AFFINITY WATER UK LABORATORY
Guidance Document: Commercial
Title: Automated Tracking Sheet
Author: David Golacis
Issue Date: 3rd February 2025

Guidance Document: Commercial

Automated Tracking Sheet

Contents

1.0 Overview

1.1 The Commercial department's Tracking Sheet is a mission-critical database which includes information about samples for external stakeholders. The complete details are provided in the Tracking Sheet guidance document.

1.2 This business improvement project successfully reduced the team's data entry requires and improved data quality for auditing purposes, saving 10+ staff from daily data entry requirements. The benefits of this project were the savings of 50 hours per month from automating admin transcription requirements, reducing the time spent executing the Commercial team's admin processes by 75%.

1.3 The data entry requirements for the project went as follows:

- New sample details (record ID, date of receival, customer ID, identity of who processed the order, resample status, weekend status, note of subcontracted requirements, description of work, and location of sample point)
- Transcription of any reported out-of-specification and non-conforming results
- Authorisation of on-site details
- Tracking of subcontracted results
- Updating of cancellation status

1.4 This was achieved by updating the Tracking Sheet at scheduled hours using Business Objects, Power Automate, and Excel Online. This document outlines how this system functions and manages the risks associated with data entry errors.

1.5 This project uses 7 flows to extract data from LIMS regarding new sample details, exceedances, non-conformances, authorised customer information, subcon receival, and cancellations.

1.6 The flows operate by monitoring David.Golacis' inbox for keywords in the email's title, sent from the Enquiries.Commercial address.

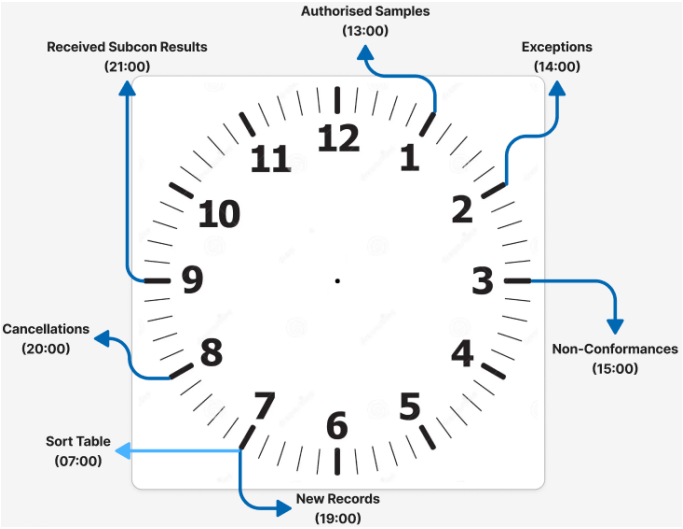
After meeting the conditions, the XLSX report attachment is saved to OneDrive for processing. This begins with converting the cell range into a table, extracting the data as JSON objects, and merging changes with the Tracking Sheet.

1.7 Process map for the order of operations:

1.8 Queries are received throughout the day to reduce conflict caused by multiple cells being edited in parallel.

The schedule of queries goes as follows:

Time of Day (24-hrs)	Process
07:00	Sort table
13:00	Authorised samples
14:00	Exceptions
15:00	Non-conformances
19:00	New records
20:00	Cancellations
21:00	Received subcon results



1.9 Each query contains data from the past 10 days that meet the specified conditions. This aids in risk management by ensuring redundancy within the system, allowing a flow to process the same data multiple times, which reduces the impact of errors that result in incomplete actions.

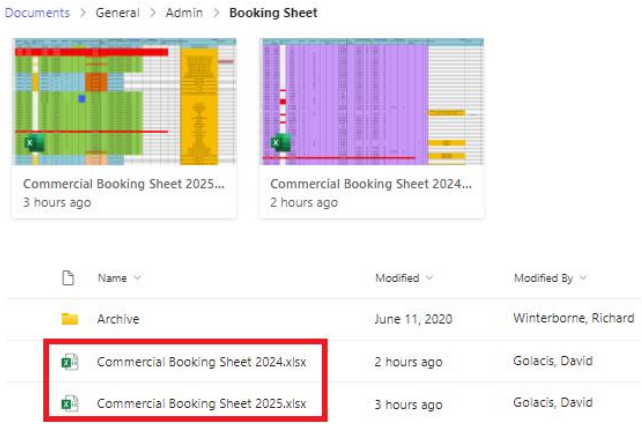
If the Tracking Sheet is prevented from updating due to IT services being down, or end users modifying the sheet during business hours, the system will refresh the sheet with the latest information over the rolling weekend. This ensures 99.9% uptime of updates week-over-week whilst minimizing internal server usage.

2.0 Maintenance

2.1 At the end of the calendar year, only the main sheets require adjustments to continue operation. This is because the year within the document's title defines which entries are allowed to be entered.

By storing the current and previous year's sheets together in a folder, both sheets can be processed in parallel and keep only relevant records from being written, enabling redundancy of data within the query and bypassing the requirement of filtering out mismatched year's data from the initial query.

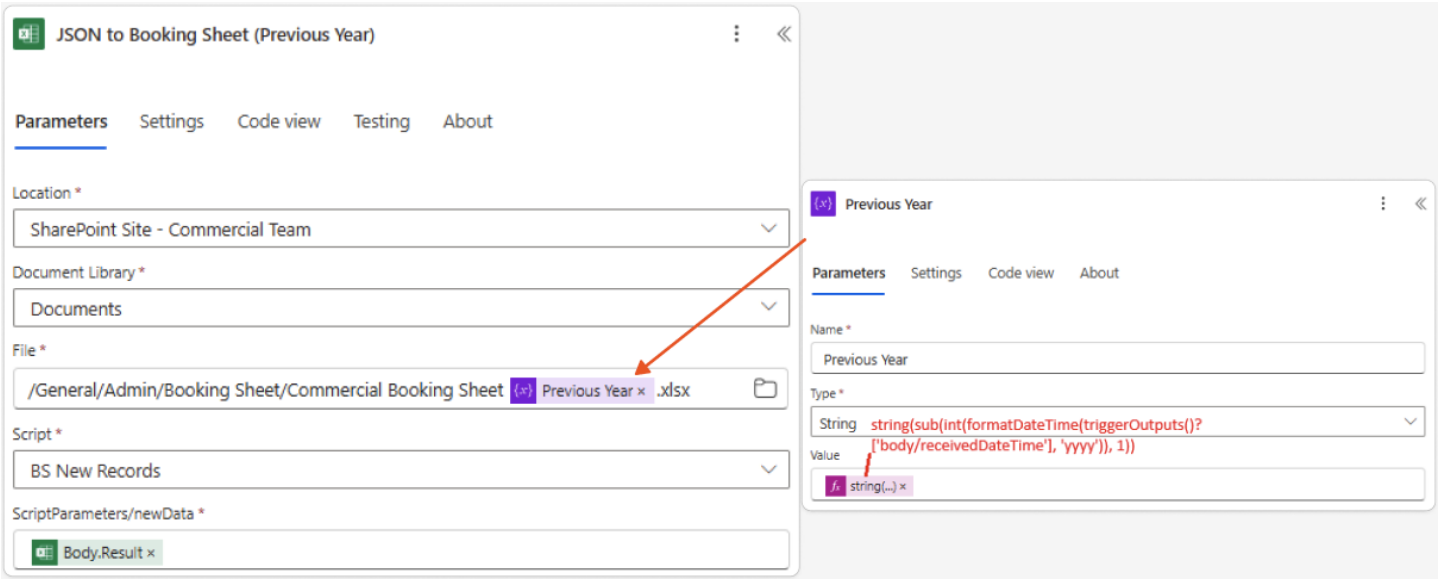
Teams: Commercial Team/ Documents/ General/ Admin/ Booking Sheet



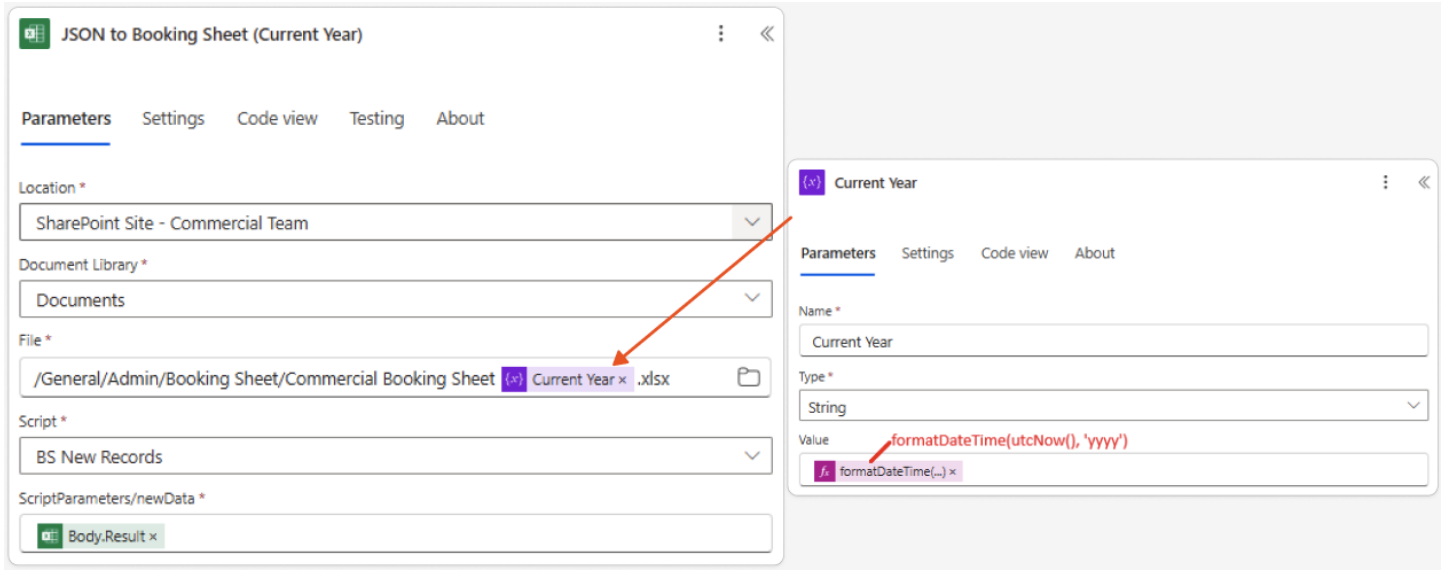
2.2 In December 2025, create a fresh copy of the Booking Sheet for the following year, 2026. Later in February, archive the 2025 sheet once all samples have been reported and invoiced.

2.3 Flows do not require any amendments due to the dynamic variables used to calculate the year in which the ending year of the current and previous years' sheets should be relative to the current UTC.

Previous year variable:



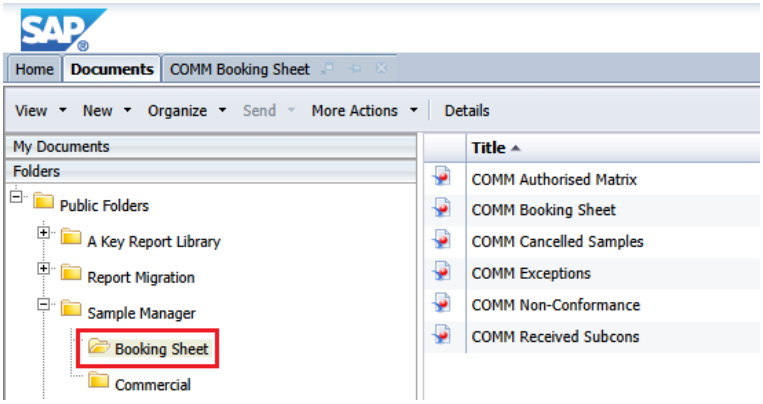
Current year variable:



3.0 Data Governance

3.1 All Business Objects files (SQL reports) are stored online at Affinity's BO Portal within this location:

Public Folders/ Sample Manager/ Booking Sheet



3.2 Queries, flows, and Office scripts are provided in the appendix.

4.0 Detailed Design

4.1 Design of reports

4.1.1 Business Objects generated and delivered scheduled queries from an Oracle database.

4.1.2 These queries shared safety features that restricted the data pulled from the cloud, reducing the server's memory usage and improving processing speed.

Techniques used to hone searches were:

- Limiting the date range used:

```
WHERE
  sample.recd_date >= TRUNC ( sysdate ) - 10
```

- Fetching samples with an associated customer ID:

```
WHERE
   LENGTH ( TRIM ( sample.customer_id ) ) > 0
```

- Specifying which sample and/ or result status was required:

```
WHERE
   result.status IN ( 'A' )
   AND sample.status NOT IN ( 'X', 'U' )
```

- Utilising parameter names:

```
WHERE
   test.analysis IN ( 'MATRIX' )
```

- Using CTEs to left-join additional information:

```
WITH subcon_tests AS (

SELECT
   DISTINCT test.sample


FROM test

INNER JOIN sample
   ON sample.id_numeric = test.sample


WHERE
   test.laboratory_id = 'SUB_CON'
   AND sample.recd_date >= TRUNC ( sysdate ) - 10
   AND LENGTH ( TRIM ( sample.customer_id ) ) >  0
)
```

```
LEFT JOIN subcon_tests
   ON subcon_tests.sample = sample.id_numeric
```

4.1.3 The selection followed a pattern of searching for an ID, a parameter of interest, and a date of when this change occurred. All samples that met the filtering criteria were passed on to the next step.

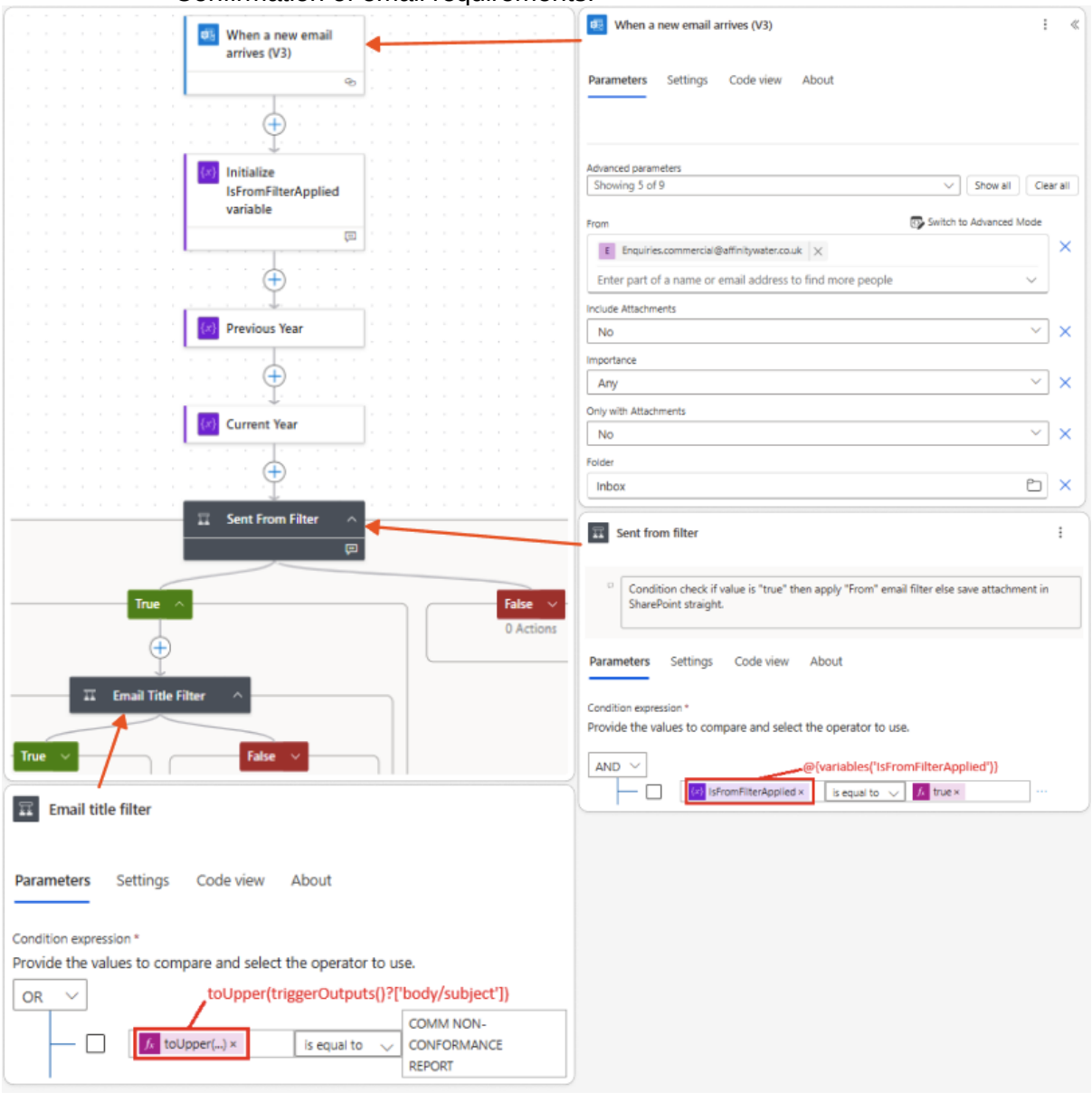
Sample No	Entered On	Parameter
2688527	14 Jan 2025	LEGIONELLA
2688527	14 Jan 2025	LEGIONELLA
2688528	14 Jan 2025	LEGIONELLA

4.2 Design of flows

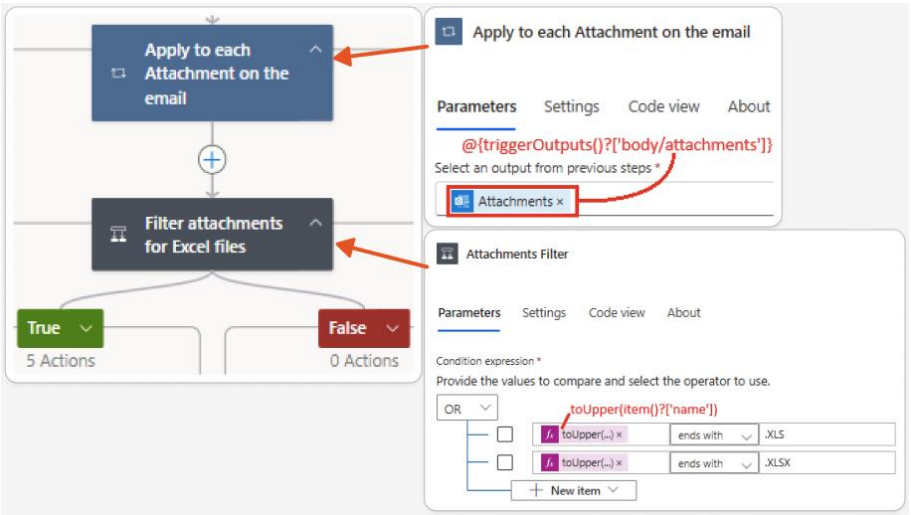
4.2.1 Once an email containing a report has been received, a Power Automate flow attempts to match the email's title to keywords. If a match is found, a series of steps take place to save the attached XLSX file for processing.

To ensure reliability, conditional filters were used to eliminate potential problems that could occur during an action. Considerations included were:

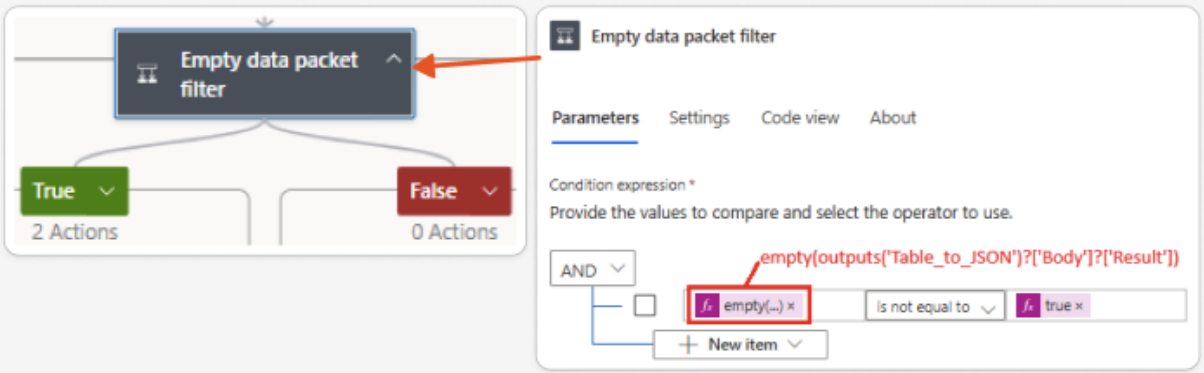
- Confirmation of email requirements:



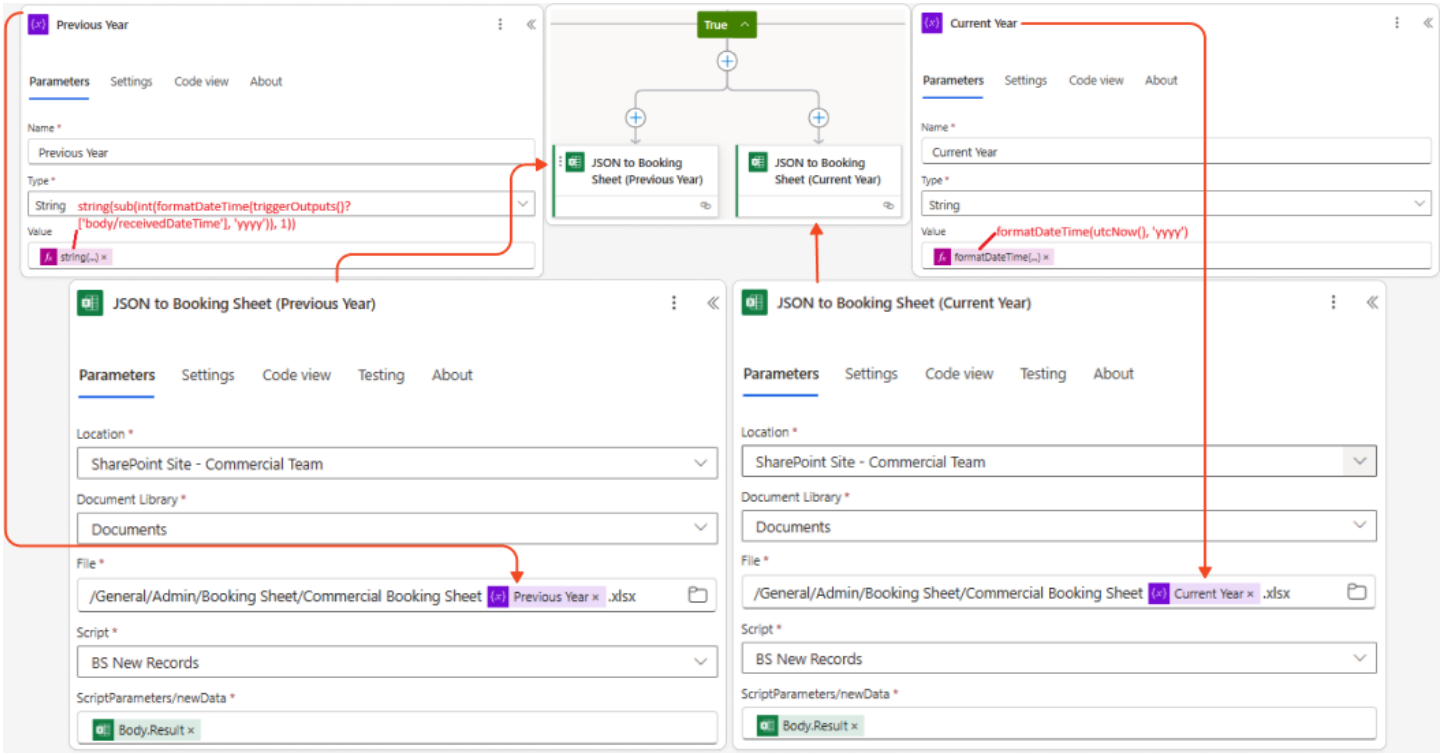
- Check for attachment(s):



- Validation of outgoing data:



To improve efficiency, both yearly spreadsheets were acted on using the data from a single query in parallel:



4.3 Design of scripts

4.3.1 Power Automate provides access to Excel Online for the use of Office Script, enabling functions to be written and executed on queries. To improve performance, efforts were directed at reducing the number of Excel API calls.

4.3.2 All flows shared one script to extract data from its initial report, which can be found in section 5.3.1. This function was designed to interact with Excel API once and pass on the contents as a string of JSON objects.

Objects were chosen for their key-value pairs, enabling future table amendments of the Tracking Sheet.

A check for data was placed at the end of the script to terminate impractical flows:

```
// If there's data, turn range into nested objects
if (rangeText.length > 1) {
   const outputData = stringToObjects(rangeText);
   return JSON.stringify(outputData);
}
// Otherwise, return an empty array to stop flow
else {
   return '';
};
```

This data was then fed to a purpose-built program to execute a singular function on the spreadsheet.

4.3.3 Beginning with the new records script, data from the Tracking Sheet was extracted in 1 API call and the year of the document in another. Then, records of the incorrect year were removed from the query, and further filtered by positive matches of binary search, leaving a query of exclusively new IDs. These items were then added to the end of the table in 1 API call, giving a total of 3 server requests for the entire report.

New records report:

Sample No	Date Received	Customer	Booked By	Resample	Weekend Work	Sub-Con	Analysis Description	Cancelled Sample	Description
2692090	14 Jan 2025	ASCOT_	GEALL			Y	RE_ASCOT1		580832, GRANDSTAND, CORE D, LAWN LEVEL, CLEANERS CUPBOARD, CWS
2692091	14 Jan 2025	ASCOT_	GEALL			Y	RE_ASCOT1		580830, GRANDSTAND, CORE C, LAWN LEVEL, CLEANERS CUPBOARD, CWS
2692092	14 Jan 2025	ASCOT_	GEALL			Y	RE_ASCOT1		584032, GRANDSTAND, CORE C, LEVEL 1, CLEANERS CUPBOARD, CWS

4.3.4 Next, the exceptions and non-conformance scripts were written to read the Tracking Sheet table in 1 API call, concatenating the new parameter/ text to any previous texts within the appropriate cell. Once all of the new, unique data had been strung together, the affected record line was replaced in 2 API calls, for a total of 3 server requests per unique record number in the report.

Exceptions report:

Sample No	Parameter	Result
2689477	Legionella species	100
2689736	Legionella species	2400

Non-conformance report:

Sample No	Entered Date	Text
2683778	03/01/2025	Giardia not complete due to external processing error
2683787	20/12/2024	Taste test removed.
2684296	20/12/2024	Could not test for Taste and Odour due to missing test bottle

4.3.4 Finally, the cancellations, subcon receival, and authorisations scripts were written to draw the Tracking Sheet data in 1 API call, match records using binary search and enter data if required. The affected rows were replaced in the table in 2 API calls, totalling 3 server requests per item in the report.

Cancellations report:

Sample No	Date Authorised
2665640	3 Jan 2025
2684715	6 Jan 2025
2688538	6 Jan 2025

Subcon Receival report:

Sample No	Entered On	Parameter
2688527	14 Jan 2025	LEGIONELLA
2688527	14 Jan 2025	LEGIONELLA
2688528	14 Jan 2025	LEGIONELLA

Authorisations report:

Sample No	Auth Date	Auth Initials
2691776	14 Jan 2025	GOLACISD
2691777	14 Jan 2025	GOLACISD
2691778	14 Jan 2025	GOLACISD

5.0 Appendix

5.1 Business Objects material

5.1.1 New Records

SQL query:

```
WITH subcon_tests AS (

SELECT
   DISTINCT test.sample


FROM test

INNER JOIN sample
   ON sample.id_numeric = test.sample


WHERE
   test.laboratory_id = 'SUB_CON'
   AND sample.recd_date >= TRUNC ( sysdate ) - 10
   AND LENGTH ( TRIM ( sample.customer_id ) ) > 0
)


SELECT
   TRIM ( sample.id_numeric ) AS sample_no,
   TRUNC ( sample.recd_date ) AS date_received,
   sample.customer_id,
   sample.login_by,
   sample.template_id,
   CASE WHEN
     INSTR ( UPPER ( sample.collected_from ), 'RESAMPLE' ) > 0
     THEN 'Y'
     ELSE ''
     END AS resample,
   CASE WHEN
     TO_CHAR ( sample.recd_date, 'Dy' ) = 'Sat'
     THEN 'Y'
        WHEN
        TO_CHAR ( sample.recd_date, 'Dy' ) = 'Sun'
        THEN 'Y'
     ELSE ''
     END AS weekend_work,
   CASE WHEN
     subcon_tests.sample > 0
     THEN 'Y'
     ELSE ''
     END AS subcon,
   CASE WHEN
     sample.status = 'X'
     THEN 'Y'
     ELSE ''
     END AS cancelled,
   sample.collected_from


FROM sample

LEFT JOIN subcon_tests
   ON subcon_tests.sample = sample.id_numeric


WHERE
   sample.recd_date >= TRUNC ( sysdate ) - 10
   AND LENGTH ( TRIM ( sample.customer_id ) ) > 0
```

Report:

Sample No	Date Received	Customer	Booked By	Resample	Weekend Work	Sub-Con	Analysis Description	Cancelled Sample	Description
2692090	14 Jan 2025	ASCOT_	GEALL			Y	RE_ASCOT1		580832, GRANDSTAND, CORE D, LAWN LEVEL, CLEANERS CUPBOARD, CWS
2692091	14 Jan 2025	ASCOT_	GEALL			Y	RE_ASCOT1		580830, GRANDSTAND, CORE C, LAWN LEVEL, CLEANERS CUPBOARD, CWS
2692092	14 Jan 2025	ASCOT_	GEALL			Y	RE_ASCOT1		584032, GRANDSTAND, CORE C, LEVEL 1, CLEANERS CUPBOARD, CWS

5.1.2 Exceptions

SQL query:

```
SELECT
   TRIM ( sample.id_numeric ),
   result.name,
   result.text


FROM sample

INNER JOIN test
   ON test.sample = sample.id_numeric

INNER JOIN result
   ON result.test_number = test.test_number


WHERE
   result.status NOT IN ( 'U', 'X' )
   AND result.result_type IN ( 'N', 'K' )
   AND result.out_of_range = 'T'
   AND LENGTH ( TRIM ( sample.customer_id ) ) > 0
   AND result.entered_on >= TRUNC ( sysdate ) - 10
```

Report:

Sample No	Parameter	Result
2689477	Legionella species	100
2689736	Legionella species	2400
2692260	Bromate as BrO3	461.2

5.1.3 Authorisations

SQL query:

```
SELECT
   TRIM ( sample.id_numeric ) AS sample_no,
   TRUNC ( result.date_authorised ) AS authorisation_date,
   TRIM ( result.authoriser ) AS authoriser

FROM
   result

INNER JOIN test
   ON test.test_number = result.test_number

INNER JOIN sample
   ON sample.id_numeric = test.sample
```

```
WHERE
   result.status IN ( 'A' )
   AND test.analysis IN ( 'MATRIX' )
   AND test.date_authorised >= ( TRUNC ( sysdate ) - 10 )
   AND LENGTH ( TRIM ( sample.customer_id ) ) > 0
   AND sample.status NOT IN ( 'X', 'U' )
```

Report:

Sample No	Auth Date	Auth Initials
2691776	14 Jan 2025	GOLACISD
2691777	14 Jan 2025	GOLACISD
2691778	14 Jan 2025	GOLACISD

5.1.4 Non-conformance

SQL query:

```
SELECT
   TRIM ( sample.id_numeric ) AS sample_no,
   TRUNC ( result.entered_on ),
   result.text


FROM sample

INNER JOIN test
   ON test.sample = sample.id_numeric

INNER JOIN result
   ON result.test_number = test.test_number


WHERE
   LENGTH ( TRIM ( sample.customer_id ) ) > 0
   AND result.entered_on >= TRUNC ( sysdate ) - 10
   AND test.analysis = 'NON_CONF_S'
   AND result.name IN ( 'Text comment 1', 'Text comment 2' )
   AND result.status IN ( 'A', 'C' )
```

Report:

Sample No	Entered Date	Text
2683778	03/01/2025	Giardia not complete due to external processing error
2683787	20/12/2024	Taste test removed.
2684296	20/12/2024	Could not test for Taste and Odour due to missing test bottle

5.1.5 Cancellations

SQL query:

```
SELECT
   TRIM ( sample.id_numeric ) AS sample_no,
   TRUNC ( sample.date_authorised ) AS cancelled_date

FROM
   test
```

```
INNER JOIN sample
   ON sample.id_numeric = test.sample

WHERE
   sample.status IN ( 'X' )
   AND (
      ( sample.date_authorised >= ( TRUNC ( sysdate ) - 10 ) )
      AND
      ( sample.date_authorised < TRUNC ( sysdate ) )
   )
   AND LENGTH ( TRIM ( sample.customer_id ) ) > 0
```

Report:

Sample No	Date Authorised
2665640	3 Jan 2025
2684715	6 Jan 2025
2688538	6 Jan 2025

5.1.6 Subcon Receival

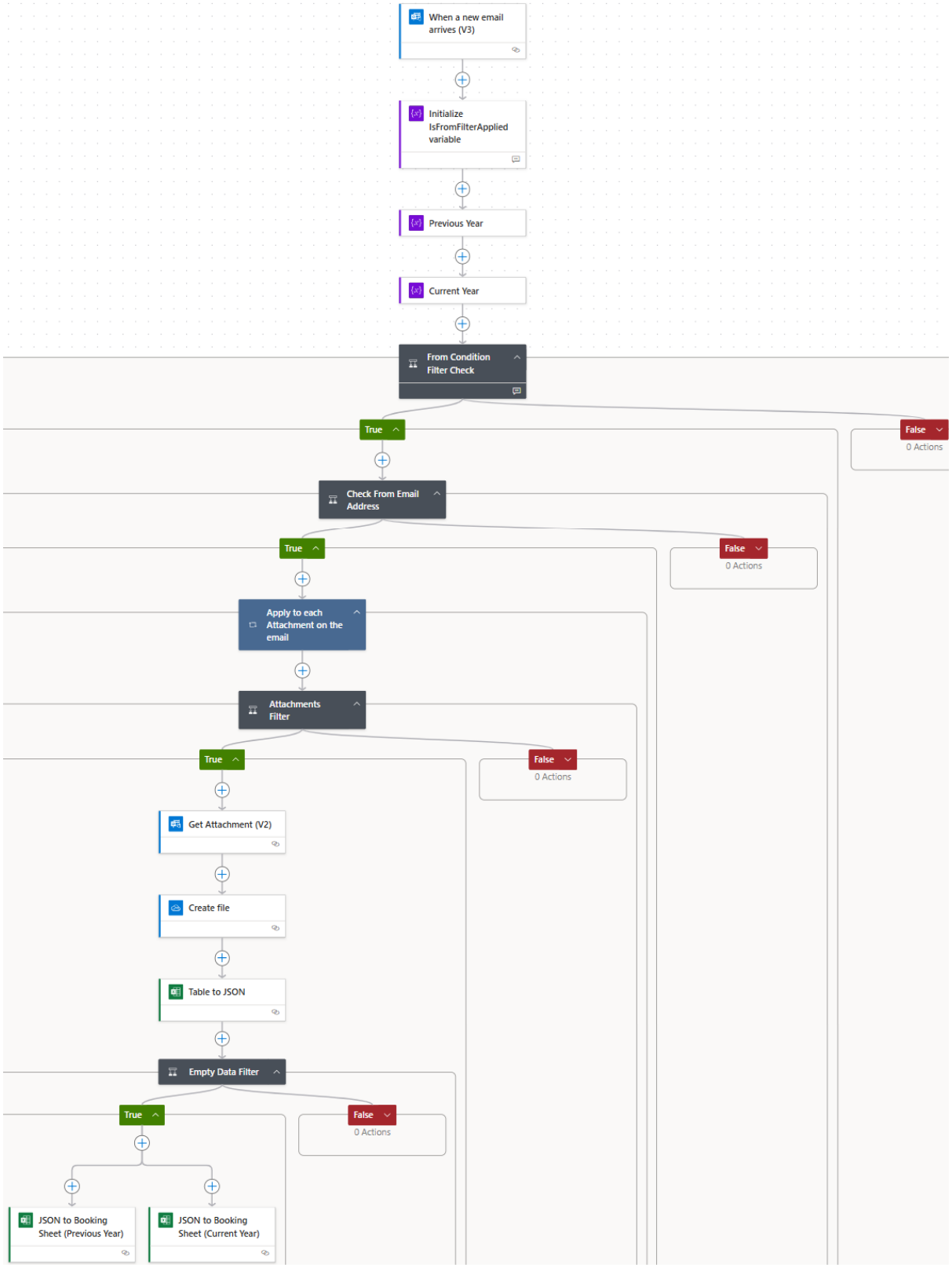
SQL query:

```
SELECT
   TRIM ( sample.id_numeric ) AS sample_no,
   result.entered_on,
   test.analysis

FROM sample

INNER JOIN test
   ON test.sample = sample.id_numeric

INNER JOIN result
   ON result.test_number = test.test_number

WHERE
   result.entered_on >= TRUNC ( sysdate ) - 10
   AND LENGTH ( TRIM ( sample.customer_id ) ) > 0
   AND test.laboratory_id = 'SUB_CON'
   AND TRIM ( result.text ) IS NOT NULL
```

Report:

Sample No	Entered On	Parameter
2688527	14 Jan 2025	LEGIONELLA
2688527	14 Jan 2025	LEGIONELLA
2688528	14 Jan 2025	LEGIONELLA

5.2 Power Automate flow



5.3 Office Script material

5.3.1 Table to Objects script:

```
// Function to extract data and output nested objects
function main(workbook: ExcelScript.Workbook): string {
   // Select 1st sheet in workbook
   const selectedSheet = workbook.getWorksheets()[0];
   // Get the working range as string
   const usedRange = selectedSheet.getUsedRange();
   let rangeText = usedRange.getTexts();
   //console.log(rangeText);

   // Cleaning string
   let length = rangeText.length;

   while (length--) {
      // Remove blank rows
      if (rangeText[length][1] === '') {
         rangeText.splice(length, 1);
         continue;
      };

      // Remove blank columns
      if (rangeText[length][-1] === '') {
         rangeText[length].splice(-1, 1);
         continue;
      };
   };

   // If there's data, turn range into nested objects
   if (rangeText.length > 1) {
      const outputData = stringToObjects(rangeText);
      //console.log(JSON.stringify(outputData));
      return JSON.stringify(outputData);
   }
   // Otherwise, return an empty array to stop flow
   else {
      return '';
   };
};

// Function to convert a 2D array string to nested objects
function stringToObjects(tableString: string[][]): string[][] {
   // Key: Value pairs
   var objectKeys: string[] = [];
   // Result
   var outputArray: string[][] = [];

   // for each element in array...
   for (var index = 0; index < tableString.length; index++) {
      // Use the 1st element of array as keys
      if (index === 0) {
         objectKeys = tableString[index];
         continue;
      };

      // Empty object to store key and values
      var tempObject: Object = {};

      // For the length of an array within nest...
      for (var element = 0; element < tableString[index].length; element++) {
         //console.log(objectKeys[element]);

         // Set the value of newObject with objectKeys at position (element);
         // Using values of tableString at position (index) at key (element)
         tempObject[objectKeys[element]] = tableString[index][element];
      };

      // Push object into output array
```

```
      outputArray.push(tempObject);
      continue;
   };
   return outputArray;
};
```

5.3.2 New Records script:

```
// Function to add new records to Commercial's tracking sheet
function main(workbook: ExcelScript.Workbook, newData: string): void {
   // Convert newData string to JSON objects
   const inputData: string[][] = JSON.parse(newData);

   // Find output table, turn it to string
   const table = workbook.getWorksheets()[0].getTables()[0];

   // Convert table to string
   const tableString = table.getRange().getTexts();

   // Convert table string to nested objects
   let tableObjects = stringToObjects(tableString);
   //console.log(tableObjects);

   // Removes objects from inputData which are already in tableObjects, or of incorrect
year
   const outputData = cleaner(workbook, tableObjects, inputData);
   //console.log(outputData);

   // Add new records to end of table
   const newObjects = addObjects(tableObjects, outputData);
   //console.log(newObjects);
   // Pasting final JSON block into end of the table
   table.addRows(-1, newObjects);

   // Sort table (for records which were received later than when they were created)
   return sort(table);
};

// Function to create objects with matching keys to output table and convert to nested
array
function addObjects(outputObjects: string[][], inputObjects: string[][]): string[][] {
   // Get keys from nested arrays of outputObjects
   const keys = Object.keys(outputObjects[0]);
   //console.log(keys);

   // Loop start value
   var a = 0;
   // Loop end value
   const b = inputObjects.length;
   // Result in nested array for Excel
   var outputArray: string[][] = [];

   while (a < b) {
      // Creating temp object to store all keys from outputObjects
      var tempObject: Object = {};

      for (var key of keys) {
         // Pasting Excel formulae in place, otherwise blank
         if (key === 'Working Days') {
            tempObject[key] = `=IF(ISBLANK([@[Cancelled Sample]]),IF(ISBLANK([@[Date
Received]]),"",IF(ISBLANK([@[Report Date]]),NETWORKDAYS([@[Date Received]],TODAY())-
1,NETWORKDAYS([@[Date Received]],[@[Report Date]])-1)),"")`;
         } else if (key === 'Total Days') {
            tempObject[key] = `=IF(ISBLANK([@[Cancelled Sample]]), IF(ISBLANK([@[Date
Received]]), "", IF(ISBLANK([@[Report Date]]), DAYS(TODAY(),[@[Date Received]]),
DAYS([@[Report Date]],[@[Date Received]]))),"")`;
         } else {
            tempObject[key] = '';
```

```
         }
         };
         //console.log(tempObject);

         // Add values from current inputObjects' item to tempObject, matched by key
         tempObject = {...tempObject, ...inputObjects[a]};
         //console.log(tempObject);

         // Temp array to store values
         var outputRow: string[] = [];

         // Extracting values from each key
         for (var key of keys) {
            //console.log(tempObject[key]);
            outputRow.push(tempObject[key]);
         };
         //console.log(outputRow);

         // Nesting array
         outputArray.push(outputRow);

         // Queue the next object in inputObjects
         a++;
   };
   //console.log(outputArray);
   return outputArray;
};

// Function to clean input nested objects of irrelevant year's records and of duplicate
records of output table
function cleaner(workbook: ExcelScript.Workbook, outputObjects: string[][],
inputObjects: string[][]): string[][] {
   // Required year found in filename
   const documentTitle = workbook.getName();
   //console.log(documentTitle);

   // Extract year from filename
   const documentYear = documentTitle.replace(/\D+/, '').replace('.', '').replace(/\D+/,
'');
   //console.log(documentYear);

   // Length of loop
   let length = inputObjects.length;

   // For all elements of inputObjects...
   while (length--) {
      // Identify date of record
      var itemDate: string = inputObjects[length]['Date Received'];
      //console.log(itemDate);

      // Extract year from record
      var itemYear = itemDate.split(' ');
      //console.log(itemYear[2]);

      // Remove unmatched years
      if (itemYear[2] !== documentYear) {
         inputObjects.splice(length, 1);
         continue;
      };

      // Position of current record of inputObjects in outputObjects
      var position = binarySearch(outputObjects, 'Sample No',
inputObjects[length]['Sample No']);
      //console.log(inputObjects[length]['Sample No']);
      //console.log(position);

      // If value from inputObjects is present in outputObjects, remove the element from
inputObjects
      if (position !== -1) {
         //console.log(inputObjects[length]);
```

```
            inputObjects.splice(length, 1);
            continue;
        } else {
            continue;
        };
    };
    //console.log(inputObjects);
    return inputObjects;
};

// Function to find the index of the ID within nested objects
function binarySearch(nestedObjects: string[][], key: string, value: string): number {
    // Range of array
    var start = 0;
    var end = nestedObjects.length - 1;
    // How many iterations did it take to find the index
    //let turns = 0;

    // Iterate while start has not met end
    while (start <= end) {
        //turns++;
        // Find the mid index, rounding down
        var mid = Math.floor(start + ((end - start) / 2));

        // If element is present at mid index, return index value
        if (nestedObjects[mid][key] === value) {
            return mid;
        }

        // Else, re-find mid in the left or right half of the array
        else if (nestedObjects[mid][key] < value) {
            start = mid + 1;
        } else {
            end = mid - 1;
        };
    };
    // If not found, break and return -1
    return -1;
};

// Function to convert a 2D array string to nested objects
function stringToObjects(tableString: string[][]): string[][] {
    // Key: Value pairs
    var objectKeys: string[] = [];
    // Result
    var outputArray: string[][] = [];

    // for each element in array...
    for (var index = 0; index < tableString.length; index++) {
        // Use the 1st element of array as keys
        if (index === 0) {
            objectKeys = tableString[index];
            continue;
        };

        // Empty object to store key and values
        var tempObject: Object = {};

        // For the length of an array within nest...
        for (var element = 0; element < objectKeys.length; element++) {
            //console.log(objectKeys[element]);

            // Pasting Excel formulae in place, otherwise blank
            if (objectKeys[element] === 'Working Days') {
                tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled
Sample]]),IF(ISBLANK([@[Date Received]]),"",IF(ISBLANK([@[Report
Date]]),NETWORKDAYS([@[Date Received]],TODAY())-1,NETWORKDAYS([@[Date
Received]],[@[Report Date]])-1)),"")`;
            } else if (objectKeys[element] === 'Total Days') {
```

```
            tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled Sample]]),
IF(ISBLANK([@[Date Received]]), "", IF(ISBLANK([@[Report Date]]), DAYS(TODAY(),[@[Date
Received]]), DAYS([@[Report Date]],[@[Date Received]]))),"")`;
         } else {
            tempObject[objectKeys[element]] = tableString[index][element];
         };
      };
      // Push object into output array
      outputArray.push(tempObject);
      continue;
   };
   return outputArray;
};

// Function to sort table before binary search
function sort(table: ExcelScript.Table) {
   // Sort table by Sample No. column
   table.getSort().apply([{ key: 0, ascending: true }]);

   // Copy table with new positions and paste in place
   const workingRange = table.getRangeBetweenHeaderAndTotal();
   return workingRange.copyFrom(workingRange, ExcelScript.RangeCopyType.all, false,
false);
};
```

5.3.3 Exceptions script:

```
// Function to add exceptions data to Commercial's tracking sheet
function main(workbook: ExcelScript.Workbook, newData: string): void {
   // Convert newData string to JSON objects
   const inputData: string[][] = JSON.parse(newData);

   // Find output table, turn it to string
   const table = workbook.getWorksheets()[0].getTables()[0];
   const tableString = table.getRange().getTexts();

   // Convert table string to nested objects
   let tableObjects = stringToObjects(tableString);
   //console.log(tableObjects);

   // Find unique ID's from inputData
   const uniqueIDs = separateIDs(inputData);
   //console.log(uniqueIDs);

   // For each unique ID...
   for (let targetID of uniqueIDs) {
      // Find the position of targetID in tableObjects
      const position = binarySearch(tableObjects, 'Sample No', targetID);
      //console.log(position);

      // If sample isn't present, go to the next entry
      if (position === -1) {
         continue;
      };
      // Placeholder for result
      let failureString = '';

      // If record's PCV Failure was blank
      if (tableObjects[position]['PCV Failure'] === '') {
         // Loop through inputData
         for (let a = 0; a < inputData.length; a++) {
            // If IDs of targetID and current item match
            if (inputData[a]['Sample No'] === targetID) {
               // If 1st failure, no comma
               if (failureString === '') {
                  failureString = inputData[a]['Parameter'];
               }
               // Otherwise, add comma
               else {
```

```
                    failureString = failureString.concat(`,
${inputData[a]['Parameter']}`);
                }
            };
        };
    };
    //console.log(failureString);

    // If record's PCV Failure was not blank
    if (tableObjects[position]['PCV Failure'] !== '') {
        // Split the PCV Failure
        const splitString = String(tableObjects[position]['PCV Failure']).split(', ');
        //console.log(splitString);

        // Loop through splitString
        for (let index = 0; index < splitString.length; index++) {
            // If 1st failure, no comma
            if (failureString === '') {
                failureString = splitString[index];
            }
            // Otherwise, add comma
            else {
            failureString = failureString.concat(`, ${splitString[index]}`);
            };
        };

        // Loop through inputData
        for (let index = 0; index < inputData.length; index++) {
            // If IDs of targetID and current item match
            if (inputData[index]['Sample No'] === targetID) {
                // Append new parameters
                failureString = failureString.concat(`,
${inputData[index]['Parameter']}`);
            };
        };

        // Split the failureString
        const splitFailures = String(tableObjects[position]['PCV Failure']).split(',
');
        //console.log(splitFailures);
        failureString = deDuplicate(splitFailures);
    };
    //console.log(failureString);
    //console.log(tableObjects[position]['PCV Failure']);

    // Check if existing string and new string are the same, skip if they are
    if (tableObjects[position]['PCV Failure'] === failureString) {
        continue;
    }
    // Otherwise, amend table
    else {
        // Set PCV Failure as the failureString
        tableObjects[position]['PCV Failure'] = failureString;
        // Convert to array of values
        const updateArray: string[] = Object.values(tableObjects[position]);
        //console.log(updateArray);

        // Delete previous item at index
        table.deleteRowsAt(position, 1);

        // Replace with updated string
        table.addRow(position, updateArray);
        continue;
    };
  };
  return;
};

// Function to add unique record dates to array
function deDuplicate(inputObjects: string[]): string {
```

```
   // Finding unique dates's from inputObjects
   const uniqueObjs: string[] = inputObjects.reduce((newArr, element) => {
      // Before element: x-1, and empty array
      //console.log(newArr);
      // Current element: x
      //console.log(element);

      // If the the current object's year is not in newArr, add it
      if (!newArr.some(item => item === element)) {
         // Add missing object to array
         newArr.push(element);
         //console.log(element);
      };
      // Return nested objects
      return newArr;
   }, []);
   //console.log(uniqueObjs);

   // Extracting the parameter values into array
   let uniqueIDs = '';
   for (let index = 0; index < uniqueObjs.length; index++) {
      //console.log(uniqueObjs[index]);

      // If uniqueIDs is blank, no comma
      if (uniqueIDs === '') {
         uniqueIDs = String(uniqueObjs[index]);
      }
      // Otherwise, add comma
      else {
         uniqueIDs = uniqueIDs.concat(`, ${String(uniqueObjs[index])}`);
      };
   };
   //console.log(uniqueIDs);
   return uniqueIDs;
};

// Function to find the index of the ID within nested objects
function binarySearch(nestedObjects: string[][], key: string, value: string): number {
   // Range of array
   var start = 0;
   var end = nestedObjects.length - 1;
   // How many iterations did it take to find the index
   //let turns = 0;

   // Iterate while start has not met end
   while (start <= end) {
      //turns++;
      // Find the mid index, rounding down
      var mid = Math.floor(start + ((end - start) / 2));

      // If element is present at mid index, return index value
      if (nestedObjects[mid][key] === value) {
         return mid;
      }
      // Else, re-find mid in the left or right half of the array
      else if (nestedObjects[mid][key] < value) {
         start = mid + 1;
      } else {
         end = mid - 1;
      };
   };
   // If not found, break and return -1
   return -1;
};

// Function to add unique record dates to array
function separateIDs(inputObjects: string[][]) {
   // Finding unique dates's from inputObjects
   const uniqueObjs = inputObjects.reduce((newArr, element) => {
      // Before element: x-1, and empty array
```

```
      //console.log(newArr);
      // Current element: x
      //console.log(element);

      // If the the current object's year is not in newArr, add it
      if (!newArr.some(item => item['Sample No'] === element['Sample No'])) {
         // Add missing object to array
         newArr.push(element);
         //console.log(element);
      };
      // Return nested objects
      return newArr;
   }, []);
   //console.log(uniqueObjs);

   // Extracting the ID values into array
   let uniqueIDs: string[] = [];
   for (let index = 0; index < uniqueObjs.length; index++) {
      //console.log(uniqueObjs[index]['Sample No']);
      uniqueIDs.push(uniqueObjs[index]['Sample No']);
   };
   //console.log(uniqueIDs);
   return uniqueIDs;
};

// Function to convert a 2D array string to nested objects
function stringToObjects(tableString: string[][]): string[][] {
   // Key: Value pairs
   var objectKeys: string[] = [];
   // Result
   var outputArray: string[][] = [];

   // for each element in array...
   for (var index = 0; index < tableString.length; index++) {
      // Use the 1st element of array as keys
      if (index === 0) {
         objectKeys = tableString[index];
         continue;
      };

      // Empty object to store key and values
      var tempObject: Object = {};

      // For the length of an array within nest...
      for (var element = 0; element < objectKeys.length; element++) {
         //console.log(objectKeys[element]);

         // Pasting Excel formulae in place, otherwise blank
         if (objectKeys[element] === 'Working Days') {
            tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled
Sample]]),IF(ISBLANK([@[Date Received]]),"",IF(ISBLANK([@[Report
Date]]),NETWORKDAYS([@[Date Received]],TODAY())-1,NETWORKDAYS([@[Date
Received]],[@[Report Date]])-1)),"")`;
         } else if (objectKeys[element] === 'Total Days') {
            tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled Sample]]),
IF(ISBLANK([@[Date Received]]), "", IF(ISBLANK([@[Report Date]]), DAYS(TODAY(),[@[Date
Received]]), DAYS([@[Report Date]],[@[Date Received]]))),"")`;
         } else {
            tempObject[objectKeys[element]] = tableString[index][element];
         };
      };
      // Push object into output array
      outputArray.push(tempObject);
      continue;
   };
   return outputArray;
};
```

5.3.4 Non-Conformance script:

```
// Function to add non-conformance data to Commercial's tracking sheet
function main(workbook: ExcelScript.Workbook, newData: string): void {
   // Convert newData string to JSON objects
   const inputData: string[][] = JSON.parse(newData);

   // Find output table, turn it to string
   const table = workbook.getWorksheets()[0].getTables()[0];
   const tableString = table.getRange().getTexts();

   // Convert table string to nested objects
   let tableObjects = stringToObjects(tableString);
   //console.log(tableObjects[310]);

   // Find unique ID's from inputData
   const uniqueIDs = separateIDs(inputData);
   //console.log(uniqueIDs);

   // For each unique ID...
   for (let targetID of uniqueIDs) {
      // Find the position of targetID in tableObjects
      const position = binarySearch(tableObjects, 'Sample No', targetID);
      //console.log(position);

      // If sample isn't present, go to the next entry
      if (position === -1) {
         continue;
      };

      // Placeholder for result
      let failureString = '';

      // If record's PCV Failure was blank
      if (tableObjects[position]['Non-conformance'] === '') {
         // Loop through inputData
         for (let a = 0; a < inputData.length; a++) {
            // If IDs of targetID and current item match
            if (inputData[a]['Sample No'] === targetID) {
               // If 1st failure, no comma
               if (failureString === '') {
                  failureString = inputData[a]['Text'];
               }
               // Otherwise, add comma
               else {
                  failureString = failureString.concat(`; ${inputData[a]['Text']}`);
               };
            };
         };
      };
      //console.log(failureString);

      // If record's Non-conformance was not blank
      if (tableObjects[position]['Non-conformance'] !== '') {
         // Split the Non-conformance
         const splitString = String(tableObjects[position]['Non-conformance']).split(';
');
         //console.log(splitString);

         // Loop through splitString
         for (let index = 0; index < splitString.length; index++) {
            // If 1st failure, no comma
            if (failureString === '') {
               failureString = splitString[index];
            }
            // Otherwise, add comma
            else {
               failureString = failureString.concat(`; ${splitString[index]}`);
            };
         };
```

```
         // Loop through inputData
         for (let index = 0; index < inputData.length; index++) {
            // If IDs of targetID and current item match
            if (inputData[index]['Sample No'] === targetID) {
               // Append new parameters
               failureString = failureString.concat(`; ${inputData[index]['Text']}`);
            };
         };

         // Split the failureString
         const splitFailures = String(tableObjects[position]['Non-
conformance']).split('; ');
         //console.log(splitFailures);
         failureString = deDuplicate(splitFailures);
      };
      //console.log(failureString);
      //console.log(tableObjects[position]['Non-conformance']);

      // Check if existing string and new string are the same, skip if they are
      if (tableObjects[position]['Non-conformance'] === failureString) {
         continue;
      }
      // Otherwise, amend table
      else {
         // Setting Non-conformance as failureString
         tableObjects[position]['Non-conformance'] = failureString;
         // Array of values
         const updateArray: string[] = Object.values(tableObjects[position]);

         // Deleting last column due to an error
         updateArray.pop();
         //console.log(updateArray);

         // Delete previous item at index
         table.deleteRowsAt(position, 1);

         // Replace with updated string
         table.addRow(position, updateArray);
         continue;
      };
   };
   return;
};

// Function to add unique record dates to array
function deDuplicate(inputObjects: string[]): string {
   // Finding unique dates's from inputObjects
   const uniqueObjs: string[] = inputObjects.reduce((newArr, element) => {
      // Before element: x-1, and empty array
      //console.log(newArr);
      // Current element: x
      //console.log(element);

      // If the the current object's year is not in newArr, add it
      if (!newArr.some(item => item === element)) {
         // Add missing object to array
         newArr.push(element);
         //console.log(element);
      };
      // Return nested objects
      return newArr;
   }, []);
   //console.log(uniqueObjs);

   // Extracting the parameter values into array
   let uniqueIDs = '';
   for (let index = 0; index < uniqueObjs.length; index++) {
      //console.log(uniqueObjs[index]);
```

```
      // If uniqueIDs is blank, no comma
      if (uniqueIDs === '') {
         uniqueIDs = String(uniqueObjs[index]);
      }
      // Otherwise, add comma
      else {
         uniqueIDs = uniqueIDs.concat(`, ${String(uniqueObjs[index])}`);
      };
   };
   //console.log(uniqueIDs);
   return uniqueIDs;
};

// Function to find the index of the ID within nested objects
function binarySearch(nestedObjects: string[][], key: string, value: string): number {
   // Range of array
   var start = 0;
   var end = nestedObjects.length - 1;
   // How many iterations did it take to find the index
   //let turns = 0;

   // Iterate while start has not met end
   while (start <= end) {
      //turns++;
      // Find the mid index, rounding down
      var mid = Math.floor(start + ((end - start) / 2));

      // If element is present at mid index, return index value
      if (nestedObjects[mid][key] === value) {
         return mid;
      }

      // Else, re-find mid in the left or right half of the array
      else if (nestedObjects[mid][key] < value) {
         start = mid + 1;
      } else {
         end = mid - 1;
      };
   };
   // If not found, break and return -1
   return -1;
};

// Function to add unique record dates to array
function separateIDs(inputObjects: string[][]) {
   // Finding unique dates's from inputObjects
   const uniqueObjs = inputObjects.reduce((newArr, element) => {
      // Before element: x-1, and empty array
      //console.log(newArr);
      // Current element: x
      //console.log(element);

      // If the the current object's year is not in newArr, add it
      if (!newArr.some(item => item['Sample No'] === element['Sample No'])) {
         // Add missing object to array
         newArr.push(element);
         //console.log(element);
      };
      // Return nested objects
      return newArr;
   }, []);
   //console.log(uniqueObjs);

   // Extracting the ID values into array
   let uniqueIDs: string[] = [];
   for (let index = 0; index < uniqueObjs.length; index++) {
      //console.log(uniqueObjs[index]['Sample No']);
      uniqueIDs.push(uniqueObjs[index]['Sample No']);
   };
   //console.log(uniqueIDs);
```

```
    return uniqueIDs;
};

// Function to convert a 2D array string to nested objects
function stringToObjects(tableString: string[][]): string[][] {
    // Key: Value pairs
    var objectKeys: string[] = [];
    // Result
    var outputArray: string[][] = [];

    // for each element in array...
    for (var index = 0; index < tableString.length; index++) {
        // Use the 1st element of array as keys
        if (index === 0) {
            objectKeys = tableString[index];
            continue;
        };

        // Empty object to store key and values
        var tempObject: Object = {};

        // For the length of an array within nest...
        for (var element = 0; element < objectKeys.length; element++) {
            //console.log(objectKeys[element]);

            // Pasting Excel formulae in place, otherwise blank
            if (objectKeys[element] === 'Working Days') {
                tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled
Sample]]),IF(ISBLANK([@[Date Received]]),"",IF(ISBLANK([@[Report
Date]]),NETWORKDAYS([@[Date Received]],TODAY())-1,NETWORKDAYS([@[Date
Received]],[@[Report Date]])-1)),"")`;
            } else if (objectKeys[element] === 'Total Days') {
                tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled Sample]]),
IF(ISBLANK([@[Date Received]]), "", IF(ISBLANK([@[Report Date]]), DAYS(TODAY(),[@[Date
Received]]), DAYS([@[Report Date]],[@[Date Received]]))),"")`;
            } else {
                tempObject[objectKeys[element]] = tableString[index][element];
            };
        };

        // Push object into output array
        outputArray.push(tempObject);
        continue;
    };
    return outputArray;
};
```

5.3.5 Authorisations script:

```
// Function to add authorisation data to Commercial's trackinging sheet
function main(workbook: ExcelScript.Workbook, newData: string): void {
    // Convert newData string to JSON objects
    const inputData: string[][] = JSON.parse(newData);

    // Find output table, turn it to string
    const table = workbook.getWorksheets()[0].getTables()[0];
    const tableString = table.getRange().getTexts();

    // Convert table string to nested objects
    let tableObjects = stringToObjects(tableString);
    //console.log(tableObjects);

    // For each entry of update table...
    for (let entry = 0; entry < inputData.length; entry++) {
        // Matching ID
        const targetID: string = inputData[entry]['Sample No'];
        // Parameter 1
        const date: string = inputData[entry]['Auth Date'];
        // Parameter 2
```

```
      const initials: string = inputData[entry]['Auth Initials'];
      //console.log(targetID);

      // Find the position of target ID in tableObjects
      const position = binarySearch(tableObjects, 'Sample No', targetID);
      //console.log(position);

      if (position === -1) {
         // If sample isn't present already, go to the next entry
         continue;
      };

      // Update the failure key of tableObjects with the new string
      if (tableObjects[position]['Auth Date'] === '') {
         tableObjects[position]['Auth Date'] = date;
         tableObjects[position]['Auth Initials'] = initials;
      } else {
         // If date is already present, go to the next entry
         continue;
      };

      // Array of values
      const updateArray: string[] = Object.values(tableObjects[position]);

      // Delete previous item at index
      table.deleteRowsAt(position, 1);
      // Replace with updated string
      table.addRow(position, updateArray);
      continue;
   };
   return;
};

// Function to find the index of the ID within nested objects
function binarySearch(nestedObjects: string[][], key: string, value: string): number {
   // Range of array
   var start = 0;
   var end = nestedObjects.length - 1;
   // How many iterations did it take to find the index
   //let turns = 0;

   // Iterate while start has not met end
   while (start <= end) {
   //turns++;
   // Find the mid index, rounding down
      var mid = Math.floor(start + ((end - start) / 2));

      // If element is present at mid index, return index value
      if (nestedObjects[mid][key] === value) {
         return mid;
      }
      // Else, re-find mid in the left or right half of the array
      else if (nestedObjects[mid][key] < value) {
         start = mid + 1;
      } else {
         end = mid - 1;
      };
   };
   // If not found, break and return -1
   return -1;
};

// Function to convert a 2D array string to nested objects
function stringToObjects(tableString: string[][]): string[][] {
   // Key: Value pairs
   var objectKeys: string[] = [];
   // Result
   var outputArray: string[][] = [];

   // for each element in array...
```

```
   for (var index = 0; index < tableString.length; index++) {
      // Use the 1st element of array as keys
      if (index === 0) {
         objectKeys = tableString[index];
         continue;
      };

      // Empty object to store key and values
      var tempObject: Object = {};

      // For the length of an array within nest...
      for (var element = 0; element < objectKeys.length; element++) {
         //console.log(objectKeys[element]);

         // Pasting Excel formulae in place, otherwise blank
         if (objectKeys[element] === 'Working Days') {
            tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled
Sample]]),IF(ISBLANK([@[Date Received]]),"",IF(ISBLANK([@[Report
Date]]),NETWORKDAYS([@[Date Received]],TODAY())-1,NETWORKDAYS([@[Date
Received]],[@[Report Date]])-1)),"")`;
         } else if (objectKeys[element] === 'Total Days') {
            tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled Sample]]),
IF(ISBLANK([@[Date Received]]), "", IF(ISBLANK([@[Report Date]]), DAYS(TODAY(),[@[Date
Received]]), DAYS([@[Report Date]],[@[Date Received]]))),"")`;
         } else {
            tempObject[objectKeys[element]] = tableString[index][element];
         };
      };

      // Push object into output array
      outputArray.push(tempObject);
      continue;
   };
   return outputArray;
};
```

5.3.6 Cancellations script:

```
// Function to add cancellation data to Commercial's tracking sheet
function main(workbook: ExcelScript.Workbook, newData: string): void {
   // Convert newData string to JSON objects
   const inputData: string[][] = JSON.parse(newData);

   // Find output table, turn it to string
   const table = workbook.getWorksheets()[0].getTables()[0];
   const tableString = table.getRange().getTexts();

   // Convert table string to nested objects
   let tableObjects = stringToObjects(tableString);
   //console.log(tableObjects);

   // For each row of update table...
   for (let entry = 0; entry < inputData.length; entry++) {
      // ID to look for
      const targetID: string = inputData[entry]['Sample No'];
      //console.log(targetID);

      // Index of targetID in output table
      const position: number = binarySearch(tableObjects, 'Sample No', targetID);
      //console.log(position);

      if (position === -1) {
         // If sample isn't present already, go to the next entry
         continue;
      };

      // Mark off at position
      if (tableObjects[position]['Cancelled Sample'] === '') {
         tableObjects[position]['Cancelled Sample'] = 'Y';
```

```
      } else {
         // If already complete, go to the next entry
         continue;
      };

      // Array of values
      const updateArray: string[] = Object.values(tableObjects[position]);

      // Delete previous item at index
      table.deleteRowsAt(position, 1);
      // Replace with updated string
      table.addRow(position, updateArray);
      continue;
   };
   return;
};

// Function to find the index of the ID within nested objects
function binarySearch(nestedObjects: string[][], key: string, value: string): number {
   // Range of array
   var start = 0;
   var end = nestedObjects.length - 1;
   // How many iterations did it take to find the index
   //let turns = 0;

   // Iterate while start has not met end
   while (start <= end) {
      //turns++;
      // Find the mid index, rounding down
      var mid = Math.floor(start + ((end - start) / 2));

      // If element is present at mid index, return index value
      if (nestedObjects[mid][key] === value) {
         return mid;
      }
      // Else, re-find mid in the left or right half of the array
      else if (nestedObjects[mid][key] < value) {
         start = mid + 1;
      } else {
         end = mid - 1;
      };
   };
   // If not found, break and return -1
   return -1;
};

// Function to convert a 2D array string to nested objects
function stringToObjects(tableString: string[][]): string[][] {
   // Key: Value pairs
   var objectKeys: string[] = [];
   // Result
   var outputArray: string[][] = [];

   // for each element in array...
   for (var index = 0; index < tableString.length; index++) {
      // Use the 1st element of array as keys
      if (index === 0) {
         objectKeys = tableString[index];
         continue;
      };

      // Empty object to store key and values
      var tempObject: Object = {};

      // For the length of an array within nest...
      for (var element = 0; element < objectKeys.length; element++) {
         //console.log(objectKeys[element]);

         // Pasting Excel formulae in place, otherwise blank
         if (objectKeys[element] === 'Working Days') {
```

```
            tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled
Sample]]),IF(ISBLANK([@[Date Received]]),"",IF(ISBLANK([@[Report
Date]]),NETWORKDAYS([@[Date Received]],TODAY())-1,NETWORKDAYS([@[Date
Received]],[@[Report Date]])-1)),"")`;
         } else if (objectKeys[element] === 'Total Days') {
            tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled Sample]]),
IF(ISBLANK([@[Date Received]]), "", IF(ISBLANK([@[Report Date]]), DAYS(TODAY(),[@[Date
Received]]), DAYS([@[Report Date]],[@[Date Received]]))),"")`;
         } else {
            tempObject[objectKeys[element]] = tableString[index][element];
         };
      };

      // Push object into output array
      outputArray.push(tempObject);
      continue;
   };
   return outputArray;
};
```

5.3.7 Subcon Receival script:

```
// Function to add subcon data to Commercial's tracking sheet
function main(workbook: ExcelScript.Workbook, newData: string): void {
   // Convert newData string to JSON objects
   const inputData: string[][] = JSON.parse(newData);

   // Find output table, turn it to string
   const table = workbook.getWorksheets()[0].getTables()[0];
   const tableString = table.getRange().getTexts();

   // Convert table string to nested objects
   let tableObjects = stringToObjects(tableString);
   //console.log(tableObjects);

   // For each row of update table...
   for (let entry = 0; entry < inputData.length; entry++) {
      // ID to look for
      const targetID: string = inputData[entry]['Sample No'];
      //console.log(targetID);

      // Index of targetID in output table
      const position: number = binarySearch(tableObjects, 'Sample No', targetID);
      //console.log(position);

      if (position === -1) {
         // If sample isn't present already, go to the next entry
         continue;
      };

      // Mark off at position
      if (tableObjects[position]['Subcon Received'] === '') {
         tableObjects[position]['Subcon Received'] = 'Y';
      } else {
         // If already complete, go to the next entry
         continue;
      };

      // Array of values
      const updateArray: string[] = Object.values(tableObjects[position]);

      // Delete previous item at index
      table.deleteRowsAt(position, 1);
      // Replace with updated string
      table.addRow(position, updateArray);
      continue;
   };
   return;
};
```

```
// Function to find the index of the ID within nested objects
function binarySearch(nestedObjects: string[][], key: string, value: string): number {
   // Range of array
   var start = 0;
   var end = nestedObjects.length - 1;
   // How many iterations did it take to find the index
   //let turns = 0;

   // Iterate while start has not met end
   while (start <= end) {
      //turns++;
      // Find the mid index, rounding down
      var mid = Math.floor(start + ((end - start) / 2));

      // If element is present at mid index, return index value
      if (nestedObjects[mid][key] === value) {
         return mid;
      }
      // Else, re-find mid in the left or right half of the array
      else if (nestedObjects[mid][key] < value) {
         start = mid + 1;
      } else {
         end = mid - 1;
      };
   };
   // If not found, break and return -1
   return -1;
};

// Function to convert a 2D array string to nested objects
function stringToObjects(tableString: string[][]): string[][] {
   // Key: Value pairs
   var objectKeys: string[] = [];
   // Result
   var outputArray: string[][] = [];

   // for each element in array...
   for (var index = 0; index < tableString.length; index++) {
      // Use the 1st element of array as keys
      if (index === 0) {
         objectKeys = tableString[index];
         continue;
      };

      // Empty object to store key and values
      var tempObject: Object = {};

      // For the length of an array within nest...
      for (var element = 0; element < objectKeys.length; element++) {
         //console.log(objectKeys[element]);

         // Pasting Excel formulae in place, otherwise blank
         if (objectKeys[element] === 'Working Days') {
            tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled
Sample]]),IF(ISBLANK([@[Date Received]]),"",IF(ISBLANK([@[Report
Date]]),NETWORKDAYS([@[Date Received]],TODAY())-1,NETWORKDAYS([@[Date
Received]],[@[Report Date]])-1)),"")`;
         } else if (objectKeys[element] === 'Total Days') {
            tempObject[objectKeys[element]] = `=IF(ISBLANK([@[Cancelled Sample]]),
IF(ISBLANK([@[Date Received]]), "", IF(ISBLANK([@[Report Date]]), DAYS(TODAY(),[@[Date
Received]]), DAYS([@[Report Date]],[@[Date Received]]))),"")`;
         } else {
            tempObject[objectKeys[element]] = tableString[index][element];
         };
      };
      // Push object into output array
      outputArray.push(tempObject);
      continue;
   };
```

```
    return outputArray;
};
```