**README**

**2024 Schedule Automation Documentation**

This document contains a set of custom-built functions to automate specific tasks at my job, such as creating and conditionally colouring a large number of Google Calendar events and converting field data to a usable format.

Read this document to understand added functionality to the document “2024 Schedule”.

**Apps Script Project “2024 Schedule Functions”:**

* [calendarFunctions.gs](#_f2yjad51jx4l)
  + [onOpen()](#_rdtt9vv0fg0d)
  + [deleteEvents()](#_mdv5k28diwqb)
  + [scheduleShifts()](#_alj38vqou4ah)
  + [colourShifts()](#_m4xq7hf9qsed)
* [dataSheetConfig.gs](#_ahlvlwlmol5d)
  + [splitRow()](#_5aruyh8bb6zc)
* [formatList.gs](#_nek6d45v3hy9)
* [eventColour.gs](#_8gqs4dbivjuk)

## calendarFunctions.gs

#### Global variables:

* sheetName = “Appscript Input” -> use getSheetByName(sheetName) to reference Appscript Input sheet
* calID -> stores ID of target calendar

#### onOpen()

* This is a simple trigger that adds the Calendar menu to the toolbar.
  + “Add Shifts to Calendar” points to function scheduleShifts()
  + “Colourize Calendar Events” points to function colourShifts()
  + “Clear Shifts from Calendar” points to deleteEvents()

#### deleteEvents()

* This functions deletes all events in the target calendar between the specified variables startTime and endTime
* Currently it is set to delete all events in the target calendar in 2024
* This function will be modified in the future to delete only events with titles containing “Registered”, “Drop In”, “Special Event”, “Sponsored”, “Outreach”, or “Giveaway”.
* This will also be modified with a warning message to avoid unwanted deletions.

#### scheduleShifts()

* This function reads data from the “Appscript Input” sheet, which queries all of the monthly schedule sheets in the 2024 Schedule document.

 var ss = SpreadsheetApp.getActiveSpreadsheet();

var sheet = ss.getSheetByName('Appscript Input');

var eventCal = CalendarApp.getCalendarById(calId);

var data = sheet.getRange("A1:F600").getValues().filter(row => row[2] !== null && row[2] !== '' && row[3] !== null && row[3] !== '');

* Variables ss and sheet refer to the current spreadsheet (which will always be “2024 Schedule”) and the sheet “Appscript Input”, respectively.
* var eventCal points to the target calendar
* var data reads all of the data in the source sheet, references each cell’s value, and filters out rows with null values in columns B and C (B = Start Date/Time, C = End Date/Time). The row[x] !== ‘’ syntax checks for invalid data types in these columns (‘’ is an empty string, these columns should only contain DateTime objects).
  + The “Data” variable captures an array containing sub-arrays of each row that doesn’t have null values in columns B and C. This makes it quick to iterate over the data using a for loop since we only need rows that specify times.

 for (x=0; x<data.length;x++){

var shift = data[x];

var title = shift[0];

var startTime = new Date(shift[1]);

var endTime = new Date(shift[2]);

var options = {

description: shift[4],

location: shift[3],

};

* This loop iterates over the rows in the variable “Data”.
* var shift captures current row in the iteration
* var title captures the first value of the list in var shift (located in column A of the sheet).
* var startTime captures the start time of each row (column B of the sheet)
* var endTime captures the end time of each row (column C of the sheet)
* var options contains description and location parameters, pointing to the values from column E and D, respectively.
  + Description and location are built in parameters for event objects in Apps Script

 eventCal.createEvent(title, startTime, endTime, options);

* This calls the built in function createEvent() for each element of var data. The parameters are built in - my variables are named after the built in parameters to make things easy.

#### colourShifts()

* This function calls a separate function, eventColour() on each event in the target calendar between specified dates. It colours calendar events conditionally based on their title.
* This function is necessary because Apps Script does not have a built in way to create and conditionally format events at the same time. It may be possible using the Google Calendar API, but I could not find a suitable solution.

 var eventCal = CalendarApp.getCalendarById(calId);

var events = eventCal.getEvents(new Date('January 1, 2024 00:00:00'), new Date('December 31, 2024 23:00:00'));

* var eventCal calls the target Calendar
* var events references all events in the target calendar between the specified start and end dates.

for (var i = 0; i < events.length; i++) {

eventColour(events[i]);

}

* This loop iterates over each of the events in the events variable and calls eventColour on the current iteration. This colourizes the events according to each event’s title.

## dataSheetConfig.gs

#### splitrow()

This function takes the Plant Tracker Form input, found in the “Input” tab and transforms it into usable database format in the “Data” tab.

 // Get the active (source) spreadsheet

var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();

var sheet = spreadsheet.getSheetByName('Input');

// Get target sheet

var targetSheetName = "Data";

var targetSheet = spreadsheet.getSheetByName(targetSheetName);

// Get timestamp of last processed form submission

var lastTimestamp = targetSheet.getRange("Z1").getValue();

* var sheet points to the source sheet by name
* var targetSheet points to the target sheet by name
* var lastTimeStamp stores the value of the last timestamp, pushed to cell Z1 later in the program

 // Function to fetch data range

var getDataRange = function(column) {

return sheet.getRange(2, column, 1000, 1).getValues().flat().filter(Boolean);

};

* var getDataRange defines an inline function on the column that returns the non-null values present in a specified column.
  + .filter(Boolean) returns only non-null values

 // Get relevant data values

var plantData = sheet.getRange('F2:F1001').getValues().filter(String);

var dateRange = getDataRange(2);

var locationRange = getDataRange(8);

var eventTypes = getDataRange(5);

var eventNames = getDataRange(7);

var eventStartTimes = getDataRange(3);

var eventEndTimes = getDataRange(4);

* These variables store the values of each relevant column in the source sheet. Names are self-explanatory.
* var plantData gets the values from column F of the sheet “Input”. This is the column that needs transformation.

 // Initialize empty arrays for storing output

var left = [];

var right = [];

var tempArray = [];

for (i=0; i < plantData.length; i++) {

left.push([dateRange[i], eventStartTimes[i], eventEndTimes[i], eventTypes[i], eventNames[i], locationRange[i]]);

for (j=0; j < plantData[i].length; j++) {

tempArray.push(plantData[i][j].split(/[,=]/).map(function(value) { return value.trim(); }));

}

}

for (i=0; i < tempArray.length; i++) {

l = formatList(tempArray[i])

right.push(l)

}

var result = [];

* var left and var right begin as empty lists to split the source data into two groups.
  + var left contains information about each event (date, start time, end time, location). This will be appended to each row that is produced from column F of the “Input” sheet.
  + var right stores a list of lists, each element containing information about a species, size of plant, and quantity planted. These will be output as rows in the target sheet.
* var tempArray is initialized to split and reformat the planting accomplishments from column F of the input sheet.
* The first for loop iterates over the length of the plantData variable, which will be the number of rows with meaningful text in the source sheet (“Input”). It stores the values from columns B, C, D, E, G, and H from the source sheet as a list of lists, with each element containing data from one row.
  + The nested for loop inside of it takes the values from column F of the source sheet, splits them into separate elements using the ‘,’ and ‘=’ operators, and trims trailing and leading white space from them.
  + It then stores the resulting values as separate lists in the tempArray variable.
* The third for loop iterates over all of the planting data elements in tempArray, then calls the formatList function on each element, which extracts the text within brackets, discards the brackets, and splits each element containing brackets (e.g. Trembling Aspen (S24)) into two new elements.
* right.push(l) moves all of the formatted values from l and stores them in the ‘right’ variable, which will now look like this: [[Trembling Aspen, S24, 48], [Balsam Poplar, S15, 60], etc.]
* The empty list var result is initialized to store the combinations of var left and var right.

 // Combine left and right arrays

for (var i = 0; i < left.length; i++) {

for (var j = 0; j < right[i].length; j++) {

var combinedList = left[i].concat(right[i][j]);

console.log(right[i][j]);

result.push(combinedList);

}

}

* The outer loop iterates over the length of var left
* The inner loop iterates over the length of one element (i) in var right
  + It then combines the current element from var left and the current element from var right into a temporary list.
  + console.log(right[i][j]) helps us keep track of which element is being added to the current element from var left
  + result.push(combinedList) adds the combined elements to the ‘result’ array.

 if (result.length > 0) {

targetSheet.getRange("Z1").setValue(new Date());

targetSheet.getRange(targetSheet.getLastRow() + 1, 1, result.length, result[0].length).setValues(result);

}

* targetSheet.getRange(“Z1”).setValue(new Date()); updates the value in cell Z1 in the spreadsheet to the current time. This lets us know when the last form submission occurred.
  + This will help us avoid duplicate submissions in the future, but I still have to add this functionality.
* The next line of code ending in .setValues() writes each element of the ‘result’ array to a new row in the spreadsheet.
  + getLastRow() + 1 finds the last row with text and selects the following row (+1).
  + the getRange() function has the following syntax:
  + getRange(row, column, optNumRows, optNumColumns)



* + So our code is specifying a range in the target sheet by specifying row = getLastRow() + 1, column as 1 (this specifies the starting column), optNumRows as result.length (each element of ‘result’ gets its own row), and optNumColumns as result[0].length (this specifies the width of the data being stored in the target spreadsheet, which is why we only need result[0] - result[0] specifies the first element of the result list.
  + Finally, .setValues(result) writes the data to the target sheet.

## formatList.gs

#### formatList(inputList)

var outputList = [];

* This initializes an empty list to store our formatted data

 // Iterate over the inputList in steps of 2 to group the elements

for (var i = 0; i < inputList.length; i += 2) {

var name = inputList[i].replace(/\s\*\(\S+\)\s\*/, ''); // Remove text inside parentheses

var size = inputList[i].match(/\(([^)]+)\)/)[1]; // Extract text inside parentheses

var quantity = parseInt(inputList[i + 1]);

// Create a subarray with the formatted values and push it to the outputList

outputList.push([name, size, quantity]);

}

return outputList;

}

* This function is designed to take the data from column F of the source sheet and output a list with three elements: [name, size, quantity]
  + It does this by performing a RegEx search in the name variable that finds parentheses and replaces the parentheses and any characters within them with empty strings (‘’)
  + It performs a RegEx match in the size variable to return only the text inside parentheses.
  + For quantity, it uses parseInt to find and return integers in every second element of the list

for (var i = 0; i < inputList.length; i += 2)

* You’ll notice that this for loop uses a step of i += 2
* This is because the original inputList will have the format [[species, (size)], quantity, [species, (size)], quantity]. Remember that list indexes begin with 0, so the first element has index 0. This means that every even index is a list with species and size information, and every odd index is an integer representing the quantity of plants.
* This is why we only need to iterate through every second element of the list to perform our RegEx functions, and then for integers we can specify:

var quantity = parseInt(inputList[i + 1]);

where i + 1 will give us an odd numbered index. This will make the function run faster.

## eventColour.gs

#### eventColour(event)

function eventColour (event) {

if (event.getTitle().toLowerCase().includes("registered")) {

event.setColor(CalendarApp.EventColor.PALE\_GREEN);

} else if (event.getTitle().toLowerCase().includes("drop in")) {

event.setColor(CalendarApp.EventColor.YELLOW);

} else if (event.getTitle().toLowerCase().includes("special event")) {

event.setColor(CalendarApp.EventColor.MAUVE);

} else if (event.getTitle().toLowerCase().includes("sponsored")) {

event.setColor(CalendarApp.EventColor.CYAN);

} else if (event.getTitle().toLowerCase().includes("outreach")) {

event.setColor(CalendarApp.EventColor.ORANGE);

} else if (event.getTitle().toLowerCase().includes("givewaway")) {

event.setColor(CalendarApp.EventColor.PALE\_RED);

}

}

* This function uses a conditional statement to set a calendar event’s colour based on its title.
* This is called by the colourShifts() function in the calendarFunctions.gs file, which requires us to schedule and then colour events using two separate functions.