**The value of easy data conversion**

The work that I have been doing lately has to do with visualizing data. Visualizing data is a very useful thing to do as it allows us to look at a large amount of data in a small amount of time, and assists us in seeing patterns in those large amounts of data. For example, if I showed you a big long list of numbers asked you what the mean, median and mode was, it would take you a while to figure that out. But if you had a tool that could graph these numbers, you could quite easily and very quickly give me a rough estimate. The problem with this is even if you had some sort of graphing tool, it probably requires a very specific format for the data it takes in, the data I give you might not be of that format. You could do manual data entry to plug the data in, or you could write code to convert it. If you choose to write this code, you end up with very specialized code that can only do this one thing. And if you decided you wanted change what data got plugged in where, it would be a rather painful process of going into your code altering it to meet the new requirements. It would be very helpful if you had some sort of system for easily converting the data you had, into something your visualization tool could use. This method should be simple and easy to understand, but also be powerful enough to support most kinds of data. The system should also be easily maintainable.

**The system**

I have begun developing the system described above. The system I have made is used for converting JSON files into a new format of JSON file usable by a data visualization tool. This is done through creating another JSON file that is called the “Visualization Strategy”. This JSON describes how the data is going to be changed to meet the requirements. The whole system is written in javascript and only works with JSON files and is meant for web use, but the concept could potentially be extended to other things with other file formats. Below is an explanation on how to use the system.

**The examples**

For the examples I am using for the rest of this paper, I will be using a dataset that represents communication within a team of six people. The data contains elements that represent communication between two people. Each element contains who the communication is from and who it is two. They contain the type of communication, which can be an email, a work item or a source code change. It also contains the number of communications and a timestamp.

Throughout this paper I will create three different ways of visualizing this one dataset. The three visualization tools are tools that I have found online, I did not create them.

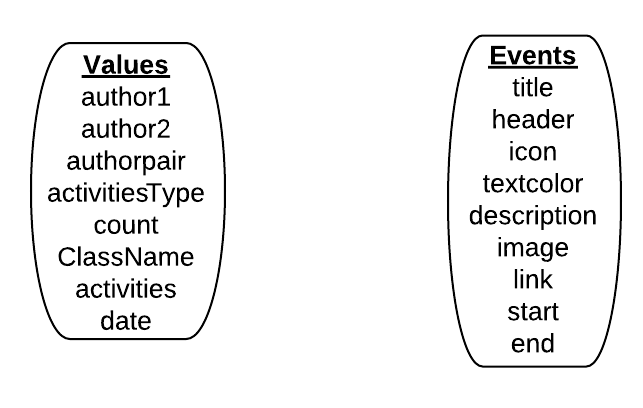
The first visualization is a timeline that requires events. Each event contains a title (with optional text color) and an icon to be displayed on the timeline itself with a start date and an end date (no end date means it is an instant event). Each event also has a header and a description that will be displayed in an info window when the event is clicked on. Each event also contains an optional link and image, for use in the info window.

The second visualization is what is called a chord diagram, which displays coloured slices in a circle that are connect by chords of varying sizes that go through the middle of the circle. Each slice contains a title and a colour. Each chord contains a source and a target with a width.

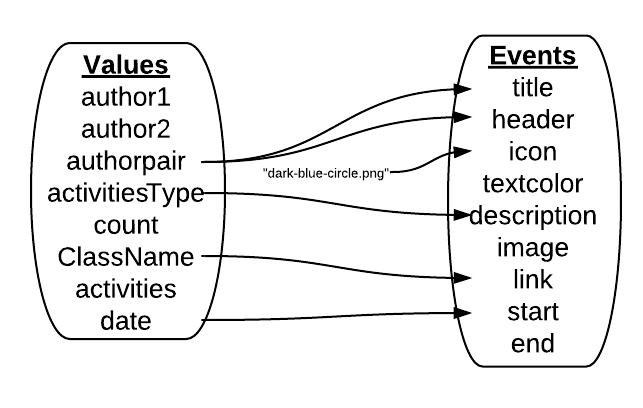
The third visualization is what is called a force directed graph, which lays out a graph based on attraction of connected nodes and repulsion of non-connected nodes. Each node contains a title, colour, shape, and size. Each link contains a source, target, distance, width, and strength.

**The concept**

The basic concept is very simple. The first thing you do is draw a diagram that will represent what is called your “Visualization Strategy”. To do this, first list all of the fields of your data on the left, and list all the required fields of your data on the right. Below is an example of the first visualization spoken of above.



Then you simply draw arrows from the left to the right, mapping fields from the dataset to fields in what will become the visualization input. As an example, let’s say I want a timeline where each event has a title of displaying the “authorpair”. This will tell us who was communicating with who for each event. We would like to see the type of activity somewhere, so let’s put it in the description so we can see it when we click on an element. Also, we haven’t mentioned “icon” yet, eventually I’d like icon to represent something, but for now let’s just make all the icons the same default blue dot. This is stored in a file called “dark-blue-circle.png”



**Creating a usable Visualization Strategy**

We now have the start of our Visualization Strategy. It’s missing a few things, but let’s try to plug it into the system anyways. This means we have to construct a JSON file based on the diagram we made. The JSON looks like this:

{

“from”:”values”,

“to”:”events”,

“iterate”:[

{

“from”:”authorpair”,

“to”:”title”

},

{

“from”:”authorpair”,

“to”:”header”

},

{

“value”:”dark-blue-circle.png”,

“to”:”icon”

},

{

“from”:”activitiesType”,

“to”:”description”

},

{

“from”:”ClassName”,

“to”:”link”

},

{

“from”:”date”,

“to”:”start”

}

]

}

As you can see, the strategy uses a simple “from” and “to” structure describing what we’re mapping from and where it is going. The first part says that we’re mapping from the list of values and that we are going to construct a new list of events. The “iterate” says, for every single element in “values” do these mappings. As you may have noticed, each of those objects within the “iterate” object represents one of those arrows we drew on our diagram. The third one does not have a “from” field, this is because the data is not actually being mapped from anywhere; it’s coming from the value specified.

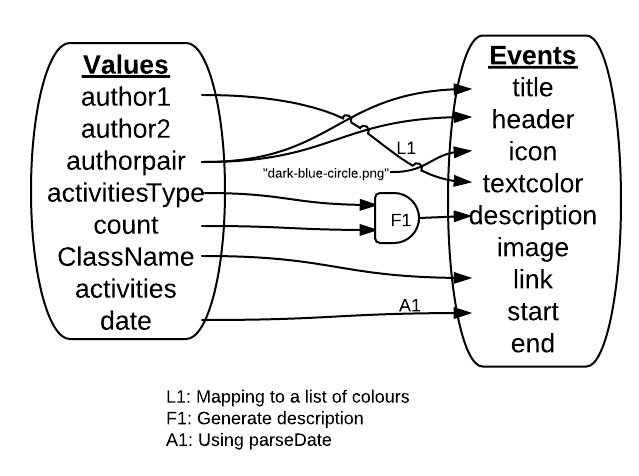
**Functions, Appends, and Lists**

So the strategy that we have is pretty good, but it’s missing a few minor things, the first is that I want to display the count in the description somewhere, but I also want to display the activitiesType. This means I am going to have to write a function that uses both the count and the activitiesType to generate some sort of description. This function can be represented in the diagram with a small half circle. In the actual code for the visualization strategy, you would replace your “from” variable with your function.

Another thing I noticed is that my date is in the wrong format, so I need to convert it to the correct format. To do this I need to use what is called and Append Function. This is a function that runs on the resulting value after the mapping is done, it then places the returned value from the function into the targeted field. To indicate that I am using an append function on the diagram, I will mark the line with an uppercase A. To do this in the code of the visualization strategy, just add an “appendFunction” variable to your mapping, with the value as a function.

Another thing that I’d like to have, is I would like to have the textcolor represent the author1 (the sender of the message). This feature will make it easy to look at the timeline and quickly gauge who was sending messages and when simply by looking at the colours. To do this I am going to start by mapping author1 to textcolor just like normal, but instead of doing a straight mapping, I am going to use a list. To indicate that I am doing this on the diagram, I’ll mark the arrow with an uppercase L. How this works, is in the strategy I am going to set a variable called “toList” to an array of values, then whenever the data conversion system is doing that mapping, instead of directly mapping it, it will pick a value from the list. If it ever sees the same value again, it will make sure to map it to the same value from the list, and it will always make sure to never map two different values to the same value in the list. So if I pass it a list colours, it should take each name that it sees and map it to colour and then put this colour in the textcolour field.

Below is an example of what the mappings discussed above would look like on our example.



Note: You can mark your functions, appends and lists with a number then refer to them at the bottom of your diagram to give more info.

Note: letters that belong with lines should always appear above the line to eliminate any confusion as to which line it belongs to.

Below is that the actual strategy looks like.

{

“from”:”values”,

“to”:”events”,

“iterate”:[

{

“from”:”author1”,

“to”:”textcolor”,

“toList”:[“#ff0000”,”#00ff00”….. etc],

“tag”:”textcolor”

},

{

“from”:”authorpair”,

“to”:”title”

},

{

“from”:”authorpair”,

“to”:”header”

},

{

“value”:”dark-blue-circle.png”,

“to”:”icon”

},

{

“function”:function(object) {

var activitiesType = object.activitiesType;

var count = object.count;

\*\*\*code here\*\*\*

return description;

}

“to”:”description”

},

{

“from”:”ClassName”,

“to”:”link”

},

{

“from”:”date”,

“to”:”start”,

“appendFunction”:function(date) {

\*\*\*code that parses date\*\*\*

return newDate;

}

}

]

}

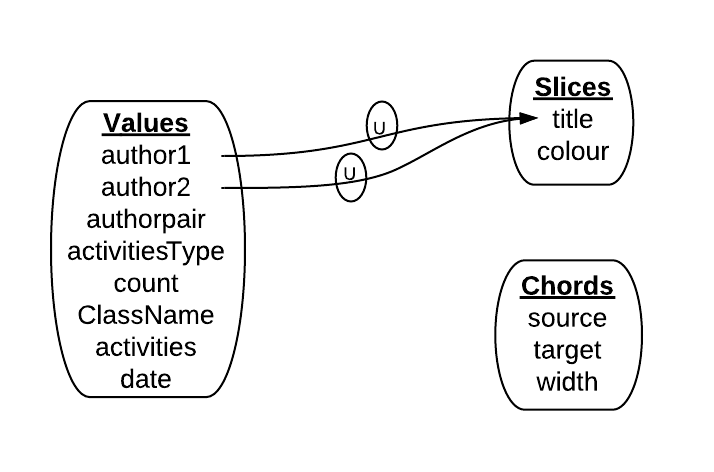
Note: Another strategy for mapping authors to colours would be to have some sort of hash function turn the author into a number then use the number to get a colour. This could easily be done using another append function.

**Unique Values**

So our timeline looks good. Now we want to try something a little different. Instead of looking at all the communications as individual events, we want to aggregate them and look at a visualization based on numbers. In other words, we want a visualization that can tell, at a glance, us who is communicating with who and how much. To do this we are going to use a chord diagram that consists of slices laid out in a circle with different sized chords between them. This visualization requires slices and chords, which means we are going to have to do multiple iterations through our data.

The first thing we need to do is create our slices. We need one slice for every person, that means we need to iterate through our data and every time we see a person we have never seen before, we have to create a new slice. This can be done using a “unique” object. To use this, simply draw your mapping like you normally would, but mark the line with a U.

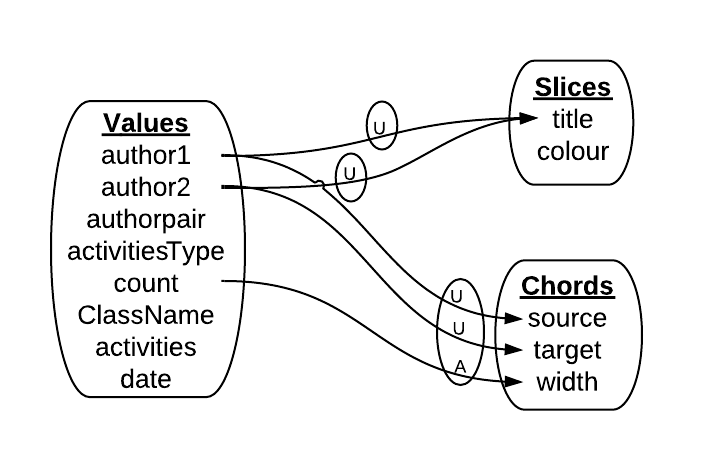
For this particular example, I want to make sure I look at both author1 and author2, meaning I’ll have to do two separate iterations.



The two separate iterations are indicated by the two separate circles. Any lines that go through the same circle are assumed to a part of the same iteration.

What this means is, go through the data, and with every element, check if author1 already exists as a title in slices, if it doesn’t create a new one and map author1 to title. Then go through the data again and do the same with author2. This will create a bunch of slices with unique titles.

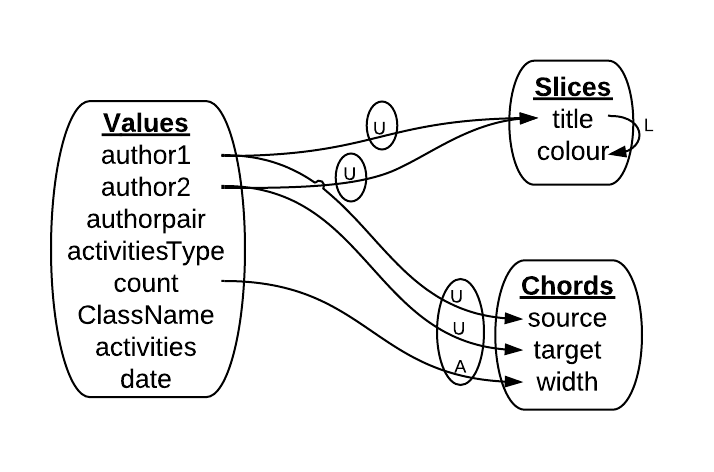
Now that we have our slices, we need to create our chords. For each chord, the source represents the sender (ie author1) and the target represents the receiver (ie author2). But we want to only create one chord for each combination of author1 and author2. So what we would like is to iterate through our data, and look at author1 and author2, if it is a pairing we have not seen before, we want to create a new chord and map count to width, but if we have seen it before, we want to add the count to the width that is already there. To do this, we can use the unique object again.



As you can see, this looks similar to above, only the two U’s are in the same circle. This means they are in the same iteration, meaning the result will be a bunch of chords that each have a unique combination of source and target.

Also notice that the arrow going to width has a familiar A on it. This means it uses an appendFunction, only this time the appendFunction gets two parameters, the first is the new value coming in from “count”, and the second is whatever value already exists at the given spot we are mapping to. We can use this function to add it to the already existing value of width.

This visualization strategy is looking good, but we’re missing one thing: colour. Each slice needs a colour, and we want every colour to represent a different person. To do this we are going to do what is called a “remap”. This is where instead of mapping from the left to the right, we’re going to make the right map to itself. We are going to use the same colour list as before and map “title” to “colour” using a toList.



Below is what the code looks like for this visualization strategy.

{

"from":"",

"to":"",

"map":[

{

"from":"values",

"to":"slices",

"unique":[

{

"from":"author1",

"to":"title"

}

]

},

{

"from":"values",

"to":"slices",

"unique":[

{

"from":"author2",

"to":"title"

}

]

},

{

"remap":1,

"from":"slices",

"to":"slices",

"iterate":[

{

"from":"title",

"to":"colour",

"toList":["#ff0000", "#00ff00", ...etc],

“tag”:”sliceColours”

}

]

},

{

"from":"values",

"to":"chords",

"unique":[

{

"from":"author1",

"to":"source"

},

{

"from":"author2",

"to":"target"

}

],

"iterate":[

{

"from":"count",

"to":"width",

"appendFunction":function(newValue, oldValue) {

if(isNaN(oldValue)) return newValue;

else return newValue + oldValue;

}

}

]

}

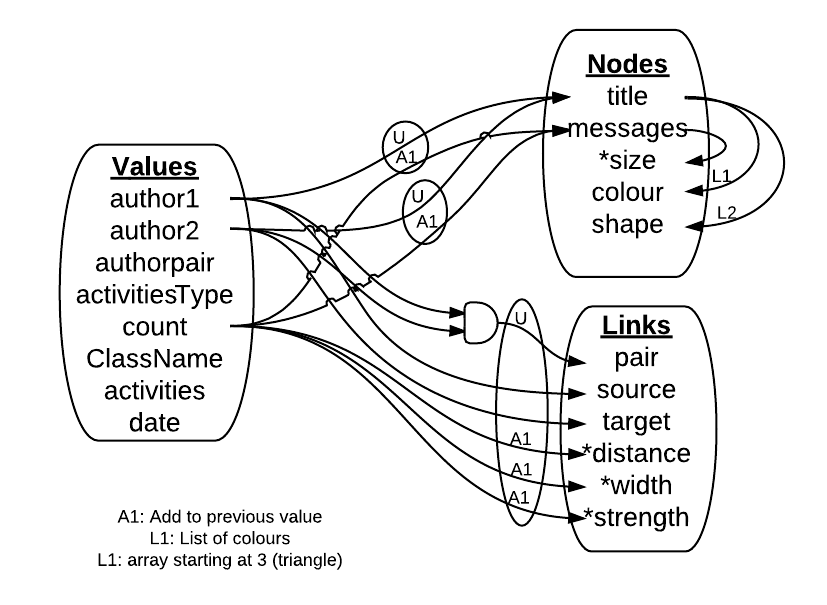
]

}

The map object, starting at the fourth line, is simply an array of mappings. This object is simply used to perform multiple different mappings.

**Formatting Numbers**

Now we want to take this same data and display it as a “force directed graph”. Below is a diagram of the visualization strategy needed to do this.



This diagram says that the number of total number of messages a person sends and receives gets mapped to the node size, and the number of messages between two people gets mapped to distance, width and strength. However, if we just leave those numbers as is, this visualization won’t make any sense… Because if we end up with, for example, 800 messages sent and received by a given person, if we put that number into “size” the node would be larger than your screen. This means we need to format these numbers to make sense in the context of what they are getting mapped to.

The values that require formatting are marked by a \* in the diagram. This indicates that after the iteration is finished, the numbers will be formatted.

The method I use for formatting is one of altering the mean and standard deviation of the data set to fit a spec given. Below is an example of how one would do this.

{

“remap”:1,

“from”:”messages”,

“to”:”size”,

“format”:1,

“mean”:8,

“deviation”:3,

“min”:0.1

}

“format” is a Boolean that says that this field should be formatted.

This says that the numbers should be altered such that the new data set has a mean of 8 and a standard deviation of 3, with a minimum of 0.1. This will change the numbers so that the numbers make sense in context to the visualization. If you make a visualization strategy but there doesn’t seem to be much difference in certain values, ie all the nodes are about the same size, you can try increasing the “deviation” value to increase the difference in sizes.

Below are examples of the three mappings in links that require formatting.

var addToOldValue = function(newValue, oldValue) {…code here…}

…

{

“from”:”count”,

“to”:”distance”,

“appendFunction”:addToOldValue,

“format”:1,

“inverted”:1,

“mean”:100,

“deviation”:15,

“min”:0

},

{

“from”:”count”,

“to”:”width”,

“appendFunction”:addToOldValue,

“format”:1,

“mean”:15,

“deviation”:8,

“min”:1

},

{

“from”:”count”,

“to”:”strength”,

“appendFunction”:addToOldValue,

“format”:1,

“mean”:4,

“deviation”:0.4,

“min”:0

}

The “inverted” Boolean in the map to “distance” says that for this particular mapping, invert the numbers to make it so the higher the number, the lower the resulting value. In the force directed graph, this achieves the result of nodes that are more “related” ending up closer to each other (ie the more messages sent between the two people, the lower the distance value becomes)

**Adding Filters**

So now we’ve got decent mappings for all of our data, but we would like to filter it on the fly. To do this, simply give any mapping a “filter” attribute with a tag as the value. Example:

{

"from":"author",

"to":"author",

"filter":"author"

}

Anything with the filter keyword “author” will be added to the author filter. This is a list of values that appears at the top of the visualization:



How this is works, is during the first creation of the visualizations, “all” is selected by default, meaning it allows all values. And it keeps track of all the values it sees during the mapping and creates a list from it. Anything with the “author” filter keyword will be added to this list.

If something is selected (or multiple things, as this is a multiselect list) then whenever the mapping comes across something with the filter keyword, if the value being mapped does not match one of the selected items, the element containing this unwanted value is deleted.

This filter can marked on the visualization diagram with an uppercase F. Like the other markings, you may also include an ID with this F to indicate which filter it is.

**Adding Legends**

Adding dynamically generated legends to your visualizations is very similar to adding filters. You simply add a “legend” attribute with a tag for the legend. Example:

{

"from":"author",

"to":"textColor",

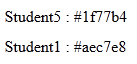
"toList":["#ff0000", "#00ff00", ...etc],

“tag”:”textcolor”

"legend":"Colours"

}

So here we’re mapping to a list of colours and we want a legend that shows the author name on the left and the colour on the right. So we just add that legend attribute there and that does exactly that. The only problem is, the system doesn’t know how to display the data unless you tell it, by default, it just displays a string, so the above code would create a legend that looks like this:



To do this, simply add “fromType” and “toType” attributes.

{

"from":"author",

"to":"textColor",

"toList":["#ff0000", "#00ff00", ...etc],

“tag”:”textcolor”

"legend":"Colours",

“fromType”:”string”,

“toType”:”colour”

}

By default, it displays these as strings anyways, so the fromType here is actually unnecessary, but the toType tells the system to display the right side as a colour rather than a string. This will make the legend look like this:



The current accepted types are : string, colour (or color), shape, and image.

Shape takes a number representing the number of sides of the shape you wish to display and it will display a regular n-gon with that many sides. That looks like this:



And image simply displays the image at the given file path.

Other legend types will be created as they are needed.

**Other Examples**

Below are a few examples that I have created with their visualization strategy and the visualization.