*‘Agavue’ Event Data Sample*

*Danyel Fisher. August 10, 2016.  
Contact:* [*danyelf@microsoft.com*](mailto:danyelf@microsoft.com) *The original of this dataset can be found at* [*http://eventevent.github.io*](http://eventevent.github.io) *and* [*https://www.microsoft.com/en-us/research/project/logan-logfile-analysis/*](https://www.microsoft.com/en-us/research/project/logan-logfile-analysis/)

The attached files are a sample data release for AgaVue data. They are meant as a representative of real event log features –warts and all. We intend this set to be a useful standard set for users working on visualizations and models of logfiles.

These are sampled from a larger dataset; these elements represent only the files from the 8th day of each month. This dataset consists of 2,211 different sessions; 46,490 total events were logged.

They follow the EventFlow format, documented at <http://www.cs.umd.edu/hcil/eventflow/manual/chapter_start.html>

# Origin

The “Agavue” project released a series of applications on the web; users have been downloading and running those visualizations as part of an Excel or Access document since their release. The application feeds back updates, called with an HTTP GET request, to the server.

You can find documentation of the **user side** of these experiences at the project webpage:

<https://www.microsoft.com/en-us/research/project/microsoft-research-data-visualization-apps-for-office/>

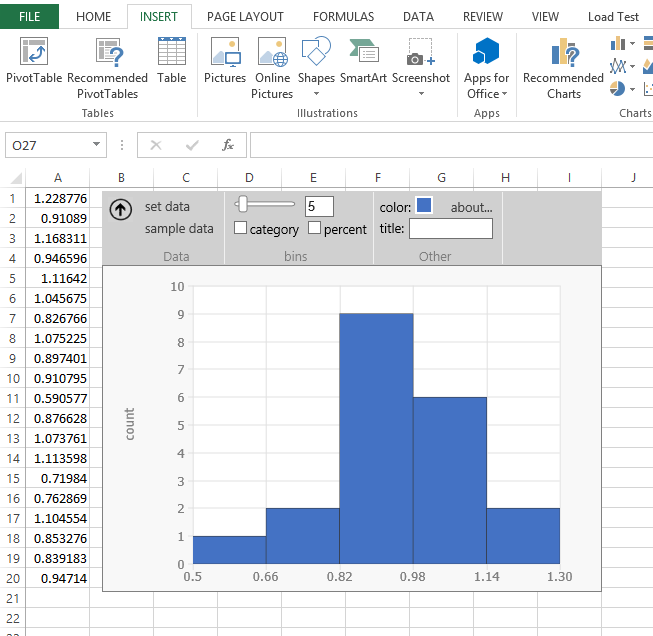
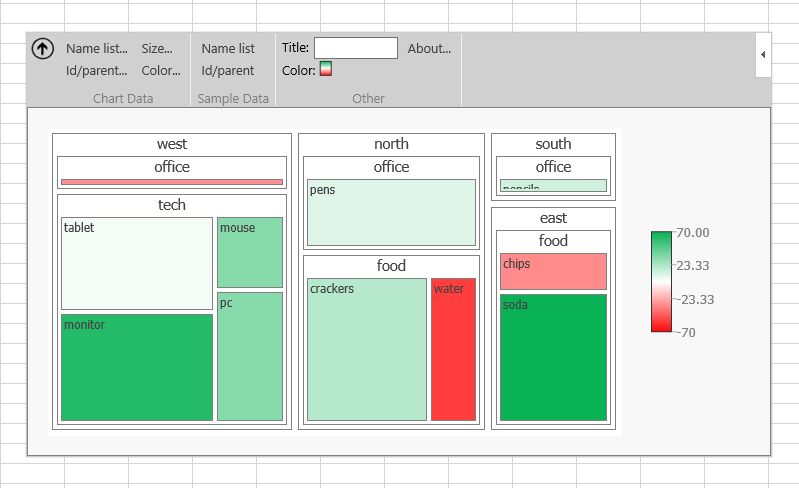
Since “GET” is being used as a logging mechanism:

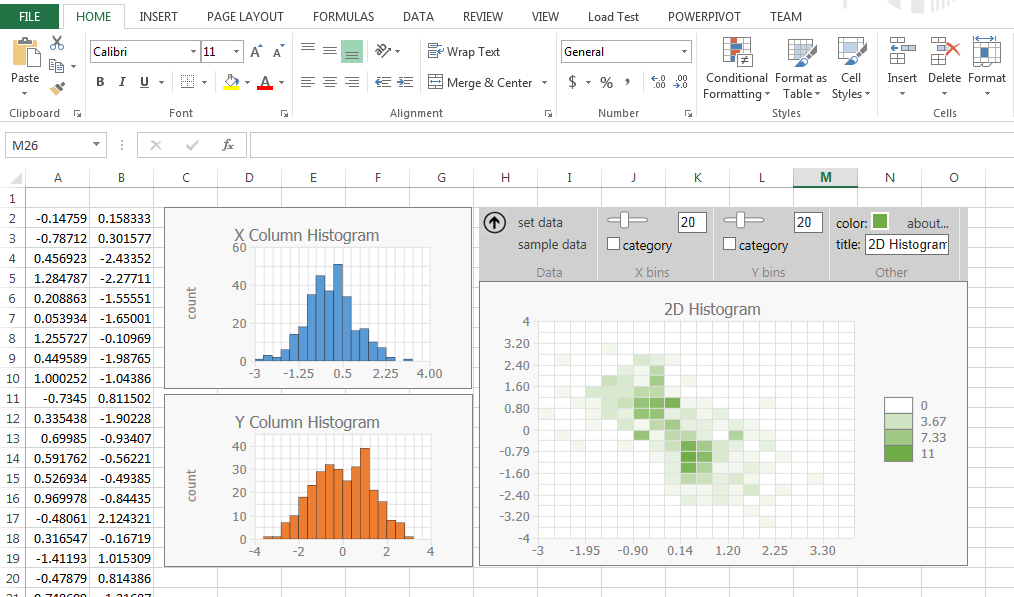
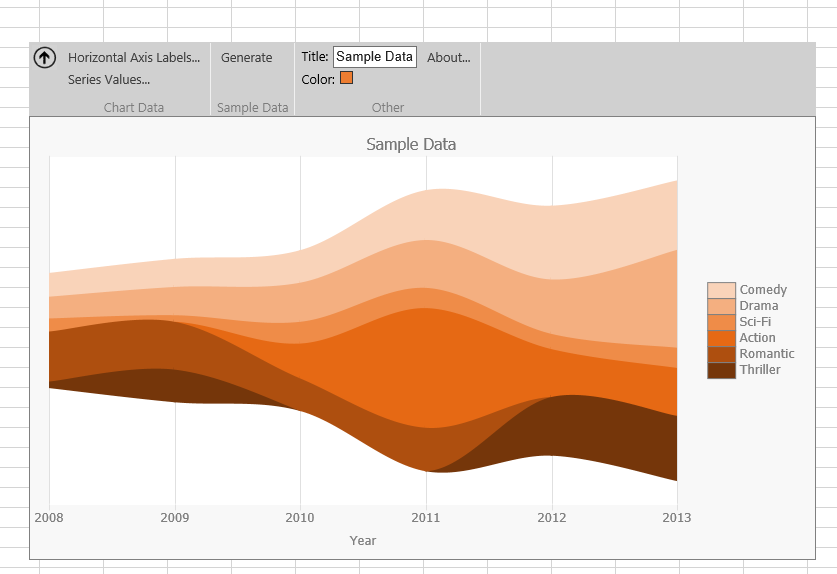
* Events are not precise to more than a second.
* Some events are known to be out of order if the packets got switched in traffic
* Some logs may be truncated, due to client or server issues.

In general, the logs are meant to preserve privacy while allowing us to track basic usage.

# The Apps

These apps were run as visualizations embedded in a sheet of Excel. A single sheet of Excel can contain more than one visualization.

# The State Machine

Roughly, a session looks like this:

A user selects data, which creates a **data binding** between that data and the visualization. The system visualizes their data (or throws an error message); the user manipulates the visualization.

## Initialization

|  |  |
| --- | --- |
| appInit( chartName, machineId, docId) | Initiates the applet with the given chartType on this unique machine and in this document. (Both are hashed). Note that many documents are named “untitled document” when created, so the docId – which is a hash of the title – might not be particularly helpful. |

## Data Binding & Create a Visual

|  |  |
| --- | --- |
| btSetData() | User presses “set data”; this triggers a bindFromPrompt |
| create( binding ) | Create a new visualization, with this binding. (xBinding = x Axis; yBinding = y Axis; etc) |
| load (binding) / setData ( binding ) | Notification that the sheet’s underlying data has changed on this binding. setData was what this was called in V1. Triggers a readBoundData |
| bindFromPrompt() | Data has been bound because the user has requested it (as opposed to loaded). This triggers a readBoundData if successful |
| readBoundData( colName, format?, rowCount, isCategorical?, min?, max?, seriesCount?) | Read in data from binding. Format is variable depending on which chart type this is. *Colname*: what data does this column correspond to?  *Format*: In a treemap, data can be entered as either nameList or idParent *rowCount*: How many rows of data? *Min*: smallest value in the dataset *Max*: the largest value in the dataset  *seriesCount:* for a streamgraph, how many series? |
| treeStats(rootCount, leaveCount, maxDepth, nameColCount, namesWithBlanks) | Statistics about the TreeMap that was just requested, executed after a readBoundData |
|  |  |
|  |  |

## Adjust the Visual

|  |  |
| --- | --- |
| btColor(isShowingPalette) / pickedColor( color ) / closeColorPicker() | Open color palette; set the color for the visualization; close the color picker |
|  |  |
| cbCategory(isChecked) | User presses the checkbox “Categorical” to toggle a histogram from categorical to numerical |
| cbPercent(isChecked) | User presses the checkbox “Percentage” to toggle a histogram from showing counts to showing a percentage |
| hasColumnHeadersToggle (value) | Does the data have column headers? |
| resize (width, height) | Automatically triggered at start, or user has manually resized the space |
| setStackType ( value ) | For lineChart, decides on a drawing type |
| setTitle (length ) | Sets the title of the chart. Length is the length of the name in characters |
| Slider( value ); xSlider (value); ySlider (value) | Fired when the user changes the number of buckets on the histogram (or the X and Y sliders on a 2D histogram) |
| tbBins (value); xTbBins (value); yTbBins (value) | After the slider has stopped, this event shows the number of bins on a histogram (or the X and Y bin counts on a 2D histogram) |
| toolTip (id) | The user hovers over the control *id*, and a tooltip appears |

## UI

*These are functions that describe interactions with the system*

|  |  |
| --- | --- |
| error(msg) / closeErrorBox() | Show an error message; close the modal box |
| warning (message) | Shows a warning message |
| aboutBox() / closeAboutBox() | **Shows an informational message; close the modal box** |
| writeSampleData (); writeSampleData2 () | Writes a set of sample data to the sheet. Histogram and 2D Histogram toggle through several different sets of data. Treemap has sample data in Name-Value form (“writeSampleData”) and ID-Parent form (“writeSampleData2”) |
| toolBarToggle (value) | Shows or hides the toolbar |

# Let’s Read A Session

**Events Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Guid1004 | appInit | 7/8/2015 17:56:48 |  | chartName=treemap; machineId=machineId\_1436375531310\_0.04533913009629037; docId=docId\_kDFiHl37xmhx1WzAwnLn4KckxzpsVtiWhkvHuDctXjc^ |
| Guid1004 | create | 7/8/2015 17:56:48 |  | bindingName=colorBindingId |
| Guid1004 | create | 7/8/2015 17:56:48 |  | bindingName=primaryBinding |
| Guid1004 | create | 7/8/2015 17:56:48 |  | bindingName=sizeBindingId |
| Guid1004 | resize | 7/8/2015 17:56:48 |  | width=800; height=600 |
| Guid1004 | resize | 7/8/2015 17:56:48 |  | width=800; height=600 |
| Guid1004 | treeStats | 7/8/2015 17:56:48 |  | rootCount=0; leaveCount=0; maxDepth=0; nameColCount=0; namesWithBlanks=0 |
| Guid1004 | bindFromPrompt | 7/8/2015 17:56:56 |  |  |
| Guid1004 | readBoundData | 7/8/2015 17:56:56 |  | colName=nodes; format=nameList; rowCount=443 |
| Guid1004 | treeStats | 7/8/2015 17:56:58 |  | rootCount=8; leaveCount=443; maxDepth=2; nameColCount=2; namesWithBlanks=0 |
| Guid1004 | readBoundData | 7/8/2015 17:57:13 |  | colName=size; rowCount=443; isCategorical=false; min=1; max=1242 |
| Guid1004 | readBoundData | 7/8/2015 17:57:27 |  | colName=color; rowCount=443; isCategorical=false; min=0; max=0 |
| Guid1004 | warning | 7/8/2015 17:57:27 |  | msg=Warning:%201%20or%20more%20non-numeric%20values%20were%20converted%20to%200. |
| Guid1004 | closeErrorBox | 7/8/2015 17:57:32 |  |  |
| Guid1004 | btColor | 7/8/2015 17:57:34 |  | isShowingPalette=false |
| Guid1004 | pickedColor | 7/8/2015 17:57:34 |  | color=rgb(237,%20125,%2049) |
| Guid1004 | pickedColor | 7/8/2015 17:57:40 |  | color=rgb(165,%20165,%20165) |
| Guid1004 | pickedColor | 7/8/2015 17:57:42 |  | color=rgb(68,%20114,%20196) |
| Guid1004 | toolTip | 7/8/2015 17:57:44 |  | id=toolBar |
| Guid1004 | toolTip | 7/8/2015 17:57:49 |  | id=btColorCol |

**Attributes Table**

|  |  |  |
| --- | --- | --- |
| Guid1004 | anonIP | ip539 |
| Guid1004 | ChartId | chartId\_1436378208885\_0.8642284872660698 |
| Guid1004 | vers | v4 |

*A user reloads a chart – we know it’s a reload because there’s an existing series of* create *operations. The system gets started up. The existing chart has no data, though: the treestats says that it’s got 0 roots.*

*The user clicks “read data” and binds 443 rows to the nodes column. This makes a tree with 8 roots, 2 levels, and 443 leaves. The user binds in a Size column, with sizes ranging from 1 to 1242; and then a color column. The color column was probably text; they get an error message.*

*They play with color schemes, and then quit. The full session ran two minutes.*

# Some Additional Notes

1. The current dataset takes a combination of an IP address, the version of the software running, and the ChartId *together* to be a unique ID. **ChartId** is a **globally-unique** ID; therefore, the same chart might have occurred across multiple sessions. You can find the ChartId in the Attributes table.
2. The IP address is anonymized; they appear in time order. If two records have the same ip address, their client reported the same IP address – however, that might mean a VPN or a Comcast aggregator.
3. Version=v1 of the data named all parameters P1, P2, and so on. It was extremely annoying to parse; as such, parameters are more correctly named as of V2.

# Some Sample Analysis Tasks

**Basic Usage**

How many different visualizations have been created? Of what sorts?

Is there a change in interest over time?

**Sample Data**

How many of our instances started with inserting sample data?

How often did people then modify it, and still have a successful visualization?

Did users who had used sample data have a better success rate at making the visualization work?

**Creation**

How many attempts did it take to get a visualization rather than an error message?

How many people gave up?

**Reuse**

How many instances were reloaded?

How many instances were opened in a different document ID than the original? Different system ID than the original?

**Resize**

Can we come to some ideas about what the *right* size of a visual is based on the visualization, amount of data, and so on?

**Errors**

For each different error, what happens immediately before that?

What visualization type drives each error?

What *data shape* goes with each error?

**Data Shape**

How much non-sample data are people looking at?