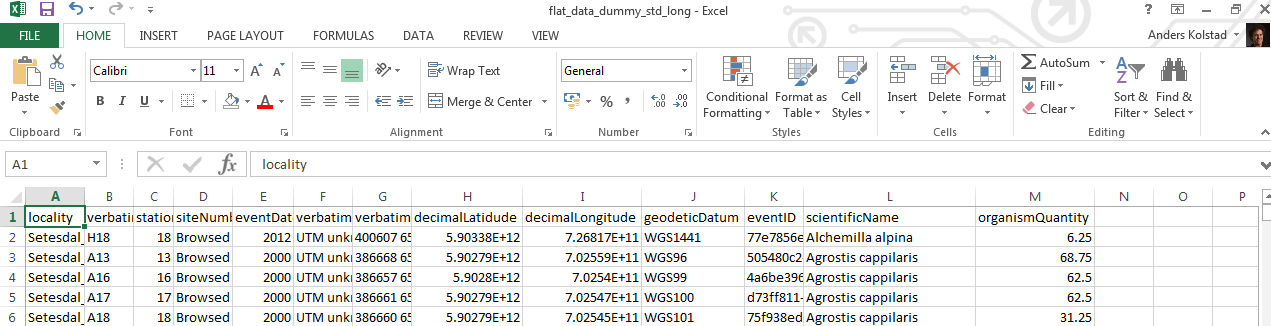
Structure and map workflow

The raw unpublished dataset is in a standardised flat format or this type:



This is a long format (i.e. species name is a column, not one column per species). Column names are the same as those already in use in Natron. EventID is a UUID generated in an earier step before transposing the dataset from wide to long.

Question 1:

In locations (or even more tricky, some of the locations) exist in NaTron from before, how do we get hold of the UUIDs so that we can reuse them?

*We can screen the locations table for similar locality, verbatimLocality, stationNumber or a combination of these. This would be much easier if one of these terms (probably verbatimLocality) was unique in NaTron. However, it very easy to make a typo in the text string which would result in R not finding the duplicate locality. Another option is to filter existing locations by the coordinates, so that R return a list of candidate locations that are within say 100 m of the present locations. Then one would have to manually check if some of these locations can be reused. If so, change the verbatimLocality text so that it matches exactly the NaTron locality and run through the first option with screening for similar locations in NaTron. In reality one often knows if the location exists from before or not. If you know it doesn’t exists you can go straight to making the locations table and give them new UUIDs. If you know they exists (or some of them exists) the question is how many locations are we talking about, because the manual work can be time consuming for large numbers of locations. However, resampling of locations is usually done for rather small experimental datasets (where verbatimLocality is consistent to begin with).*

The next step is to produce the locations table (if the locations are new) or extract locationIDs for the event table if locations are pre-existing (see above).For making any table (locations, events, occurrences) the procedure is similar. First we get the column names from Natron:

tableinfo **<-** dbGetQuery**(**con,

"select table\_name,column\_name,data\_type

from information\_schema.columns

where table\_name = 'Events' OR

table\_name = 'Occurrences' OR

table\_name = 'Locations'

;"**)**

We then use these to create an empty data frame using the correct column headers in the correct order, before we paste the data we have into the corresponding columns, leaving the columns we don’t have data for as blank or NA (example here from event table):

# select all terms (columns) for event table as they are in NaTron

event\_db\_terms **<-** tableinfo**$**column\_name**[**tableinfo**$**table\_name**==**"Events"**]**

# filter leaving those that match our dummy flat data

event\_terms **<-** names**(**flatt\_data**)[**names**(**flatt\_data**)** %in% event\_db\_terms**]**

# recreate the temporary event table with the right column names (this step removes columns that are not defined in NaTron)

event\_data\_temp **<-** flatt\_data**[**event\_terms**]**

# create empty dataframe with all event table terms

event\_data **<-** data.frame**(**matrix**(**ncol **=** length**(**event\_db\_terms**)**, nrow **=** 0**)**,stringsAsFactors**=FALSE)**

colnames**(**event\_data**)** **<-** event\_db\_terms

# rowbind event data from import to the empty data.frame

# in order to create generic event table for import

event\_data **<-** bind\_rows**(**event\_data,event\_data\_temp**)**

This data now needs locationIDs and then it’s ready for upsert.