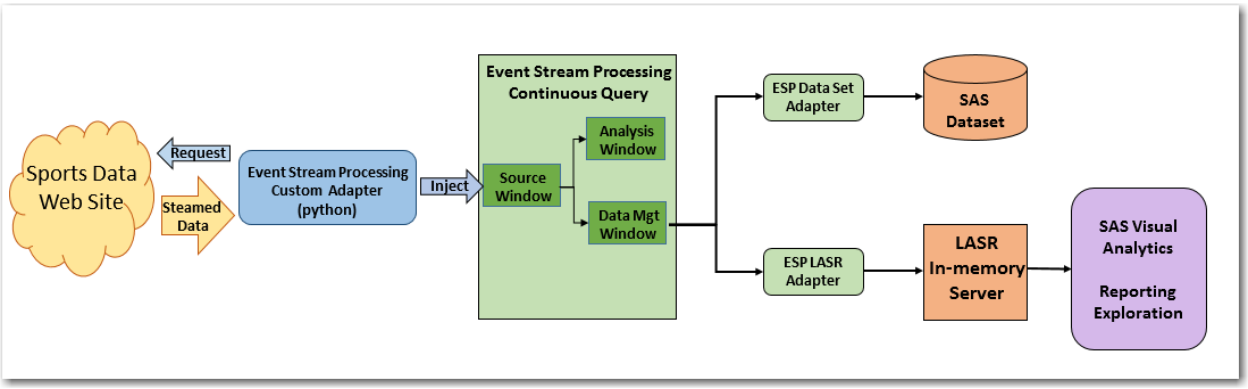

Figure 7. Project Data Flow

STREAM DATA FORMAT

To initiate the data stream from the website, an HTTP request must be sent to a website URL requesting the specific data that is desired. For example, to request play-by-play data for a specific NCAA basketball game, the HTTP request is formatted:

```
http://<sports website URL>/<API call>/<request>.json
```

The game ID and the desired format, along with the user's access key is sent to the website. Because the request specified JSON format, this request results are streamed back in that format. A specific organizational structure or schema is used for the play-by-play data stream. We can choose all or just parts of this data as we continue the downstream processing of the data.

HTTP REQUESTS TO WEBSITE API

To manage the requests for data, to perform error handling in the communications path, and to inject the data stream into the Event Stream Processing engine, I developed a custom Event Stream Processing adapter written in Python. The adapter issues a request for the data and receives the streamed events:

```
f = urllib.request.urlopen("http://<sports-website-api>/games/<game-
id>/pbp.json?api_key=<api_key>")
json_string = f.read().decode('utf8')
```

The program parses the data stream, identifying the information on the individual plays, and then formats the elements of the stream that are going to be used downstream. These elements are formatted into a message which is inserted into the Event Stream Processing engine:

```
pbpurl='http://<esp_server>:<port>/SASESP/windows/SportsRadar_NCAAM_Game_PBP/CQ_NCAAM/Source_PBP/
state?value=injected'
headers = {'Content-Type': 'application/xml'}

<… Logic to extract data elements from JSON string …>

     requests.put(pbpurl, data=str('''
        <events>
            <event>
                <opcode>p</opcode>
                    <id key='true'>''' + event_id               + '''</id>
                    <a_seq_num>'''     + str(seq_num)           + '''</a_seq_num>
                    <period_type>'''   + period_type            + '''</period_type>
                    <period_number>''' + str(period_number)     + '''</period_number>
                    <event_type>'''    + event_type             + '''</event_type>
                    <event_description>''' + event_description  + '''</event_description>
                    <event_x_coord>'''     + str(event_x_coord) + '''</event_x_coord>
                    <event_y_coord>'''     + str(event_y_coord) + '''</event_y_coord>
                    <stat_type>'''         + stat_type          + '''</stat_type>
                    <stat_team_market>'''  + stat_team_market   + '''</stat_team_market>
                    <stat_player_name>'''  + stat_player_name   + '''</stat_player_name>
                </event>
          </events>'''), headers=headers)
```

Wrapped around this message is additional code that will resend the request to the website on a periodic basis. While the game is in progress, this stream will be updated as new plays are completed to give up-to-date game status.

PROCESSING DATA STREAM USING SAS EVENT STREAM PROCESSING®

When the data is injected into the Event Stream Processing engine, it is processed by the continuous query designated in the adapter program. An element of the query called a “source” window receives the data stream. In the source window (Figure 8 - Source_PBP), a schema is defined which matches the format of the incoming transactions.

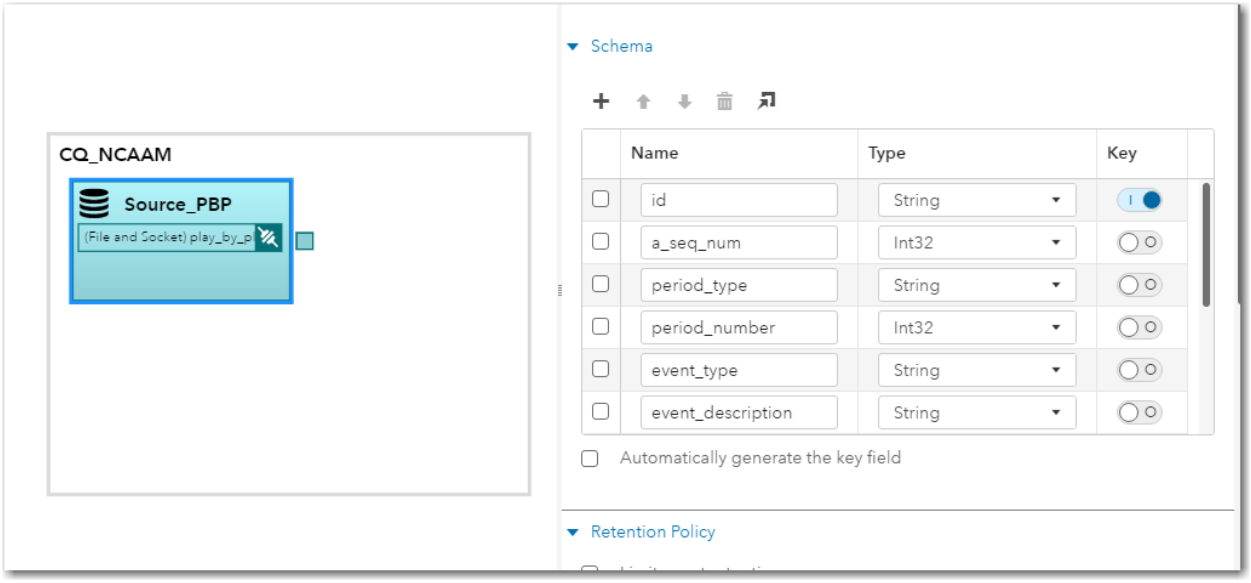

Figure 8. Source Window for data stream

From the source window, the data stream can be passed to a variety of windows to process the data stream. These windows can send email alerts, execute simple and complex statistics on the stream in real time, and perform data management functions including joins and filters.

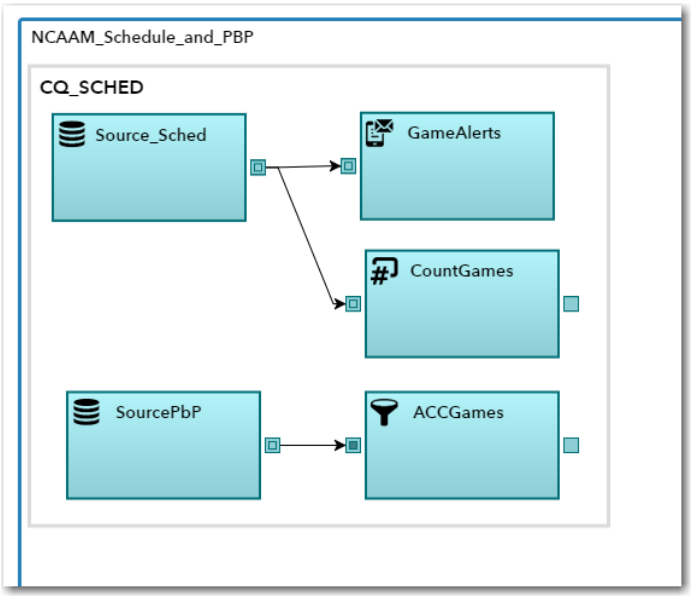

Figure 8. Continuous Query Processing Windows

The next step in the process is to pass the data stream out of the continuous query window to a data store. The data stores chosen for this project were SAS datasets and the SAS LASR Analytic Server. The LASR server is an in-memory data store which supports the SAS Visual Analytics suite of applications. Using the data we store in the LASR server, we can use Visual Analytics to create reports and explore the data. To load the data into the LASR server, we will use Event Stream Processing's LASR Analytic Server Adapter. It subscribes to the data stream from the continuous window, and pushes the data into the LASR server. It is invoked by a command line script which defines the source stream as well as the target LASR server:

```
$DFESP_HOME/bin/dfesp_lasr_adapter -k sub -h
"dfESP://<esp_server>:<port>/SportsRadar_NCAFF_PBP/CQ_PBP/Source_PBP/?snapshot=false"      -H
<LASR_Server>:<port> -t hps.streaming -X $DFESP_HOME/bin/tklasrkey.sh -n true -a 5 -A 5
```

As new data is streamed into SAS Event Stream Processing during a game in progress, the content of the LASR server will be updated with new plays from the game. This data will then be reflected in the reports and explorations of SAS Visual Analytics. To archive the data, we use the Event Stream Processing Data Set Adapter to write the data stream out to a SAS dataset. We can then use this data to re-load the LASR server without having to query the website again for the data.

REPORTING WITH SAS VISUAL ANALYTICS

SAS Visual Analytics is a reporting product built to allow reporting and analysis on large volumes of data very quickly. In this project, we want to be able to see the progress of a game reflected in the reports that we are viewing as the underlying LASR Analytic server is updated with new game data. Combining the real-time streaming with the reporting capabilities of Visual Analytics, it is possible to monitor the status of a game, as the game progresses.

game_date	Visitor Team	Visitor score	Home Team	Home score	game_id
2017-01-07	Kennesaw State Owls	79	Lipscomb Bisons	82	a333d65a-59cf-41fe-a0a8-efa...
2017-01-07	UC Riverside Highlanders	64	Long Beach State 49ers	70	c0dd6623-c290-4dd6-a85e-7...
2017-01-07	UTSA Roadrunners	69	Louisiana Tech Bulldogs	68	9a06766b-2d7c-46a2-b6d3-0...
2017-01-07	Louisiana-Monroe Warhawks	60	Louisiana-Lafayette Ragin'	69	8431175e-5df3-4efd-bb6d-20...
2017-01-07	Santa Clara Broncos	56	Loyola Marymount Lions	66	6dee1c84-bc4c-4bb9-9c26-4...
2017-01-07	Mississippi State Bulldogs	95	LSU Tigers	78	a00df515-e566-4743-92c7-a3...
2017-01-07	Charlotte 49ers	93	Marshall Thundering Herd	110	7f3c47c0-f60a-4540-af52-8f9...
2017-01-07	East Tennessee State Buccaneers	67	Mercer Bears	58	9ab318f1-1e5d-47f0-9262-c7f...
2017-01-07	Maryland Terrapins	77	Michigan Wolverines	70	5d43c6bd-fa43-44ae-ab8b-ae...
2017-01-07	Southern Illinois Salukis	75	Missouri State Bears	67	22dd3b50-73cb-4b04-bfa4-02...
2017-01-07	SIU-Edwardsville Cougars	65	Morehead State Eagles	73	97bff934-5ffc-4f28-a4eb-121...
2017-01-07	Nevada Wolf Pack	105	New Mexico Lobos	104	6d285c73-f067-4792-92de-5...
2017-01-07	Southern Utah Thunderbirds	65	North Dakota Fighting Hawks	95	6ff5ea3e-02e1-4669-8d88-97...
2017-01-07	Middle Tennessee Blue Raiders	79	North Texas Mean Green	68	e02a8501-bfa2-489e-b4df-08...
2017-01-07	James Madison Dukes	54	Northeastern Huskies	64	4ad7c734-025d-4edf-9281-00...
2017-01-07	Northern Arizona Lumberjacks	79	Northern Colorado Bears	83	14db8a16-e174-44e6-bfb1-84...
2017-01-07	Central Michigan Chippewas	83	Northern Illinois Huskies	87	0e992906-cd92-4d5e-9cc8-a...
2017-01-07	Cleveland State Vikings	75	Northern Kentucky Norse	83	28349cce-673b-4aa3-8c8d-ed...
2017-01-07	Clemson Tigers	70	Notre Dame Fighting Irish	75	c299129f-7f85-404c-9428-4e...
2017-01-07	Michigan State Spartans	63	Penn State Nittany Lions	72	6568f574-a9a2-4bf1-9865-2a...
2017-01-07	San Diego Toreros	76	Pepperdine Waves	68	a7e4334b-5735-4396-b16e-a...
2017-01-07	Sacramento State Hornets	76	Portland State Vikings	83	3a6e8aca-79ac-4c81-8553-59f...
2017-01-07	Radford Highlanders	76	Presbyterian Blue Hose	63	f1c0b214-c267-46c0-b6bb-fe...

Figure 9. Game Score Report

Figures 9 and 10 show two sample reports that are generated from the streamed data. The first figure

(Figure 9) shows the scores from games, either complete or in progress. Selecting one of these games links to another report (Figure 10) which shows the play-by-play events for the game up to the current point in the game, or for the full game if it has already been completed.

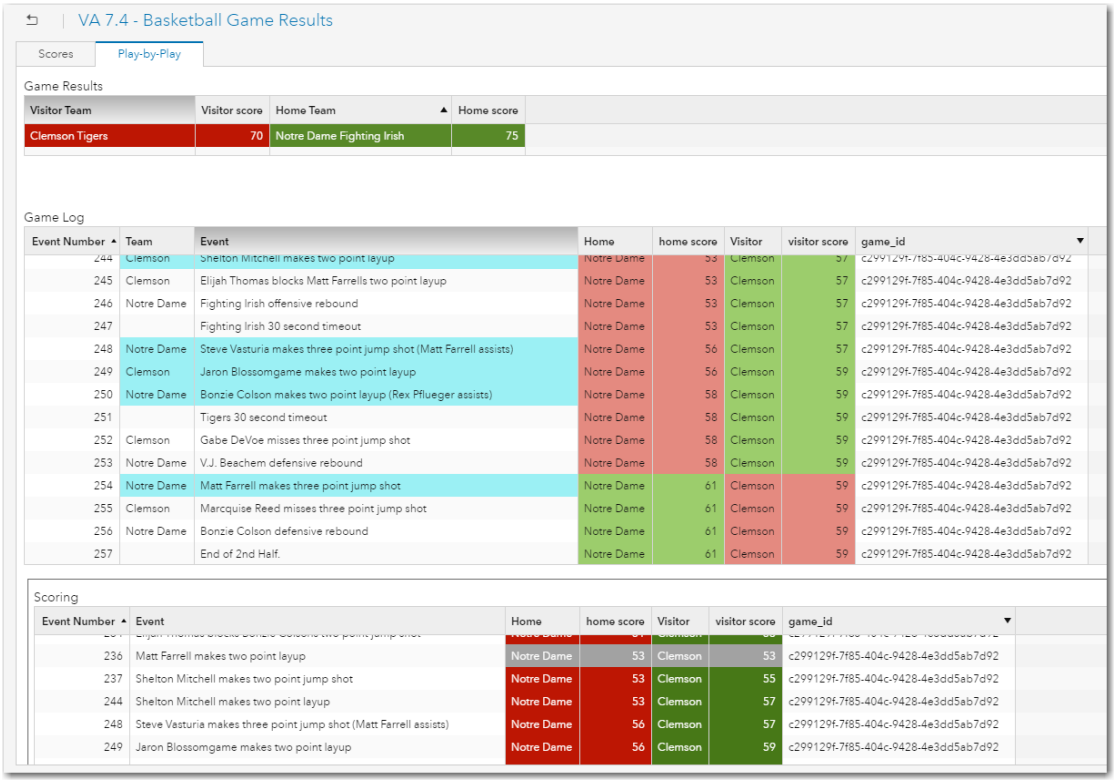

Figure 10. Play-by-Play Game Status

CONCLUSION

This project illustrates one way that SAS software can be used to capture and use data that is streamed from a real-time source. SAS Event Stream Processing is the tool that was designed for and is best suited for processing events in real-time. Event Stream Processing has many more features for the analysis and processing of this data than we have time to cover in this paper. If you are interested in more information on the capabilities and application of SAS Event Stream Processing, please attend other papers and demos on the subject during SAS Global Forum, and visit the Event Stream Processing areas of The Quad.

REFERENCES

Websites that stream sports data:

https://www.programmableweb.com/category/sports/apis?category=20016 a listing of possible streaming sports data websites. Some are not current, and some are more fitness oriented.

https://www.mysportsfeeds.com/ My Sports Feeds: Data for National Football League, Major League Baseball, National Basketball League, and National Hockey League

http://developer.sportradar.com/ SportRadar: A website with a paid-for service to stream live and archived data from 40 different sports. These are professional and collegiate levels and is US and World-wide sporting events.

CONTACT INFORMATION

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

John Davis
SAS Institute
john.davis@sas.com