**National Park Service**

**U.S. Department of the Interior**

Bingham, S. N., C. C. Young, J. L. Haack-Gaynor , L. W. Morrison, and G. A. Rowell. 2016. Wetland monitoring protocol for Cuyahoga Valley National Park: Narrative. Natural Resource Report NPS/HTLN/NRR—2016/ 1336. National Park Service, Fort Collins, Colorado.

# Standard Operating Procedure XX: Loading VIBI Survey123 Data into HTLNWetlands Database

**Version 1.00 (12/27/2016)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Previous Version # | Revision Date | Author | Changes Made | Reason for Change | New Version # |
|  | 3/31/2024 | G. Rowell | Initial version |  | 1.0 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Wetland Monitoring Protocol for Cuyahoga Valley National Park**

*Standard Operating Procedures*

**Contents**

Page

[Standard Operating Procedure xx: Uploading Survey123 Data 3](#_bookmark0)

[Heartland Wetlands Geodatabase (HTLNWetland.gdb) 3](#_bookmark1)

[Heartland Wetlands Database (HTLNWetland.mdb) 4](#_bookmark2)

[NPSTORET Database (NPSTORET.mdb) 5](#_bookmark3)

[Aquarius Database 6](#_bookmark4)

[Heartland Water Depth Spreadsheets 7](#_bookmark5)

**Figures**

**Figure XX.X**

. 5

**Standard Operating Procedure XX: Uploading VIBI Survey123 Data**

This SOP describes procedures for uploading VIBI Survey123-generated Excel spreadsheets into the HTLNWetlands MS Access database.

Overview

This SOP describes the uploading process of Excel files generated by Survey123 surveys for VIBI field data into the HTLNWetlands database . The survey data is collected for VIBI herbaceous and VIBI woody analysis. VIBI woody data also includes data for big trees (> 40 cm in diameter). For all three datasets (herbaceous, woody and big trees) the workflow is the same. Excel spreadsheets derived from Survey123 surveys are loaded into R scripts and reformatted to match their destination tables in Access. The R scripts generate “loadfiles” which are imported into the HTLNWetlands Access database. The imported loadfiles are appended to each respective table (tbl\_BigTrees, tbl\_VIBI\_herb and tbl\_VIBI\_woody). The complete code for each script is provided in Appendix A. Once the data is appended to Access tables, it is exported back out as .csv files for end-to-end validation. The validation tests consist of (1) verifying no duplicate records are contained in either the loadfile or the exported datasets, (2) verifying no missing values in the measured fields and (3) matching record counts between loadfiles and data exports. The complete code for end-to-end scripts is provided in Appendix B.

PROCEDURES

Review the Survey123 datasheets:

1. Make working copies of the original spreadsheets and store the original files in a safe location.
2. Open the spreadsheets using Excel.
3. Column headings should be machine-readable. Remove any spaces or special characters. Spaces should be replaced with underscores. Special characters such as commas, apostrophes, percent signs, and math symbols should be replaced with text.
4. Ensure that all columns contain data. Columns that do not contain data should be deleted.
5. Removed all columns in the spreadsheet records. Replace commas with a non-parsing symbol such as period.

Generating the load file: "Load\_VIBI\_herb\_2023.xlsx"

1. Export the VIBI spreadsheets from Excel into csv format. Use the CSV UTF-8 format.
2. Run the script Survey123\_VIBI\_herb\_final.R
3. The following is a description of the code in Survey123\_VIBI\_herb\_final.R

The following code is broken into functional steps. Download the package – tidyverse from CRAN prior to running the code. The first few lines describe the purpose of the script.

################################################################################

#

# Survey123\_VIBI\_herb\_final.R

#

# Gareth Rowell, 2/16/2024

#

# This script converts csv files exported from Survey123 VIBI herbaceous

# data to create a file that can be loaded into MS Access and directly

# appended to the tbl\_VIBI\_herb table.

#

################################################################################

library(tidyverse)

#setwd("./VIBI-herbaceous")

The first step is to load the csv files into dataframes. Use the bind\_rows() function to append multiple csv files.

#################

#

# Step 1 - load spreadsheet csv files and appended them

#

#################

# load the Survey123 data

load\_file1 <- read\_csv("CUVA\_VIBI\_herb1.csv")

load\_file2 <- read\_csv("CUVA\_VIBI\_herb2.csv")

load\_file3 <- read\_csv("CUVA\_VIBI\_herb3.csv")

glimpse(load\_file1)

glimpse(load\_file2)

glimpse(load\_file3)

Access\_data <- bind\_rows(load\_file1,load\_file2)

glimpse(Access\_data)

Access\_data <- bind\_rows(Access\_data,load\_file3)

glimpse(Access\_data)

In the next code, I am renaming columns to more closely match the table tbl\_VIBI\_herb. The mm/dd/yyyy date format is converted to the universal date format yyyy-mm-dd.

#################

#

# Step 2 - select and rename columns, convert date to yyyy-mm-dd

#

#################

Access\_data <- Access\_data |>

select(Species, Comments, Module, CoverClass\_LT\_6m,

CoverClassAll, EditDate, HerbSiteName) |>

mutate(

FeatureID = HerbSiteName,

CoverClass = CoverClass\_LT\_6m,

EditDate = (EditDate <- as.Date(EditDate, format = "%m/%d/%Y"))

)

glimpse(Access\_data)

In Step 3, I am removing missing records and duplicate records. In Step3a, I am replacing the “NA” symbols. NA means “not available”. Its a special symbol in R that’s generated when values are missing. Here, I substitute it with -9999 in numerical columns. Then I inspect the -9999 rows. If they represent records with missing data, I remove them from the dataframe.

#################

#

# Step 3a - Substitute NA with -9999 in CoverClass and CoverClassAll

# Then remove those with -9999 in CoverClass

#

#################

Access\_data$CoverClass <- Access\_data$CoverClass |> replace\_na(-9999)

Access\_data$CoverClassAll <- Access\_data$CoverClassAll |> replace\_na(-9999)

Access\_data |>

filter(CoverClass == -9999)

Access\_data <- Access\_data |>

filter(CoverClass != -9999)

# then test

Access\_data |>

filter(CoverClass == -9999)

In Step 3b, I am inspecting and removing duplicate rows.

##########

#

# Step 3b - check for duplicates

#

##########

# test for dups

Access\_data |>

count(Species, Comments, Module, CoverClass\_LT\_6m, CoverClassAll,

EditDate, HerbSiteName, FeatureID, CoverClass

) |>

filter(n > 1)

# Remove dups with distinct()

Access\_data <- Access\_data |>

distinct(Species, Comments, Module, CoverClass\_LT\_6m, CoverClassAll,

EditDate, HerbSiteName, FeatureID, CoverClass

)

Access\_data

# test for dups

Access\_data |>

count(Species, Comments, Module, CoverClass\_LT\_6m, CoverClassAll,

EditDate, HerbSiteName, FeatureID, CoverClass

) |>

filter(n > 1)

In Step 4, I am generating the EventID. In Step 4a, I’m concatenating the park and project code ‘CUVAWetlnd’ to the date value, removing the dashes ‘-‘, and creating a separate column NumMonth. The NumMonth is used in Step4b to insert the text month. In Step 4b, I am importing a look-up table that contains the month as a three character string, for example, July is represented as ‘Jul’ and August is represented as ‘Aug’. This lookup table is joined to the dataframe using a left\_join(). Once the look-up table is imported and attached, I can extract the front half of the EventID which contains the park, project id and the year as YYYY, and extract the back half of the EventID which contains the day in DD format, and insert the text month between them. Its a bit complicated but it does generate the standard EventID column.

#################

#

# Step 4a - Generate EventID from EditDate

#

#################

Access\_data <- Access\_data |>

mutate( EventID = str\_c( 'CUVAWetlnd', EditDate)) |>

mutate(EventID = str\_replace\_all(EventID, "-", "")) |>

mutate(NumMonth = str\_sub(EventID, start = 15L, end = -3L))

#################

#

# Step 4b - Replace numeric month with text month abbreviation

#

#################

Months\_LUT <- read\_csv("Months\_LUT.csv")

Access\_data <- Access\_data |>

left\_join(Months\_LUT, join\_by(NumMonth))

Access\_data <- Access\_data |>

mutate(EventID\_left = str\_sub(EventID, start = 1L, end = -5)) |>

mutate(EventID\_right = str\_sub(EventID, start = 17, end = -1))

Access\_data <- Access\_data |>

mutate(EventID = str\_c(EventID\_left, TxtMonth, EventID\_right))

In step 5, I am also using a lookup table. This time, I am generating a LocationID column based on the FeatureID. The left\_join() is used again to attach the desired column from the lookup table to the dataframe.

#################

#

# Step 5 - Create the LocationID column from the FeatureID column

#

#################

Locations\_LUT <- read\_csv("tbl\_Locations\_20230316.csv")

glimpse(Locations\_LUT)

Access\_data <- Access\_data |>

left\_join(Locations\_LUT, join\_by(FeatureID))

Finally, in this last step, Step 6, I am just selecting the columns that will be needed to append the data to tbl\_VIBI\_herb in the database. The last line writes the dataframe to the loadfile.

#################

#

# Step 6 - Clean up columns and write load file

#

#################

# clean up columns

Access\_data <- Access\_data |>

select(EventID, FeatureID, LocationID, Species, Comments, Module,

CoverClass, CoverClassAll, EditDate )

writexl::write\_xlsx(Access\_data, "Load\_VIBI\_herb\_2023.xlsx"

The load file should be ready to import into Access to be joined to tbl\_VIBI\_herb.

Generating the load file: "Load\_VIBI\_woody\_2023.xlsx"

In the following script, I am taking Survey123 data for VIBI woody observations and converting them to a load file to append to tbl\_VIBI\_woody in the Access database.

################################################################################

#

# Survey123\_VIBI\_woody\_final.R

#

# Gareth Rowell, 2/16/2024

#

# This script converts csv files exported from Survey123 VIBI woody

# data to create a file that can be loaded into MS Access and directly

# appended to the tbl\_VIBI\_woody table.

#

################################################################################

library(tidyverse)

#setwd("../HTLN-Data-Capture-Scripts/wetlands/src")

#setwd("./VIBI-woody")

In Step 1 below, I am loading the VIBI woody spreadsheet data. This is almost identical to Step 1 in the script above except I am dealing with species codes as well as scientific names. The records with species codes have been placed into their own spreadsheets with a column called SpeciesCode. The scientific name is then attached to the dataframe by way of a left\_join() from the lookup table”WoodySpecies\_LUT2.csv”

#################

#

# Step 1 - load spreadsheet csv files and appended them

# Note - species codes were only used in CUVA\_VIBI\_woody1.csv

# and also CUVA\_VIBI\_woody3.csv

# They were joined to WoodySpecies\_LUT2.csv to create WoodySpecies

#

#################

# species codes were only used in CUVA\_VIBI\_woody1.csv

# join to WoodySPecies\_LUT2.csv to create WoodySpecies

load\_file1 <- read\_csv("CUVA\_VIBI\_woody1.csv")

problems(load\_file1)

WoodySpecies\_LUT <- read\_csv("WoodySpecies\_LUT2.csv")

load\_file1 <- load\_file1 |>

left\_join(WoodySpecies\_LUT, join\_by(SpeciesCode))

glimpse(load\_file1)

glimpse(WoodySpecies\_LUT)

load\_file2 <- read\_csv("CUVA\_VIBI\_woody2.csv")

problems(load\_file2)

load\_file3 <- read\_csv("CUVA\_VIBI\_woody3.csv")

problems(load\_file3)

load\_file3 <- load\_file3 |>

left\_join(WoodySpecies\_LUT, join\_by(SpeciesCode))

glimpse(load\_file3)

load\_file4 <- read\_csv("CUVA\_VIBI\_woody4.csv")

problems(load\_file4)

glimpse(load\_file1)

glimpse(load\_file2)

glimpse(load\_file3)

glimpse(load\_file4)

Access\_data <- bind\_rows(load\_file1,load\_file2)

glimpse(Access\_data)

Access\_data <- bind\_rows(Access\_data,load\_file3)

glimpse(Access\_data)

Access\_data <- bind\_rows(Access\_data,load\_file4)

glimpse(Access\_data)

load\_file <- Access\_data # used for normalization test in step 8

Step 2 below cleans things up with sel.ect() and renaming columns using mutate(). The date is converted to YYYY-MM-DD

#################

#

# Step 2 - select and rename columns, convert date to yyyy-mm-dd

#

#################

Access\_data <- Access\_data |>

select(WoodyModule, WoodySpecies, EditDate, WoodySiteName, ShrubClump, D0to1,

D1to2\_5, D2\_5to5, D5to10, D10to15, D15to20, D20to25, D25to30, D30to35,

D35to40, Dgt40, Dgt40\_1, Dgt40\_2, Dgt40\_3, Dgt40\_4, Dgt40\_5)

Access\_data <- Access\_data |>

mutate( FeatureID = WoodySiteName) |>

mutate(EditDate = (EditDate <- as.Date(EditDate, format = "%m/%d/%Y"))) |>

mutate( Module\_No = WoodyModule )

glimpse(Access\_data)

Step 3 below introduces the processing steps to normalize the diameter columns in the spreadsheet. The goal is to go from … ShrubClump, D0to1, D1to2\_5, Diam2\_5to5… to two columns, one containing the category of diameter called DiamID and a second column called Count. This process starts in Step 3 by renaming columns as Access\_data$Colx. The process is completed in Step 6 using the pivot longer function. Steps 4 and 5 generate the EventID and LocationID and are identical to the methods used for VIBI herbaceous above.

#################

#

# Step 3 - Rename columns using DiamID values for pivot\_longer

#

#################

Access\_data$Col1 <- Access\_data$ShrubClump

Access\_data$Col2<- Access\_data$D0to1

Access\_data$Col3 <- Access\_data$D1to2\_5

Access\_data$Col4 <- Access\_data$D2\_5to5

Access\_data$Col5 <- Access\_data$D5to10

Access\_data$Col6 <- Access\_data$D10to15

Access\_data$Col7 <- Access\_data$D15to20

Access\_data$Col8 <- Access\_data$D20to25

Access\_data$Col9 <- Access\_data$D25to30

Access\_data$Col10 <- Access\_data$D30to35

Access\_data$Col11 <- Access\_data$D35to40

Access\_data$Col12 <- Access\_data$Dgt40

glimpse(Access\_data)

Steps 4 and 5 generate the EventID and LocationID and are identical to the methods used for VIBI herbaceous above.

#################

#

# Step 4a - Generate EventID from EditDate

#

#################

Access\_data <- Access\_data |>

mutate( EventID = str\_c( 'CUVAWetlnd', EditDate)) |>

mutate(EventID = str\_replace\_all(EventID, "-", "")) |>

mutate(NumMonth = str\_sub(EventID, start = 15L, end = -3L))

glimpse(Access\_data)

#################

#

# Step 4b - Replace numeric month with text month abbreviation

#

#################

Months\_LUT <- read\_csv("Months\_LUT.csv")

Access\_data <- Access\_data |>

left\_join(Months\_LUT, join\_by(NumMonth))

Access\_data <- Access\_data |>

mutate(EventID\_left = str\_sub(EventID, start = 1L, end = -5)) |>

mutate(EventID\_right = str\_sub(EventID, start = 17, end = -1))

Access\_data <- Access\_data |>

mutate(EventID = str\_c(EventID\_left, TxtMonth, EventID\_right))

#################

#

# Step 5 - create the LocationID column from the FeatureID column

# and a lookup table from HTLNWetlands

#

#################

Locations\_LUT <- read\_csv("tbl\_Locations\_20230316.csv")

glimpse(Locations\_LUT)

Access\_data <- Access\_data |>

left\_join(Locations\_LUT, join\_by(FeatureID))

glimpse(Access\_data)

In Step 6, we are taking the columns created in Step 3 and running them through the pivot\_longer() function. Step 3 and step 6 together generate a normalized form that can be attached to tbl\_VIBI\_woody.

#################

#

# Step 6 - set up columns before normalization, then pivot longer

#

#################

Access\_data <- Access\_data |>

select(EventID, LocationID, FeatureID, Module\_No, WoodySpecies,

EditDate, WoodySiteName, Col1, Col2, Col3, Col4, Col5,

Col6, Col7, Col8, Col9, Col10, Col11, Col12)

glimpse(Access\_data)

# pivot longer (normalize)

Access\_data <- Access\_data |>

pivot\_longer(

cols = starts\_with("Col"),

names\_to = "DiamID",

values\_to = "Count",

values\_drop\_na = TRUE

)

glimpse(Access\_data)

Step 7 attaches the diameter code values which are required in the database to join tlu\_WoodyPlants to tbl\_VIBI\_woody. The lookup table tlu\_WoodyPlants in the database contains diameter ranges and midpoint values.

#################

#

# Step 7 - Join the diameter information from a LUT

#

#################

Diam\_LUT <- read\_csv("Diam\_LUT.csv")

glimpse(Diam\_LUT)

Access\_data <- Access\_data |>

left\_join(Diam\_LUT, join\_by(DiamID))

Step 8 serves as a end-to-end test of the pivot\_longer process in steps 3 and 6. The control is the loadfile dataframe which was created in Step 1. The test is based on the total count for each diameter class before and after the pivot\_longer process is applied. The totals for each diameter class before and after must match exactly.

#################

#

# Step 8 - Validate normalization and join using

# Sum of counts in initial load file

# against total\_counts for each diameter

# in final version

#

#################

Initial\_load <- load\_file |>

select(ShrubClump, D0to1,

D1to2\_5, D2\_5to5, D5to10, D10to15, D15to20, D20to25, D25to30, D30to35,

D35to40, Dgt40)

colSums(Initial\_load, na.rm=TRUE)

Access\_data |>

group\_by(Diam\_Desc) |>

summarize(

total\_count = sum(Count)

)

Steps 9a and 9b below process missing values and duplicate records using the same methods as those in the VIBI herbaceous script. Step 10 loads the files into the load file "Load\_VIBI\_woody\_2023.xlsx".

#################

#

# Step 9a - Substitute NA with -9999 in CoverClass and CoverClassAll

# Then remove those with -9999 in CoverClass

#

#

#################

glimpse(Access\_data)

Access\_data$Count <- Access\_data$Count |> replace\_na(-9999)

Access\_data |>

filter(Count == -9999)

Access\_data <- Access\_data |>

filter(Count != -9999)

# then test

Access\_data |>

filter(Count == -9999)

##########

#

# Step 9b - check for duplicates

#

##########

# test for dups

Access\_data |>

count(EventID, LocationID, FeatureID, Module\_No, WoodySpecies, DiamID,

Count, Diam\_Code, Diam\_Desc) |>

filter(n > 1)

# Remove dups with distinct()

Access\_data <- Access\_data |>

distinct(EventID, LocationID, FeatureID, Module\_No, WoodySpecies, DiamID,

Count, Diam\_Code, Diam\_Desc)

Access\_data

# test for dups

Access\_data |>

count(EventID, LocationID, FeatureID, Module\_No, WoodySpecies, DiamID,

Count, Diam\_Code, Diam\_Desc) |>

filter(n > 1)

##########

#

# Step 10 - Write load file

#

##########

Access\_data <- Access\_data |>

select(EventID, LocationID, FeatureID, Module\_No, WoodySpecies, Diam\_Code,

Count)

writexl::write\_xlsx(Access\_data, "Load\_VIBI\_woody\_2023.xlsx")

Generating the load file: "Load\_VIBI\_BigTrees\_2023.xlsx

The load file for the database table tbl\_BigTrees is described below.

################################################################################

#

# Survey123\_VIBI\_Big\_trees\_final.R

#

# Gareth Rowell, 2/29/2024

#

# This script converts csv files exported from Survey123 VIBI woody big tree

# data to create a file that can be loaded into MS Access and directly

# appended to the tbl\_BigTrees table.

#

################################################################################

library(tidyverse)

# setwd("./VIBI-BigTrees")

Step 1 below shows the code for loading the big tree csvs. Note that these are the exact same csv files used for tbl\_VIBI\_Woody except we will only be using the last columns for diameter type > 40 cm. So Step 1 is essentially identical to Step 1 in the VIBI woody section above.

#################

#

# Step 1 - load spreadsheet csv files and appended them

# Note - species codes were only used in CUVA\_VIBI\_woody1.csv

# and also CUVA\_VIBI\_woody3.csv

# They were joined to WoodySpecies\_LUT2.csv to create WoodySpecies

#

#################

# species codes were only used in CUVA\_VIBI\_woody1.csv

# join to WoodySPecies\_LUT2.csv to create WoodySpecies

load\_file1 <- read\_csv("CUVA\_VIBI\_woody1.csv")

problems(load\_file1)

WoodySpecies\_LUT <- read\_csv("WoodySpecies\_LUT2.csv")

load\_file1 <- load\_file1 |>

left\_join(WoodySpecies\_LUT, join\_by(SpeciesCode))

glimpse(load\_file1)

glimpse(WoodySpecies\_LUT)

# view(load\_file1)

# check for NAs in WoodySpecies

load\_file1 |>

select(SpeciesCode, WoodySpecies) |>

filter(is.na(WoodySpecies)) |>

distinct()

load\_file2 <- read\_csv("CUVA\_VIBI\_woody2.csv")

problems(load\_file2)

load\_file3 <- read\_csv("CUVA\_VIBI\_woody3.csv")

problems(load\_file3)

load\_file3 <- load\_file3 |>

left\_join(WoodySpecies\_LUT, join\_by(SpeciesCode))

glimpse(load\_file3)

load\_file4 <- read\_csv("CUVA\_VIBI\_woody4.csv")

problems(load\_file4)

glimpse(load\_file1)

glimpse(load\_file2)

glimpse(load\_file3)

glimpse(load\_file4)

Access\_data <- bind\_rows(load\_file1,load\_file2)

glimpse(Access\_data)

Access\_data <- bind\_rows(Access\_data,load\_file3)

glimpse(Access\_data)

Access\_data <- bind\_rows(Access\_data,load\_file4)

glimpse(Access\_data)

At Step 2, we begin processing the columns specifically for big trees. Note that the data columns used here are Dgt40\_1, Dgt40\_2, Dgt40\_3, Dgt40\_4, Dgt40\_5.

##########

#

# Step 2 - select columns for Access import

# for big trees, need columns 40\_1 onwards

#

##########

Access\_data <- Access\_data |>

select(WoodyModule, WoodySpecies, EditDate, WoodySiteName,

Dgt40\_1, Dgt40\_2, Dgt40\_3, Dgt40\_4, Dgt40\_5)

glimpse(Access\_data)

Steps 3 through 5 include renaming columns and creating EventID and LocationID. These are essentially the same as those described for VIBI herbaceous and VIBI woody.

##########

#

# Step 3 - create columns for FeatureID and Module\_No

# and convert date format

#

##########

Access\_data <- Access\_data |>

mutate( FeatureID = WoodySiteName) |>

mutate(EditDate = (EditDate <- as.Date(EditDate, format = "%m/%d/%Y"))) |>

mutate( Module\_No = WoodyModule )

Access\_data

##########

#

# Step 4 - Generate EventID from EditDate

#

##########

Access\_data <- Access\_data |>

mutate( EventID = str\_c( 'CUVAWetlnd', EditDate)) |>

mutate(EventID = str\_replace\_all(EventID, "-", "")) |>

mutate(NumMonth = str\_sub(EventID, start = 15L, end = -3L))

glimpse(Access\_data)

#################

#

# Step 4b - Replace numeric month with text month abbreviation

#

#################

Months\_LUT <- read\_csv("Months\_LUT.csv")

glimpse(Months\_LUT)

Access\_data <- Access\_data |>

left\_join(Months\_LUT, join\_by(NumMonth))

Access\_data <- Access\_data |>

mutate(EventID\_left = str\_sub(EventID, start = 1L, end = -5)) |>

mutate(EventID\_right = str\_sub(EventID, start = 17, end = -1))

Access\_data <- Access\_data |>

mutate(EventID = str\_c(EventID\_left, TxtMonth, EventID\_right))

glimpse(Access\_data)

##########

#

# Step 5 - create the LocationID column from the FeatureID column

# and a lookup table from HTLNWetlands

#

##########

Locations\_LUT <- read\_csv("tbl\_Locations\_20230316.csv")

glimpse(Locations\_LUT)

Access\_data <- Access\_data |>

left\_join(Locations\_LUT, join\_by(FeatureID))

glimpse(Access\_data)

Step 6 uses the pivot\_longer() function in the same manner as Steps 3 and 6 for woody plants. Here we are converting Dgt40\_1, Dgt40\_2, Dgt40\_3 … to two columns, one called Trees and the second one called DBH which contains measures of DBH in inches??.

##########

#

# Step 6 - Normalize DBH columns

#

#

##########

# Rename columns using DiamID values for pivot\_longer

Access\_data$Tree1 <- Access\_data$Dgt40\_1

Access\_data$Tree2 <- Access\_data$Dgt40\_2

Access\_data$Tree3 <- Access\_data$Dgt40\_3

Access\_data$Tree4 <- Access\_data$Dgt40\_4

Access\_data$Tree5 <- Access\_data$Dgt40\_5

Access\_data <- Access\_data |>

mutate( Scientific\_Name = WoodySpecies) |>

mutate( SampleDate = EditDate) |>

select(EventID, LocationID, Module\_No, Scientific\_Name, Tree1,

Tree2, Tree3, Tree4, Tree5)

glimpse(Access\_data)

# pivot longer (normalize)

Access\_data <- Access\_data |>

pivot\_longer(

cols = starts\_with("Tree"),

names\_to = "TreeName",

values\_to = "DBH",

values\_drop\_na = TRUE

)

glimpse(Access\_data)

Step 7 is a check for duplicate records which is basically the same code as used in previous scripts for detecting and eliminating duplicate records. It ends with writing the dataframe to "Load\_VIBI\_BigTrees\_2023.xlsx".

##########

#

# Step 7 - check for duplicates

#

##########

# test for dups

Access\_data |>

select(EventID, LocationID, Module\_No, Scientific\_Name, DBH)

Access\_data |>

group\_by(EventID, LocationID, Module\_No, Scientific\_Name, DBH) |>

summarize(

n = n(),

) |>

filter(n > 1)

# Remove dups with distinct()

Access\_data <- Access\_data |>

distinct(EventID, LocationID, Module\_No, Scientific\_Name, DBH)

Access\_data

# test for dups

Access\_data |>

group\_by(EventID, LocationID, Module\_No, Scientific\_Name, DBH) |>

summarize(

n = n(),

) |>

filter(n > 1)

#view(Access\_data)

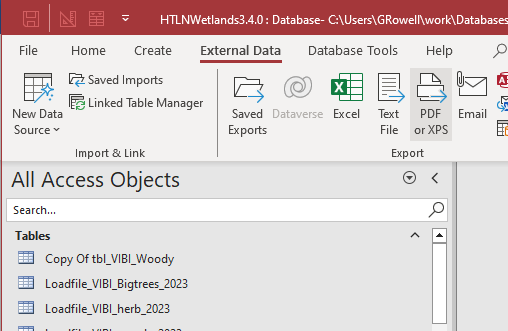
writexl::write\_xlsx(Access\_data, "Load\_VIBI\_BigTrees\_2023.xlsx")

PROCEDURES – CONTINUED

Importing the load files into MS Access

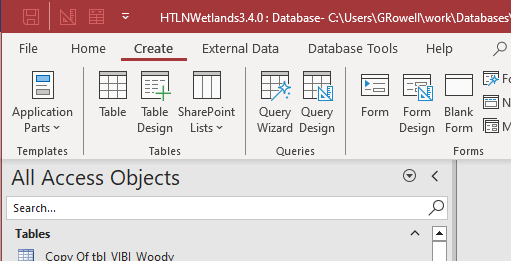
The process for importing the load files is basically the same for tbl\_BigTrees, tbl\_VIBI\_herb and tbl\_VIBI\_woody. We will import the load file generated by the R script then append it to the data table. The detailed steps are explained below for tbl\_VIBI\_herb, but they are basically the same for tbl\_BigTrees and tbl\_VIBI\_woody.

1. Backup the original table tbl\_VIBI\_herb in Access using copy and paste in the Navigation pane. Name the backup table tbl\_VIBI\_herb\_orig. This will allow you do go back and re-run the append operation if anything goes wrong.
2. Import the spreadsheet Load\_VIBI\_BigTrees\_2023.xlsx into Access. Go to the External Data ribbon and click on New Data. Follow the wizard to import a spreadsheet into a table. Name the new table Loadfile\_VIBI\_herb\_2023.



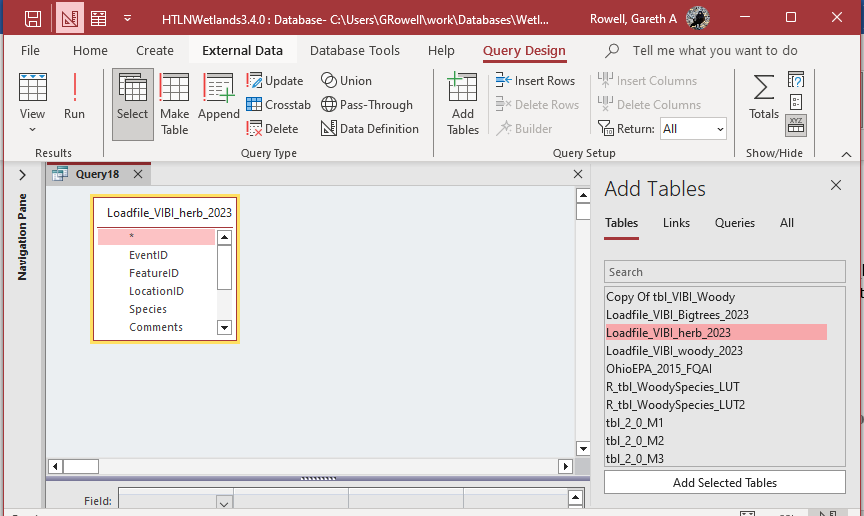


1. Append the table Loadfile\_VIBI\_herb\_2023 to the data table called tbl\_VIBI\_herb. We will use an append query to accomplish this. Go to the Create ribbon and click on Query Design.



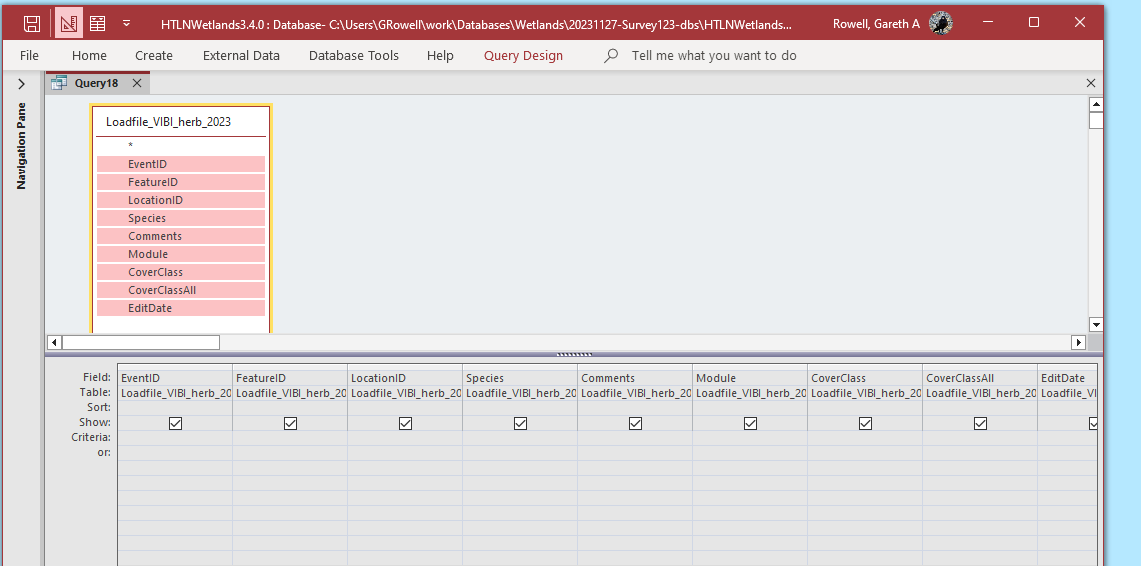


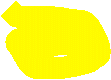
1. Once you are in the Design view of the query editor, double click on the Loadfile\_VIBI\_herb\_2023 table in the Add Tables box.



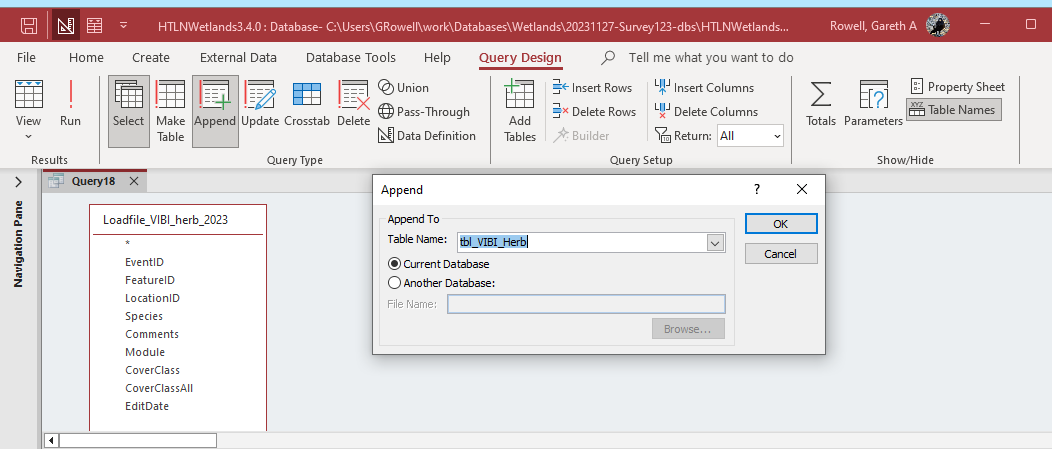


1. Select all the fields in the table box and drag them to the field design panel.



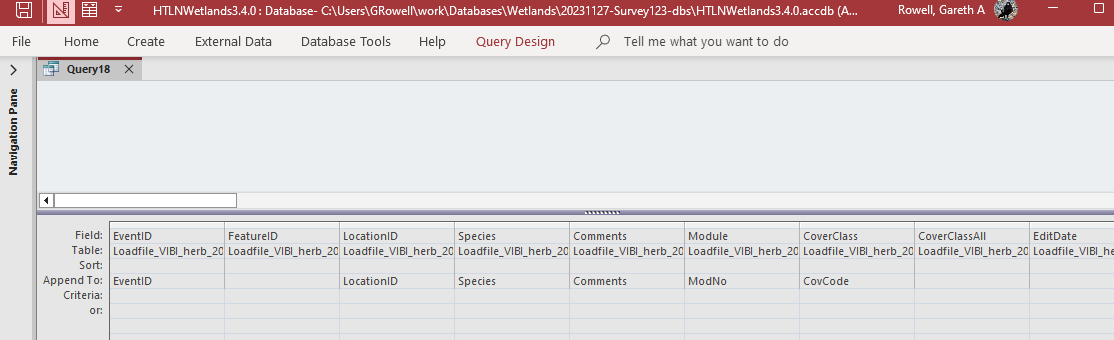


1. Click on the Append Query button on the query design ribbon and select the table tbl\_VIBI\_herb



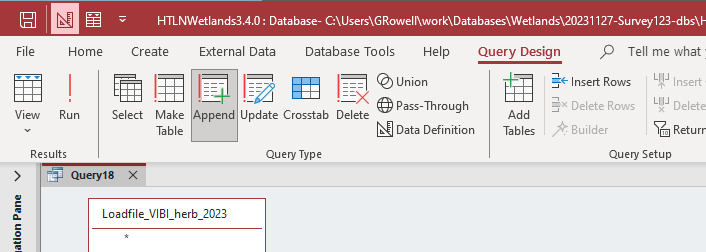


1. When you click on the OK button, Access will attempt to match column names. For other columns, you will have to match them up manually (see highlighted columns).





1. Once you have matched all the columns, you can click on the run icon. Be sure to click on it just once. If you click on it multiple times, it will append additional copies of your load file to tbl\_VIBI\_herb. One of the end2end tests (below) is to verify that you have only appended once (test for duplicates).





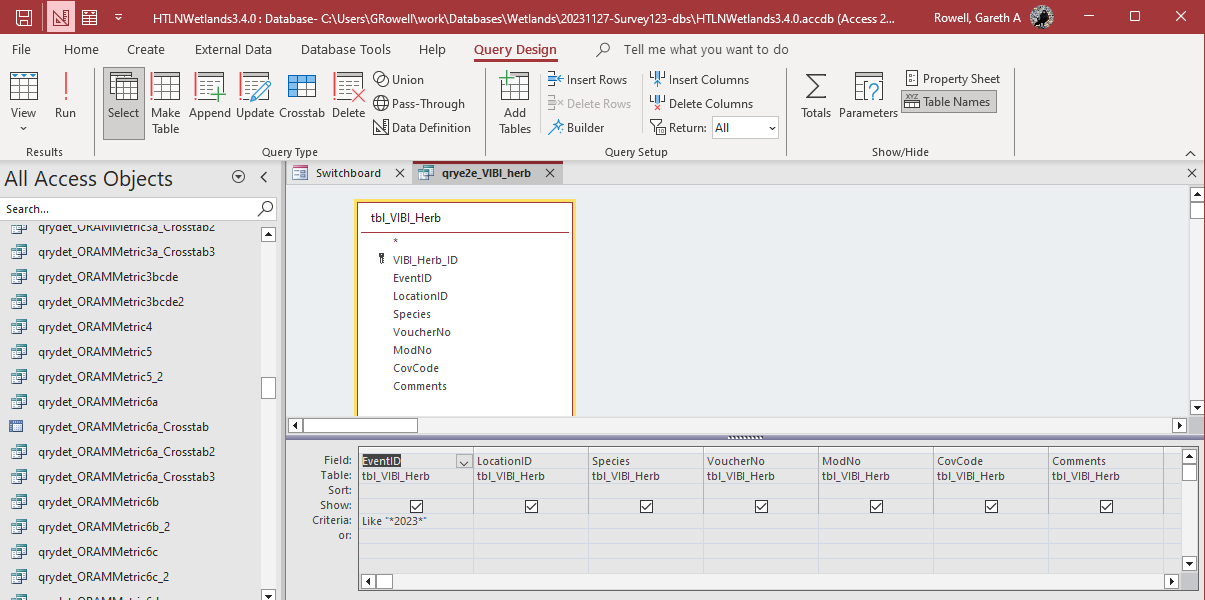
1. You can repeat these same procedures to append "Load\_VIBI\_woody\_2023.xlsx” to tbl\_VIBI\_woody and “"Load\_VIBI\_BigTrees\_2023.xlsx" to tbl\_BigTrees.

PROCEDURES – CONTINUED

Running End-to-End tests

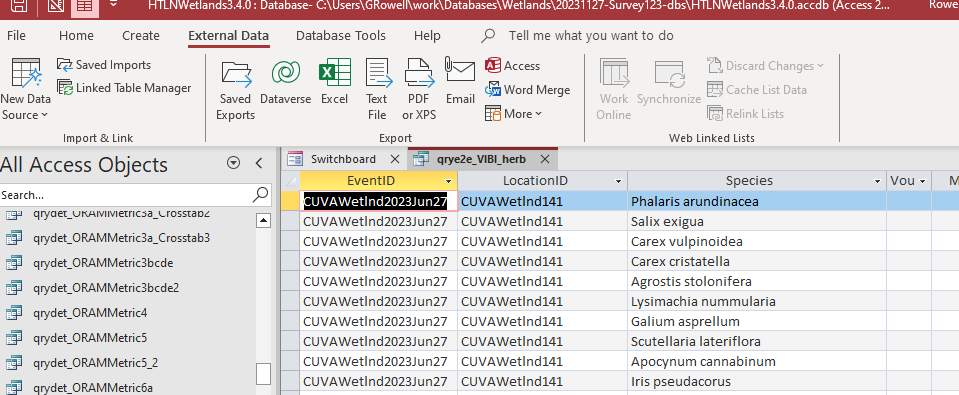
Once you have appended data to the VIBI tables, you really need to validate the new data. To do this, we compare the dataframe generated by R script with the data loaded in the VIBI table. We will step through this process for VIBI herbaceous data but the process is the same for VIBI woody and Big Trees. (The complete scripts for all three end-to-end tests are in Appendix 3).

1. Open the query called qrye2e\_VIBI\_herb in design view. Note that the EventID is filtered for Like “\*2023\*”.





1. This will pull out just the 2023 data. Change the query from Design to View mode. Be sure to save the query just in case you’ve made any modifications you need to keep. Go to the External Data ribbon and the Text File export button. Follow the export wizard steps to export the data to csv format. Save the file as “qrye2e\_VIBI\_herb.csv”





1. Load the script VIBI\_herb\_end2end.R in RStudio and examine the script. The first 6 steps are identical to the script Survey123\_VIBI\_herb\_final.R Run the first 6 steps to generate the load file Access\_data dataframe. Starting at the comment “End2End test begins here”, we load the export data from Access into an R dataframe called end2end. We make some minor changes in column names so they match up exactly.

#################

#

# Step 6 - Clean up columns and write load file

#

#################

# clean up columns

Access\_data <- Access\_data |>

select(EventID, FeatureID, LocationID, Species, Comments, Module,

CoverClass, CoverClassAll, EditDate )

# writexl::write\_xlsx(Access\_data, "Load\_VIBI\_herb\_2023.xlsx")

#------------------------------------------------------------------------------

# End2End test begins here

# load end2end data

end2end <- read\_csv("qrye2e\_VIBI\_herb.csv")

problems(end2end)

glimpse(end2end)

glimpse(Access\_data)

# matching column names

Access\_data <- Access\_data |>

mutate(

ModNo = Module,

CovCode = CoverClass

)

glimpse(Access\_data)

# Selecting the columns that matter

Access\_data <- Access\_data |>

select(EventID, LocationID, Species, ModNo, CovCode)

end2end <- end2end |>

select(EventID, LocationID, Species, ModNo, CovCode)

1. Next, we test for duplicates in both the initial load file (Access\_data) and the end2end exported data from MS Access.

# test for duplicates in Access\_data and End2end dataframes.

Access\_data |>

count(EventID, FeatureID, Species, Module, CoverClass) |>

filter(n > 1)

# Need to resolve these duplicates with Sonia.

# Also, there's a bunch of null values in the CoverClass variable

Access\_data |>

filter(CoverClass == -9999)

glimpse(Access\_data)

# matching column names

Access\_data <- Access\_data |>

mutate(

ModNo = Module,

CovCode = CoverClass

)

# Selecting the columns that matter

Access\_data <- Access\_data |>

select(EventID, LocationID, Species, ModNo, CovCode)

end2end <- end2end |>

select(EventID, LocationID, Species, ModNo, CovCode)

1. Compare record counts for the load dataframe Access\_data and the export dataframe called end2end

# compare record counts

Access\_data |>

count(EventID, LocationID, Species, ModNo, CovCode)

end2end |>

count(EventID, LocationID, Species, ModNo, CovCode)

1. Next, test for presence of duplicates both in the source Access\_data dataframe and in the export end2end dataframe end2end

# test for duplicates in the Access data

Access\_data |>

count(EventID, FeatureID, Species, Module, CoverClass) |>

filter(n > 1)

end2end |>

count(EventID, FeatureID, Species, Module, CoverClass) |>

filter(n > 1)

# Test for missing data in cover class

Access\_data |>

filter(CoverClass == -9999)

end2end |>

filter(CoverClass == -9999)

end2end <- read\_csv("qrye2e\_VIBI\_herb.csv")

problems(end2end)

glimpse(end2end)

glimpse(Access\_data)

# matching column names

Access\_data <- Access\_data |>

mutate(

ModNo = Module,

CovCode = CoverClass

)

# compare record counts

Access\_data |>

count(EventID, LocationID, Species, ModNo, CovCode)

end2end |>

count(EventID, LocationID, Species, ModNo, CovCode)

# Selecting the columns that matter

Access\_data <- Access\_data |>

select(EventID, LocationID, Species, ModNo, CovCode)

end2end <- end2end |>

select(EventID, LocationID, Species, ModNo, CovCode)

# testing for PK - unique no-nulls

Access\_data |>

count(EventID, LocationID, Species, ModNo, CovCode) |>

filter(n > 1)

end2end |>

count(EventID, LocationID, Species, ModNo, CovCode) |>

filter(n > 1)

# need to delete duplicates from database and reexport <<<<<<<<<<<<<<<<<<<

# manually delete duplicates from Access\_data and retest

#view(end2end)