Standard operating procedure

Uploading of data into CKAN and GCP

Table of Contents

[Introduction 2](#__RefHeading___Toc229_1508744584)

[Data formats 2](#__RefHeading___Toc2033_652564788)

[Excel 2](#__RefHeading___Toc623_3899410093)

[Multiple table in an Excel sheet 2](#__RefHeading___Toc625_3899410093)

[Field names 3](#__RefHeading___Toc627_3899410093)

[Empty rows and columns 4](#__RefHeading___Toc1522_3899410093)

[Merged columns and rows 5](#__RefHeading___Toc1524_3899410093)

[Graphs and other objects 5](#__RefHeading___Toc1526_3899410093)

[Excel sheet formulas 6](#__RefHeading___Toc1528_3899410093)

[CSV 6](#__RefHeading___Toc2035_652564788)

[Field names 7](#__RefHeading___Toc716_3899410093)

[Field content 8](#__RefHeading___Toc718_3899410093)

[CSV parsing 9](#__RefHeading___Toc856_3899410093)

[Text format 10](#__RefHeading___Toc1143_3899410093)

[Shapefile 11](#__RefHeading___Toc2037_652564788)

[Geojson 12](#__RefHeading___Toc1145_3899410093)

[Field names 12](#__RefHeading___Toc1147_3899410093)

[Compressed files 12](#__RefHeading___Toc252_2668390096)

# Introduction

This standard operating procedure (SOP) will supply the user with instructions and standards that should be adhered to when uploading data using the Water Research Observatory (WRO) site (uses CKAN) into Google Cloud Platform (GCP). Following this guide will allow the user to avoid any issues while uploading data, especially when the data needs to be stored as a BigQuery table.

The first section deals with data formats.

# Data formats

The rules or standards a user will need to follow depends on the format. For instance a shapefile contains the field type for each field, but this is not the case with comma-separated values (CSV) files as it is a text-based format. These rules will enable the user to make sure the data is correct prior to uploading it.

The first section deals with Excel, followed by the CSV format. Text format has a short discussion on steps which needs to be followed. This is followed by shapefiles and geojson. The last section deals with compressed files.

## Excel

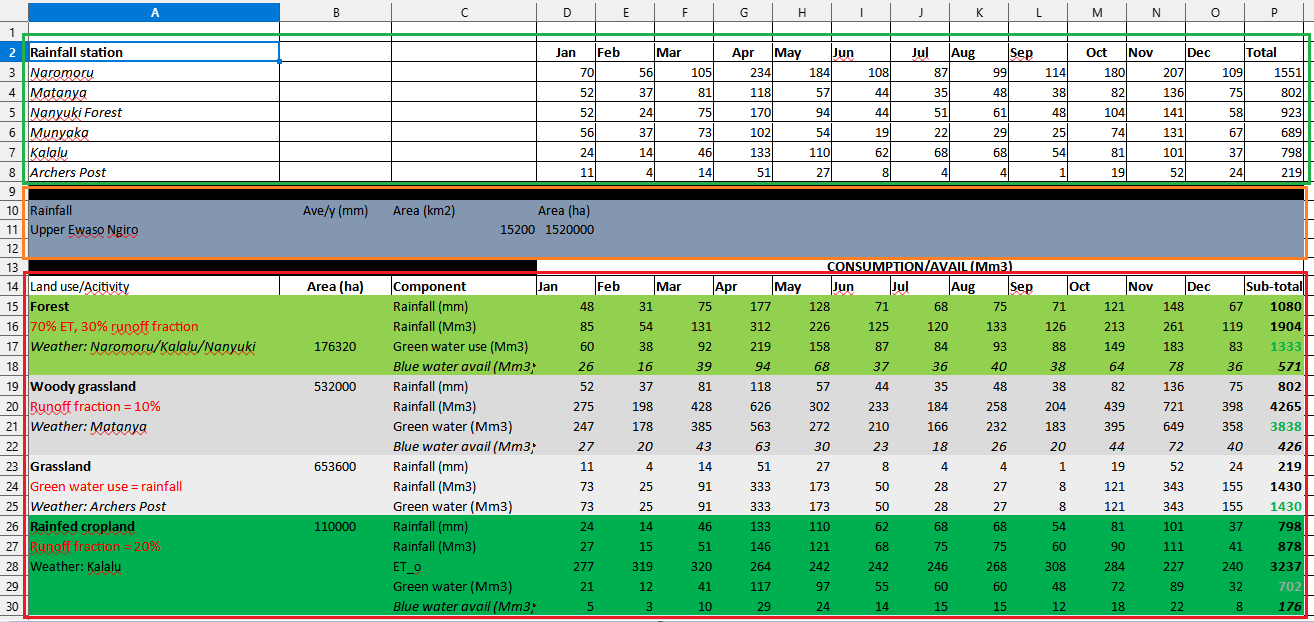
Excel files can be imported into BigQuery. The latest excel format (\*.xlsx) and the older excel format (\*.xls) are supported, but best will be to make use of \*.xlsx (\*.xls is more than two decades old). All Excel applications allow users to save in the \*.xlsx format, so it is easy to convert. But otherwise \*.xls is still supported and should work with no issue when uploading data into BigQuery.

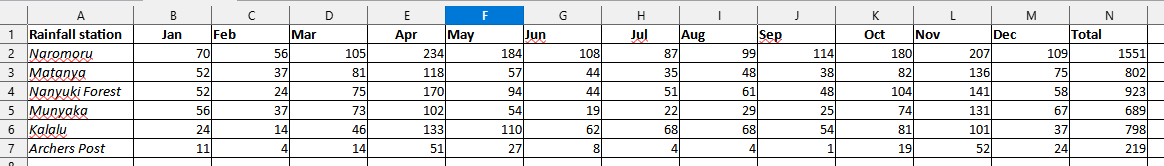
The first section deals with multiple tables in a single sheet, followed by a section on field names, empty rows/columns, merged rows/columns and graphs/objects stored in the Excel file. The last section covers formulas.

### Multiple table in an Excel sheet

A sheet which contains multiple tables cannot be processed, as its extremely challenging to accurately split the contents into each table automatically. The recommendation is to place the respective tables on separate worksheets. Figure 1 illustrates an example of an Excel worksheet which contains multiple tables (highlighted in green, orange, and red).

**Solution**: Copy each table to its own worksheet. The code supports multiple sheets, so each table will then be loaded into a separate BigQuery table. Figure 2 shows the top table Figure 1 copied to its own sheet.

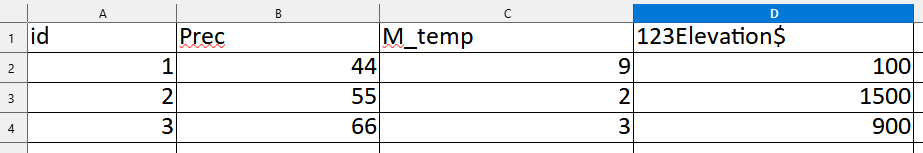
Figure 1: Excel sheet which contains multiple tables

Figure 2: Sheet containing a single table

### Field names

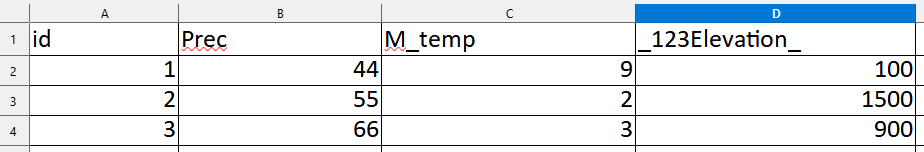
Excel allows the user to make use of special characters (‘&’, ‘)’, etc.) in field names, but note this is not supported by BigQuery. Users should also make use of descriptive field names (e.g. ‘M\_temp’ for minimum temperature can be confusing) when loading an Excel table into BigQuery. See Figure 3.

**Solution**: The code automatically attempts to “fix” any field names which contains unwanted characters by replacing the unwanted character with an ‘\_’ (e.g. ‘123Elevation$’ would be changed to ‘123Elevation\_’). But best practice will be to avoid such cases and use descriptive names. Here is a list of suggestions for when naming fields:

Figure 3: Table which contains poor field names

* Never start a field name with a numeric value (e.g. ‘123Elevation’ would be changed to ‘\_123Elevation’);
* The following characters should not be used: ‘@’, ‘#’, ‘$’, ‘%’, ‘^’, ‘&’, ‘\*’, ‘(’, ‘)’, etc. There are a large number of other characters which should be avoided, too many to list all of the cases;
* No spaces in field names, and note the code will replace them with ‘\_’; and
* Always use descriptive names, ‘precipitation’ will be better than ‘prec’. BigQuery has a long field name length limit, so there are no concerns there.

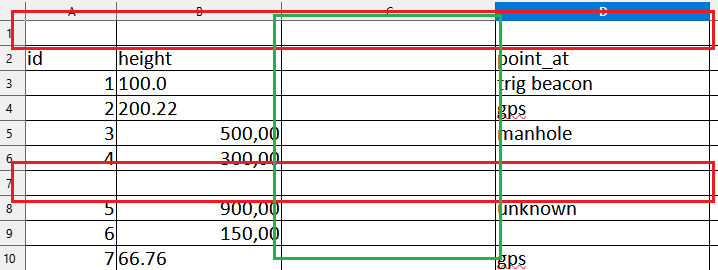
Figure 4 shows an example of parsed field names. The only field which changed in this example is at column D. ‘123\_Elevation$’ was changed to ‘\_123Elevation\_’ so that it is a valid field name for BigQuery.

Figure 4: Parsed field names

### Empty rows and columns

Empty rows and columns will cause a problem. Such cases are difficult to process, because it can be interpreted to mean there is no data, that it is the start of another table, or it is just a poorly structured table. See Figure 5. Missing rows highlighted in red, missing columns in green.

**Solution**: The suggested best approaches to fix issues related to empty columns and rows:

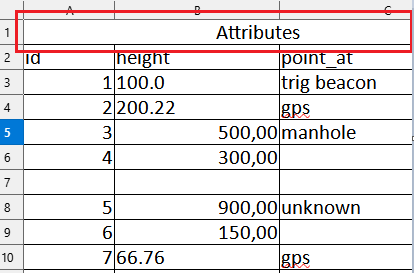
Figure 5: Excel table with missing rows and columns

* Delete all unwanted empty rows and columns;
* If its several tables in one sheet, copy each of the tables to their own respective sheet (see Multiple tables in an Excel sheet);
* Add a column heading if the field should be retained; and
* **Important**: Never start a sheet with an empty row followed by the actual table.

### Merged columns and rows

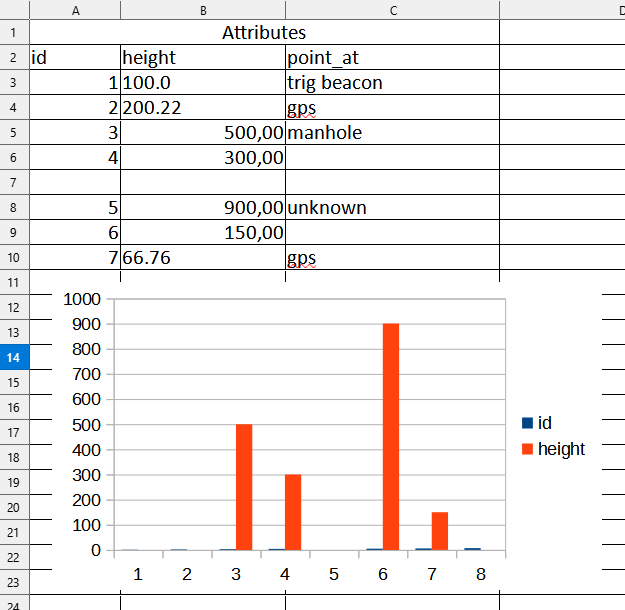
Merged columns are common practice in Excel, but BigQuery cannot store a merged column. See Figure 6. The merged row is highlighted in red.

**Solution**: Simply remove merged columns and rows. Either split the merged cells into individual cells or delete the merged cell(s) (row with the merged cells) from the table. For instance, the ‘Attribute’ row in Figure 6 can be removed as it has no use.

Figure 6: Excel table which contains a merged field name

### Graphs and other objects

When importing an Excel file which contains graphs or other objects, the graphs and objects are ignored. Therefore, the only content that will be loaded and processed is the cell content of the Excel sheet, and the table will be stored in BigQuery. See Figure 7.

Figure 7: Excel sheet with a bar graph

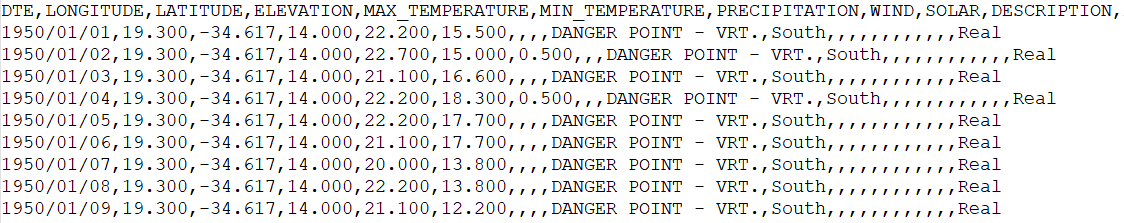
### Excel sheet formulas

In terms of the formulas contained in Excel cells, only the resulting values from the formulas will be imported. Therefore, be sure that the calculations are done correctly prior to loading the Excel file into BigQuery, as the formula cannot be changed in BigQuery.

## CSV

A CSV is a delimited text file, which uses a text character (e.g. “,”, “;”, tabs, spaces, etc.) to separate and distinguish between columns. An XYZ file can also be seen as a text delimited format using a comma as the delimiter.

The example which follows (Figure 8) shows a CSV file which stores data with a comma (“,”) as the delimiter. Therefore “1950/01/01” goes with the “DTE” column, “19.300” with longitude, and so on.

Figure 8: CSV file example

To be as robust as possible, it will be best for the user to make sure there are no issues in the CSV file itself prior to uploading the file.

**Important**: Always aim to use a comma (“,”) as the delimiter for CSV files. Not using a comma will cause issues and erroneous results when processing the CSV table.

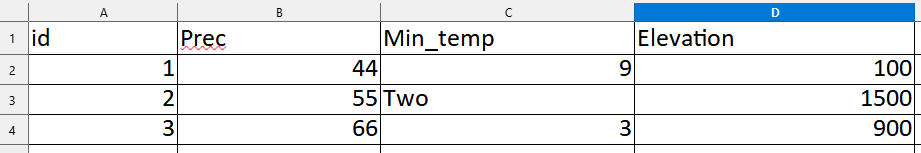
### Field names

As with Excel files, there are characters which need to be avoided and rules to follow when importing from GeoJSON. See the section ‘Field names’ for a detailed discussion on the best practice to avoid issues.

### Field content

Issues in the field content can have a large impact on the resulting BigQuery table. For instance, if a field should be numeric, but there is a row which has a value “One”, then it will be set as a string because BigQuery only allows numeric values (“5”, “10.6”, etc.). In Figure 8 column ‘Min\_temp’ will be set to string, because there is a value ‘two’.

**Solution**: Here are some best practice rules a user can follow when working with CSV files:

Figure 11: CSV example

* + Be sure there are no alphabet characters or other non-numeric characters (e.g. ‘$’, ‘#’, etc.) in the field contents if it should be a numeric field;
  + Dates should always be **YYYY-MM-DD** (e.g. 2006-01-15);
    - **Important**: BigQuery can only work with the above format. If the above format is not used, it will most likely be stored in a string or numeric format;
  + Do not use **YYYYMMDD** (e.g. 20060115) as its difficult to automatically distinguish such values from normal numeric values;
  + Commas should be avoided in field content when possible. A workaround will be to use quotation marks (e.g. ‘two, three, four’ should be changed to ‘”two, three, four”’);
  + Empty values should be avoided if possible, but the code will set such cases to *nulll*;
  + Be sure that each row has the same number of columns; and
  + No empty columns or rows.

Not adhering to these rules will cause issues when loading data into BigQuery.

### CSV parsing

The code will also parse the CSV data, which attempts to remove any unwanted characters and automatically determine the field type (e.g. float, string, etc.). The parsing will do the following:

* Remove any spaces, tabs, and new lines from numeric column content;
* Replace spaces and other non-alphanumeric characters with “\_” in field names;
* If any numeric value cannot be converted to a numeric type, that field will be set as type string;
* Spaces, and non-numeric characters will not be removed from a string;
* Set a column type to date if the correct date formatting is used. BigQuery requires **YYYY-MM-DD**;
* Check if the number of values of a row agrees with the number of field names at the first row of the CSV file;
* If a field name starts with a numeric character, “\_” will be added to the start of the field name;
* Empty fields will be left as-is. BigQuery sees such cases as *null*; and
* Store the parsed data in BigQuery as a table.

Table 1 shows examples of possible updates the CSV parses will perform before loading the CSV file into BigQuery.

Table 1: Examples of possible changes the CSV parser will implement

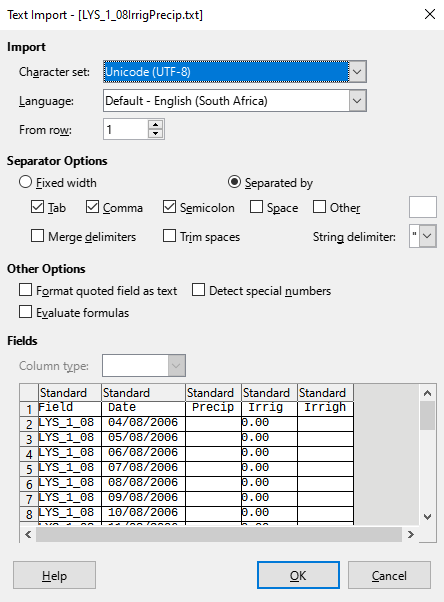
|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Value** | **Updated value** | **Description** |
| Field name | max temperature | max\_temperature | Space removed |
| Field name | 123\_precipitation | \_123\_precipitation | Starts with numeric characters |
| Field name | rainfall# | rainfall\_ | Unwanted character removed |
| Numeric | 30 000 | 30000 | Space removed |
| Date | 2019/01/15 | 2019-01-15 | Date converted to BigQuery format |
| String | Danger point | Danger point | No changes, as it’s a string |
| Empty |  | Null | This will be null in a BigQuery table |

## Text format

A text file (\*.txt) can contain either a table or standard text. This makes it difficult to determine if a text file contains a table. See the solution on steps that can be taken if the text file contains a table. If it only contains standard text, the text file can be left as is.

**Solution**: If the text file contains a table, it’s likely a delimited text file, and can therefore be converted to a CSV file. Here is a list of possible steps a user can take to accomplish this:

* Change the extension of the file (e.g. ‘table.txt’ to ‘table.csv’). Be sure that the file already make use of a comma (‘,’) as the delimiter;
* Open the file in Excel or Libre Calc. When doing so, the application will ask the user what the ‘Separator’ or ‘Delimiter’ is (see Figure 12). Then save the file as a CSV which makes use of a comma as the delimiter; and

Figure 12: Delimited text file opened in Libre Calc

* The user can also manually use the ‘Replace all’ feature using a text editor such as Notepad++.

**Important**: Remember to change the file extension to .csv.

## Shapefile

Uploaded shapefiles can also be loaded into BigQuery. Adhere to the following rules:

* Store all of the shapefile files in a zip file. Upload this zip file. CKAN only allows the upload of one file. The files will be extracted automatically;
* Table 2 shows the possible file types which a shapefile can consist of, and which files are compulsory and have to be included for the functioning of the shapefile;

Table 2: Shapefile files and requirements

|  |  |
| --- | --- |
| **Extension** | **Compulsory?** |
| Shapefile (\*.shp) | Yes |
| Position index file (\*.shx) | Yes |
| Dbase database file (\*.dbf) | Yes |
| Projection file (\*.prj) | Yes |
| Spatial index (\*.sbn) | No |
| Read-only spatial index (\*.fbn/\*.fbx) | No |
| Attribute index file (\*.ain/\*.aih) | No |
| Geocoding index (\*.ixs) | No |
| Metadata (\*.xml) | No |
| Codepage (\*.cpg) | No |

* **IMPORTANT**: If any of the required files are missing (\*.shp, \*.shx, \*.dbf, and \*.prj), the shapefile is broken and cannot be used;
* Shapefile already stores the field types, which will be used as the field types for the BigQuery table; and
* Shapefile field names have the same limitations as BigQuery table, so there should be no issue with the field names.

## Geojson

GeoJSON, based on the JSON format, can store geographic features with non-spatial attributes. The WRO can automatically load this format into BigQuery by converting it to newline JSON, which is a supported format for when importing into BigQuery.

Figure 13: GeoJSON example

### Field names

As with Excel files, there are characters which need to be avoided and rules to follow when importing from GeoJSON. See the section ‘Field names’ for a detailed discussion on the best practice to avoid issues.

## Compressed files

Currently only ZIP files (\*.zip) are allowed for upload. When a ZIP file is stored in a bucket and the user has selected the import to BigQuery option, the code will trigger. This code will automatically extract all the files stored in the ZIP, which will then be stored in the same bucket. When this data is stored in the bucket, it will again trigger the code. The code will then parse each file as usual – basically as if the user uploaded each file in the compressed file individually. This can be done with all file types.