

fisheryStandardization

Francesca Rosa

February 5, 2019

Introduction

Preliminary operations

Load the necessary packages:

```
suppressMessages( {  
  library(data.table)  
  library(shiny)  
  library(faosws)  
  library(data.table)  
  library(shiny)  
  library(faoswsStandardization)  
  library(faoswsProcessing)  
  library(faoswsUtil)  
  library(faoswsModules)  
  library(faosws)  
  library(faoswsFlag)  
  library(DT)  
  library(dplyr)  
  library(faoswsFisheryStandardization)  
})
```

The version of the Global Production dataset currently stored in QA does not match with the last release. The first GetData pulls data from the Prod env (see as confirmation the the sws1.yml file):

```
if(CheckDebug()){  
  
  SETTINGS = ReadSettings("sws1.yml")  
  
  ## If you're not on the system, your settings will overwrite any others  
  R_SWS_SHARE_PATH = SETTINGS[["share"]]  
  
  ## Define where your certificates are stored  
  SetClientFiles(SETTINGS[["certdir"]])  
  
  ## Get session information from SWS. Token must be obtained from web interface  
  GetTestEnvironment(baseUrl = SETTINGS[["server"]],  
                     token = SETTINGS[["token"]])  
  
  startYear = 2000  
  endYear   = 2016  
}
```

The list of countries the plugin will work on, depends on the query the user set in creating the session. The user may select one or more countries that will be processed in order to produce consistent SUA tables and FBS. The following lines will be used once the SWS plugin will be created. Several parameters should be

associated to the parameter. The parameters will allow to choose the country and the time-window to be processed.

This information are currently hard-coded in this script: see for example startYear and endYear in the previous chunk.

```
## sessionKey = swsContext.datasets[[1]]
## countries=sessionKey@dimensions$geographicAreaM49_fi
## startYear= swsContext.computationParams@startYear
## endYear= swsContext.computationParams@endYear

if(CheckDebug()){
batchNumb=992

dir.create(paste0("C:/Users/ROSA/Desktop/Fisheries/batch", batchNumb, "/SUA_unbalanced"), recursive = TRUE)
dir.create(paste0("C:/Users/ROSA/Desktop/Fisheries/batch", batchNumb, "/SUA_balanced"), recursive = TRUE)
dir.create(paste0("C:/Users/ROSA/Desktop/Fisheries/batch", batchNumb, "/FBS"), recursive = TRUE)
dir.create(paste0("C:/Users/ROSA/Desktop/Fisheries/batch", batchNumb, "/plot"), recursive = TRUE)

directory=paste0("C:/Users/ROSA/Desktop/Fisheries/batch", batchNumb)
}
```

Hard-coded list of countries:

```
countries=c("300", "203", "40", "246", "156", "792", "643", "268")
```

Loop by country to obtain Supply-Utilization Account balanced equations:

```
sua_unb=list()

for(j in seq_along(countries)) {

  currentCountry=countries[j]

  keyDim=c("geographicAreaM49_fi", "fisheriesAsfis", "measuredElement", "timePointYears")

  KeyGlobal = DatasetKey(domain = "Fisheries", dataset = "fi_global_production", dimensions = list(
    #geographicAreaM49_fi = Dimension(name = "geographicAreaM49_fi", keys = GetCodeList("Fisheries", "geographicAreaM49_fi"),
    geographicAreaM49_fi = Dimension(name = "geographicAreaM49_fi", keys = currentCountry),
    fisheriesAsfis = Dimension(name = "fisheriesAsfis", keys = GetCodeList("Fisheries", "fi_global_production_fisheriesAsfis")),
    fisheriesCatchArea = Dimension(name = "fisheriesCatchArea", keys = GetCodeList("Fisheries", "fi_global_production_fisheriesCatchArea")),
    measuredElement = Dimension(name = "measuredElement", keys = c("FI_001")),
    #timePointYears = Dimension(name = "timePointYears", keys = GetCodeList("Fisheries", "fi_global_production_timePointYears")),
    timePointYears = Dimension(name = "timePointYears", keys = as.character(c(startYear:endYear)))
  ))

  ##Get Global Production Data
  globalProduction=GetData(KeyGlobal)
  # globalProduction[geographicAreaM49_fi=="156", geographicAreaM49_fi=="1248"]

  #if(currentCountry=="156"){currentCountry="1248"}
```

After pulling the Global production data for the country of interest, move to the QA env (look at the sws.yml file):

```

if(CheckDebug()){

  SETTINGS = ReadSettings("sws.yml")

  ## If you're not on the system, your settings will overwrite any others
  R_SWS_SHARE_PATH = SETTINGS[["share"]]

  ## Define where your certificates are stored
  SetClientFiles(SETTINGS[["certdir"]])

  ## Get session information from SWS. Token must be obtained from web interface
  GetTestEnvironment(baseUrl = SETTINGS[["server"]],
                     token = SETTINGS[["token"]])

}

```

Aggregate the data referring to FishingArea:

```

globalProduction=globalProduction[,sum(Value, na.rm = TRUE), by=c("geographicAreaM49_fi",
                                                                    "fisheriesAsfis",
                                                                    "measuredElement",
                                                                    "timePointYears")]

setnames(globalProduction, "V1", "Value")

## Work on flags: aggregate observationFlag!!
## All Values different from zero are flagged as "". Are they all OFFICIAL??
## globalProduction[,flagObservationStatus==""]
## globalProduction[,flagMethod=="c"]

```

Reading the mapping table from the SWS. The following datatables:

1. fishery_item_mapping
2. fishery_primary_mapping

These datatables have been created by the SWS team (Francesca) starting from the file Mapping- ICS - ISSCAAP - stored in the SharePoint (Fisheries - Food Balance - Sheets - FIAS team document).

```

mapping=ReadDatatable("fishery_item_mapping")
primaryMapping=ReadDatatable("fishery_primary_mapping")

#primaryMapping = fread("data/PrimaryProduction_mapping.csv")
setnames(primaryMapping, "alphacode","fisheriesAsfis")

globalProductionMapping = merge(
  globalProduction,
  primaryMapping,
  by = c( "fisheriesAsfis"),
  all.x = TRUE)

```

TIP: Integrate this info in the data.table fishery_primary_mapping

```

primary = c("1501", "1514", "1527","1540","1553", "1562", "1579", "1570", "1587", "1594")

```

Populate the SUA unbalanced table: the ICS FAOSTAT classification (section 3.1: Populate the SUA unbalanced table: the ICS FAOSTAT classification)

```
globalProductionMapping= as.data.table(inner_join(globalProductionMapping,
                                                    unique(mapping[ics %in% primary,.(ics,isscaap)]),
                                                    by = "isscaap" )

globalProductionMapping=globalProductionMapping[ics %in% primary]

##
globalProductionMapping=globalProductionMapping[,.(geographicAreaM49_fi,
                                                    timePointYears,
                                                    ics,
                                                    measuredElement,
                                                    Value)]

globalProductionMapping[, Value := sum(Value, na.rm = TRUE), by = list(geographicAreaM49_fi,
                                                                    timePointYears,
                                                                    measuredElement,
                                                                    ics)]

globalProductionMapping = globalProductionMapping[!duplicated(globalProductionMapping)]

globalProductionMapping=globalProductionMapping[!is.na(ics)]
```

Pull the commodity DB, currently stored locally in the project folder repository:

```
commodityDB_quantity = fread("data/commodityDB_Quantity.csv", header = TRUE)
commodityDB_quantity = melt(
  commodityDB_quantity,
  id.vars = colnames(commodityDB_quantity)[c(1:7)],
  measure.vars = colnames(commodityDB_quantity)[c(8:48)],
  variable.name = "timePointYears",
  value.name = "Value"
)

commodityDB_quantity[,Value:=gsub("F","",Value)]

commodityDB_quantity[, Value := as.numeric(Value)]
```

Please note that the warning is normal: the “...” are transformed in NA which is exactly what we want!

```
commodityDB_quantity[, measuredElement := as.character(measuredElement)]

#####
#commodityDB = rbind(commodityDB_value, commodityDB_quantity)
commodityDB = rbind( commodityDB_quantity)
commodityDB=commodityDB[Country_code==currentCountry]
#####
```

Aggregate the ISSCFC_Code according to the mapping on the ICS classification Commodity Database:

ISSCFC_Code on the ICS_Code:

```
commodityDB[,.(Country_code, ISSCFC_Code, measuredElement, timePointYears,Value)]
setnames(commodityDB, "ISSCFC_Code", "isscfc")
commodityDB_aggregation = merge(commodityDB, mapping, by = "isscfc", all.x = TRUE)
commodityDB_aggregation = commodityDB_aggregation[!is.na(Value)]
```

Hard- coded solution for China (check if it fits also for other country). This chunk highlight the need to develop a pre-link tables. This means that the deviation of some flows occurs before the aggregation in ICS, but at the level of ISSCFC codes. At the moment this table does not exist.

```
if(commodityDB_aggregation[, unique(Country_code)]=="156") {commodityDB_aggregation[isscfc=="034.4.1.1"]
```

Some commodities are imported but not for food purposes, main example “ornamental fish”. Those flow are deviated directly to “other utilizations”:

```
otherUses=fread("data/otherUses.csv")

for(otherUsesItem in seq_along(unique(otherUses[,isscfc])) ){
  commodityDB_aggregation[isscfc %in% otherUses[,unique(isscfc)] [otherUsesItem] & measuredElement=="6"]
  measuredElement:=otherUses[isscfc== otherUses[,unique(isscfc)] [otherUsesItem]]
}

sua_commodityDB = commodityDB_aggregation[, .(Country_code,
                                              ics,
                                              Trade_flow,
                                              measuredElement,
                                              timePointYears,
                                              Value)]

sua_commodityDB[, Value := sum(Value, na.rm = TRUE), by = list(Country_code,
                                                                ics,
                                                                measuredElement,
                                                                timePointYears)]

sua_commodityDB = sua_commodityDB[!duplicated(sua_commodityDB)]
sua_commodityDB=sua_commodityDB[,.(Country_code,
                                    ics,
                                    measuredElement,
                                    timePointYears,
                                    Value)]
```

The Link table (see section 3.1: Populate the SUA unbalanced table: the ICS FAOSTAT classification)

Once the link table has been read from the SWS, a number of checks must be performed (still not fully developed):

1. Check for percentages, ensure that sum of percentages by
2. Ensure that the column flow only contains PRD, IMP, EXP, TRD or ALL. No other flows are admitted.

Since this data.table is manually updated, directly in the SWS, these checks are extremely important because it is might be possible to find typos in the flow column or mistakes in the compilation of the shares.

If the link table does not pass all the checks, the process stops and the user should be warned by an email containing as attachment the problematic lines. Ideally, the checks should be performed only on the slice of

data referring to the current country, in order to not block the process for a country if the mistake is not related to it.

```
link=ReadDatatable("link_mapping")
link[, check:=sum(percentage), by=c("geographic_area_m49","flow","from_code","start_year","end_year")]

link[check!=1] ##send a warning: there is a double counting.
## send email containing as attachment those lines of the link table than did not pass the check.

link=link[check<=1,]

split=link[check!=1]
split=unique( split[,.(geographic_area_m49 ,flow ,start_year, end_year ,from_code, check)] )

split[,percentage:=1-check]
split[,to_code:=from_code]
link=rbind(link, split)
link[,check:=NULL]

link=link[geographic_area_m49==currentCountry]
link=(unique(link))
if(nrow(link)>0){

  lastYear=max(as.numeric(as.character(commodityDB[,timePointYears])))

  all=c("PRD", "IMP", "EXP")
  all=data.table(flow=rep("ALL", length(all)),all=all)
  link_all=merge(link, all, by="flow")
  link_all[,flow:=all]
  link_all[,all:=NULL]

  trade=c( "IMP", "EXP")
  trade=data.table(flow=rep("TRD", length(trade)),trade=trade)
  link_trade= merge(link, trade, by="flow", allow.cartesian = TRUE)
  link_trade[,flow:=trade]
  link_trade[,trade:=NULL]

  link=link[!flow %in% c("ALL", "TRD"),]
  link=rbind(link,link_trade,link_all)

  link[end_year=="LAST", end_year:=lastYear]
  link[,start_year:=as.numeric(as.character(start_year))]
  link[,end_year:=as.numeric(as.character(end_year))]

  link[,nYear:=(end_year-start_year)+1]
  linkNew=list()

  for(i in 1:dim(link)[1]){
    timePointYears=data.table(timePointYears=c(link[i,start_year]:link[i,end_year]))
    iter= link[i]
    if(iter[,nYear]>1){
      iter= data.table(apply(iter, 2, rep, iter[,nYear]))
      linkNew[[i]]=data.table(iter, timePointYears=c(unique(iter[1,start_year]):unique(iter[1,end_year])))
    }else{
```

```

    linkNew[[i]]=data.table(iter, timePointYears=c(unique(iter[1,start_year]):unique(iter[1,end_year])))
  }

}

linkNew=rbindlist(linkNew)

linkNew[,start_year:=NULL]
linkNew[,nYear:=NULL]
#linkNew[,to_code:=NULL]
linkNew[,start_year:=NULL]
linkNew[,end_year:=NULL]

setnames(linkNew, "from_code", "ics")
linkNew[flow=="IMP",flow:="61"]
linkNew[flow=="EXP",flow:="91"]
linkNew[flow=="PRD",flow:="51"]

### Modify the ICS_Code according to the linkNew file:
setnames(linkNew,"geographic_area_m49" ,"Country_code")
setnames(linkNew,"flow" ,"measuredElement")

```

In the previous chunk the link table that has been created to be exactly the same as the one currently in use in the (old) FIAS Working System, is expanded to be used to deviate the proper flow to ad hoc ICS codes.

```

sua_commodityDB[,Country_code:=as.character(Country_code)]
sua_commodityDB[,timePointYears:=as.integer(as.character(timePointYears) )]

sua_commodityDB=merge(sua_commodityDB,
                      linkNew, by=c("Country_code",
                                     "timePointYears",
                                     "measuredElement", "ics"),
                      all.x = TRUE)

sua_commodityDB[, percentage:=as.numeric(percentage)]

sua_commodityDB[!is.na(percentage), Value:=Value*percentage]
sua_commodityDB[!is.na(to_code), ics:=to_code]
sua_commodityDB[, percentage:=NULL]
sua_commodityDB[, to_code:=NULL]
sua_commodityDB[, Value:=sum(Value, na.rm = TRUE), by=c("Country_code",
                                                         "measuredElement",
                                                         "ics",
                                                         "timePointYears" )]

sua_commodityDB=sua_commodityDB[!duplicated(sua_commodityDB)]

}

setnames(sua_commodityDB,c("Country_code"), c("geographicAreaM49_fi"))
sua_commodityDB=sua_commodityDB[,.(geographicAreaM49_fi,
                                   measuredElement,

```

```
ics,
timePointYears,
Value)]
```

Data coming from Global Production and from the Commodity DB are put together in the same table:

```
SUA=rbind(sua_commodityDB, globalProductionMapping)
```

```
SUA[measuredElement=="FI_001", measuredElement:="51"]
```

```
### Step 1. balance the equation if the imbalance<0 (add the imbalance to the PROD)
### I do not have obs and method flag columns (I add fake 2 columns)
### This problem will be directly solved when I
```

```
##SUA[,flagObservationStatus:=""]
##SUA[,flagMethod:="q"]
```

Expand the SUA: many SUA will be populated in the next steps. The most basic operation to populate a missing SUA component is balancing the SUA equation and assign to the missing component the overall amount of the imbalance.

To perform this operation it is necessary to create the empty cells to be filled:

```
key = c("timePointYears", "geographicAreaM49_fi", "ics")
keyDataFrame = SUA[, key, with = FALSE]
keyDataFrame=keyDataFrame[with(keyDataFrame, order(get(key)))]
keyDataFrame=keyDataFrame[!duplicated(keyDataFrame)]
elDataFrame = c("51","61","91","71","111","121", "141", "151", "131", "101", "31", "131")
##elDataFrame = c("51","61","91","141")
#elDataFrame = unique(SUA[,.(measuredElement)])
elDataFrame=data.table(elVar=elDataFrame)
colnames(elDataFrame) = "measuredElement"

completeBasis = data.table(merge.data.frame(keyDataFrame, elDataFrame))
expandedData = merge(completeBasis, SUA, by = colnames(completeBasis), all.x = TRUE)
expandedData = fillRecord(expandedData)
# expandedData[is.na(flagObservationStatus), flagObservationStatus:="M"]
# expandedData[is.na(flagObservationStatus), flagMethod:="u"]
SUA=expandedData
```

Protected figures had been extracted from the already existing SUAs (provided as excel file from the FIAS team) and are currently stored in a local folder (in the module repository).

This issue will be managed through the FLAGS. Those figures flagged as protected (see faosws-Flag::flagValidTable) will be kept and used in the next steps (see also the package: faoswsFlag).

```
protected=fread(paste0("data/localRun/protected/",currentCountry,"_protected.csv"), header = TRUE)
protected[,geographicAreaM49_fi:=as.character(geographicAreaM49_fi)]
protected[,ics:=as.character(ics)]
protected[,measuredElement:=as.character(measuredElement)]
protected[,timePointYears:=as.factor(timePointYears)]

SUA=merge(SUA, protected, by=c("geographicAreaM49_fi","ics","timePointYears","measuredElement"),suffix="protected")
SUA[!is.na(Value_protected),Value:=Value_protected]
SUA[,Value_protected:=NULL]
```



```
SUA=SUA[timePointYears %in% c(2000:2016)]
```

Compute the imbalance. It is looked as the **availability**. It is the amount that has not been allocated yet in any utilization and it is still available to be consumed or used as input to produce derived items.

```
SUA[, availability:=sum(iffelse(is.na(Value),0,Value)*
  iffelse(measuredElement=="51", 1,
  iffelse(measuredElement=="61",1,
  iffelse(measuredElement=="91", -1,
  iffelse(measuredElement=="101", -1,
  iffelse(measuredElement=="71", -1,
  by=c("geographicAreaM49_fi","ics", "timePointYears")]
```

Production can be touched only in case of processed items. The PRIMARY negative availabilities at this stage MUST be properly taken into account. In case we identify (in table toBeChecked) some primary negative availabilities, Send Email and STOP the process:

```
toBeChecked=SUA[ics %in% primary & availability<0]
## if(!CheckDebug() & nrow(toBeChecked)>0){
##   sendmailR::sendmail("jjjjj")
##   stop("There are negative primary availabilities, please check your mailbox!")
##
## }else{
```

Balance the line characterized by negative availabilities increasing the PRODUCTION: Before sending all the negative imbalance to Production, it would be possible also to deviate a small portion of the TOT imbalance to stockVariation. This means that stock is a supply-component (if negative). Anyway we have to check that the sum of the series of stock variation is still positive (or equal to zero).

```
SUA[measuredElement=="51" & availability<0 ,
  Value:=iffelse(is.na(Value), -availability, Value-availability)]
```

Compute again the imbalance==availability:

```
SUA[, availability:=sum(iffelse(is.na(Value),0,Value)*
  iffelse(measuredElement=="51", 1,
  iffelse(measuredElement=="61",1,
  iffelse(measuredElement=="91",-1,
  iffelse(measuredElement=="101", -1,
  iffelse(measuredElement=="71", -1,
  by=c("geographicAreaM49_fi","ics", "timePointYears")]
```

Compute Food Processing (Look at the section 3.3.)

The Tree structure is quite strange, each processed product can be produced at many different processing levels according to the availability of the primary, secondary,... For example, a canned item can be produced starting from fresh fish (if it is available) but also from frozen fish (which might have been imported)

```
commodityTree=fread(paste0("data/localRun/commodityTree/tree_", currentCountry, ".csv") )

#commodityTree=commodityTree[!(parent=="1515" & child=="1518")]
#commodityTree=commodityTree[!(parent=="1517" & child=="1518")]

commodityTree[,geographicAreaM49_fi:=as.character(geographicAreaM49_fi)]
commodityTree[,parent:=as.character(parent)]
commodityTree[,child:=as.character(child)]
```

```
commodityTree[,timePointYears:=as.character(timePointYears)]

SUA[,timePointYears:=as.character(timePointYears)]

SUA = processingCompute(SUA,commodityTree)
#SUA = processingComputeSimplified(SUA,commodityTree)
```

Compute again the Availability including Food Processing in the equation:

```
SUA[, availability:=sum(ifelse(is.na(Value),0,Value)*
                        ifelse(measuredElement=="51", 1,
                        ifelse(measuredElement=="61",1,
                        ifelse(measuredElement=="91", -1,
                        ifelse(measuredElement=="131", -1,
                        ifelse(measuredElement=="71", -1,
                        ifelse(measuredElement=="101", -1, 0)))))),
                        by=c("geographicAreaM49_fi","ics", "timePointYears")]
```

In theory at the end of this process the sua_unbalanced SWS table should have been created in the meantime, the output populates a local folder

```
# SaveData(domain = "fisheries", dataset = "fi_sua_unbalanced", data = SUA, waitTimeout = 20000)
write.csv(SUA, paste0(directory,"/SUA_unbalanced/SUA_unbalanced_",currentCountry , ".csv") )
```

Allocate the positive imbalance to the possible utilizations:

```
# balancingItems=fread(paste0("C:/Users/ROSA/Desktop/Fisheries/batch0_SUA_pilotCounties/utilizationTab
balancingItems=fread(paste0("data/localRun/balancingElement/",currentCountry,"_balancingElements.csv")

balancingItems[, geographicAreaM49_fi:=as.character(geographicAreaM49_fi)]
balancingItems[, ics:=as.character(ics)]
balancingItems[, geographicAreaM49_fi:=as.character(geographicAreaM49_fi)]
balancingItems[, measuredElement:=as.character(measuredElement)]

SUA[,measuredElement:=as.character(measuredElement) ]
SUA[,timePointYears:=as.integer(as.character(timePointYears) )]
balancingItems[, timePointYears:=as.integer(timePointYears) ]

SUA=merge(SUA, balancingItems, by=c("geographicAreaM49_fi",
                                "ics",
                                "timePointYears",
                                "measuredElement"),
          suffixes = c("", "_balancing"), all.x = TRUE)
SUA[Value_balancing=="B", Value:=availability]

SUA[, availability:=sum(ifelse(is.na(Value),0,Value)*
                        ifelse(measuredElement=="51", 1,
                        ifelse(measuredElement=="61",1,
                        ifelse(measuredElement=="91", -1,
                        ifelse(measuredElement=="131", -1,
                        ifelse(measuredElement=="71", -1,
                        ifelse(measuredElement=="141", -1,
                        ifelse(measuredElement=="151", -1,
```

```

        ifelse(measuredElement=="101", -1,0) ))) )))),
by=c("geographicAreaM49_fi","ics", "timePointYears")

```

```
SUA[, Value_balancing:=NULL]
```

Add NutrientFactors:

```

nutrientFactors=ReadDatatable("fishery_nutrient")
SUA_with_nutrient=merge(SUA, nutrientFactors, by="ics", all.x = TRUE)
SUA_with_nutrient[measuredElement=="141", calories:=Value*calories]
SUA_with_nutrient[measuredElement=="141", proteins:=Value*proteins]
SUA_with_nutrient[measuredElement=="141", fats:=Value*fats]
SUA_with_nutrient[measuredElement!="141", `:=`(c("calories", "proteins", "fats"),list(0,0,0) )]

```

Deviate the output to an intermediate file:

```

sua_unb[[j]]=SUA

# SaveData(domain = "fisheries", dataset = "fi_sua_balanced", data = SUA_with_nutrient, waitTimeout =
write.csv(SUA_with_nutrient, paste0(directory,"/SUA_balanced/SUA_balanced_",currentCountry , ".csv")
}

```

Here I am considering the default commodity tree-no country-year specificities: this choice it is not correct. The commodity tree that has to be used to finally perform the standardization had to meet the following requirements: 1. the so-called ONE-LEVEL tree, all the processed items must be reported in terms of their primary equivalent. 2. if a country provides figures for both input(31) and production (51) the extraction rate is computed (and not the input) on the fly by the module and this latter extraction rate has to be used in the standardization.

```

commodityTree = ReadDatatable("fisheries_commodity_tree")
lev=findProcessingLevel(commodityTree, from="parent", to="child", aupusParam = list(itemVar="parent"))
commodityTreeLev0=commodityTree[parent %in% lev[processingLevel==0, parent],]

final=rbindlist(sua_unb)

```

Set the parameters to perform the function “standardizeTree”:

```

fisheryParams = fisheryStandardizationParameters()
zeroWeight=commodityTree[weight==0, unique(child)]

```

Loop again along the years: the standardization works on country-year basis.

```

out=list()
out_nutrient=list()
additiveElements=c("calories", "proteins", "fats")

for(t in seq_along(unique(final$timePointYears)) ){
  currentY=unique(final$timePointYears)[t]
  final_time=final[timePointYears==currentY]

  final_time[measuredElement=="51" & (!ics %in% primary), Value:=0]
  final_time=final_time[measuredElement!="131"]
  final_time=final_time[measuredElement!="31"]
  current = final_time[, fisheryStandardizeTree(data = .SD, tree = commodityTreeLev0,

```

```

standParams = fisheryParams, elements = "Value", zero
by = c(fisheryParams$elementVar)]

```

Nutritive elements are obviously associated only to the Food component (141).

```

final_time=merge(final_time, nutrientFactors, by="ics", all.x = TRUE)
final_time[measuredElement=="141", calories:=Value*calories]
final_time[measuredElement=="141", proteins:=Value*proteins]
final_time[measuredElement=="141", fats:=Value*fats]
final_time[measuredElement!="141", `:=`(c("calories", "proteins", "fats"),list(0,0,0) )]
final_time[is.na(calories), calories:=0]
final_time[is.na(proteins), proteins:=0]
final_time[is.na(fats), fats:=0]

```

Additive elements are Calories, Fats and Proteins associated to all the items (primary and derived). No need to standardize these quantities since for both primary and processed items the amount of Calories, Fats and Proteins can be directly summed (it's already expressed in the same unit of measurement).

```

if(length(additiveElements) > 0){
  additiveTree = copy(commodityTreeLev0)
  additiveTree[, c(fisheryParams$extractVar) := 1]
  nutrients = lapply(additiveElements, function(nutrient){
    temp = final_time[get(fisheryParams$elementVar) == fisheryParams$foodCode,
                      fisheryStandardizeTree(data = .SD, tree = additiveTree,
                      standParams = fisheryParams, elements = nutrient )]

    temp[, Value := get(nutrient)]
    temp[, c(fisheryParams$elementVar) := nutrient]
    temp[, c(nutrient) := NULL]
    temp
  })
  fromProcessed= rbind(current, do.call("rbind", nutrients))

  fromPrimary=final_time[ics %in% primary]
  fromPrimary[,availability:=NULL]

  primaryNutrients= melt.data.table(
    data = fromPrimary, measure.vars = c("calories", "proteins","fats"),
    id.vars = c("geographicAreaM49_fi", "timePointYears", "ics"),
    variable.name = "measuredElement", value.name = "Value")
  primaryNutrients=primaryNutrients[Value!=0]
  fromPrimary[,calories:=NULL]
  fromPrimary[,proteins:=NULL]
  fromPrimary[,fats:=NULL]

  primaryEq=rbind(fromProcessed, fromPrimary, primaryNutrients)

  primaryEq=primaryEq[,sum(Value, na.rm = TRUE), by=c(fisheryParams$mergeKey, "measuredElement")]
  setnames(primaryEq, "V1", "Value")

  primaryEq=primaryEq[ics %in% primary]

  out[[t]]=primaryEq
}

```

```
}
```

```
out=rbindlist(out)
```

Compute, once again, the final imbalance/availability that should finally be **ZERO**.

```
out[, availability:=sum(ifelse(is.na(Value),0,Value)*
                        ifelse(measuredElement=="51", 1,
                        ifelse(measuredElement=="61",1,
                        ifelse(measuredElement=="91", -1,
                        ifelse(measuredElement=="131", -1,
                        ifelse(measuredElement=="141",-1,0)))))),
      by=c("geographicAreaM49_fi","ics", "timePointYears")]
```

Convert the ics codes into the FBS codes:

1. 1501 goes to 010
2. 1514 goes to 020
3. 1527 goes to 030
4. 1540 goes to 040
5. 1553 goes to 050
6. 1562 goes to 060
7. 1570 goes to 070
8. 1587 goes to 090

```
sua_fbs_mapping=data.table(ics=c( "1501" ,"1514" ,"1527" ,"1540" ,"1553", "1562", "1570", "1587"),
                             fbs=c( "010", "020", "030", "040", "050", "060", "070", "090"))
```

```
out=merge(out, sua_fbs_mapping, by="ics", all.x = TRUE)
out[, ics:=NULL]
setnames(out, "fbs", "ics")
```

DES computation (section 4. Nutritive Values and DES calculation)

Get population data. Note that the Population to be taken in consideration comes from the dataset population UNPD - 2017 release.

```
elemKeys="511"
keyPop = DatasetKey(domain = "population", dataset = "population_unpd", dimensions = list(
  geographicAreaM49 = Dimension(name = "geographicAreaM49", keys = c("300", "203", "40")),
  measuredElementSuaFbs = Dimension(name = "measuredElement", keys = elemKeys),
  timePointYears = Dimension(name = "timePointYears", keys = as.character(c(2000:2016)))
))

popSWS=GetData(keyPop)

setnames(popSWS, "geographicAreaM49", "geographicAreaM49_fi")
```

Divide the overall amount of calories (by ICS group) by the population and by 365 (number of days in one year):

```
out[,measuredElement:=as.character(measuredElement)]

out = merge(out, popSWS, by=c("geographicAreaM49_fi","timePointYears"), suffixes = c("", "_pop"))
out[, measuredElement %in% c("calories","proteins","fats"), Value_caput_day:= (Value/Value_pop)/365]
nutrient_caput_day = out[,measuredElement %in% c("calories","proteins","fats"),
                          ,.(ics,geographicAreaM49_fi,timePointYears,measuredElement,Value_caput_day)]

nutrient_caput_day[measuredElement=="calories",measuredElement:="calories_caput_day"]
nutrient_caput_day[measuredElement=="fats",measuredElement:="fats_caput_day"]
nutrient_caput_day[measuredElement=="proteins",measuredElement:="proteins_caput_day"]

setnames(nutrient_caput_day,"Value_caput_day", "Value")

out=out[,.(ics,geographicAreaM49_fi,timePointYears,measuredElement, Value)]

out=rbind(out, nutrient_caput_day)
```

Save data back to the SWS (at the moment the output is deviated to a csv file):

```
# SaveData(domain = "fisheries", dataset = "fi_fbs", data = out, waitTimeout = 20000)

write.csv(out,paste0(directory,"/FBS/FBS.csv" , row.names = FALSE))
```