Fishnet2 Data Migration

Jeremy Holden

2022-07-11

Contents

Chapter 1

About

The goal of this project is to document the workflow of migrating FN2 projects from the existing *DATA.ZIP* archive to a modern data product. The manual will outline tools for importing, cleaning and outputting a project or series of projects.

1.1 Usage

This manual will focus on the code and associated R packages used to migrate data.

Chapter 2

Introduction

Fishnet2 (FN2) is custom database software used as a Ontario provincial standard for over two decades. The system applied a common database structure to accommodate a variety of fishery surveys (i.e. gill net, trawl, creel).

2.1 Overview

The goal of this manual is to document the process used to develop an R-based reproducible workflow for importing, cleaning and migrating archived FishNet2 project data.

2.2 Basic workflow

1. The R package development structure is used as a framework for organizing the files and workflow.
2. FN2 archive files store relational data tables in a DBF format in a compressed format (*DATA.ZIP*). These files are unzipped to memory then read in to an R environment.

3. Data types are imported as *strings* and are converted to usable data types (i.e. *integers*, *dates*, *double*).

4. Error checking and data correction is conducted using a series of predefined and project specific queries.

5. Data variables names are standardized across projects.

6. At this point the data can be used or distributed as an R data package.
7. Data can also be exported from the package to other database formats as needed.

2.3 Why use R?

Generally, a code-based approach (R or other programming languages like Python) are advantageous over other graphic user interface (GUI) approaches that rely on point-and-click actions that are often not documented and are certainly not reproducible. The primary advantage of using R is that it is a commonly used programming language among fisheries biologists. The data package approach provides a uniform structure and additionally means that the data is immediately accessible in the desired format for data analysis and other analytical products (Rmarkdown reports, flexdashboards). It also means that the workflow is in a familiar environment (RStudio) and can leverage existing programming skills for data cleaning. The R package environment also provides the advantage of version control (using git) and deployment (from github).

Chapter 3

R package set up

Building R packages requires that Rtools be installed.

Additionally, there are several R packages that will be required throughout the workflow.
1. devtools
2. usethis
3. roxygen

Data import and analysis packages will be discussed in later sections.

3.1 Basic package initialization

1. From RStudio, select "create new project" -> "New Directory" -> "New R Package". It is recommended that during this process a git repo also be created.

2. Edit the DESCRIPTION file. At this point you can do a *build and reload* to check the package structure is working.

3. Create a README file (`usethis::use_readme_rmd`). Edit details as required. Render the document to create the README.md file that is used by github to display project details on the repo start up page.

4. FishNet2 archive files are stored in a special directory (*data-raw*) that will be ignored in the package build process. This allows the package structure to be self-contained but not included in the distributed package files. The directory structure is created using (`usethis::use_data_raw()`).

The package structure is now complete. Successive steps will be discussed in the following chapters.

Chapter 4

Data Import

4.1 Project Structure

A *FN2* directory should be created within the *data-raw* folder. This provides a package relative directory structure to access the raw data. Each FishNet project should be stored in an individual directory named according to the *PRJ_CD* naming convention. All the import steps and migration of data to the package structure is contained in a script (i.e. `import_data_build_data_package.R`) saved in the *data-raw* directory.

```
+-- data-raw
|   \-- FN2
|       +-- IA00_FW4
|       |   \-- DATA.ZIP
|       +-- IA01_FW7
|       |   \-- DATA.ZIP
|   \-- import_data_build_data_package.R
```

4.2 Importing Data

The *DATA.ZIP* file contains the *dbf* files for each of the tables of the relational database structure in the FishNet project structure. `gfsR` package has been developed with several functions designed to unzip each *DATA.ZIP* file in to memory and then import each of the appropriate *FN* tables (ex. *FN121*, *FN125*, ...) as an object in R. Additionally, many of the *import_* functions have pre-processing code that applies common code fixes to the structure that results from the *dbf* file format (ex. all fields stored as character strings). The available import functions and code can be found at the gfsR github page.

Multiple projects can be read in and compiled with a few lines of code.

```
# Import and clean raw FWIN data
# Generate project list ----
fndata_folder <-"data-raw/FN2"
fndata_zips <- dir(fwin_folder, recursive = T, full.names = T)
fndata_zips <- grep(fwin_zips, pattern = "DATA.ZIP", value = T)

# Import FN2 files ----
library(gfsR)
fndata_raw <- import_index_series(fndata_zips)

# View sample of the data ----
lapply(fndata_raw, head)

# move all tables to global environment ----
list2env(fndata_raw, envir = .GlobalEnv)
```

4.3 data-raw to data

At this point, the data can be made available in the R package but generally there will be some data cleaning required (see data cleaning)[1]. However, since the process of moving data from *raw-data* to *data* that is more generically part of the package building process; those general steps will be covered here.

The general convention that has been adopted is to store the data in an R list with the list given a descriptive name and then each table saved as a list item named with the FN2 table name convention similar to the structure used by `gfsR` when the data is imported.

```
fndata <- list(
  FN011 = FN011,
  FN012 = FN012,
  FN026 = FN026,
  FN121 = FN121,
  FN123 = FN123,
  FN125 = FN125,
  FN126 = FN126,
  FN127 = FN127
)
```

[1]If not for the data cleaning the `fndata_raw` variable could be used directly in `use_data()` and explains why there appears to be some redundancy here between moving tables from `fndata_raw` to an identical list for export `fndata`.

Once the main data object is created for export the `usethis::use_data()` command is used to make the object available in the package.

```
usethis::use_data(fndata, overwrite = TRUE)
```

This function will write `fndata` to a new directory in the package as an rda file that is included in the package once compiled.

```
+-- data
|   +-- fndata.rda
```

4.4 Data Documentation

At this point the function provides a reminder that the data needs documentation. The documentation page is build starting with an R script that includes special header information that is used by *devtools* and *roxygen2* to develop the documentation files. The file should be saved in the *R* directory.

```
+-- R
|   +-- fndata.R
```

The general *roxygen2* for documentation is:

```
#' fndata
#' @description give a description of the data here
#' @format list containing FN2 data tables
#' @examples
#' data(fndata)
#' lapply(fndata, head)

"fndata"
```

Once completed, the `devtools::document()` function will created the necessary changes in the package structure. It's likely that the first time this process is conducted it will generate a warning message related to the NAMESPACE file. At least point, it is best to delete the existing NAMESPACE file and re-run the `devtools::document()` command which will generate a new NAMESPACE file. In addition to updating the NAMESPACE file, `document()` generates the help file in the *man* folder.

```
+-- man
|   +-- fndata.Rd
```

At this point, the package is ready to be built. This can be done in RStudio using the *Install and Restart* button or using `devtools::build()` command. At this point it is a good idea to check the package build by opening a new RStudio environment (not in the existing package environment) and check the package contents.

```
library(fndata)
data(fndata)
lapply(fndata, head)
```

Chapter 5

Data cleaning

This chapter will discuss the error checking functions available in `rprocval` or other tools that might be used on a project by project case. Spatial queries are discussed in a future chapter. Additionally, changes to variable names to meet broader program area standards (i.e. `SITRAN` renamed as `SECCHI`) are included here as part of the data cleaning process as are changes to coded variables (i.e. *SEX* = UNK vs *SEX* = 9).

5.1 Data Cleaning Structure

Scripts used to identify and correct errors that remain in the FN2 *DATA.ZIP* file. By conducting the cleaning in the package build process changes to the raw data (the *DATA.ZIP*) are done in a manner that is documented through the code base. Additionally, by establishing the data cleaning pipeline in a reproducible way is allows collaboration and future changes to be easily incorporated in to the package build structure in a reproducible manner. Data changes (with appropriate versioning) either in data tables, fields or values are easily added or removed in the package build process.

5.2 Identifying Errors

Data errors can be introduced in a number of manners throughout the data collection and data entry process. Some error types are structural (i.e. orphaned child record), systemic (i.e. fork length recorded as total length, units reported in grams rather than kilograms), or transcription or data entry errors. The extent to which these can be identified and corrected varies. This section is not intended to provide a complete data cleaning recipe book, rather to provide some general tools for common error checking (`rprocval`) but more specifically to develop a structure of data cleaning that is consistent with the package structure pipeline and, most importantly, is documented and reproducible.

5.3 Cleaning Scripts Structure

As alluded to in the data import chapter, cleaning occurs following the import of the raw FN2 data but before writing the *rda* file to the *data* folder. The proposed work flow is to create separate cleaning files for (at least) each FN table. Each of these scripts should be sourced (i.e. `source("data-raw/clean_FN121.R")`) from `import_data_build_data_package.R`.

```
+-- data-raw
|   \-- FN2
|       +-- IA00_FW4
|       |   \-- DATA.ZIP
|       +-- IA01_FW7
|       |   \-- DATA.ZIP
|   \-- import_data_build_data_package.R
|   \-- clean_FN121.R
|   \-- clean_FN125.R
```

5.4 Two approaches to changing data

Changes to variable names and coded variables are easily done using many conventional R commands (see: `dplyr::rename`; `dplyr::case_when`). These types of changes are easily included within the cleaning script.

Larger or more complex changes often requires several intermediate steps and possibly the creation of intermediate variables. Similarly, test done in `rprocval` often return a complete data frame of data that has failed a test. When such changes are required it is suggest that `dplyr::rows_update` be used. This approach utilizes key field matching to overwrite a single field in the main table of interest (i.e *FN125*) rather than rather other approaches that require removing data and then rebuilding the table (using `bind_rows` or `rbind`).

Some systemic errors are most efficiently made directly in the code base. Frequently however there will be no easy programatic fix and individual records and values will need to be made. In such instances, it is helpful to create a look up table of data changes. The format of the look up table should include the key fields, the column that requires updating and the value to be updated. This minimal table is preferred over a spreadsheet with all the records and all the fields as it is far more explicit as to what value is changed and is less prone to accidentally inducing more errors through data editing.

Table 5.1: Example look-up table structure

PRJ_CD	SAM	EFF	SPC	FISH	Attribute	Value
LOA_IA12_001	0011	038	334	001	FLEN	123
LOA_IA12_001	0011	038	334	006	RWT	1254
LOA_IA12_001	0021	038	131	005	TLEN	345

CAUTION: editing data in Excel, even using the above approach can lead to unexpected format changes in the data. *SQLite* is a lightweight database platform that connects seamlessly with R. Several open source *SQLite* software packages exist [https://sqlitebrowser.org/] or see DataChangesDB for a lightweight custom GUI designed to make FN125 data changes quick and efficient.

With data changes recorded using the suggested structure each variable requires separate `rows_update` queries which adds slightly more code than a single update query of a table with all columns but the explicit nature reduces errors and provides better documentation of the changes that were made.

5.5 FN121 Errors

- list some common errors
- explain the test and output
- methods for fixing the data

Chapter 6

Data distribution

This could address a few things like uploading to GLIS, writing files to Access or storing data as an R package.

Chapter 7

Spatial tests

This a good section to describe how to do spatial queries to check data with custom shapefiles and kml polygons.