FinCatch Documentation

Keith Hurley et al

Invalid Date

Table of contents

Home

This is documentation for the FinCatch Data System of the Nebraska Game and Parks Fishery Division. FinCatch stores and provides analysis of standard lentic fisheries population surveys. This set of documentation is an accumulation of both developmental and instructional documentation.

Part I

FinCatch System

1 System Components

The FinCatch data system is comprised of a number of separate components, including:

1.1 FinCatch

FinCatch is the central website that provides data management capabilities as well as links to other components. FinCatch is written in the asp.mvc framework of .NET 6.

1.2 FinCatch Database

The backend database for FinCatch is built in Microsoft SQL Server.

1.3 FinCatchDE

1.4 FinCatchAG

1.5 FinCatchRA

1.6 FinCatchAnalysis R Package

1.7 FinCatchAccess R Package

1.8 FinCatchWebApi

2 Sampling Restrictions

- All length subsampling should occur at the SAMPLE level.
- All weight and age subsampling may occur at the SAMPLE or SURVEY level.
- Every individual of a species in the processed species list has either been measured or counted. If "All Species" (code 0) is in the processed species list, EVERY individual encountered during sampling was either measured or counted.
- Odd species (those not found in the processed species list) may be measured or counted and entered into FinCatch. However, they will only be reported in raw data summaries and NOT included in calculations and analysis as they cannot be assumed to have been collected in their entirety (i.e. all individuals in all samples).
- If the "Use For CPUE" flag is FALSE (unchecked), then NONE of the data for any species in the sample will be used for catch rate calculations. If the flag is TRUE, then ALL species in the sample will be included for catch rate calculations.

3 Data Design

3.1 Processed Species

A list of species processed during sampling is collected at the survey level and applies to all fish samples attached to the survey. Any species included in the processed species list and NOT found in the data carries the assumption of a zero-catch. If a species is found in the data and NOT in the process species list, no assumption is made that the species was consistently processed throughout the survey.

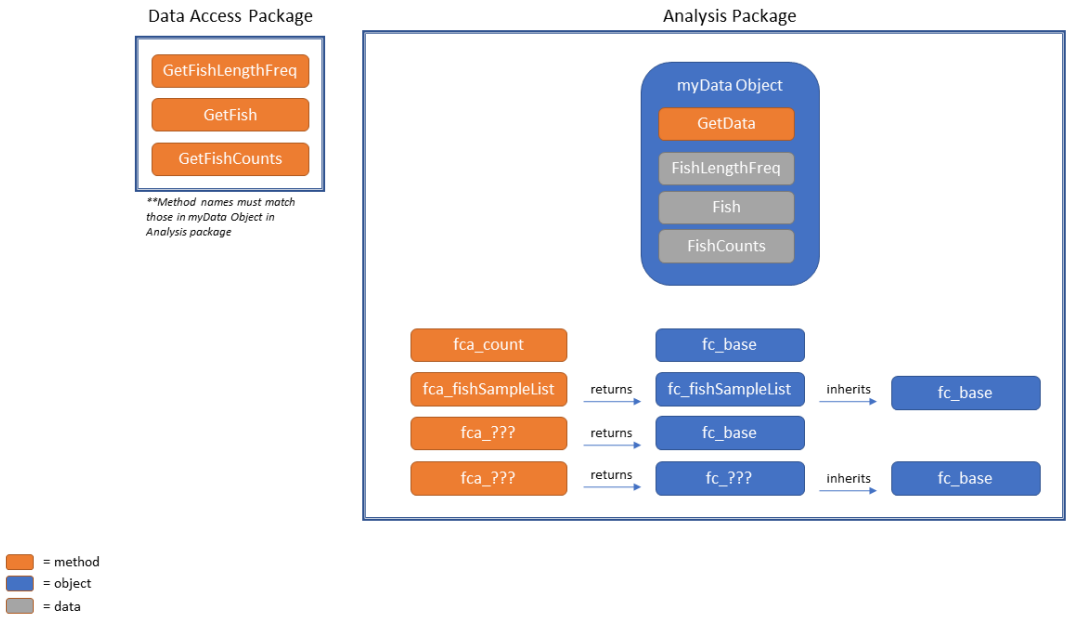
All species (code 0) can be included in the processed species list. The inclusion of this code carries the assumption that ALL fish collected during the survey (in ALL samples) were processed. During analysis, the processed species list will be expanded to include all species found in the data.

> **Note**
>
> Important caveat: expanded to include all species in the analysis and not just in the survey. This allows the following example to work: all Omaha area surveys were analyzed for red ear population structure - but redear were only found in half the reservoirs. By expanding the processed species list by analysis data rather than grouped within survey, zero's will be included for redear in all surveys in the analysis that have all species (code 0) in the processing list.

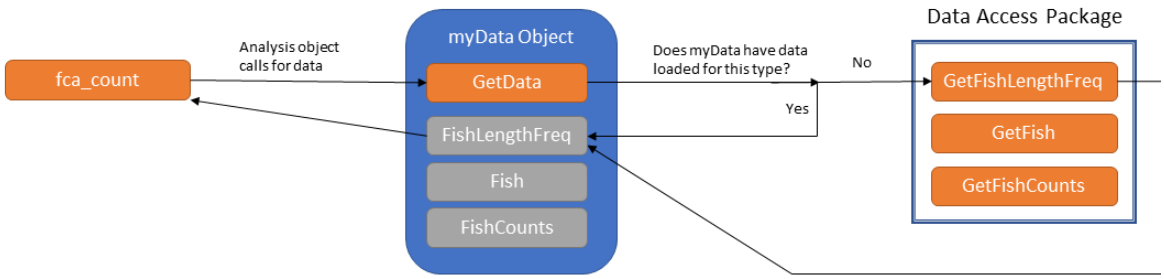
3.2 Extrapolation of Length Subsampling

4 Architecture

4.1 Packages



4.2 WorkFlow



4.3 Principles

By Using myData Object:

- Only data that's needed is loaded
- Data is cached
- Data is only retrieved once for entire analysis string

By Using Separate Data Access Package:

- Isolates and generalizes data access
- Allows analysis code to use different sources of data
- Allows different scripts to use different authentications for data access

5 FinCatch Users

5.1 Auth0 User Authentication

FinCatch uses the Auth0 (https://www.auth0.com) for authentication and authorization. At-will, user-requested new user creation is not supported. New users can be created by a tenant admin with Auth0. Users will be sent an email to verify their address and future FinCatch usage is not possible until the verification is complete. Users should also be assigned the appropiate authorization roles.

5.2 FinCatch User

In addition to Auth0 user authentication, FinCatch website, FinCatchAG, and FinCatchRA users will also need a profile in the FinCatch system. This profile is meant to decouple FinCatch from the Auth0 system in case a new authentication service needs to be implemented. Upon the first login to the FinCatch website, a new Auth0 user with a verified email will be redirected to a page to create their profile. After creation of their profile, a FinCatch DataAdmin will need to log in and "Activate" their profile.

5.3 How to Create A New User

1) FinCatch administrator logs into Auth0 and creates new user. The appropiate roles (DataAdmin, DataWorker, DataSupervisor) should be added to the new user. The new user will need to be supplied their password.

2) New user will receive an email with instructions on how to verify their email address. The must complete the verification process.

3) New user should log into https://FinCatch.outdoornebraska.gov using the password supplied by the FinCatch administrator.

4) Upon logging in, the new user will be redirected to a page to complete their FinCatch profile.

5) A FinCatch administrator must login and activate the new user's profile.

Part II

FinCatchDE

6 FinCatchDE Deployment

6.1 Before Deployment

Change:

1. ConnectionString in AppSettings.json
2. redirectUri at line 45 in previousFishSamples
3. redirectUri at line 164 in previousWaterSamples
4. auth0 settings in clsAuthFunctions, fetchController.cs, and index.tsx

6.2 Deployment Steps

The application needs to be self-contained (See REF-3 and REF-4); you might want to investigate publishing the application through Visual Studio's IIS publishing, but this is the way that has worked best for our development.

1. Open the Projects in Visual Studio
2. Ensure the application runs locally through Visual Studio
3. Right click the solution and select Publish...
4. Select the Web Server (IIS) Target
5. Select Web Deploy Package
6. Put the package location in the same folder as the project
7. Any name will do for the Site Name, we did Fish-Test, and click finish
8. Select Show All Settings
9. Change Deployment mode to Self-contained
10. Change the target runtime to the required runtime (for us that was win-x64)
11. Select Save
12. Click the Publish button

13. Wait for the Publish to be successful

14. Open the Console/Git Bash/PowerShell

15. Navigate to fisheries-field-sampling\FisheriesFieldSampling\FisheriesFieldSampling\client-app\

16. Run npm run build

17. Open File Explorer

18. Navigate to fisheries-field-sampling\FisheriesFieldSampling\FisheriesFieldSampling\client-app\

19. Copy the build folder within the project

20. Navigate to the newly created zip file from the Publish button

21. Extract into a folder named publish

22. Navigate to publish\Content\D_C\GitHub\fish-test\fisheries-field-sampling\FisheriesFieldSampling\Fish x64\PubTmp\Out\client-app folder

23. Replace the build folder with the copied build folder

24. Copy everything within the publish\Content\D_C\GitHub\fish-test\fisheries-field-sampling\FisheriesFieldSampling\FisheriesFieldSampling\obj\Release\netcoreapp3.1\win-x64\PubTmp\Out\ folder

25. Open Remote Desktop

26. Connect to the fincatchag.fishstaff.info computer

27. Open the file explorer on the Remote Desktop

28. Replace the contents of C:\Websites\FinCatchDE with the copied files/folders

29. Open the IIS Manager

30. Restart the FinCatchDE website

Part III

Analysis R Package

7 Overview

Overview

The FinCatchAnalysis (FCA) R package centralizes standard analysis functions for the FinCatch system and promotes DRY and reusable analysis code practices. The FCA package is built on top of R6 classes which provides a standard programming interface for users of the R package. Analysis functions are available for each individual analysis available and results from each analysis function returns results encapsulated in an R6 class object. All public functions in the package are prefixed with "fca_". Package R6 objects are prefixed with "fco_".

7.1 Analysis Functions

Each analysis function in the FCA package provides a single call for an independent analysis and returns a function specific R6 object built on the base fco_ object that provides standard methods and data objects.

7.2 Analysis Return Objects

8 Data Object

The data object serves a couple of functions. First, it provides a central method and common approach for data retrieval from the FinCatch system. It stores the filters used for retrieval and provides a standard diction for calling the data retrieval functions in the FinCatchAccess package. Second, it provides a caching mechanism so that during an analysis chain or report, data will only be downloaded/retrieved from the database once. Additionally, any commonly used calculations (i.e. length-frequency extrapolations) can be defined and cached as well.

> ⚠ Warning
>
> If the filter objects used in creating the data object change, the data object should be destroyed and re-created to ensure the cached data and filters remain synchronized!

8.1 Creating FinCatch Data Objects

8.2 Data Caching And Retrieval

Data is cached in the "data_..." and "codes_..." properties of the object. Data and codes are retrieved from the data object using the "get_data_..." and "get_codes_..." functions. These "get_" functions first check for cached data in the "data_" objects and returns it; otherwise they will retrieve it from the database, cache it in the "data_" objects, and return it. The names of "get_" functions match their counterparts in the "data_" objects. The "data_" objects are exposed to users (rather than hiding them as private objects) so that advanced users can utilize the FinCatchAnalysis package and the data object by directly saving data to the cache.

> ❗ Retrieving Data
>
> Data should always be retrieved using "get_" functions, so that caching works. Trying to retrieve data directly from the "data_" and "code_" cache objects opens up logic errors where lack of data and yet-to-be retrieved data are confused and confounded.

> **Caution**
>
> While data column names may be renamed and table structures combined and split during caching operations, data should not be modified in any way that introduce assumptions or potential bias. These type of operations should use "get_calc_" and "calc_" names as described below!

8.3 Calculated Data

Commonly used calculations are defined in the the data object so that they do not need to be repeated in the same analysis chain or report. One example would be length-frequency extrapolations when part of a fish sample are sub-sampled for length and then extrapolated by fish counts. These calculations are defined and retrieved in "get_calc_..." functions with their results cached in "data_..." objects similar to the "data_" and "code_" objects.

8.4 Warnings and Errors

Warning and error properties exist in the data object. These accumulate potential issues encountered when retrieving data/codes from the database or when performing calculations (such as extrapolated length-freqencies from a small subsample). These errors are printed to the console when the FinCatchAnalysis package is used interactively.

> **Important**
>
> When the FinCatchAnalysis package is used in a non-interactive environment like markdown, quarto, or shiny, it is important that these warnings and errors are explicitly and intentionally shown to the end user as part of the output!

8.5 Available Data

8.6 Available Code

8.7 Available Calcs

9 Warnings and Errors

10 Example Script

```
library(FinCatchAccess)
library(FinCatchAnalysis)

#use filters gadget to make a filter object
myFilters<-fcacc_show_FilterSelector_gadget()

#feed the filter object with your ID values into a data object
myDataObject<-fc_data$new(myFilterObject=myFilters, myGroupSurveys=TRUE)

#run an analysis by feeding it the data object
myAnalysis<-fca_meanLength(myDataObject)

#print output tables with printTablesAuto/Html/Latex
myAnalysis$printTablesAuto()

#if you want canned plots
myAnalysis$plots

#view potential data problems
myDataObject$warnings
myDataObject$errors

#view analysis issues
myAnalysis$warnings
myAnalysis$errors
```

11 Base Object

11.1 Purpose

Provides a common interface for results regardless of what analysis is run.

11.1.1 Variables

- analysisTitle - (string) Description of the analysis stored in the object, used for headers and titles in reports and displays
- exportName - (string) Name used when generating file download names, no spaces or punctuation should be used
- descriptionText - (string) Markdown text to be displayed in outputs prior to the output tables and figures
- tableTitle- (list<string>) Title text to be used on object tables, each item in list corresponds to one table output in printTables
- groupByVars - (string) Comma-separated string of variable name to use as grouping variables within the tables
- surveys - (dataframe) Dataframe containing survey-level data
- samples - (dataframe) Dataframe containing samples
- results - (list<dataframes>) List of dataframes containing results of analysis
- plots - (list<ggplot objects>) List of ggPlot objects created from analysis
- errors - (list<string>) Error messages created during analysis
- warnings- (list<string>) Warning messages created during analysis
- groupHeaderBackgroundColor - (string) Color to be used in background of group header within table, groups are determined by the groupByVars variable, used in HTML outputs only

- groupSummaryBackgroundColor - (string) Color to be used in the summary row of each group within table, groups are determined by the groupByVars variable, used in HTML outputs only
- gtTheme - (string) Name of gt tables theme from the gtExtras package, used in HTML outputs only

11.1.2 Methods

- print - method used by R to print the results variable
- exportJson - method used to create and save Json file of results, no default implementation
- exportCsv - method used to create and save Csv file of results, no default implementation
- createTable - generic table to create both Latex and Html tables for object
- createTableLatex - creates table formatted for Latex, default implementation simply calls and returns createTable
- createTableHtml - creates table formatted for Html, default implementation simply calls and returns createTable
- printTablesLatex - called to output all tables in a LaTex format, this does not include any formatting or table specific code which is included in the createTable? functions, but instead iterates through multiple table outputs
- printTablesHtml - called to output all tables in a Html format, this does not include any formatting or table specific code which is included in the createTable? functions, but instead iterates through multiple table outputs
- printTablesAuto - tests the incoming call for a LaTex environment and calls either printTablesLatex or printTablesHtml as appropiate, used for markdown reports that can be user-generated in mulitple formats
- iterateSurvey - a generic function that crawls through two loops, one for survey groups, one for specific tables (one analysis may output mulitiple tables) and is called by the printTable functions
- printPlots - called to output ggplot outputs stored as part of analysis

11.2 Overriding Base Object

The fc_base object can be extended and customized to produce objects for specific analysis. See details in “Creating Analyis Chapter”.

12 Objects

12.1 fc_counts Object

12.2 fc_fishSampleMedtadata

13 Methods

13.1 fca_counts

- summarizes number of fish caught in samples by species and gender
- returns fc_counts object

13.2 fca_fishSampleMetadata

- returns metadata for fish samples included in analysis
- returns fc_fishSampleMetadata object

14 Helpers

There are a number of helper functions available in the FinCatch Analysis package.

14.1 Dplyr Verbs

14.1.1 AddStockCategory

This function works as a dplyr verb and adds a column containing a factor of stock-length categories from Gabelhouse's 5-cell model. The function takes the name of the column containing the species code, the column containing the fish length (in mm), and a reference to the data object for the analysis (needed to retrieve species code and stock length values). The function defaults to full group names as factor labels; use abbrieviations=TRUE to use category abbrieviations instead.

Example:

```
some_data %>%
  AddStockCategory(sppCode, fishLen, someDataObject, useAbbreviations = TRUE)
```

14.1.2 AddWrParameters

This dplyr verb has yet to be implented

14.1.3 AddAgeIntercept

This dplyr verb has yet to be implemented

14.2 Functions

14.2.1 weighted.se.mean

This function calculates standard error around a weighted mean. The function assumes a weighted mean calculated by the "weighted.mean()" function in the stats package of R and accepts the same arguements.

Example:

```
some_data=data.frame(xValue=c(3,4,6,3,2,3),
                     xWeights=c(0.2, 0.2, 0.3, 0.1, 0.1, 0.1))
some_data %>%
  mutate(myWeightedMean=weighted.mean(xValue, xWeights, na.rm=TRUE),
         myWeightedMeanSE=weighted.se.mean(xValue, xWeights, na.rm=TRUE))
```

14.2.2 fca_getAnalysisFunctions

This function returns a list and description of the analysis functions available in the FinCatch-Analysis package.

Example:

```
fca_getAnalysisFunctions()
```

14.3 Labelers

14.3.1 fc_labeler_Survey

This function creates a label for each survey to be used in outputs such as reports and plots. The arguments are a vector of surveyUid values and a reference to the analysis data object. The label returned is structured like:

Title1 (WB=5110 | Method=21 | Year=2022 | Season=Spring)

Example:

```
some_data %>%
  mutate(surveyLabel=fc_labeler_Survey(surveyUid, aDataObject))
```

14.3.2 fc_labeler_fishSample

This function creates a label for each fish sample to be used in outputs such as reports and plots. The arguments are a vector of surveyUid values and a dataframe of fish samples in the analysis taken from the data object. The label returned is structured like:

> WB=2832 | Method=45 | 2022-03-11 | Station=627

Example:

```
some_data %>%
  mutate(sampleLabel=fc_labeler_fishSample(surveyUid, aDataObject$get_data_samplesFish))
```

14.3.3 fc_labeler_wqSample

This function creates a label for each fish sample to be used in outputs such as reports and plots. The arguments are a vector of surveyUid values and a dataframe of fish samples in the analysis taken from the data object. The label returned is structured like:

> WB=2832 | 2022-03-11 | Station=627

Example:

```
some_data %>%
  mutate(sampleLabel=fc_labeler_wqSample(surveyUid, aDataObject$get_data_samplesWq))
```

14.3.4 fc_matchCodes

www

14.3.5 fc_createCodeLabel

www

14.3.6 fc_createCodeLabelReversed

www

15 Create Analysis

15.0.1 General

Analysis function calls are prefixed with “fca_” and object names are prefixed with “fc_”

Make sure to test each analysis function for:

- Data selected by surveys only
- Data selected by samples only
- Data selected by both surveys and samples
- Filters that return NO data
- Species in processing list and not in data
- Species not in processing list and in data
- Works for both grouped by survey and ungrouped analysis, if not grouped by survey in fc_data$groupSurveys, surveyUid is set to “-1” during data download by the data object

15.0.2 Flow and Critical Issues

- Import Data
  - Make sure to use remove_nonCPUE_data as appropiate in get_data_ functions

    > **!** Important
    >
    > use remove_nonCPUE_data=FALSE to get all data if appropriate, specify remove_nonCPUE_data=TRUE for readability (is default if not specified)

  - Check For Empty/Missing Stations using ??? function
- Conduct Analysis
  - Expand to account for missing 0’s using AddZerosForMissingSpecies() function
- Finish Analysis

- Label survey UIDs, sample UIDs, species, and methods
-

> **Note**
>
> Useful Functions:
> fc_labeler_Survey
> fc_labeler_fishSample
> fc_labeler_wqSample
> fc_matchCodes
> fc_createCodeLabel
> fc_createCodeLabelReversed

15.0.3 Steps

1. Create new r file names fc_analysisName.R (Easiest To Copy Existing Analysis and Modify)
2. No library statements should be included in R file. Instead, they need to be included in the package DESCRIPTION file.
3. Create/Modify the roxygen comments for procedure
4. Name/Rename function. Analysis functions are prefixed with "fca_" and the same base name as the "fc_" file.
5. All function calls require a FinCatch Data Object (fc_data) to be passed to an arguement named "myData"
6. Check that fish (or Wq) samples exist in the current dataset.
7. Set grouping variables.

- To group analysis calculations during analysis, use the dplyr verbs

```
    group_by(across(all_of(myGroups))
```

- add addition fields as necessary "group_by(across(all_of(myGroups), anotherFieldHere)"

8. Write analysis code

- Always include Standard Error and Sample size (if appropriate), this allow users to calculate difference confidence intervals post hoc
- When including confidence intervals, include 95% and 80%

- Make sure to account for missing data in any input

9. Label values like survey, sample, species, waterbody, etc.

- Labelers exist for samples and surveys (make sure surveyUid's are set to -1 if not grouping by survey
- Helper functions are available for coded values

10. If analysis is calculated (as opposed to raw data summations), screen out species NOT in the processing list. This is necessisary because species NOT in the list do not carry the assumption that all specimans for that species were processed.

11. Attach all results to an analysis object (either base or custom). Example:

```
#create return object at beginning of function
(this allows errors and warnings to be added throughout survey)
op <- fc_meanWeight$new()
op$analysisTitle = "Mean Weight (Weighted)"
op$exportName = "MeanWeight"
op$descriptionText = "These results display weighted mean
                      weight.  Calculations are weighted as
                      only a non-proportional (i.e. first 10 fish
                      per 10mm length group) number of fish
                      are subsampled."
op$tableTitle <- list("Weighted Mean Weight")
op$groupByVars <- "speciesCode"
op$groupSurveys <- myData$groupSurveys

#during function, add warnings and errors to return object
op$errors <- rlist::list.append(op$errors, "Error1 Message", "Error2 Message")
op$warnings <- rlist::list.append(op$warnings, "Warning1")

#at end of function, add results to return object
op$results <- list(data.frame(d))
op$plots <- myPlots
```

12. Add the function to the list of analysis functions available in the package (found in inst folder). This list is used to populate the FinCatchRA UI and to feed the fc_getAvailableAnalysis function.

15.0.4 Create Custom Analysis R6 Object

All analysis results are returned using R6 objects. This allows for consistent use and implementation of different analysis functions by parent applications and code. A base R6 object, "fc_base", provides all the basic functions and structure for FinCatch analysis objects. Custom objects can be created in the analysis files to allow customized output tables and plots and MUST inherit from the fc_base object.

Column names will often need to be altered to provided user friendly text in the outputs. In addition, sometimes valuable columns are dropped for display purposes. Both of these should be done in the analysis output object createTable functions, NOT in the analysis function itself or in the "results" property of the output object. This is to provide for consistency between analyses and is important for the download function of FinCatchRA.

Every effort is made to ensure tables produced by analysis objects work in both HTML (which allows more formatting options and used by FinCatchRA) and in LaTex (for PDF report generation). Basic table creation should happen by providing a "CreateTable" function. Any additional work needed for specific HTML or LaTex output should be included in overridden "CreateTableHtml" or "CreateTableLatex" functions….which otherwise just call and return the "CreateTable" function by default.

```
fc_counts <- R6Class("fc_counts",
                     inherit = fc_base,
                     public = list(
                       createTable = function(mySurveyUid, myTableNumber) {
                           #if data was selected by samples only…all surveyUids will be blank
                       op <- NA

                       thisSurveyLabel <- (self$results[[myTableNumber]] %>%
                         pull(surveyLabel))[[1]]

                       op <- self$results[[myTableNumber]] %>%
                         filter(surveyUid == mySurveyUid) %>%
                         group_by(countSpeciesLabel, sampleLabel) %>%
                         summarise(FishCount = sum(FishCount, na.rm = TRUE)) %>%
                         mutate(countGenderLabel = "Total") %>%
                         bind_rows(self$results[[myTableNumber]]  %>%
                         filter(surveyUid == mySurveyUid)) %>%
                         arrange(countSpeciesLabel, sampleLabel, countGenderLabel) %>%
                         select(-countSpeciesCode,-countGenderCode,-sampleUid,-surveyLabel,-su
                         spread(countGenderLabel, FishCount, fill = 0) %>%
                         relocate("Total", .after = last_col()) %>%
                         gt(rowname_col = colnames(self$results[[1]])[8]) %>%
```

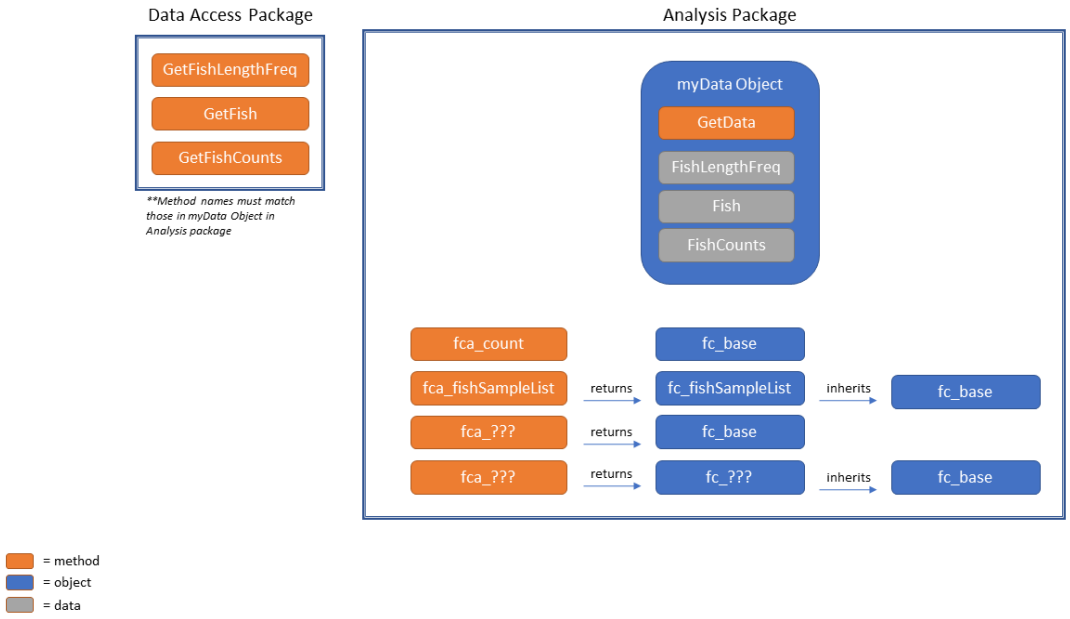
```
                    tab_options(
                      row_group.background.color = self$groupHeaderBackgroundColor,
                      summary_row.background.color = self$groupSummaryBackgroundColor
                      ) %>%
                    tab_header(title = md(self$tableTitle),
                               subtitle = thisSurveyLabel) %>%
                    summary_rows(groups = TRUE,
                                 columns = everything(),
                                 fns = list("Total" = "sum"),
                                 formatter = fmt_integer,
                                 use_seps = TRUE,
                                 missing_text = "") %>%
                    self$gtTheme()

                  return(op)
                }
              ))
```

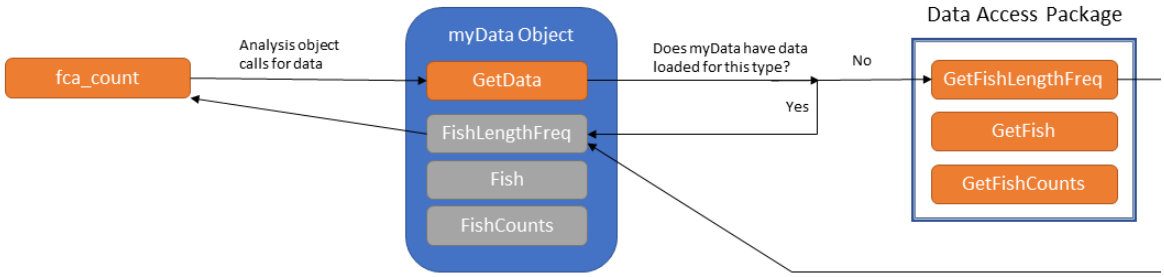
Part IV

Data Access R Package

Packages



WorkFlow



Principles

By Using myData Object:

- Only data that's needed is loaded

- Data is cached
- Data is only retrieved once for entire analysis string

By Using Separate Data Access Package:

- Isolates and generalizes data access
- Allows analysis code to use different sources of data
- Allows different scripts to use different authentications for data access

Part V

References