Flaremap® Development Kit
Technical Reference

Visual Action Software, Inc.

Version 7.2

Introduction

The Flaremap Development Kit Technical Reference is a reference guide for the technical configuration and use of Flaremap JS. Flaremap JS is pure JavaScript and compatible with any web serving platform. This document will introduce the major parts of the Flaremap JS by walking through the salesforce demo that is included with the Flaremap JS software download.

We'll present these parts in the following order:

Part 1: The File Structure

Part 2: The Core HTML

Part 3: The Configuration Parameters

Part 4: The Data

Part 5: The Mouseover Box

Part 6: Configuration Parameters in detail

Part 7: JavaScript Interface

Part 8: URL features – Live URL, URL Pre-filter, and URL filter

Once you are familiar with these parts, you should be able build your own treemaps with the aid of this documentation and the sample salesforce demo.

Nomenclature and characters used in the Guide: The use of curly braces {} denotes items that are to be replaced (including the curly braces). Replace the {item} with the appropriate value. The use of square brackets [] indicates optional items, those that are not required for the parameter value to work correctly. Any words not written in curly braces or square brackets should be used literally in the parameters.

Part 1: The File Structure

In its simplest form, a Flaremap JS application contains four unique files that reference a set of application support files. For the salesforce demo, the unique files are:

- salesforce.html: The HTML page serves up the salesforce treemap by bringing all the Flaremap JS elements together. salesforce.html also contains the cell and group mouseover box details.
- salesforceConfig.js: Contains the Flaremap configuration parameters and initialization code.
- salesforceData.js: Contains the data used by Flaremap JS.
- salesforce.css: Contains additional CSS formatting that can be used by the treemap, usually by the mouseover boxes.

The application support files are located in the "css" and "js" directories. These directories/files are referenced by each Flaremap JS application and are generally not modified.

Part 2: The Core HTML

Each Flaremap JS application has its own HTML file, the "core HTML". The core HTML contains several parts – some which are unique to each application and some that are the same for each application. The core HTML:

- Contains the HTML structure for the application layout. (Same for each application.)
- References all the application support files. (Same for each application.)
- References the Flaremap configuration file – See Part 3. (Unique to each application.)
- References the Flaremap style sheet (CSS). (Unique to each application.)
- Contains the cell mouseover box details – See Part 5. (Unique to each application.)
- Contains the group mouseover box details – See Part 5. (Unique to each application.)

For an example Flaremap JS core HTML file, see the file "salesforce.html".

Modifying the HTML structure is not recommended. However, in versions 6.2+, a blank DIV has been added to the HTML to allow for a custom heading to be added. Also in versions 6.2+, an animated "Loading..." screen has been added. See below for details.

How to add a custom heading to your application

Available in versions 6.2+: A blank DIV has been added to the HTML to allow for a custom heading. Anything entered inside this DIV will appear above the application – on all devices. The DIV has the ID #flaremapHeadingContainer and should be placed as the first DIV inside the #pageContainer DIV. See the snippet below as an example.

```
<body>
<div id="pageContainer">
      <div id="flaremapHeadingContainer">
            Insert custom heading here
      </div>
      <div id="gscContainer" style="display:none;">
      ...
      </div>
...
```

Animated "Loading..." screen

The "Loading..." screen is meant to provide the user with a simple interface while the application and data load. The "Loading..." screen will automatically fade out when the page has finished loading. The "Loading" screen is purely aesthetic and can be disabled by deleting the entire contents of the DIV .loading_indicator.

Part 3: The Configuration Parameters

In the file "salesforceConfig.js", you'll find many Flaremap parameters such as:

```
var viewConfig = {
    "tm.group.gutter.size.default": "1" ,
    "tm.group.bg.color.default": "DADADA" ,
    "tm.group.label.color.default": "4F5155" ,
}
```

The parameters above are identical to Flaremap Thin parameters and can be interchanged.

The parameters above are also identical to Flaremap Rich parameters, except for the packaging: Rich parameters come in HTML format while JS parameters come in a JavaScript object. The above parameters, if formatted for Flaremap Rich, would look like this:

```
<param name="tm.group.gutter.size.default" value="1"/>
<param name="tm.group.bg.color.default" value="DADADA"/>
<param name="tm.group.label.color.default" value="4F5155"/>
```

If you are building a treemap from Flaremap Rich parameters, keep in mind that Flaremap Rich contains additional parameters that are not used in Flaremap JS.

A complete description of each configuration parameter used by Flaremap JS can be found in Part 6.

At the bottom of "salesforceConfig.js", you'll find this initialization function:

```
$(function(){
       onLoad(viewConfig);
});
```

The initialization function is required for Flaremap JS to load. The initialization function invokes the "onLoad" command, which passes along the configuration parameters for generating the Flaremap View. In addition, the initialization function can contain customization parameters – customization parameters will be mentioned throughout the User's Guide.

Part 4: The Data

Flaremap JS can connect to many types of data formats.

Since Flaremap JS is built with JavaScript, the native Flaremap JS data format is essentially a CSV file expressed as a JavaScript "array of arrays". A CSV file such as:

```
Sales_Rep,Sales_Manager,Sales_Region,Total_Sales_Q4,Quota_Q4,Quota_Percent
Brett Black,Albert Bachman III,Mountain West,478920,614000,78
Vincent Iorio,Albert Bachman III,Mountain West,353100,535000,66
```

Will be expressed in JavaScript as:

```
var flaremapData = [
['Sales_Rep', 'Sales_Manager', 'Sales_Region', 'Total_Sales_Q4', 'Quota_Q4',
'Quota_Percent'],
['Brett Black', 'Albert Bachman III', 'Mountain West', 478920, 614000, 78],
['Vincent Iorio', 'Albert Bachman III', 'Mountain West', 353100, 535000, 66],
];
```

For an example Flaremap JS data set, see the file "salesforceData.js".

For information on connecting to different data formats see Part 6.1 – Connecting to Data.

Part 5: The Mouseover Box

The cell and group mouseover box are specified in the core HTML of each application.

Each mouseover box consists of regular HTML markup along with variables that are populated, on-the-fly, with values from the active cell or active group.

Clicking on a cell or group reveals the cell/group action menu – where customized actions can be launched directly from the application.

5.1 – Cell Mouseover Box

The cell mouseover box is located in the following div tag of the HTML – anything entered in this div tag will be displayed in the cell mouseover box:

#mouseOverContainer .cellMouseOver

Standard HTML markup can be used to customize the look and layout of the cell mouseover box.

To insert the value of an attribute in the cell mouseover box, enclose the attribute name in curly braces '{}'. In the snippet below we use the attributes Sales_Rep, Sales_Manager, and Sales_Region as variables.

Sample cell mouseover box:

```
<table cellpadding="0" width="100%">
      <tr><td class="labelBold blue" colspan="2">{Sales_Rep}</td></tr>
      <tr><td colspan="2">Manager: {Sales_Manager}</td></tr>
      <tr><td colspan="2">Region: {Sales_Region}</td></tr>

      <tr><td colspan="2"> </td></tr>

      <tr class="stat"><td>Total Sales (Q4):</td><td>${Total_Sales_Q4}</td></tr>
      <tr class="stat"><td>Quota (Q4):</td><td>${Quota_Q4}</td></tr>
      <tr class="stat"><td>% of Quota Met:</td><td>{Quota_Percent}%</td></tr>

      <tr><td colspan="2"> </td></tr>
</table>
```

5.2 – Cell Mouseover Box Actions

When a user clicks the mouse on a cell, a menu will appear from the mouseover box. This menu contains actions the end-user can make from the cell, including dynamic links and special Flaremap actions like tracking and zooming.

Cell actions are located in the following div tag of the HTML.

#mouseOverContainer .cellMouseOver .moMenuContainer .moMenu .moMenuItem a

All actions are formatted as hyperlinks. Standard HTML markup for the <a> tag may be used. To insert the value of an attribute from the cell in a link, enclose the attribute name in curly braces '{}'.

Cell Tracking

To add Cell Tracking to a cell mouseover box action menu include the following code. Once added, the Cell Tracking action item will automatically change between "Track this cell" and "Untrack this cell" depending upon the state of the current cell.

```
<div class="moMenuItem"><a href="javascript:MouseOverActions.trackCell(-
1);"><span id="spanTrackOrUntrack">Track</span> this cell</a></div>
```

Group Tracking

To add Group Tracking to a cell mouseover box action menu include the following code. Both items below must be included for group tracking to work properly.

```
<div class="moMenuItem"><a
href="javascript:MouseOverActions.trackParentGroup(1);">Track visible cells
in this group</a></div>

<div class="moMenuItem"><a
href="javascript:MouseOverActions.trackParentGroup(0);">Untrack visible cells
in this group</a></div>
```

Cell Zooming

To add zooming to a cell mouseover box action menu include the following code. Both items below must be included for group tracking to work properly. Note: The "Zoom Out" option will only appear when the treemap is "zoomed in".

```
<div class="moMenuItem"><a
href="javascript:MouseOverActions.zoomToGroup();">Zoom in</a></div>

<div class="moMenuItem zoomOutAction" style="display:none;"><a
href="javascript:MouseOverActions.zoomOut();">Zoom out</a></div>
```

Example

Sample cell mouseover box action menu:

```
<div id="mouseOverContainer">
...
<div class="moMenu">
       <div class="moMenuItem"><a
       href="javascript:MouseOverActions.trackCell(-1);"><span
       id="spanTrackOrUntrack">Track</span> this cell</a></div>

       <div class="moMenuItem"><a
       href="javascript:MouseOverActions.zoomToGroup();">Zoom in</a></div>

       <div class="moMenuItem zoomOutAction" style="display:none;"><a
       href="javascript:MouseOverActions.zoomOut();">Zoom out</a></div>

       <div class="moMenuItem"><a href="mailto:{mgr_email}">Email Sales
       Manager</a></div>

       <div class="moMenuItem"><a
       href="http://company.com/scorecard?id={product_id}
       &mgr={Sales_Manager}">Launch Sales Manager Scorecard</a></div>

       <div class="moMenuItem"><a
       href="javascript:MouseOverActions.trackParentGroup(1);">Track visible
       cells in this group</a></div>

       <div class="moMenuItem"><a
       href="javascript:MouseOverActions.trackParentGroup(0);">Untrack
       visible cells in this group</a></div>

</div>
...
</div>
```

5.3 – Group Mouseover Box

The group mouseover box is located in the following div tag of the HTML – anything entered in this div tag will be displayed in the group mouseover box:

```
#groupMouseOverContainer .groupMouseOver
```

Standard HTML markup can be used to customize the look and layout of the group mouseover box.

The group mouseover box can contain dynamically-generated summary statistics of the active group. The following summary statistics can be used in the group mouseover box:

Display the active group's title

Description: Displays the title of the active group. The title is the same text as the group header.

Value: {title}

{title}	Must be included exactly as written, including curly braces.

Example: `<td>{title}<br>Summary Statistics</td>`

Display the number of cells inside the active group

Description: Displays the count of the total number of cells inside the active group.

Value: {{count}}

{{count}}	Must be included exactly as written, including both sets of curly braces.

Example: `<td>Total records in this group: {{count}}</td>`

Attribute's average

Description: Calculates the mean of all cells inside the active group.

Value: {{value_name}.avg}

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. Outer curly braces must be included.

Example: `<td>Average Quota: ${Quota_Q4.avg}</td>`

Attribute's max

Description: Returns the maximum value among the cells contained by this group.

Value: {{value_name}.max}

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. Outer curly braces must be included.

Example: `<td>Max Quota: ${Quota_Q4.max}</td>`

Attribute's median

Description: Returns the value in the middle of the cells contained by this group. E.g., of N cells, ordered from minimum to maximum, returns the value of the cell number N/2.

Value: {{value_name}.median}

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. Outer curly braces must be included.

Example: `<td>Median Quota: ${Quota_Q4.median}</td>`

Attribute's min

Description: Returns the minimum value among the cells contained by this group.

Value: {{value_name}.min}

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. Outer curly braces must be included.

Example: `<td>Min Quota: ${Quota_Q4.min}</td>`

Attribute's mode

Description: Returns the most frequently occurring value among the cells contained by this group. If all values are unique, the first value is returned.

Value: {{value_name}.mode}

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. Outer curly braces must be included.

Example: `<td>Quota Mode: ${Quota_Q4.mode}</td>`

Attribute's standard deviation

Description: Returns the standard deviation of the values of the cells contained by this group.

Value: {{value_name}.stdev}

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. Outer curly braces must be included.

Example: `<td>Total Sales Standard Deviation: ${Total_Sales_Q4.stdev}</td>`

Attribute's sum

Description: Calculates the sum of all cells inside the active group.

Value: {{value_name}.sum}

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. Outer curly braces must be included.

Example: `<td>Total Sales: ${Total_Sales_Q4.sum}</td>`

Attribute's variance

Description: Returns the variance of the values of the cells contained by this group.

Value: {{value_name}.variance}

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. Outer curly braces must be included.

Example: `<td>Variance of Total Sales: ${Total_Sales_Q4.variance}</td>`

Sample group mouseover box:

```
<table cellpadding="0" width="100%">

       <tr><td colspan="2">{title}<br>Summary Statistics</td></tr>

       <tr><td colspan="2" class="statLabel"> </td></tr>

       <tr><td>Total Sales:</td><td>${Total_Sales_Q4.sum}</td></tr>
       <tr><td>Total Quota:</td><td>${Quota_Q4.sum}</td></tr>
       <tr><td>Avg Quota Met:</td><td>{Quota_Percent.avg}%</td></tr>

</table>
```

5.4 – Group Mouseover Box Actions

When a user clicks the mouse on a group, a menu will appear from the mouseover box. This menu contains actions the end-user can make from the group, including dynamic links and special Flaremap actions like tracking.

Group actions are located in the following div tag of the HTML.

 # groupMouseOverContainer .groupMouseOver .moMenuContainer .moMenu .moMenuItem a

All actions are formatted as hyperlinks. Standard HTML markup for the <a> tag may be used.

Group Tracking

To add Group Tracking to a group mouseover box action menu include the following code. Both items below must be included for group tracking to work properly.

```
<div class="moMenuItem"><a
href="javascript:MouseOverActions.trackCell(1);">Track visible cells in this
group</a></div>

<div class="moMenuItem"><a
href="javascript:MouseOverActions.trackCell(0);">Untrack visible cells in
this group</a></div>
```

Group Zooming

To add zooming to a group mouseover box action menu include the following code. Both items below must be included for group tracking to work properly. Note: The "Zoom Out" option will only appear when the treemap is "zoomed in".

```
<div class="moMenuItem"><a
href="javascript:MouseOverActions.zoomToGroup();">Zoom in</a></div>

<div class="moMenuItem zoomOutAction" style="display:none;"><a
href="javascript:MouseOverActions.zoomOut();">Zoom out</a></div>
```

Example

Sample group mouseover box action menu:

```
<div id="mouseOverContainer">
...
<div class="moMenu">
       <div class="moMenuItem"><a
       href="javascript:MouseOverActions.trackCell(1);"> Track visible cells
       in this group</a></div>

       <div class="moMenuItem"><a
       href="javascript:MouseOverActions.trackCell(0);"> Untrack visible
       cells in this group</a></div>

       <div class="moMenuItem"><a
       href="javascript:MouseOverActions.zoomToGroup();">Zoom in</a></div>

       <div class="moMenuItem zoomOutAction" style="display:none;"><a
       href="javascript:MouseOverActions.zoomOut();">Zoom out</a></div>

       <div class="moMenuItem"><a href="http://company.com/">Launch Corporate
       Website</a></div>
</div>
...
</div>
```

5.5 – Adding Conditional Logic

Conditional logic provides additional format flexibility and can increase the utility and readability of mouseover box information. For example, suppose you are creating a supply chain Flaremap and only want to show the re-order costs when the current supply is less than 500 units. This can be done using a simple conditional statement. It is also possible to define more complex IF – THEN – ELSE statements.

Both the cell mouseover box and group mouseover box can have conditional logic added to individual elements. Conditional logic can be used on any HTML element. When conditional logic is used, the HTML element will display ONLY when the logic statement specified is true.

Conditional logic can be added in two steps:

1. Add ONE of the following classes to the HTML element:
    - `flaremap-if` Use this class to create simple IF statements.
        - An HTML element with the class `flaremap-if` will execute independent of any other conditional logic present in the mouseover box.
    - `flaremap-else` Use this class to create IF – THEN – ELSE statements.
        - An HTML element with the class `flaremap-else` will look for a preceding HTML element that contains the class `flaremap-if` OR `flaremap-else`.
        - Any `flaremap-else` element that follows an element with the class `flaremap-if` OR `flaremap-else` will only execute its logic statement if the preceding element's logic statement returned false.
        - An HTML element with the class `flaremap-if` will always begin a new logic statement.
        - An HTML element with the class `flaremap-if` is required to begin a new logic statement.
        - If only HTML elements with the class `flaremap-else` are used all of the `flaremap-else` elements will be hidden.

2. Add the following new attribute to the HTML element:

    if="{value_name}|{operation}|{operation_values}"

{value_name}	Any attribute from "CSV_DataTypes.cells"	Specifies the attribute that will be used in the logic statement.
{operation}	See table below for list of operations	Specifies the logical operator that will be used against {value_name}
{operation_values}	See table below for how different operations require different values	Specifies what values the operation will use to determine if the logic statement is true or false.

Table of operations and operation values

Description	{operation}	{operation_values}
equals	EQ	Any number or string. Will only return results that exactly match the value. Use HASTEXT for more granular text searching.
not equals	NEQ	Any number or string. Will only return results that exactly match the value. Use HASTEXT for more granular text searching.
less than	LT	Any number
less than or equals	LTE	Any number
greater than	GT	Any number
greater than or equals	GTE	Any number
set membership (in a set)	SET	Any pipe separated list of numbers or strings. Will only return results that exactly match the value. Use HASTEXT for more granular text searching. Values within SET have an OR relationship.
from (inclusive)… to (inclusive)	[]	Any two pipe separated numbers (the second should be larger than the first)
from (not inclusive)… to (inclusive)	(]	Any two pipe separated numbers (the second should be larger than the first)
from (inclusive)… to (not inclusive)	[)	Any two pipe separated numbers (the second should be larger than the first)
from (not inclusive)… to (not inclusive)	()	Any two pipe separated numbers (the second should be larger than the first)
text search	HASTEXT	Any pipe separated list of numbers or strings. Will return any results that match the value, including partial matches. For example “Dan” will return: “Dan”, “Daniel”, and “Dan Smith”.
current option – checks if {value_name} is currently used as the SIZE or COLOR option	CURRENT_ OPTION	Three possible values: SIZE, COLOR, or SIZE|COLOR
adding an exclamation point immediately before any of the operations above will negate the filter rule	!	n/a

Examples:

Example 1: Display the phrase "North Region" when hovered over a cell with a region that contains the word "North".

Example 1 code:

```
<div class="flaremap-if" if="Sales_Region|HASTEXT|North">North Region</div>
```

Example 2: Display the phrase "North Region" when hovered over a cell with a region that contains the word "North". Display the phrase "South Region" when hovered over a cell with a region that contains the word "South".

Example 2 code:

```
<div class="flaremap-if" if="Sales_Region|HASTEXT|North">North Region</div>
<div class="flaremap-if" if="Sales_Region|HASTEXT|South">South Region</div>
```

Example 3: Display the phrase "North Region" when hovered over a cell with a region that contains the word "North". Display the phrase "North Region Over $500k" when hovered over a cell with a region that contains the word "North" AND has total sales greater than or equal to 500,000.

Example 3 code:

```
<div class="flaremap-if" if="Sales_Region|HASTEXT|North">
      North Region <span class="flaremap-if"
      if="Total_Sales_Q4|GTE|500000">Over $500k</span>
</div>
```

Example 4: Display the phrase "Above Quota" when hovered over a cell with a quota greater than or equal to 100. Display the phrase "Below Quota" in red lettering when hovered over a cell with a quota less than 100. Otherwise display the phrase "No Quota Data Provided".

Example 4 code:

```
<div class="flaremap-if" if="Quota_Percent|GTE|100">Above Quota</div>
<div class="flaremap-else" if="Quota_Percent|LT|100" style="color:red">Under
Quota</div>
<div class="flaremap-else">No Quota Data Provided</div>
```

5.6 – Mouseover Box Width

The width of the mouseover box can be changed by adding a simple JQuery function to the initialization function located at the bottom of Configuration file (e.g. "salesforceConfig.js").

Cell Mouseover box width

Customization Parameter:

$('#mouseOverContainer .cellMouseOver').css('width', '{width_in_px}');

{width_in_px}	Specifies the width of the cell mouseover box. The value must end in "px" (e.g. 310px)

Example:

```
$(function(){
       onLoad(viewConfig);
       $('#mouseOverContainer .cellMouseOver').css('width', '310px');
});
```

Group Mouseover box width

Customization Parameter:

$('# groupMouseOverContainer .groupMouseOver).css('width', '{width_in_px}');

{width_in_px}	Specifies the width of the group mouseover box. The value must end in "px" (e.g. 310px)

Example:

```
$(function(){
       onLoad(viewConfig);
       $('#groupMouseOverContainer . groupMouseOver).css('width', '360px');
});
```

5.7 – Mouseover Box Variables Formatting

The *appearance* of mouseover box variables can be modified using standard HTML and CSS. In addition, there are several options available for adjusting the *format* of mouseover box variables.

The *format* of all mouseover box variables are adjusted by adding customization parameters to the initialization function located at the bottom of Configuration file (e.g. "salesforceConfig.js").

```
$(function(){
       onLoad(viewConfig);
       {customization_parameter_1};
       {customization_parameter_2};
       {customization_parameter_3};
       {customization_parameter_N};
});
```

Date format

Customization Parameter:

MouseOverPainter.setFieldFormatOption('{value_name}', 'format', '{date_format}');

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties.
{date_format}	To specify the date/time format use a pattern string made up of the symbols specified in Section 6.1, subsection {date_format}.

Example:

```
$(function(){
       onLoad(viewConfig);
       MouseOverPainter.setFieldFormatOption('last_sold_date', 'format',
       'd MMM yyyy');
       MouseOverPainter.setFieldFormatOption('end_date', 'format',
       'yyyy MM-dd');
});
```

Decimal places

Customization Parameter:

MouseOverPainter.setFieldFormatOption('{value_name}', 'sigdigits', {number_of_decimal_places});

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties.

{number_of_decimal_places}	Any positive integer. This is the number of decimal places that will be displayed.

Example:

```
$(function(){
       onLoad(viewConfig);
       MouseOverPainter.setFieldFormatOption('Quota_Percent', 'sigdigits',
       1);
       MouseOverPainter.setFieldFormatOption('Quota_Percent.avg',
       'sigdigits', 2);
});
```

Comma separator in large numbers

Customization Parameter:

MouseOverPainter.setFieldFormatOption('{value_name}', 'no_comma', {true_false});

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties.
{true_false}	true – separating commas are not used in large numbers false – separating commas are used in large numbers

Example:

```
$(function(){
       onLoad(viewConfig);
       MouseOverPainter.setFieldFormatOption('Total_Sales_Q4', 'no_comma',
       true);
});
```

"no data" text for blank values: global setting

Customization Parameter:

MouseOverPainter.setNoDataText('{no_data_text}');

{no_data_text}	Specifies the string to use when no data is available.

Special Note: In the case of no data, any prefix or suffix is combined with the {no_data_text} string.

Example:

```
$(function(){
       onLoad(viewConfig);
```

```
        MouseOverPainter.setNoDataText('No Data Available');
});
```

"no data" text for blank values: individual attribute setting and/or global setting override

Customization Parameter:

MouseOverPainter.setFieldFormatOption('{value_name}', 'no_data', '{no_data_text}');

{value_name}	Any attribute from "CSV_DataTypes.cells".
{no_data_text}	Specifies the string to use when no data is available.

Special Note: In the case of no data, any prefix or suffix is combined with the {no_data_text} string.

Example:

```
$(function(){
        onLoad(viewConfig);
        MouseOverPainter.setFieldFormatOption('Quota_Q4', 'no_data', 'n/a');
});
```

Part 6: Configuration Parameters in detail

The Flaremap JS configuration parameters are at the core of building and modifying a Flaremap JS treemap. The configuration parameters can be categorized into the following groups:

6.1 – Connecting to data

6.2 – Creating group options

6.3 – Creating group pre-filters

6.4 – Creating size options

6.5 – Creating color options

6.6 – Setting up group, size, and color controls

6.7 – Adding cell labels

6.8 – Adding filters, indicators, and sliders

6.9 – Modifying look and feel options of the treemap field

6.10 – Localization options

6.11 – Priority Rendering Engine options

Using salesforceConfig.js as an example, we will go through each configuration parameter in detail below.

6.1 – Connecting to data

There are two required parameters for connecting to data: **CSV_DataTypes.cells** (specifies the data attributes) and **tm.data_provider.info** (specifies the location of the data). Both parameters are described in detail below. In addition, there are optional parameters for enabling live updates to the data (no page refresh).

CSV_DataTypes.cells

Description: Specifies the data type for each column of data. Entries do not have to be in the same order as those in the data source header row.

Value: {value_name1}|{data_type_definition},{value_name2}|{data_type_definition}, ... {value_nameN}|{data_type_definition}

<table>
<tr><td>{value_name}</td><td>Any attribute from the dataset.</td></tr>
<tr><td>{data_type_definition}</td><td>There are four possible values, depending on how {value_name} will be used in the treemap:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>NUMBER</td><td>Numeric data</td></tr>
<tr><td>TEXT</td><td>Text data</td></tr>
<tr><td>TEXT|ENUM</td><td>Enumerated text data, allows Flaremap to automatically create groups and filters. Required for any attribute that will be automatically grouped by Flaremap.</td></tr>
<tr><td>DATE|{date_format}</td><td>Date and time formatted data. See below for details on {date_format}</td></tr>
</table></td></tr>
</table>

{date_format}

Description: When using date/time formatted attributes, the date/time format used by the database (i.e. the input format) needs to be specified. Flaremap JS supports a wide range of date/time formats. To specify the date/time format use a pattern string made up of the following symbols.

Symbol	Meaning	Presentation	Example
G	era designator	Text	AD
y	year	Number	1996
M	month in year	Text & Number	July & 07
d	day in month	Number	10
h	hour in am/pm (1~12)	Number	12
H	hour in day (0~23)	Number	0
m	minute in hour	Number	30
s	second in minute	Number	55

S	millisecond	Number	978
E	day in week	Text	Tuesday
D	day in year	Number	189
F	day of week in month	Number	2 (2nd Wed in July)
w	week in year	Number	27
W	week in month	Number	2
a	am/pm marker	Text	PM
k	hour in day (1~24)	Number	24
K	hour in am/pm (0~11)	Number	0

Note: The pattern of symbols determine the format. For example:

d = 2
dd = 02

M = 8
MM = 08
MMM = Aug
MMMM = August

Example: `"CSV_DataTypes.cells": "Sales_Rep|TEXT,Sales_Manager|TEXT|ENUM, Sales_Region|TEXT|ENUM,Total_Sales_Q4|NUMBER,Quota_Q4|NUMBER, Quota_Percent|NUMBER,Start_Date|DATE|yyyy-MM-dd",`

tm.data_provider.info

Description: Specifies the type of data connection being made and the location of the data. For details on complying with the "Same-Origin" Policy, see the special note at the end of this section.

Value: The value will depend on the format of data being used. See below.

Value: *For JS formatted data:*

JS_URL={path_to_data}[|JS_VAR_NAME={js_var}]

{path_to_data}	Path to JS formatted data.
{js_var}	Optional: Specifies the name of the variable used in the JS data file to define the data. If no variable is defined, the variable "flaremapData" is expected.

Example: *For JS formatted data:*

```
"tm.data_provider.info": "JS_URL=salesforceData.js",

"tm.data_provider.info": "JS_URL=salesforceData.js|
JS_VAR_NAME=myCustomDataVariable",
```

Value: *For JSON formatted data:*

JS_URL={path_to_data}

{path_to_data}	Path to JS formatted data.

Example: *For JSON formatted data:*

```
"tm.data_provider.info": "JS_URL=data/salesforce.json",
```

Note: *For JSON formatted data:* Nested structures are supported by using a period to separate each level of the hierarchy. For example, the attributes present in the data below are: "Sales_Rep", "Sales_Manager.mgr1", "Sales_Manager.mgr2", "Sales_Manager.mgr3", "Quota_Q4" .

```
var flaremapData = [
{
'Sales_Rep': 'Vincent Iorio',
'Sales_Manager': {
        'mgr1': 'Albert',
        'mgr2': 'Dennis',
        'mgr3': 'Jack',
},
'Quota_Q4': 535000,
},
];
```

Value: *For CSV formatted data:*

CSV_URL={path_to_data}

{path_to_data}	Path to CSV formatted data.

Example: *For CSV formatted data:*

```
"tm.data_provider.info": "CSV_URL=data/salesforce.csv",
```

Value: *For Copy/Paste data:*

DIV_ID={div_in_html}

In the core HTML, insert a hidden DIV (see example below), then paste CSV formatted data into the DIV. The data must be in comma separated format.

{div_in_html}	Name of the DIV ID in the core HTML that contains the data.

Example: *For Copy/Paste data:*

```
"tm.data_provider.info": "DIV_ID=flaremapData",
```

In the core HTML, an example DIV (including data) is below:

```
<div id="flaremapData" style="display:none;">
Sales_Rep,Sales_Manager,Quota_Q4,Quota_Percent
Brett Black,Albert Bachman III,614000,78
Vincent Iorio,Albert Bachman III,535000,66
Henry Bateman,Albert Bachman III,433000,117
Mike Cowgill,Albert Bachman III,641000,97
</div>
```

tm.data_provider.loader

Description: The "Data Loader" enables the connection to an external data source, and complies with the "Same-Origin Policy" implemented in modern browsers. For a sample Data Loader, see the file: csvConnect.jsp. This parameter can be omitted in many circumstance – please read the special note on "Same-Origin" Policy (located at the end of this section) for details on when this parameter can be omitted.

Value: CSV_LOADER={path_to_loader}

{path_to_loader}	Path to Data Loader.

Example: `"tm.data_provider.loader": "CSV_LOADER=https://myserver.com/data_loader/csvConnect.jsp",`

Special Note: "Same-Origin" Policy

Summary: The "Same-Origin" Policy is implemented in all modern browsers. The Policy enforces that all data accessed by a web page must come from the same origin. An origin is defined as a combination of URI scheme, hostname, and port number. This is important to keep in mind when accessing data from different servers and when accessing local data. For example, most browsers consider all file:// URLs as having different origins. Complying with the "Same-Origin" Policy is easy. See the table below for some of the options available.

Quick reference sheet for complying with "Same-Origin" Policy

Type of data and URL used by "tm.data_provider.info" parameter	Application is running from local desktop	Application is running on a server
Copy/Paste (DIV_ID=...)	✓ Complies	✓ Complies
JS, relative URL (JS_URL=...)	✓ Complies	✓ Complies
JS, http URL (JS_URL=http://...)	✓ Complies	✓ Complies
CSV, relative URL (CSV_URL=...)	Does not comply with "Same-Origin" Policy	✓ Complies
CSV, http URL (CSV_URL=http://...)	✓ Complies with use of Data Loader	✓ Complies with use of Data Loader

Data Loader parameter (tm.data_provider.loader)	Application is running from local desktop	Application is running on a server
URL Options	URL must start with http	URL can be relative or start with http

tm.data_provider.refresh.basic.interval

Description: Specifies a time interval (in seconds) at which the data set should be re-loaded and merged into the existing set. New rows are added, existing are updated, and missing rows are removed. In order to define how a row is determined to be new, existing, or missing, a list of "unique key" columns must be specified by the "tm.data_provider.unique_key" parameter.

Value: {number_of_seconds}

{number_of_seconds}	An integer corresponding to the number of desired seconds between each data refresh attempt.

Example: `"tm.data_provider.refresh.basic.interval": "60",`

tm.data_provider.unique_key

Description: Specifies a list of column names that, collectively, uniquely define a row: if a row added after page load shares identical values with an existing row for each of these columns, the new row will replace the existing row. If such a row does not match any existing rows in all of these columns, it will be considered new and added to the existing data set.

Value: {column_name_1},...,{column_name_N}

{column_name}	The name of a column that exists in the list specified by the CSV_DataTypes.cells parameter.

Example: `"tm.data_provider.unique_key": "Sales_Rep,Sales_Manager",`

6.2 – Creating group options

Note: Two options exist for creating a group: 1) Auto group generation, and 2) Manually defined groups. Option 1 is recommended for most implementations.

Option 1 (Auto groups): tm.group.{group_name}

Description: Defines each group – where {group_name} is a unique name used internally to reference the group option, and the value of the parameter defines the group option.

Value: {value_name}[;{override_list}]|{AUTO_ENUM_LIST}

{value_name}	Any attribute from "CSV_DataTypes.cells" that has ENUM properties.
{override_list}	Optional: Additional tags that override the default group parameters. See the "Group Override Table" below for list and format of overrides.
{AUTO_ENUM_LIST}	Must be included exactly as written, including curly braces, so that groups are automatically generated.

Group Override Table
An optional section of the group definition.
The format for listing multiple group overrides is: {tag1}={value1},{tag2}={value2},...,{tagN}={valueN}

Property	Tag	Value	Parameter it overrides
Group background color	c	hexadecimal color value	tm.group.bg.color.default
Label font	lf	any font defined by one of the tm.font.size.{font_name} parameters	tm.group.label.font.default

Example: `"tm.group.Sales_ManagerGRP": "Sales_Manager|{AUTO_ENUM_LIST}",`
`"tm.group.Sales_RegionGRP": "Sales_Region;c=0000ff|{AUTO_ENUM_LIST}",`

Option 2 (Manually defined groups): tm.group.{group_name}

Description: Defines each group – where {group_name} is a unique name used internally to reference the group option, and the value of the parameter defines the group option.

Value: {value_name}[;{override_list}]|{group_spec1}|...|{group_specN}

{value_name}	Any attribute from "CSV_DataTypes.cells" that has ENUM properties.
{override_list}	Optional: Additional tags that override the default group parameters. See the "Group Override Table" below for list and format of overrides.

{group_spec}	Manually defines a group. See {group_spec} table below for format.

Group Override Table

An optional section of the group definition.

The format for listing multiple group overrides is: {tag1}={value1},{tag2}={value2},...,{tagN}={valueN}

Property	Tag	Value	Parameter it overrides
Group background color	c	hexadecimal color value	tm.group.bg.color.default
Label font	lf	any font defined by one of the tm.font.size.{font_name} parameters	tm.group.label.font.default

{group_spec}

Value: [{sub_group}];{label};[{description}];{operation};{operation_values}

{group_spec} item	Value	Explanation
{sub_group}	Any value may be entered, or this field can be left blank.	Not used in current version of Flaremap JS.
{label}	any string	This is the group name that a user sees for the group
{description}	Any value may be entered, or this field can be left blank.	Not used in current version of Flaremap JS.
{operation}	See table below for list of operations	Specifies the operation used to define group membership
{operation_values}	See table below for how different operations require different values	Specifies what values the group operation uses to determine group membership

Group operations and values

Description	{operation}	{operation_values}
equals	EQ	Any number or string. Strings are assumed to be enum values
not equals	NEQ	Any number or string. Strings are assumed to be enum values
less than	LT	Any number
less than or equals	LTE	Any number
greater than	GT	Any number
greater than or equals	GTE	Any number
set membership (in a set)	SET	Any comma separated list of numbers or strings. Strings are assumed to be enum values

from (inclusive)... to (inclusive)	[]	Any two comma separated numbers (the second should be larger than the first)
from (not inclusive)... to (inclusive)	(]	Any two comma separated numbers (the second should be larger than the first)
from (inclusive)... to (not inclusive)	[)	Any two comma separated numbers (the second should be larger than the first)
from (not inclusive)... to (not inclusive)	()	Any two comma separated numbers (the second should be larger than the first)
the rest of the items not already in a group	ELSE	No operation value necessary

Example: `"tm.group.Quota_PercentGRP": "Quota_Percent|;Need Help;;LT;100|;Above Quota;;[);100,150|;Superstars;;GTE;150|;Other;;ELSE;|",`

tm.hierarchy.{hierarchy_name}

Description: Specifies a hierarchical list of groups, with the first listed at the top of the hierarchy, and the last listed at the bottom. Also allows for specification of group option overrides.

Value: {group_name}[`{OPTIONS}]^{group_name2}[`{OPTIONS}]^{group_name3}...

Item	Function	Values
{group_name}	The group to use at this level in the hierarchy. NOTE that this cannot be another GROUP_HIERARCHY definition.	Any {group_name} from the tm.group.{group_name} defined groups
{OPTIONS}	Additional tags that override the default group parameters.	Comma separated list of override options from Group Override Table. See table below.

Group Override Table

An optional section of the group definition.

The format for listing multiple group overrides is: {tag1}={value1},{tag2}={value2},...,{tagN}={valueN}

Property	Tag	Value	Parameter it overrides
Group background color	c	hexadecimal color value	tm.group.bg.color.default
Label font	lf	any font defined by one of the tm.font.size.{font_name} parameters	tm.group.label.font.default

Example: `"tm.hierarchy.HierGrp": "Sales_RegionGRP^Sales_ManagerGRP`c=999999,lf=size9" ,`

6.3 – Creating group pre-filters

Optional: Group pre-filters provide a convenient way to modify the pre-filter settings of a Flaremap when changing group options. For example, if your dataset contains the same data aggregated in multiple ways: sales by region and sales by manager; you can use group pre-filters to display the relevant data based on the current group option. When group is set to Region, a group pre-filter can be set so that only manager aggregate data is visible. Conversely, when group is set to Manager, a group pre-filter can be set so that only region aggregate data is visible.

Note: Group pre-filters are always applied after URL pre-filters. Group pre-filters cannot change URL pre-filters.

tm.group_prefilters_off

Description: Turns group pre-filter functionality on/off.

Value: true – turns off group pre-filter functionality (default)
false – turns on group pre-filter functionality

Example: `"tm.group_prefilters_off":"false",`

tm.group.{group_name}.prefilterN

Description: Defines a group pre-filter. A group pre-filter must be associated with an existing group option. In the parameter, {group_name} needs to be replaced with the name of an existing group option. In the parameter, the letter "N" needs to be replaced with a unique number (e.g. 1, 2, 3, etc.)

Value: See Section 8.2: Group pre-filters take the exact same format and values as URL pre-filters.

Example: `"tm.group.Sales_ManagerGRP.prefilter1":"Sales_Region|EQ|US",`
`"tm.group.Sales_ManagerGRP.prefilter2":"QuotaQ4|GTE|10000",`

6.4 – Creating size options

Note: This parameter is optional. In versions 6.2+ a size option can be declared directly in the "Size" parameter by entering the {value_name} for the {size_option} in the "Size" parameter. See section 6.5 for details.

tm.size.option.{size_option}

Description: Defines a size option – where {size_option} is a unique name used internally to reference the size option, and the value of the parameter defines the size option.

Value: {value_name}

{value_name}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. NOTE: To force all cells to be the same size, use "all" in replace of {value_name}

Example:
```
"tm.size.option.Sales": "Sales_Q4" ,
"tm.size.option.Quota": "Quota_Q4" ,
"tm.size.option.SameSize": "all" ,
```

6.5 – Creating color options

tm.color.range.{color_option}

Description: Defines a color option – where {color_option} is a unique name used internally to reference the color option; and the value of the parameter specifies the exact colors used by a {color_option} and optionally assigns a specific value to one or more of the colors.

Value: {color_value1}[:{data_value}], … ,{color_valueN}[:{data_value}]
A comma separated list of {color_values} with a {data_value} optionally assigned to a {color_value}.

{color_value}	Any hexadecimal color.
{data_value}	For a numeric color spectrum: any number. For a discrete color spectrum (based on enumerated text): any word/phrase found in the data.

Example:
```
"tm.color.range.Quota_PercentCLR":"ff0000:-5,000000:0,00ff00:5",
"tm.color.range.Total_SalesCLR": "990000,e33500,ffff00,78F065,0F6A00",
"tm.color.range.Quota_MetCLR": "ff0000:No,00ff00:Yes",
"tm.color.range.Sales_RegionCLR": "78F065:Mountain West,ffff00:
Southeast,0000dd:Southwest,990000:Northeast,e33500:Great Lakes,
0F6A00:Northwest",
```

tm.color.nodata

Description: Specifies the color to use for a cell when a cell has invalid data or no data for the currently selected color option.

Value: Any hexadecimal color value.

Example: `"tm.color.nodata": "cccccc",`

Optional: Custom-defined color legend

In Flaremap JS, the color legend is automatically generated based on the "tm.color.range.{color_option}" parameter. Occasionally, it may be useful to use a custom-defined color legend instead of the automatically generated one. For example, when a color option includes ranges for outlier values, the majority of the color legend may be obscured because the colors are being spaced on a linear scale. The custom-defined color legend allows designers to create a color legend independent of the color scale used by treemap (i.e. independent of the "tm.color.range.{color_option}" parameter). The custom-defined color legend can display a simplified legend that connotes the colors being used by the treemap.

To create a custom-defined color legend (and override the automatically generated color legend), include the following two parameters (both parameter must be used).

color.range.{color_option}

Description: Sets the legend color values for a {color_option}.

Value: {color_value1}[:{data_value}], … ,{color_valueN}[:{data_value}]
A comma separated list of {color_values} with a {data_value} optionally assigned to a {color_value}.

{color_value}	Any hexadecimal color.
{data_value}	For a numeric color spectrum: any number. For a discrete color spectrum (based on enumerated text): any word/phrase found in the data.

Example: `"color.range.Quota_PercentCLR":"ff0000:-5,000000:0,00ff00:5",`
`"color.range.Sales_RegionCLR": "78F065,ffff00,0000dd,990000,e33500,0F6A00",`

color.legend.{color_option}

Description: Specifies a list of labels that map to the corresponding colors as specified by the color.range.{color_option} parameter.

Value: {legend_type},{label1},…,{labelN}
A comma separated list of {labelN}.

Special Note: The value of each "color.legend.{color_option}" parameter must be have the same number of labels as there are points in the associated color range. If the label for a particular point is not to appear (to avoid clutter, for example), an empty entry must be used.

{legend_type}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{labelN}	Any string.

Example: `"color.legend.Quota_PercentCLR": ",-5%,At Quota,5%",`
`"color.legend.Sales_RegionCLR": "Mt West,SE,SW,NE,Gt Lakes,NW",`

Color blind support

Color blind support can be optionally enabled in Flaremap JS. When the color blind functionality is turned on a hyperlink appears near the color legend that allows the user to switch the color options between the standard colors and the color blind safe colors. By default, the color blind functionality operates in a smart-auto mode and automatically converts all color options to color blind safe color options. The smart-auto color blind options can also be overridden by manually defining color blind options

The smart-auto functionality is designed to create a color option that makes it easy for people with color deficiencies to use the interface. To be most effective, the smart-auto functionality simplifies existing color options and replaces the colors with a set of high contrast, color blind safe colors. During our trials, this method was far superior to every other option tested and was the most logical for users with color deficiencies to understand. The smart-auto functionality works well for all typical color options. For highly customized and intricate color options it is expected that designers will manually create a second color option suitable for people with color deficiencies.

The smart-auto functionality works by scanning the standard color option parameters and then creating color blind safe alternative color options: The smart-auto option will create a smooth, color blind safe gradient between the endpoints of the standard color option. All colors listed in between the endpoints of the standard color option will be discarded – unless one of the middle colors is tied to the numerical value "0" or "100". If one of the middle colors of the standard color option are tied to "0" or "100" the smart-auto option will create a three-color gradient that pivots on the "0" or "100" – smart-auto will preserve the numeric value of the middle number. If the endpoints of the standard color option are tied to a numerical value, the smart-auto option will preserve the numerical values of those endpoints in the smart-auto option.

tm.color.cb.toggle

Description: Toggles color blind support on and off.

Value: "on" or "off". The value "on" will turn on color blind support.

Example: `"tm.color.cb.toggle": "on",`

tm.strings.cb.control.turn_on.label

Description: In the Flaremap interface, near the color legend, a link will appear (default text is "Click here for color safe version") that the user can click to toggle between color blind mode and full-color mode. The text of the hyperlink can be changed with the following parameter.

Value: Any string – basic html markup is allowed.

Example:
```
"tm.strings.cb.control.turn_on.label": "Click here for <br>color
safe version",
```

tm.strings.cb.control.turn_off.label

Description: In the Flaremap interface, near the color legend, a link will appear (default text is "Click here for color safe version") that the user can click to toggle between color blind mode and full-color mode. The text of the hyperlink can be changed with the following parameter.

Value: Any string – basic html markup is allowed.

Example: `"tm.strings.cb.control.turn_off.label": "Click here for <br>standard version",`

Manually defined color blind options (overrides): The smart-auto color blind options can be overridden by manually defining color blind options. To manually create a color option that overrides the smart-auto option use the standard color parameters, but insert ".cb" after the word "color" in each parameter. For example:

```
//Sample standard color parameters
"tm.color.range.Quota_MetCLR": "9a0000,9a0000:99.9,009100:100,009100",
"color.range.Quota_MetCLR": "9a0000:0,9a0000:5,009100:5,009100:10",
"color.legend.Quota_MetCLR": "Below Quota,,,Above Quota",

//Sample color blind override parameters
"tm.color.cb.range.Quota_MetCLR":
"ffff00,ffff00:99.9,0000ff:100,0000ff",
"color.cb.range.Quota_MetCLR": "ffff00:0,ffff00:5,0000ff:5,0000ff:10",
"color.cb.legend.Quota_MetCLR": "Below Quota,,,Above Quota",
```

6.6 – Setting up group, size, and color controls

group

Description: Specifies the group options available to Flaremap end-users in the "Group by" dropdown list.

Value: group|{position_info}|{label_info}|{initial_option}|{written_name1},{group_name1}; ... ; {written_nameN},{group_nameN}

{position_info}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{label_info}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{initial_option}	Specifies the option that is first displayed when the application loads. If nothing is entered the first option is displayed. {initial_option} must match one of {written_name} options.
{written_name}	Specifies the name of an option which appears in the dropdown box control. {written_name} can be any string.
{group_name}	Specifies the internal name of a unique group option previously defined by one of the tm.group.{group_name} parameters or tm.hierarchy.{hierarchy_name} parameters.

Example: `"group": "group|-|-|Sales Manager|Sales Region,Sales_RegionGRP; Sales Manager,Sales_ManagerGRP",`

size

Description: Specifies the size options available to Flaremap end-users in the "Size by" dropdown list.

Value: size|{position_info}|{label_info}|{initial_option}|{written_name1},{size_option1}[^{override}]; ... ; {written_nameN},{ size_optionN}[^{override}]

{position_info}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{label_info}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{initial_option}	Specifies the option that is first displayed when the application loads. If nothing is entered the first option is displayed. {initial_option} must match one of {written_name} options.
{written_name}	Specifies the name of an option which appears in the dropdown box control. {written_name} can be any string.
{size_option}	Specifies the size option. There are two options for this value: 1) Any attribute from "CSV_DataTypes.cells" that has NUMBER properties.

	2) The internal name of a unique size option previously defined in one of the tm.size.option.{size_option} parameters.
{override}	Optional, overrides the size parameter settings for an individual size option. See values below.

{override}

Value	Description	Parameter it overrides
a	Sizes cells based on their absolute values (rather than their original values).	tm.cell.size.abs
i	Inverts the size values.	tm.cell.size.invert
r{#}	Sets the cell size ratio between the smallest and largest cells to the specified number. The values of {#} are described below: {#} = 0: this parameter is ignored {#} = 1: all cells are the same size {#} = 2-50: cells will scale between relatively similar in size (2) and significantly dissimilar in size (50)	tm.cell.size.ratio

Example: `"size": "size|-|-|-|Total Sales Q4,Total_Sales_Q4;Quota Q4,Quota_Q4^r9",`

color

Description: Specifies the color options available to Flaremap end-users in the "Color by" dropdown list.

Value: color|{position_info};{gradient_position_info}[;{enum_position_info}]|{label_info}|{initial_option}|{written_name1},{color_def1}; … ; {written_nameN},{ color_defN}}|{group_written_name1},{group_color_def1}; … ; {group_written_nameN},{group_color_defN}

{position_info}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{gradient_position_info}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{enum_position_info}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{label_info}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{initial_option}	Specifies the option that is first displayed when the application loads. If nothing is entered the first option is displayed. {initial_option} must match one of {written_name} options.
{written_name}	Specifies the name of an option which appears in the dropdown box control. {written_name} can be any string.

{color_def}	Defines how the color option will appear. {color_def} is described in detail below.
{group_written_name}	(Optional) For coloring groups (instead of cells). Same format as {written_name},
{group_color_def}	(Optional) For coloring groups (instead of cells). Same format as color_def, except an aggregate data attribute must be referenced using the a^ override option..

{color_def}
Description: Matches a data attribute with a coloring option.
Value: {color_attribute}^{option_color_method}^p{color_option}[^{color_scale_override}]

{color_attribute}	The data attribute that the cell's color will be based upon. Must be an attribute from "CSV_DataTypes.cells" that has NUMBER or ENUM properties.
{option_color_method}	Must be set to *NodeColorRangeEx*
{color_option}	Any {color_option} previously defined by one of the tm.color.range.{color_option} parameters
{color_scale_override}	Optional. By default colors are plotted along a linear color scale. Alternatively, colors can be plotted along a logarithmic color scale. To change the color scale for ALL color options to a logarithmic color scale use the parameter: "tm.color.logscale": "true", For more details on this parameter see section 6.8. To change the color scale for an individual color option use one of the overrides in the table below.

{color_scale_override}	Description
l	Forces the color option to be plotted along a logarithmic color scale
o	Forces the color option to be plotted along a linear color scale (i.e. uses "original values")
a	Specify an aggregate data value (e.g. 'sum', 'min', 'max', 'avg', etc) to be used for coloring groups.

Example: `"color": "color|-|-|Quota Percent|Quota Percent,Quota_Percent^ NodeColorRangeEx^pQuota_PercentCLR;Total Sales Q4,Total_Sales_Q4^ NodeColorRangeEx^pTotal_Sales_Q4CLR; Sales Region,Sales_Region^ NodeColorRangeEx^pSales_RegionCLR;",`

6.7 – Adding cell labels

celltext

Description: The cell label control lets users change what data is displayed on the cells directly. This allows users to scan for particular data points rapidly. Cell labels can include static text as well as any cell-specific data.

Value: celltext|{position_info}|{label_info}|{control_bgcolor}|{initial_option}|{option1_name},{celltext_spec1}; ... ;{optionN_name},{celltext_specN}

{position_info}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{label_info}	Specifies the text that will appear as the label of the filter. {label_info} can be any string. If a caret (^) is used within the {label_info} field, all characters after the caret will be ignored.
{control_bgcolor}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{initial_option}	Specifies the option that is first displayed when the application loads. If nothing is entered the first option is displayed. {initial_option} must match one of {option_name} options.
{option_name},{celltext_spec}	A semicolon separated list of names and definitions. {option_name} is the text that will appear in the control box. {celltext_spec} defines what appears on cells. The format of {celltext_spec} is discussed next.

Celltext Specification {celltext_spec}

Description: A celltext definition can be either a single data item name, or a definition organized by lines.

Value: [<font_name>`][<font_style>]{data_def} OR [<font_name>`][<font_style>]{line1}[\\n[<font_name>`][<font_style>]{line2}...{lineN}]

<font_name>	Optional: For changing the font size of a line of cell text. Use any font defined by one of the tm.font.size.{font_name} parameters.
<font_style>	Optional: Surrounding text and/or attribute(s) with one of the tags below will format the cell text bold, italics, or underline. If a close tag is not used, the remaining line(s) will inherit the style specified. Use for bold Use for italics Use <u></u> for underline
{data_def}	Value: {attribute} OR {sigdigits({Value_Name}, #)} OR {int({Value_Name})}
{attribute}	Any attribute from "CSV_DataTypes.cells". NOTE that the {} braces ARE necessary here.

{Value_Name}	Any attribute from "CSV_DataTypes.cells". The {} braces are NOT necessary here. However, remember to enclose the sigdigits() or int() function in {} braces.
#	The number of significant digits to display.
{line}	{line} represents one or more attributes. {line} is described in detail below.
\\n	Each "\\n" specifies a line break, so each {line} will appear on a different line within the cell. Multiple "\\n"s in a row result in blank lines.

{line}
Description: {line} represents one or more attributes, with optional supporting text.
Value: [{static_text1}]{data_def1}[{static_text2}]...[{data_def2}[{static_text3}]]...

{static_text}	This is any static text which will appear in the given location in the line. Spaces are not lost, so they can be used to separate multiple items on a single line. This is also where characters such as $ or % can be prepended or appended to data values.
{data_def}	Value: {attribute} OR {sigdigits({Value_Name}, #)} OR {int({Value_Name})}
{attribute}	Any attribute from "CSV_DataTypes.cells". NOTE that the {} braces ARE necessary here.
{Value_Name}	Any attribute from "CSV_DataTypes.cells". The {} braces are NOT necessary here. However, remember to enclose the sigdigits() or int() function in {} braces.
#	The number of significant digits to display.

Example: `"celltext": "celltext|-|Cell Labels:|-|-| Sales Rep,{Sales_Rep}; Rep - Sales - Quota,size11`{Sales_Rep}\\nsize9`${sigdigits (Total_Sales_Q4,0)} Sales\\nsize9`{sigdigits(Quota_Percent,0)}% Quota; Rep - Manager,size11`{Sales_Rep}\\n<i>Mgr: {Sales_Manager}</i>; Rep - Quota,size11`{Sales_Rep}\\n${int(Quota_Q4)} Quota" ,`

6.8 – Adding filters, indicators, and sliders

Special note about filter, indicator, and slider parameters: Each filter/indicator/slider parameter must be appended with an integer. For instance: filter1, filter2, filter3, ... filterN; slider1, slider2, slider3, ... sliderN. In the code, filter/indicator/slider parameters do not need to be in numerical order. The order of the filter/indicator/slider parameters in the code will specify the order of the filters/indicator/sliders in the application.

filterN

Description: A filter searches on text-based data and shows only the cells that match the criteria.

Value: check|{control_type}|{position_info}|{label_info}|{check_flag}|{control_bgcolor}|{control_group}|{initial_option}|{filter_spec-select} OR {filter_spec-text}

<table>
<tr><td>{control_type}</td><td>Dictates the type of filter to be created:
1. list.multi - creates a multi-select list
2. combo - creates a single-select dropdown box
3. text - creates a text search box</td></tr>
<tr><td>{position_info}</td><td>{x},{y},{w},{h} – See the table below for values:
<table><tr><td>{x}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr><tr><td>{y}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr><tr><td>{w}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr><tr><td>{h}</td><td>When using list.multi, {h} specifies the number of items visible in the list at any time.*</td></tr></table>
*When using list.multi, if the value of {h} is greater than the number of unique options, {h} will be automatically adjusted to match the number of unique options. i.e. the multi-select filter height will automatically resize, up until the number of lines specified by {h}, to fit the number of unique options.</td></tr>
<tr><td>{label_info}</td><td>Specifies the text that will appear as the label of the filter. {label_info} can be any string. If a caret (^) is used within the {label_info} field, all characters after the caret will be ignored.</td></tr>
<tr><td>{check_flag}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr>
<tr><td>{control_bgcolor}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr>
<tr><td>{control_group}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr>
<tr><td>{initial_option}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr>
</table>

{filter_spec-select}	Use when {control_type} = *list.multi* or *combo* The value of {filter_spec-select} is: AUTO_ENUM_LIST,{attribute}
{filter_spec-text}	Use when {control_type} = *text* The value of {filter_spec-text} is: {option_name},{operation},{invert_flag},{value_name}, {attribute1},{attribute2}, … ,{attributeN} The following fields are not used in the current version of Flaremap JS. Any value may be entered, or they can be left blank: {option_name} {operation} {invert_flag} {value_name},
{attribute}	Any attribute from "CSV_DataTypes.cells" that has TEXT properties. When {control_type} = *list.multi* or *combo* the attribute must also have ENUM properties. This is the attribute(s) that will be filtered on.

Examples:

```
"filter1": "check|text|-|Search|-|-|-|-|-,-,-,-,Industry,Sector",
"filter2": "check|list.multi|,,,6|Industry|-|-|-|-
|AUTO_ENUM_LIST,Sector",
"filter3": "check|combo|-|Region|-|-|-|-|AUTO_ENUM_LIST,Sales_Region",
```

Special note about text search: The text search filters support the use of conditional operators for advanced searching. In addition, specific phrases can be matched by surrounding the phrase in double quotes (e.g. "phrase here"). The conditional operators supported are: and, or, not, -. "Not" and "-" have the same meaning. Conditional operators are case insensitive. If a conditional operator is used inside of a phrase (e.g. "this phrase or that phrase"), the conditional operator is ignored.

indicatorN

Description: An indicator highlights cells that match the criteria by placing a small indicator icon on each matching cell.

Value: check|{control_type}|{position_info}|{label_info}|{check_flag}|{control_bgcolor}|{icon_location}|{control_group}|{initial_option}|{indicator_spec-select} OR {indicator_spec-text}

{control_type}	Dictates the type of indicator to be created: 1. *list.multi* - creates a multi-select list 2. *combo* - creates a single-select dropdown box 3. *text* - creates a text search box
{position_info}	{x},{y},{w},{h} – See the table below for values:

<table>
<tr><td></td><td>
<table>
<tr><td>{x}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr>
<tr><td>{y}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr>
<tr><td>{w}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr>
<tr><td>{h}</td><td>When using list.multi, {h} specifies the number of items visible in the list at any time.*</td></tr>
</table>

*When using list.multi, if the value of {h} is greater than the number of unique options, {h} will be automatically adjusted to match the number of unique options. i.e. the multi-select indicator height will automatically resize, up until the number of lines specified by {h}, to fit the number of unique options.
</td></tr>
<tr><td>{label_info}</td><td>Specifies the text that will appear as the label of the indicator. {label_info} can be any string. If a caret (^) is used within the {label_info} field, all characters after the caret will be ignored.</td></tr>
<tr><td>{check_flag}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr>
<tr><td>{control_bgcolor}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr>
<tr><td>{icon_location}</td><td>Specifies the location of the indicator icon on the face of cell. Accepts any of the following two-letter values listed in italics below:
tl - top left
tr - top right
br - bottom right
bl - bottom left
ct – top center</td></tr>
<tr><td>{control_group}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr>
<tr><td>{initial_option}</td><td>Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.</td></tr>
<tr><td>{indicator_spec-select}</td><td>Use when {control_type} = *list.multi* or *combo*

The value of {indicator_spec-select} is:
AUTO_ENUM_LIST,{attribute},{icon_path}</td></tr>
<tr><td>{indicator_spec-text}</td><td>Use when {control_type} = *text*

The value of {indicator_spec-text} is:
{option_name},{operation},{invert_flag},{value_name},
{icon_path},{attribute1},{attribute2}, … ,{attributeN}
The following fields are not used in the current version of Flaremap JS. Any value may be entered, or they can be left blank:
{option_name}
{operation}</td></tr>
</table>

	{invert_flag} {value_name},
{attribute}	Any attribute from "CSV_DataTypes.cells" that has TEXT properties. When {control_type} = *list.multi* or *combo* the attribute must also have ENUM properties. This is the attribute(s) that will be searched.
{icon_path}	Specifies the location of the indicator icon image file.

Examples:

```
"indicator1": "check|text|-|Search Indicator|-|-|tr|-|-|-,-,-,-,
images/indicator_yellow.gif,Industry,Sector",
"indicator2": "check|list.multi|,,,6|Industry|-|-|br|-|-|
AUTO_ENUM_LIST,Sector,images/indicator_yellow.gif",
"indicator3": "check|combo|-|Region|-|-|bl|-|-|
AUTO_ENUM_LIST,Sales_Region, images/indicator_yellow.gif",
```

Special note about text search: The text search indicators support the use of conditional operators for advanced searching. In addition, specific phrases can be matched by surrounding the phrase in double quotes (e.g. "phrase here"). The conditional operators supported are: and, or, not, -. "Not" and "-" have the same meaning. Conditional operators are case insensitive. If a conditional operator is used inside of a phrase (e.g. "this phrase or that phrase"), the conditional operator is ignored.

sliderN

Description: A slider searches on numeric-based data and shows only the cells that match the criteria.

Value: {position_info}|{label_info}|{attribute}|{scale}[,{bar_grab_scale}]|{date_in},{date_out}|{min},{max}|{L},{R}|{fill}|{snapto}|{N/A setting}|{inset}|{sigDigits}|{barcolor},{labelcolor}|{loc,imagepath}|{subTicks}|{pollInterval}|{control_group}|{divider labels}|{fromX},{toX},{label_multiplier},{label_prefix},{label_suffix}

{position_info}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{label_info}	Specifies the text that will appear as the label of the filter. {label_info} can be any string. If a caret (^) is used within the {label_info} field, all characters after the caret will be ignored.
{attribute}	Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. This is the attribute(s) that will be filtered on.
{scale}[,{bar_grab_scale}]	Specifies the scale applied to the range drawn on the slider control. A value of 1 (default value) will draw a linear range. A value of 2 will draw the first 25% of values over approximately the first 50% of the slider range. A value of 0 will be ignored.
{date_in},{date_out}	These entries specify how the slider is to handle dates, as opposed to ordinary numbers. This field can be left blank when date-formatted attributes are not being used.

	{date_in} defines how slider will parse dates which are used in {min},{max} parameters – this can be different than the format used in the database. {date_out} defines how slider will display the slider values as dates – this can be different than the format used in the database. See "Date Format" below this table for details on specifying the date format.
{min},{max}	Specifies the minimum and maximum values on the slider, defining its functional range. Values may be either numbers or the keywords "_MIN" or "_MAX". A number may be used in combination with a keyword. If the _MIN keyword is used, the slider's minimum range value will automatically correspond to the minimum value from the data set. If the _MAX keyword is used, the slider's maximum range value will automatically correspond to the maximum value from the data set.
{L},{R}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{fill}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{snapto}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{N/A setting}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{inset}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{sigDigits}	Specifies the number of decimal places that the slider will display. Can be any positive integer.
{barcolor},{labelcolor}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{loc,imagepath}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{subTicks}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{pollInterval}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{control_group}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.

{divider labels}	Specifies the number of subticks displayed. This field cannot be left blank. See the table below for more details: `,;` → 1 `,;,;` → 3 `,;,;,;` → 5 `Nx[,;]` → Each additional `[,;]` will create 2 more subticks
{fromX}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{toX}	Not used in current version of Flaremap JS. Any value may be entered, or this field can be left blank.
{label_multiplier}	Specifies a value to multiply the actual slider value by for display (for percentages, use 100: 0.24 becomes 24)
{label_prefix}	Specifies a prefix to prepend to the value.
{label_suffix}	Specifies a suffix to append to the value.

Date Format

Description: Flaremap JS supports a wide range of date/time formats. To specify the date/time format use a pattern string made up of the following symbols.

Symbol	Meaning	Presentation	Example
G	era designator	Text	AD
y	year	Number	1996
M	month in year	Text & Number	July & 07
d	day in month	Number	10
h	hour in am/pm (1~12)	Number	12
H	hour in day (0~23)	Number	0
m	minute in hour	Number	30
s	second in minute	Number	55
S	millisecond	Number	978
E	day in week	Text	Tuesday
D	day in year	Number	189
F	day of week in month	Number	2 (2nd Wed in July)
w	week in year	Number	27
W	week in month	Number	2
a	am/pm marker	Text	PM
k	hour in day (1~24)	Number	24

K	hour in am/pm (0~11)	Number	0

Note: The pattern of symbols determine the format. For example:

d	=	2
dd	=	02
M	=	8
MM	=	08
MMM	=	Aug
MMMM	=	August

Example: `"slider1": "-|Quota Percent Change|quota_chg|-|-|_MIN,_MAX|-|-|-|-|-|0|-|-|-|-|-|,;,|-,-,100,,%",`
`"slider2": "-|Total Sales|Sales_Q4|2|-|_MIN,_MAX|-|-|-|-|-|1|-|-|-|-|-|,;,;,;,;|-,-,.001,$,K",`
`"slider3": "-|Program Start Date|Start_Date|-|,MMM d yyyy|_MIN,_MAX|-|-|-|-|-|0|-|-|-|-|-|,;,;,;|-,-,1,,",`
`"slider4": "-| Program Start Date|Start_Date|-|M/d/yyyy,MMM d yyyy|1/1/2019,12/31/2019|-|-|-|-|-|0|-|-|-|-|-|,;,;,;|-,-,1,,",`

6.9 – Modifying look and feel options of the treemap field

tm.bg.color

Description: Sets the background color of the field (includes the color between groups).

Value: Any hexadecimal color value.

Example: `"tm.bg.color": "ffffff",`

tm.cell.gutter.size

Description: Sets the width of the gutter space between cells in the treemap.

Value: Any integer pixel value greater than or equal to 0.

Example: `"tm.cell.gutter.size": "0",`

tm.cell.indicator_track.loc

Description: Sets the location of the tracking icon on the cell.

Value: Any of the following:

- c - center (default)
- ct - center top
- tl - top left
- tr - top right
- br - bottom right
- bl - bottom left

Example: `"tm.cell.indicator_track.loc": "c",`

tm.cell.size.abs

Description: Sets size to map to absolute values of data values.

Value: true – maps size to absolute values
false – maps size to actual values (default)

Example: `"tm.cell.size.abs": "false",`

tm.cell.size.invert

Description: Inverts the size of cells: the smallest values will be the largest cells.

Value: true – inverts the values
false – does not invert the values (default)

Example: `"tm.cell.size.invert": "false",`

tm.cell.size.min

Description: Sets the minimum width or height of a cell in pixels. No cell will be narrower or shorter than this value.

Value: Any integer pixel value.

Example: `"tm.cell.size.min": "10",`

tm.cell.size.min.enforce

Description: Allows designers to toggle the enforcement of min cell size (tm.cell.size.min). A value of "true" will ensure that min cell size is enforced.

Value: t/f/true/false/1/0

Example: `"tm.cell.size.min.enforce": "true",`

tm.cell.size.ratio

Description: Sets an approximate ratio between the smallest and largest cells in the field.

Value: There are three options:
0 – this parameter is ignored
1 – all cells are the same size
2-50 – cells will scale between relatively similar in size (2) and significantly dissimilar in size (50)

Example: `"tm.cell.size.ratio": "8",`

tm.cell.type

Description: Adjusts the look of each cell.

Value: 0,3,4

0	No adjustments are made. The cell is a solid color.
1	Not used in current version of Flaremap JS.
2	Not used in current version of Flaremap JS.
3	Applies a gradient starting in the upper-left corner (lighter) and modulating to the lower-right corner (darker).
4 – default	Applies a gradient starting in the upper-left corner (darker) and modulating to the lower-right corner (lighter).

Example: `"tm.cell.type": "3",`

tm.color.logscale

Description: Sets all color options to use a logarithmic color scale.

Value: true/t/1 (forces all color options to use a logarithmic color scale) or false/f/0 (default, all color options use a linear color scale).

Example: `"tm.color.logscale": "false",`

tm.failure_message.text

Description: Sets the message that appears when a filter operation results in no cells to be displayed.

Value: Any string.

Example: `"tm.failure_message.text": "No data to display.",`

tm.font.size.{font_name}

Description: Defines the font size – where {font_name} is a unique name used internally to reference a font option.

Value: An integer font size.

Example: `"tm.font.size.medium": "11",`

tm.group.bg.color.default

Description: Sets the background color of the group.

Value: Any hexadecimal color value.

Example: `"tm.group.bg.color.default": "3a3a3a",`

tm.group.padding.default

Description: Sets the default padding size for all groups.

Value: Any integer pixel value greater than or equal to 0.

Example: `"tm.group.padding.default": "10",`

tm.group.gutter.size.default

Description: Sets the default size of the gutter between groups.

Value: Any integer pixel value greater than or equal to 0.

Example: `"tm.group.gutter.size.default": "1",`

tm.group.label.color.default

Description: Sets the color of the text used on group names.

Value: Any hexadecimal color value.

Example: `"tm.group.label.color.default": "ffffff",`

tm.group.label.font.default

Description: Sets the font size used on group titles.

Value: Any font defined by one of the tm.font.size.{font_name} parameters.

Example: `"tm.group.label.font.default": "size11",`

tm.group.label.heading_tags

Description: Adds HTML heading tags to group labels. The group label of the parent-level group will be surrounded by the <h1></h1> tag. The header tags of subsequent child groups will increment by one. The heading tags are inserted into the code and are not visible to the end users. However, the heading tags can be useful for people navigating the treemap with screen readers.

Value: "true" or "false"

Example: `"tm.group.label.heading_tags": "true",`

auto_layout

Description: When enabled, the application layout will be automatically modified to best fit the device type and screen size (e.g. a mobile layout will be used when a mobile device is detected). By default this parameter is set to "on".

Value: "on" or "off"

Example: `"auto_layout": "off",`

tm.live_url

Description: Enables the Live URL feature. See section 8.1 for more details.

Value: "on" or "off"

Example: `"tm.live_url": "off",`

6.10 – Localization options

In versions 6.3+ all static text used throughout the UI (e.g. the label "Group by" in the controls pane or the title "Filter" at the top of the filter pane) can be customized. These localization parameters allow designers to customize the language used in the UI.

Group, Size, Color

tm.strings.group.control.label

Description: Within the controls panel, specifies the label of the "Group by" control.

Value: Any string.

Example: `"tm.strings.group.control.label": "GRUPO DA",`

tm.strings.size.control.label

Description: Within the controls panel, specifies the label of the "Size by" control.

Value: Any string.

Example: `"tm.strings.size.control.label": "DIMENSIONE DA",`

tm.strings.color.control.label

Description: Within the controls panel, specifies the label of the "Color by" control.

Value: Any string.

Example: `"tm.strings.color.control.label": "COLORE DA",`

Cell Labels

tm.strings.group.celltext.label

Description: Within the controls panel, specifies the label of the "Cell Text" control.

Value: Any string.

Example: `"tm.strings.celltext.control.label": "TESTO CELLA",`

tm.strings.celltext.control.none_option

Description: Within the cell text drop down box, specifies the text of the "None" option.

Value: Any string.

Example: `"tm.strings.celltext.control.none_option": "none",`

tm.celltext.no_none_option

Description: Within the cell text drop down box, specifies whether to include the "None" option.

Value: "true" or "false" (default is false)

Example: `"tm.celltext.no_none_option": "true",`

Filters

tm.strings.filters.control.label

Description: Within the filters panel, specifies the text of the heading.

Value: Any string.

Example: `"tm.strings.filters.control.label": "FILTRI",`

tm.strings.filters.control.deselect_all

Description: Within the filters panel, specifies the text of the "Deselect All" link underneath text filters.

Value: Any string.

Example: `"tm.strings.filters.control.deselect_all": "Tout Déselectionner",`

Track / Untrack Cell Actions

tm.strings.mo.track_action

Description: Within the cell mouseover box action menu, specifies the verb used in the "Track this cell" action item – replaces the word "Track" with the specified word or phrase.

Value: Any string.

Example: `"tm.strings.mo.track_action": "Piste",`

tm.strings.mo.untrack_action

Description: Within the cell mouseover box action menu, specifies the verb used in the "Untrack this cell" action item – replaces the word "Untrack" with the specified word or phrase.

Value: Any string.

Example: `"tm.strings.mo.untrack_action": "Une Piste",`

Additional

tm.strings.treemap.status.label

Description: Within the controls panel, specifies the text used by the visible cell counter (e.g. Displaying X of Y).

Value: See example, both parameters in curly braces must be included.

Example: `"tm.strings.treemap.status.label": "MOSTRANDO: {visibleCells} di {totalCells}",`

tm.strings.group.label.blank_display_text

Description: Within the treemap, when a group ID is blank, specifies the text of the group name.

Value: Any string.

Example: `"tm.strings.group.label.blank_display_text": "n/a",`

6.11 – Priority Rendering Engine options

Available in versions 6.3 and later: The Priority Rendering Engine (PRE engine) dramatically improves overall rendering speed, especially for large datasets. Applications load up to 5x faster over previous versions, and can support tens of thousands of records. The PRE engine works by prioritizing visible cells during the rendering phase and utilizing a progressive loader to paint the cells onto the application. The progressive loader immediately displays the majority of visible cells – at which point the application is fully interactive; then smaller cells are progressively painted until finished. The progressive loader roughly utilizes the 80/20 rule: The cells that make up 80% of the visible area, but 20% of all records, are painted first.

The PRE engine is turned on by default and benefits applications of all sizes. There are additional parameters that can be added to further customize the PRE engine and improve performance as desired. These additional parameters are described below.

tm.perf.progressive_painting

Description: Turns progressive painting functionality on/off.

Value: true – turns on progressive painting functionality (default)
false – turns off progressive painting functionality

Example: `" tm.perf.progressive_painting":"true",`

tm.perf.min_gradient_cell_area

Description: Specifies the minimum cell size to be painted with a gradient overlay. Depending on the value used for the parameter "tm.cell.type", each cell may have a gradient (from light to dark) painted over it, instead of a flat color. Painting a gradient takes slightly more processing than a flat color. Therefore, for small cells, the usefulness of the gradient may not be worth the time spent painting the gradient. The larger the value of this parameter, the less cells that will be painted with a gradient. The value of this parameter represents the area of the smallest cell to still be painted with a gradient overlay.

Value: Any integer.

Example: `"tm.perf.min_gradient_cell_area":"10",`

tm.perf.min_painted_cell_area

Description: Specifies the minimum cell size to be painted. Cells that are not painted will appear dark grey, and will reveal their true color when hovered over. It takes a very small amount of processing time to paint a cell with the appropriate color. Therefore, it can improve processing time to skip the painting of thousands of tiny cells. The value of this parameter represents the area of the smallest cell to still be painted.

Value: Any integer.

Example: `"tm.perf.min_painted_cell_area":"2",`

tm.perf.stats.whitelist.min_max_avg_sum

Description: By default, if this parameter is not used, the min, max, average, and sum will be automatically calculated for each group and subgroup and for all attributes. If this parameter is used and the value is left blank, the calculations of min, max, average, and sum will not be performed on any attributes. If this parameter is used, only attributes listed will have their min, max, average, and sum calculated.

Calculating the min, max, average, and sum of a group of cells and for every attribute can take a noticeable amount of processing time when there are large numbers of cells, groups, and/or attributes. Therefore, excluding attributes that are not used in the group mouseover box can significantly improve overall processing time in applications with large numbers of cells, attributes, and/or groups.

Calculating the min, max, average, and sum takes approximately the same amount of processing time as calculating any one statistic independently – hence why these statistics are grouped together.

Value: Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. Multiple attributes are comma separated.

Example: `"tm.perf.stats.whitelist.min_max_avg_sum":"Quota_Q4,Sales_Q4",`

tm.perf.stats.whitelist.med_mod_var_std

Description: By default, if this parameter is not used, the median, mode, variance, and standard deviation will be automatically calculated for each group and subgroup and for all attributes. If this parameter is used and the value is left blank, the calculations of median, mode, variance, and standard deviation will not be performed on any attributes. If this parameter is used, only attributes listed will have their median, mode, variance, and standard deviation calculated.

Calculating the median, mode, variance, and standard deviation of a group of cells and for every attribute can take a noticeable amount of processing time when there are large numbers of cells, groups, and/or attributes. Therefore, excluding attributes that are not used in the group mouseover box can significantly improve overall processing time in applications with large numbers of cells, attributes, and/or groups.

Calculating the median, mode, variance, and standard deviation takes approximately the same amount of processing time as calculating any one statistic independently – hence why these statistics are grouped together.

Value: Any attribute from "CSV_DataTypes.cells" that has NUMBER properties. Multiple attributes are comma separated.

Example: `"tm.perf.stats.whitelist.med_mod_var_std":"Sales_Q4",`

Part 7: JavaScript Interface

The JavaScript Interface features a set of publicly callable JavaScript functions to retrieve the Event Stream and/or redraw the treemap according to a custom control/filter. The Event Stream is a real-time feed of all of the cell data for the currently hovered over (i.e. “active”) cell. The JavaScript Interface provides a platform for creating custom information displays, custom charts, custom treemap controls, and custom filters – which can leverage separate databases if desired.

Each JavaScript function will be described in more detail below.

- 7.1 – Cell Data
  - getCellDataForAttr(int cell_id, string attrName)
  - getCellData(int cell_id)
- 7.2 – Cell IDs
  - getAllCellIds()
  - getVisibleCellIds()
  - getTrackedCellIds()
  - getMouseOverCellId()
- 7.3 – Cell Actions
  - trackVisibleCells()
  - untrackVisibleCells()
- 7.4 – Controls Interface
  - applyControlSpec(String sXML)
  - removeFilterGroup(String filterGroup)
  - removeCelltext()
- 7.5 – Treemap Status
  - subscribe('cellHover', function(cellData))
  - subscribe('treemapInitialDraw', function())
  - subscribe('treemapDraw', function())

7.1 – Cell Data

getCellDataForAttr(int cell_id, string attrName)

This method returns a string containing the value of attrName for the given cell_id. This is useful when the developer wants to read only certain attributes of a cell and not all of the cell's data. Please note that all values are returned as a string.

Example:

```
var mouseOverCell = function() {
    try {

        var hc_js = com.visualaction.worldpop.services;

        // get info from event payload
        var cellId = parseInt(fieldApplet.getMouseOverCellId());

        if (cellId != null && cellId != undefined && !isNaN(cellId)) {

              alert("The data is: " + hc_js.getCellDataForAttr(cellId,
              "name"));

        }
    } catch(err) {
        alert(err.message);
    }
};
```

getCellData(int cell_id)

This method returns all data for a given cell_id in the form of name:value pairs where name is the attribute name and value is the value of that field.

Example:


```
{
"columns":["l1_str","l2_str","image","l3_str","dateStr","desc","dateStr2","in
dex","l1_enum","l2_enum","l3_enum","b1","v1","v2",
"v3","v4","pct","time","rating"],
"Data":["L1_I","L2_B","test.gif","L3_A","03/31/2004","Note that this is in
use.","20040331","256.0","8.0","1.0","0.0","1.0",
"0.95829","-0.34893","0.4857","630.0","0.7","0.49003","1.0"]
}
```

registerAggregateValue(string agg_value_name, function agg_function)

This method registers a javascript function that implements a custom aggregation value. The supplied function will take a set of data values associated with a group and compute (and return) a new single value that represents the desired aggregation.

Example:

```
// Create a function to compute an aggregate value
function myNewAggregateFunction(groupLevel, nameOfColumn, dataTypeOfColumn,
listOfValues) {
    // groupLevel is 1, 2, 3, ... depending on nesting
    // nameOfColumn is the column name, such as "Quota_Percent"
    // dataTypeOfColumn is the column type, such as NUMBER, TEXT, or DATE
    // listOfValues is an array of column values, such as [93, 99, 103, 109]
or ["John Doe", "Jane Doe"]

    // Loop through the listOfValues array to compute a custom aggregate
value.
    ...

    return theAggregateValue;
}

// Register the new aggregate function and provide a name for the new value
// Make sure to call registerAggregateValue() before calling onload()
flaremapServices.registerAggregateValue("nameOfNewAttribute",
myNewAggregateFunction);
```

updateRows(array rows)

This method updates the data with the provided rows. The first item in the rows array is an array of strings containing all the column names, and each subsequent array contains data values that correspond to the columns defined in the first row. All of the columns defined in CSV_DataTypes.cells must be present in the rows array. The provided data rows will replace the corresponding existing data row; if one is not present then a new data row is created. It's thus only necessary to include rows for which data has changed.

Example:

```
// Update two data rows

updateRows([

['Sales_Rep', 'Sales_Manager', 'Sales_Region',
'Total_Sales_Q4', 'Quota_Q4', 'Quota_Percent'],

['Brett Black', 'Albert Bachman III', 'Mountain
West',400000,500000, 80],

['Vincent Iorio', 'Albert Bachman III', 'Mountain West',
300000,400000, 75]

]);
```

removeRows(array rows)

This method removes the provided rows from the data. The first item in the rows array is an array of strings containing all the column names, and each subsequent array contains data values that correspond to the columns defined in the first row. All of the columns defined in CSV_DataTypes.cells must be present in the rows array. Any existing rows that correspond to the provided data rows will be removed.

Example:

```
// Remove two data rows
removeRows([
['Sales_Rep', 'Sales_Manager', 'Sales_Region', 'Total_Sales_Q4', 'Quota_Q4',
'Quota_Percent'],
['Brett Black', 'Albert Bachman III', 'Mountain West',400000,500000, 80],
['Vincent Iorio', 'Albert Bachman III', 'Mountain West', 300000,400000, 75]
]);
```

7.2 – Cell IDs

getAllCellIds()

This method returns an array of all cells Ids (both visible and invisible) in JSON format as shown below.

Example:

```
["0.0","1.0","2.0","3.0","4.0","5.0","6.0","7.0","8.0","9.0","10.0","11.0","1
2.0","13.0","14.0","15.0","16.0","17.0","18.0","19.0",
"20.0","21.0","22.0","23.0","24.0","25.0","26.0","27.0","28.0","29.0","30.0",
"31.0","32.0","33.0","34.0","35.0","36.0","37.0","38.0",
"39.0"]
```

getVisibleCellIds()

This method returns an array of all visible cells Ids in JSON format as shown below.

Example:

```
["32.0","33.0","34.0","35.0","39.0"]
```

getTrackedCellIds()

This method returns an array of tracked cells Ids in JSON format as shown below.

Example:

```
["14.0","19.0","25.0","33.0","35.0","39.0"]
```

getMouseOverCellId()

This method returns the cell_id of the treemap cell the mouse is currently hovered over.

Example:

```
"14.0"
```

7.3 – Cell Actions

trackVisibleCells()

This method is used to track all the cells visible on the treemap. This is useful when a user has changed or modified the support filters or when a user has zoomed into a group where only a subset of all the cells are being shown on the screen.

untrackVisibleCells()

This method is used to untrack all the visible cells on the treemap. This is useful when a user has changed or modified the support filters or when a user has zoomed into a group where only a subset of all the cells are being shown on the screen.

7.4 – Controls Interface

applyControlSpec(String sXML)

This method call is used trigger a redraw of the treemap according to the control spec that is passed in as argument.

Examples of XML control specs applications as follows.

Example XML control spec for **group**:

```
<?xml version='1.0' encoding='UTF-8'?>
<ControlSpec>
    <HierarchySpec>
        <GroupSpec level="1" attribute="Task" enum="true"><option
background_color="0000ff" gutter_size="3"/></GroupSpec>
    </HierarchySpec>
</ControlSpec>
```

Example XML control spec for **size**:

```
<?xml version='1.0' encoding='UTF-8'?>
<ControlSpec>
    <SizeSpec attribute='Total_Alloc'/>
</ControlSpec>
```

Example XML control spec for **color**:

```
<?xml version='1.0' encoding='UTF-8'?>
<ControlSpec>
    <ColorSpec attribute='Total_Alloc_Percent'
color_range='Total_Alloc_PercentCLR'>
    </ColorSpec>
</ControlSpec>
```

Example XML control spec for **filter**:

```
<?xml version='1.0' encoding='UTF-8'?>
<ControlSpec>
    <FilterSpec type='filter'>
        <FilterDef filterGroup='filterControl1' id='id2'
attribute='Organization' operation='EQ'>
            <param value='E-Solutions'/>
        </FilterDef>
    </FilterSpec>
</ControlSpec>
```

Example XML control spec for **filter**:

```
<ControlSpec>
    <FilterGroupSpec name="fg1" type="filter" multiSelectOp="or">
        <FilterDef attribute="Sales_Region" operation="EQ"
        invert="false"><param value="Northeast"/></FilterDef>
        <FilterDef attribute="Sales_Region" operation="EQ"
        invert="false"><param value="Northwest"/></FilterDef>
    </FilterGroupSpec>
</ControlSpec>
```

Example XML control spec for **slider**:

```
<ControlSpec>
    <FilterSpec type='filter'>
        <FilterDef filterGroup='hc_slider1a' attribute='Total_Alloc_Percent'
operation='GTE'>
            <param value='78.12'/>
        </FilterDef>
    </FilterSpec>
    <FilterSpec type='filter'>
        <FilterDef filterGroup='hc_slider1b' attribute='Total_Alloc_Percent'
operation='LTE'>
            <param value='185.08'/>
        </FilterDef>
    </FilterSpec>
</ControlSpec>
```

Example XML control spec for **cell text**:

```
<ControlSpec>
    <CelltextSpec value="Project Manager: {Title}\nTotal Emp Alloc:
{sigdigits(Total_Alloc_Percent,0)}%\nCurrent Alloc:
     {sigdigits(Current_Alloc,0)}Hrs\nTotal Hrs Alloc:
{sigdigits(Total_Alloc,0)}Hrs\n" />
</ControlSpec>
```

removeFilterGroup(String filterGroup)

This method call is used to remove the application of the given filter group, passed in as argument **filterGroup** to the call.

removeCelltext()

This method disables the visibility of text on the cells.

7.5 – Treemap Status

subscribe('cellHover', function(cellData))

This method indicates when a cell is hovered over such that the mouse over box is displayed.

Example:

```
// Cell hover event
flaremapServices.subscribe('cellHover', function(cellData) {
    console.log('Cell hover event!', cellData);
});
```

subscribe('treemapInitialDraw', function())

This method indicates when the treemap field is drawn for the first time.

Example:

```
// Initial treemap drawing event
flaremapServices.subscribe('treemapInitialDraw', function() {
    console.log('Initial treemap draw event!');
});
```

subscribe('treemapDraw', function())

This method indicates when the treemap field is re-drawn.

Example:

```
// Every treemap drawing event
flaremapServices.subscribe('treemapDraw', function() {
    console.log('Another treemap draw event!');
});
```

Part 8: URL features – Live URL, URL Pre-filter, and URL Filter

Starting with version 6.2, the Flaremap JS URL has three new features:

1. The Flaremap JS URL can be optionally enabled to instantaneously update every time a control and/or filter is changed. This Live URL allows users to share an exact view with other users and/or reference a custom configuration simply by clicking on a URL.

2. The Flaremap JS URL can be used as a clever mechanism for pre-filtering data before the data is loaded into the application. This URL pre-filter allows users to focus on a specific part of a dataset without having to first apply filters, and/or to analyze a slice of a large database without having to understand or perform complicated database transformations.

3. The Flaremap JS URL can be used for creating impromptu filters that are not defined in the application. This URL filter allows users to quickly and easily create new filters by simply updating the URL anchor. Important to note, URL filters act like native filters – i.e. URL filters are applied after the data is loaded into the application. For instance, the application will prepare the min/max values of a slider with the original dataset in mind – before applying any URL filters.

See below for details.

8.1 – Live URL

The Flaremap JS URL can be optionally enabled to instantaneously update every time a control and/or filter is changed. This Live URL allows users to share an exact view with other users and/or reference a custom configuration simply by clicking on a URL.

To enable Live URL, add the following parameter to the config file and set its value to "on".

```
"tm.live_url": "on",
```

Upon loading an application with the above parameter turned on, the application's URL will automatically show the current group, size, color, and cell text option. If a cell has been selected (clicked), the URL will also show that cell. Changing a control, selecting a filter, or selecting a cell will automatically update the URL.

8.2 – URL Pre-filter

The Flaremap JS URL can be used as a clever mechanism for pre-filtering data before the data is loaded into the application. This URL pre-filter allows users to focus on a specific part of a dataset without having to first apply filters, and/or to analyze a slice of a large database without having to understand or perform complicated database transformations.

No parameter needs to be added to the config file to use this feature.

To use a URL pre-filter, simply use the following format to append the base URL of the application:

{base_url}#tm.prefilter1[.{orGroup}]={filter_spec1}&tm.prefilterN[.{orGroup}]={filter_specN}

Note: By default, multiple pre-filters have an AND relationship; i.e. only results that match all of the pre-filter rules will be returned.

{base_url}	The core URL of the application. For example: http://www.company.com/flaremap/application.html
{orGroup}	Optional. When used, pre-filters with matching {orGroup}s will have an OR relationship – i.e. results that match any of the pre-filters with common {orGroup}s will be returned. The value of {orGroup} can be any unique string.
{filter_spec}	Specifies the pre-filter to apply. See {filter_spec} table below for format.

{filter_spec}
Value: {value_name}|{operation}|{operation_values}

{filter_spec} item	Value	Explanation
{value_name}	Any attribute from "CSV_DataTypes.cells"	Specifies the attribute that will be used to pre-filter the data.
{operation}	See table below for list of operations	Specifies the operation used to define group membership
{operation_values}	See table below for how different operations require different values	Specifies what values the operation uses to determine group membership

Group operations and values

Description	{operation}	{operation_values}
equals	EQ	Any number or string. Will only return results that exactly match the value. Use HASTEXT for more granular text searching.
not equals	NEQ	Any number or string. Will only return results that exactly match the value. Use HASTEXT for more granular text searching.
less than	LT	Any number

less than or equals	LTE	Any number
greater than	GT	Any number
greater than or equals	GTE	Any number
set membership (in a set)	SET	Any pipe separated list of numbers or strings. Will only return results that exactly match the value. Use HASTEXT for more granular text searching. Values within SET have an OR relationship.
from (inclusive)... to (inclusive)	[]	Any two pipe separated numbers (the second should be larger than the first)
from (not inclusive)... to (inclusive)	(]	Any two pipe separated numbers (the second should be larger than the first)
from (inclusive)... to (not inclusive)	[)	Any two pipe separated numbers (the second should be larger than the first)
from (not inclusive)... to (not inclusive)	()	Any two pipe separated numbers (the second should be larger than the first)
text search	HASTEXT	Any pipe separated list of numbers or strings. Will return any results that match the value, including partial matches. For example "Dan" will return: "Dan", "Daniel", and "Dan Smith".
adding an exclamation point immediately before any of the operations above will negate the filter rule	!	n/a

Examples:

http://www.company.com/flaremap/application.html#tm.prefilter1=Sales_Region|EQ|southeast

http://www.company.com/flaremap/application.html#tm.prefilter1=Sales_Region|EQ|southeast&tm.prefilter2=Quota_Percent|GTE|100

http://www.company.com/flaremap/application.html#tm.prefilter1=Sales_Region|HASTEXT|north|south

http://www.company.com/flaremap/application.html#tm.prefilter1.groupA=Sales_Region|EQ|southeast&tm.prefilter2.groupA=Sales_Manager|EQ|Jennifer Artz

8.3 – URL Filter

The Flaremap JS URL can be used for creating impromptu filters that are not defined in the application. This URL filter allows users to quickly and easily create new filters by simply updating the URL anchor. Important to note, URL filters act like native filters – i.e. URL filters are applied after the data is loaded into the application. For instance, the application will prepare the min/max values of a slider with the original dataset in mind – before applying any URL filters.

No parameter needs to be added to the config file to use this feature.

To use a URL filter, simply use the following format to append the base URL of the application (URL filters follow the same {filter_spec} as pre-filters):

{base_url}#urlfilter1[.{orGroup}]={filter_spec1}&urlfilterN[.{orGroup}]={filter_specN}

Note: By default, multiple URL filters have an AND relationship; i.e. only results that match all of the URL filter rules will be returned.

{base_url}	The core URL of the application. For example: http://www.company.com/flaremap/application.html
{orGroup}	Optional. When used, URL filters with matching {orGroup}s will have an OR relationship – i.e. results that match any of the URL filters with common {orGroup}s will be returned. The value of {orGroup} can be any unique string.
{filter_spec}	Specifies the URL filter to apply. See {filter_spec} table below for format.

{filter_spec}
Value: {value_name}|{operation}|{operation_values}

{filter_spec} item	Value	Explanation
{value_name}	Any attribute from "CSV_DataTypes.cells"	Specifies the attribute that will be used to filter the data.
{operation}	See table below for list of operations	Specifies the operation used to define group membership
{operation_values}	See table below for how different operations require different values	Specifies what values the operation uses to determine group membership

Group operations and values

Description	{operation}	{operation_values}
equals	EQ	Any number or string. Will only return results that exactly match the value. Use HASTEXT for more granular text searching.
not equals	NEQ	Any number or string. Will only return results that exactly match the value. Use HASTEXT for more granular text searching.

less than	LT	Any number
less than or equals	LTE	Any number
greater than	GT	Any number
greater than or equals	GTE	Any number
set membership (in a set)	SET	Any pipe separated list of numbers or strings. Will only return results that exactly match the value. Use HASTEXT for more granular text searching. Values within SET have an OR relationship.
from (inclusive)... to (inclusive)	[]	Any two pipe separated numbers (the second should be larger than the first)
from (not inclusive)... to (inclusive)	(]	Any two pipe separated numbers (the second should be larger than the first)
from (inclusive)... to (not inclusive)	[)	Any two pipe separated numbers (the second should be larger than the first)
from (not inclusive)... to (not inclusive)	()	Any two pipe separated numbers (the second should be larger than the first)
text search	HASTEXT	Any pipe separated list of numbers or strings. Will return any results that match the value, including partial matches. For example "Dan" will return: "Dan", "Daniel", and "Dan Smith".
adding an exclamation point immediately before any of the operations above will negate the filter rule	!	n/a

Examples:

http://www.company.com/flaremap/application.html#urlfilter1=Sales_Region|EQ|southeast

http://www.company.com/flaremap/application.html#urlfilter1=Sales_Region|EQ|southeast&urlfilter2=Quota_Percent|GTE|100

http://www.company.com/flaremap/application.html#urlfilter1=Sales_Region|HASTEXT|north|south

http://www.company.com/flaremap/application.html#urlfilter1.groupA=Sales_Region|EQ|southeast&urlfilter2.groupA=Sales_Manager|EQ|Jennifer Artz