Excel code for check processing

Thursday, June 30, 2022 8:29 AM

Power Query

```
Source = Excel.CurrentWorkbook() { [Name=Table1"] } [Content],
#"Changed Type" = Table.TransformColumnTypes (Source, {{ "EMPLOYEE ID", Int64.Type}, { "SALARY START DATE", type date}}),
```

```
let
   Source = Excel.CurrentWorkbook(){[Name="Table1"]}[Content],
   #"Changed Type" = Table.TransformColumnTypes(Source,{{"EMPLOYEE ID", Int64.Type}, {"SALARY", Int64.Type}, {"SALARY START DATE", type date}}),
   #"Inserted Year" = Table.AddColumn(#"Changed Type", "Year", each Date.Year([SALARY START DATE]), Int64.Type),
   #"Removed Columns" = Table.RemoveColumns(#"Inserted Year",{"SALARY START DATE"}),
   #"Pivoted Column" = Table.Pivot(Table.TransformColumnTypes(#"Removed Columns", {{"Year", type text}}, "en-US"), List.Distinct(Table.TransformColumnTypes(#"Removed Columns", {{"Year", type text}}, "en-US")[Year]), "Year", "SALARY", List.Sum),
   #"Reordered Columns" = Table.ReorderColumns(#"Pivoted Column",{"EMPLOYEE ID", "2018", "2019", "2020", "2021"})
in
   #"Reordered Columns"
```

```
let
   Source = Excel.CurrentWorkbook(){[Name="Table1"]}[Content],
   #"Changed Type" = Table.TransformColumnTypes(Source,{{"EMPLOYEE ID", Int64.Type}, {"SALARY", Int64.Type}, {"SALARY START DATE", type date}}),
   #"Inserted Year" = Table.AddColumn(#"Changed Type", "Year", each Date.Year([SALARY START DATE]), Int64.Type),
   #"Removed Columns" = Table.RemoveColumns(#"Inserted Year",{"SALARY START DATE"}),
   #"Pivoted Column" = Table.Pivot(Table.TransformColumnTypes(#"Removed Columns", {{"Year", type text}}, "en-US"), List.Distinct(Table.TransformColumnTypes(#"Removed Columns", {{"Year", type text}}, "en-US")[Year]), "Year", "SALARY", List.Sum),
   #"Reordered Columns" = Table.ReorderColumns(#"Pivoted Column",{"EMPLOYEE ID", "2018", "2019", "2020", "2021"})
in
   #"Reordered Columns"
```

```
function main(workbook: ExcelScript.Workbook) {
  // Get all the worksheets in the workbook.
  let sheets = workbook.getWorksheets();

  // Get a list of all the worksheet names.
  let names = sheets.map ((sheet) => sheet.getName());

  // Write in the console all the worksheet names and the total count.
  console.log(names);
  console.log(`Total worksheets inside of this workbook: ${sheets.length}`);

  // Set the tab color each worksheet to a random color
  for (let sheet of sheets) {
    // Generate a random color hex-code.
    let colorString = `#${Math.random().toString(16).substr(-6)}`;

    // Set the color of the current worksheet's tab to that random hex-code.
    sheet.setTabColor(colorString);
  }
}
```

Display data

Apply conditional formatting

This sample applies conditional formatting to the currently used range in the worksheet. The conditional formatting is a green fill for the top 10% of values.

```
function main(workbook: ExcelScript.Workbook) {
  // Get the current worksheet.
  let selectedSheet = workbook.getActiveWorksheet();

  // Get the used range in the worksheet.
  let range = selectedSheet.getUsedRange();

  // Set the fill color to green for the top 10% of values in the range.
  let conditionalFormat = range.addConditionalFormat(ExcelScript.ConditionalFormatType.topBottom)
  conditionalFormat.getTopBottom().getFormat().getFill().setColor("green");
  conditionalFormat.getTopBottom().setRule({
    rank: 10, // The percentage threshold.
    type: ExcelScript.ConditionalTopBottomCriterionType.topPercent // The type of the top/bottom
condition.
  });
}
```

Log the "Grand Total" values from a PivotTable

This sample finds the first PivotTable in the workbook and logs the values in the "Grand Total" cells (as highlighted in green in the image below).

```
function main(workbook: ExcelScript.Workbook) {
  // Get the first PivotTable in the workbook.
  let pivotTable = workbook.getPivotTables()[0];

  // Get the names of each data column in the PivotTable.
  let pivotColumnLabelRange = pivotTable.getLayout().getColumnLabelRange();

  // Get the range displaying the pivoted data.
  let pivotDataRange = pivotTable.getLayout().getBodyAndTotalRange();

  // Get the range with the "grand totals" for the PivotTable columns.
  let grandTotalRange = pivotDataRange.getLastRow();

  // Print each of the "Grand Totals" to the console.
  grandTotalRange.getValues()[0].forEach((column, columnIndex) => {
    console.log(`Grand total of ${pivotColumnLabelRange.getValues()[0][columnIndex]}:
${grandTotalRange.getValues()[0][columnIndex]}`);
    // Example log: "Grand total of Sum of Crates Sold Wholesale: 11000"
  });
}
```

Query and delete from a collection

```
function main(workbook: ExcelScript.Workbook) {
  // Name of the worksheet to be added.
  let name = "Index";

  // Get any worksheet with that name.
  let sheet = workbook.getWorksheet("Index");

  // If `null` wasn't returned, then there's already a worksheet with that name.
  if (sheet) {
    console.log(`Worksheet by the name ${name} already exists. Deleting it.`);
    // Delete the sheet.
    sheet.delete();
  }

  // Add a blank worksheet with the name "Index".
  // Note that this code runs regardless of whether an existing sheet was deleted.
  console.log(`Adding the worksheet named ${name}.`);
  let newSheet = workbook.addWorksheet("Index");

  // Switch to the new worksheet.
  newSheet.activate();
```

Handle a #SPILL! error returned from a formula

transposes the range "A1:D2" to "A4:B7" by using the TRANSPOSE function. If the transpose results in a #SPILL error, it clears the target range and applies the formula again.

}

```
function main(workbook: ExcelScript.Workbook) {
 let sheet = workbook.getActiveWorksheet();
 // Use the data in A1:D2 for the sample.
 let dataAddress = "A1:D2"
 let inputRange = sheet.getRange(dataAddress);

 // Place the transposed data starting at A4.
 let targetStartCell = sheet.getRange("A4");

 // Compute the target range.
 let targetRange = targetStartCell.getResizedRange(inputRange.getColumnCount() - 1,
inputRange.getRowCount() - 1);

 // Call the transpose helper function.
 targetStartCell.setFormula(`=TRANSPOSE(${dataAddress})`);

 // Check if the range update resulted in a spill error.
 let checkValue = targetStartCell.getValue() as string;
 if (checkValue === '#SPILL!') {
  // Clear the target range and call the transpose function again.
  console.log("Target range has data that is preventing update. Clearing target range.");
  targetRange.clear();
  targetStartCell.setFormula(`=TRANSPOSE(${dataAddress})`);
 }
```

1

input-table-
filters

Sample Excel file

Download tables-copy.xlsx for a ready-to-use workbook. Add the following scripts to try the sample yourself!

Sample code: Combine data from multiple Excel tables into a single table

```
function main(workbook: ExcelScript.Workbook) {
  // Delete the "Combined" worksheet, if it's present.
  workbook.getWorksheet('Combined')?.delete();

  // Create a new worksheet named "Combined" for the combined table.
  const newSheet = workbook.addWorksheet('Combined');

  // Get the header values for the first table in the workbook.
  // This also saves the table list before we add the new, combined table.
  const tables = workbook.getTables();
  const headerValues = tables[0].getHeaderRowRange().getTexts();
```

```
  // Select the transposed range to highlight it.
  targetRange.select();
}
```

Move rows using range values

input-table-
filters (1)

```
function main(workbook: ExcelScript.Workbook) {

  // You can change these names to match the data in your workbook.
  const TARGET_TABLE_NAME = 'Table1';
  const SOURCE_TABLE_NAME = 'Table2';

  // Select what will be moved between tables.
  const FILTER_COLUMN_INDEX = 1;
  const FILTER_VALUE = 'Clothing';

  // Get the Table objects.
  let targetTable = workbook.getTable(TARGET_TABLE_NAME);
  let sourceTable = workbook.getTable(SOURCE_TABLE_NAME);

  // If either table is missing, report that information and stop the script.
  if (!targetTable || !sourceTable) {
    console.log(`Tables missing - Check to make sure both source (${TARGET_TABLE_NAME}) and target
table (${SOURCE_TABLE_NAME}) are present before running the script. `);
    return;
  }

  // Save the filter criteria currently on the source table.
  const originalTableFilters = {};
  // For each table column, collect the filter criteria on that column.
  sourceTable.getColumns().forEach((column) => {
    let originalColumnFilter = column.getFilter().getCriteria();
    if (originalColumnFilter) {
      originalTableFilters[column.getName()] = originalColumnFilter;
    }
  });

  // Get all the data from the table.
  const sourceRange = sourceTable.getRangeBetweenHeaderAndTotal();
  const dataRows: (number | string | boolean)[][] =
sourceTable.getRangeBetweenHeaderAndTotal().getValues();

  // Create variables to hold the rows to be moved and their addresses.
  let rowsToMoveValues: (number | string | boolean)[][] = [];
  let rowAddressToRemove: string[] = [];

  // Get the data values from the source table.
```

```
  const headerValues = tables[0].getHeaderRowRange().getTexts();
  console.log(headerValues);

  // Copy the headers on a new worksheet to an equal-sized range.
  const targetRange = newSheet.getRange('A1').getResizedRange(headerValues.length-1,
headerValues[0].length-1);
  targetRange.setValues(headerValues);

  // Add the data from each table in the workbook to the new table.
  const combinedTable = newSheet.addTable(targetRange.getAddress(), true);
  for (let table of tables) {
    let dataValues = table.getRangeBetweenHeaderAndTotal().getTexts();
    let rowCount = table.getRowCount();

    // If the table is not empty, add its rows to the combined table.
    if (rowCount > 0) {
      combinedTable.addRows(-1, dataValues);
    }
  }
}
```

tables-copy

2

Sample code: Combine data from multiple Excel tables in select worksheets into a single table

```
function main(workbook: ExcelScript.Workbook) {
  // Set the worksheet names to get tables from.
  const sheetNames = ['Sheet1', 'Sheet2', 'Sheet3'];

  // Delete the "Combined" worksheet, if it's present.
  workbook.getWorksheet('Combined')?.delete();

  // Create a new worksheet named "Combined" for the combined table.
  const newSheet = workbook.addWorksheet('Combined');

  // Create a new table with the same headers as the other tables.
  const headerValues = workbook.getWorksheet(sheetNames[0]).getTables()
[0].getHeaderRowRange().getTexts();
  const targetRange = newSheet.getRange('A1').getResizedRange(headerValues.length-1,
headerValues[0].length-1);
  targetRange.setValues(headerValues);
  const combinedTable = newSheet.addTable(targetRange.getAddress(), true);

  // Go through each listed worksheet and get their tables.
  sheetNames.forEach((sheet) => {
    const tables = workbook.getWorksheet(sheet).getTables();
    for (let table of tables) {
      // Get the rows from the tables.
      let dataValues = table.getRangeBetweenHeaderAndTotal().getTexts();
      let rowCount = table.getRowCount();

      // If there's data in the table, add it to the combined table.
      if (rowCount > 0) {
        combinedTable.addRows(-1, dataValues);
      }
```

```
 for (let i = 0; i < dataRows.length; i++) {
   if (dataRows[i][FILTER_COLUMN_INDEX] === FILTER_VALUE) {
     rowsToMoveValues.push(dataRows[i]);

     // Get the intersection between table address and the entire row where we found the match. This
provides the address of the range to remove.
     let address = sourceRange.getIntersection(sourceRange.getCell(i,0).getEntireRow()).getAddress();
     rowAddressToRemove.push(address);
   }
 }

 // If there are no data rows to process, end the script.
 if (rowsToMoveValues.length < 1) {
   console.log('No rows selected from the source table match the filter criteria.');
   return;
 }

 console.log(`Adding ${rowsToMoveValues.length} rows to target table.`);

 // Insert rows at the end of target table.
 targetTable.addRows(-1, rowsToMoveValues)

 // Remove the rows from the source table.
 const sheet = sourceTable.getWorksheet();

 // Remove all filters before removing rows.
 sourceTable.getAutoFilter().clearCriteria();

 // Important: Remove the rows starting at the bottom of the table.
 // Otherwise, the lower rows change position before they are deleted.
 console.log(`Removing ${rowAddressToRemove.length} rows from the source table.`);
 rowAddressToRemove.reverse().forEach((address) => {
   sheet.getRange(address).delete(ExcelScript.DeleteShiftDirection.up);
 });

 // Reapply the original filters.
 Object.keys(originalTableFilters).forEach((columnName) => {
     sourceTable.getColumnByName(columnName).getFilter().apply(originalTableFilters[columnName]);
   });
}
```

```
    }
  });
}
```

event-data

speaker-registrations

Cross-reference Excel files with Power Automate

how to compare data across two Excel files to find discrepancies. It uses Office Scripts to analyze data and Power Automate to communicate between the workbooks.

This sample passes data between workbooks using JSON objects. For more information about working with JSON, read Use JSON to pass data to and from Office Scripts.

```
function main(workbook: ExcelScript.Workbook): string {
  // Get the first table in the "Keys" worksheet.
  let table = workbook.getWorksheet('Keys').getTables()[0];

  // Get the rows in the event table.
  let range = table.getRangeBetweenHeaderAndTotal();
  let rows = range.getValues();

  // Save each row as an EventData object. This lets them be passed through Power Automate.
  let records: EventData[] = [];
  for (let row of rows) {
    let [eventId, date, location, capacity] = row;
    records.push({
      eventId: eventId as string,
      date: date as number,
      location: location as string,
      capacity: capacity as number
    })
  }

  // Log the event data to the console and return it for a flow.
  let stringResult = JSON.stringify(records);
  console.log(stringResult);
  return stringResult;
}

// An interface representing a row of event data.
interface EventData {
  eventId: string
  date: number
  location: string
  capacity: number
}
```