Scotland survey stock conversion

Centre for Sustainable Energy

Tue May 10 16:53:30 2016

Contents

Scotland stock conversion function

The following section of code is generated in scotland/main.R

This script calls all the functions that create each .csv file. The make.scotland function is run from the main.R script which runs both Scotland and England.

Libraries

Loading the libaries required to run script

Note: these libraries must already be installed. To install libraries use install.packages("nameoflibrary) in R console

Source functions

These files can be found in the scotland folder and can be opened in R studio.

The assumptions made to form the stock are found in these files within functions. Some minor edits to the files can be made if a different assumption is required.

```
#Sources (makes available) all functions required to make
#the stock.
source("scottishsurveyprocessing.R",chdir = TRUE)
source("spaceheating.R",chdir = TRUE)
source("roofs.R",chdir = TRUE)
source("cases.R",chdir = TRUE)
source("lighting.R",chdir = TRUE)
source("occupants.R",chdir = TRUE)
source("ventilation.R",chdir = TRUE)
source("waterheating.R",chdir = TRUE)
source("storeys.R",chdir = TRUE)
source("elevations.R",chdir = TRUE)
source("additional-properties.R",chdir = TRUE)
source("DTO-import-files.R",chdir = T)
#functions required to test the stock
#source("test-results.R", chdir=TRUE)
```

Make Scottish stock

The main function which makes the scottish stock, this is run from main.R

The data is first loaded from the survey and cleaned using the function clean.survey.

This data is then passed to each of the functions (i.e. save.dtofilename) in turn. Inside these functions the survey data is processed and turned into a dataframe of the correct format which is then written to a .csv file stored in the outputs/scotland folder.

Finally the survey data is used by the test function to compare the stock data that has been created by the stock creator.

@param path.to.shcs - file path to SHCS SPSS file

@param path.to.output - directory where output files are put

```
make.scotland <- function(path.to.shcs, path.to.output) {
  #Load scottish survey data and preprocess some cases which have inconsistent
  #values i.e. an open fire fueled by electricity
  print(paste("Loading", path.to.shcs))
  shcs <- read.spss(path.to.shcs, to.data.frame = TRUE, reencode='utf-8')
  shcs <- clean.survey(shcs)

  #Creates the spaceheating file
  spaceheating.path <- file.path(path.to.output, "space-heating.csv")
  print(paste("Creating", spaceheating.path))
  save.spaceheating(shcs, spaceheating.path, dirname(scotland.survey))

  #Creates the roofs file
  roofs.path <- file.path(path.to.output, "roofs.csv")
  print(paste("Creating", roofs.path))
  save.roofs(shcs, roofs.path)

  #Creates the cases file
  cases.path <- file.path(path.to.output, "cases.csv")
  print(paste("Creating", cases.path))
  save.cases(shcs, cases.path)

  #Creates the lighting file
  lighting.path <- file.path(path.to.output, "lighting.csv")
  print(paste("Creating", lighting.path))
  save.lighting(shcs, lighting.path)

  #Creates the occupants file
  occupants.path <- file.path(path.to.output, "occupants.csv")
  print(paste("Creating", occupants.path))
  save.occupants(shcs, occupants.path)

  #Creates the ventilation file
  ventilation.path <- file.path(path.to.output, "ventilation.csv")
  print(paste("Creating", ventilation.path))
  save.ventilation(shcs, ventilation.path)

  #Creates the waterheating file
  waterheating.path <- file.path(path.to.output, "water-heating.csv")
```

```
  print(paste("Creating", waterheating.path))
  save.waterheating(shcs,waterheating.path, dirname(scotland.survey),path.to.output)

  #Creates the storeys file
  storeys.path <- file.path(path.to.output, "storeys.csv")
  print(paste("Creating", storeys.path))
  save.storeys(shcs, storeys.path)

  #Creates the elevations file
  elevations.path <- file.path(path.to.output, "elevations.csv")
  print(paste("Creating", elevations.path))
  save.elevations(shcs, elevations.path)

  #Creates the additional properties file
  additionalproperties.path <- file.path(path.to.output, "additional-properties.csv")
  print(paste("Creating", additionalproperties.path))
  save.additionalproperties(shcs, additionalproperties.path)

  #create additional csv files for stock import
  IStockImportMetadataDTO.path <- file.path(path.to.output,
                                            "IStockImportMetadataDTO.csv")
  print(paste("Creating", IStockImportMetadataDTO.path))
  save.IStockImportMetadataDTO(shcs, IStockImportMetadataDTO.path)

  IImportLogDTO.path <- file.path(path.to.output, "IImportLogDTO.csv")
  print(paste("Creating", IImportLogDTO.path))
  save.IImportLogDTO(shcs, IImportLogDTO.path)

  metadata.path <- file.path(path.to.output, "metadata.csv")
  print(paste("Creating", metadata.path))
  save.metadata(shcs, metadata.path)

  # run end-to-end tests
  #test.allshcsdtotests(shcs, path.to.output)
}
```

Preprocessing Scottish Survey Data

The following section of code is generated in scottishsurveyprocessing.R

The inconsistent values in the survey are preprocessed

The values that are fixed are those that have an impact on stock creation, the survey data has not been checked for complete consistancy, only those variables that have been used; i.e. an open fire fueled by electricity is not a physical possibility and so assumptions have been made for this case about what is likely to exist in order to create spaceheating

@param shcs - the scottish survey data

```
clean.survey <- function(shcs){
  #The scottish survey data has 22 cases where the only data that exists is
  #the uprn_new (aacode). These cases are removed prior to processing.This
  #should not be altered.
  shcs <- subset(shcs,is.na(shcs$year)=="FALSE")

  #These cases have a second extension with area and external perimeter
  #variables, but a missing ceiling height.These have been filled with the
  #standard height of 2.4m
  shcs$N7_C[shcs$uprn_new == 4950999308] <- 2.4
  shcs$N7_C[shcs$uprn_new == 3485528102] <- 2.4
  shcs$N7_C[shcs$uprn_new == 9153471654] <- 2.4

  #This case is a flat comprised entirely of a room in roof, with two extensions
  #in order to correctly calculate the position of the extensions this variables
  #stored in the room in roof variables (N5_A and N5_C) are moved to those for level
  #one. The external perimiter is calculated using the area and assuming the ratio
  #length/width is 1.5, this is the only figure that should be changed.
  dimension.ratio <- 1.5
  shcs$N1_A[shcs$uprn_new == 8859831108] <- shcs$N5_A[shcs$uprn_new == 8859831108]
  shcs$N1_C[shcs$uprn_new == 8859831108] <- shcs$N5_C[shcs$uprn_new == 8859831108]
  shcs$N1_D[shcs$uprn_new == 8859831108] <-
    (sqrt(shcs$N5_A[shcs$uprn_new == 8859831108]*dimension.ratio)+
       ((sqrt(shcs$N5_A[shcs$uprn_new == 8859831108]/dimension.ratio))*2))

  #This case is missing level one external perimiter, this is replaced with a value
  #(33) that was calculated from the area, using the assumption that length/width is
  #1.5
  shcs$N1_D[shcs$uprn_new == 460193204] <- 33

  #This case is missing a value for the extent of primary wall. This has been set
  #to 10 as it is the most common extent in the survey data
  shcs$Q10[shcs$uprn_new == 4196979699] <- 10

  #The principal hot water heating source value for the following cases were
  #inconsistent with the other values of heating, they are changed here to values
  #that are both consistant and most common.
  shcs$M17[shcs$uprn_new==6379341481] <- "Room heater BB"
  shcs$M17[shcs$uprn_new==1060005515] <- "Room heater BB"
  shcs$M17[shcs$uprn_new==9472432770] <- "Room heater BB"
  shcs$M17[shcs$uprn_new==7353370291] <- "Room heater BB"
  shcs$M17[shcs$uprn_new==7950759180] <- "Elec immersion"
```

```
shcs$M17[shcs$uprn_new==2676458351] <- "Room heater BB"
shcs$M17[shcs$uprn_new==2177514151] <- "Room heater BB"
shcs$M17[shcs$uprn_new==1676502308] <- "Room heater BB"
shcs$M17[shcs$uprn_new==8873452423] <- "Room heater BB"
shcs$M17[shcs$uprn_new==5700173539] <- "Room heater BB"

#The water heating fuel value for the following case was set as unobtainable.
#It has been changed to off-peak electric to match that of the spaceheating fuel.
shcs$M18[shcs$uprn_new==7403106567] <- "Off-peak electric"


#The primary heating fuel in these two cases are changed to oil as the primary
#heating fuel in the survey is "other" and the primary form of heating is a boiler
shcs$M5[shcs$uprn_new == 6115831852] <- "Oil"
shcs$M5[shcs$uprn_new == 4837178045] <- "Oil"

#The primary heating fuel in these two cases are changed to Gas (mains) as the
#primary heating fuel was communal heating,no CHP with a warm air system
shcs$M5[shcs$uprn_new == 1585265934] <- "Gas (mains)"

#The year of heating system is set to +1998 as the survey records it as not
#applicable and the flue type is set to balanced as it is also recorded as not
#applicable
shcs$M6[shcs$uprn_new == 6265548876] <- "1998+"
shcs$M8[shcs$uprn_new == 6265548876] <- "Balanced"

#The flue type is changed from balanced to fan assisted as the combination of
#heating system (condensing combi, gas boiler, pre 1998) with a balanced flue is
#not an option in SAP heating tables.
shcs$M8[shcs$uprn_new == 122595215] <- "Fan assisted"
shcs$M8[shcs$uprn_new == 2065580083] <- "Fan assisted"
shcs$M8[shcs$uprn_new == 8846926949] <- "Fan assisted"
shcs$M8[shcs$uprn_new == 2071095953] <- "Fan assisted"
shcs$M8[shcs$uprn_new == 7052823383] <- "Fan assisted"
shcs$M8[shcs$uprn_new == 155273248] <- "Fan assisted"

#The flue type is changed from balanced to fan assisted as the combination of
#heating system (condensing standard, gas boiler, pre 1998) with a balanced flue is
#not an option in SAP heating tables.
shcs$M8[shcs$uprn_new == 3127122104] <- "Fan assisted"
shcs$M8[shcs$uprn_new == 8853207500] <- "Fan assisted"

#Value in solid fuel heater is set to not applicable (changed from closed fire)
#as the heating system is defined as a room heater using peak electric
shcs$M15[shcs$uprn_new == 2314543027] <- "Not applicable"

#Value in solid fuel heater is set to closed fire (changed from not applicable)
#as the heating system is run on house coal in heated space
shcs$M15[shcs$uprn_new == 7212136000] <- "Closed fire"
shcs$M15[shcs$uprn_new == 1600903823] <- "Closed fire"

#Value in solid fuel heater is set to closed fire (changed from not applicable)
#as the heating system is run on Antracite nuts and grain in heated space
```

```
  shcs$M15[shcs$uprn_new == 8311930088] <- "Closed fire"

  #The percentage of double glazing column (Q47) contains 17 cases where the tenths
  #have been set to 55, this is not a valid option (10 is the highest value allowed)
  #due to the position of 5 and 8 on the numpad it has been assumed that these cases
  #should have been 88 (not applicable). Both of these values have been set to 0 for
  #calculations during elevations. This number can be set to any integer between 0
  #and 10
  shcs$Q47[shcs$Q47 == 55] <- 0
  shcs$Q47[shcs$Q47 == 88] <- 0

  #There are 22 cases which have no habitable floors (excluding room in roof)
  #for stock creation they have been given one floor - this value should not be
  #changed as it ensures the polygons for storeys are created correctly.
  shcs$J2[shcs$J2 == 0] <- 1

  #16 cases have missing entries (i.e. no levels) for floor ground floor type. These
  #should be set as 'unobtainable' and will then be allocated a ground floor type
  #according to the grndfloor.type function defined in cases
  shcs$N1_E[is.na(shcs$N1_E) == TRUE] <- "Unobtainable"
  return(shcs)
}
```

Cases

The following section of code is generated in cases.R

Make and save cases data

Create a .csv file using the function make.cases, which creates a dataframe containing a complete set of populated variables for the cases.csv stock file.

The cases stock file contains a series of information about the dwelling, including type, age, location, number of rooms, number of bedrooms, number of inhabitants, external outdoor space.

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the extension .csv

```
save.cases <- function(shcs, output) {
  write.csv(make.cases(shcs), file=output, row.names=FALSE)
}
```

Make the dataframe that contains all the information required for the cases.csv file

The dataframe is constructed and returned (as it is the last thing in the function that is assigned)

@param shcs - the scottish survey data

```
make.cases <- function(shcs) {
  #If a column in the dataframe is not taken directly from the survey the vector to
  #fill the column must first be generated.
  #Each column is created using a function:
  #For example, region.type is a function which requires the vector la from
  #the shcs dataframe (the survey data). The region.type function is defined in
  #this script.
  the.buildingtype <- building.type(shcs$C1, shcs$C2)
  the.builtformtype <- builtform.type(the.buildingtype)
  the.grndfloortype <- grndfloor.type(shcs$N1_E, shcs$C4)
  the.regiontype <- region.type(shcs$la)
  the.tenuretype <- tenure.type(shcs$tenure)
  the.loftexistence <- loft.existence(shcs$M28, shcs$C1, shcs$C2)
  the.draftlobbyexistence  <- draftlobby.existence(the.builtformtype)
  the.morphologytype <- morphology.type(shcs$rururb)
  the.livingareafaction <- living.areafaction(shcs$J1)
  the.frontplotdepth <- front.plotdepth(the.morphologytype,the.builtformtype)
  the.frontplotwidth <- front.plotwidth(the.morphologytype,the.builtformtype)
  the.backplotdepth <- back.plotdepth(the.morphologytype,the.builtformtype)
  the.backplotwidth <- back.plotwidth(the.morphologytype,the.builtformtype)
  the.hasaccesstooutsidespace <- has.accesstooutsidespace(the.morphologytype,
                                                          the.builtformtype)
  the.ongasgrid <- on.gasgrid(shcs$L1,shcs$M5)
  the.partlyownsroof <- partly.ownsroof(shcs$uprn_new)
  #creates a dataframe containing column names (word before the = sign) and a vector
  #to fill that column (vector after the = sign)
  data.frame(aacode = shcs$uprn_new
             ,adults = shcs$numadult
             ,backplotdepth = the.backplotdepth
```

```
              ,backplotwidth = the.backplotwidth
              ,buildyear = shcs$C4
              ,builtformtype = the.builtformtype
              ,children = shcs$numchild
              ,dwellingcaseweight = shcs$laWghtP
              ,grndfloortype = the.grndfloortype
              ,frontplotdepth = the.frontplotdepth
              ,frontplotwidth = the.frontplotwidth
              ,hasaccesstooutsidespace = the.hasaccesstooutsidespace
              ,hasdraftlobby = the.draftlobbyexistence
              ,hasloft = the.loftexistence
              ,householdcaseweight = shcs$laWghtP
              ,livingareafaction = the.livingareafaction
              ,morphologytype = the.morphologytype
              ,numofhabitalrooms = shcs$J1
              ,numofbedrooms = shcs$bedrooms
              ,ongasgrid = the.ongasgrid
              ,partlyownsroof = the.partlyownsroof
              ,regiontype = the.regiontype
              ,tenuretype = the.tenuretype
              )
}
```

Region

Scottish local authority (la field in shcs) to NHM region type

@param la - la column from SHCS

```
region.type <- function(la) {
    as.factor(checked.revalue(
        la,
        c("Aberdeen City" = "easternscotland",
          "Aberdeenshire" = "easternscotland",
          "Angus" = "easternscotland",
          "Argyll and Bute" = "westernscotland",
          "Clackmannanshire" = "easternscotland",
          "Dumfries and Galloway" = "westernscotland",
          "Dundee City" = "easternscotland",
          "East Ayrshire" = "westernscotland",
          "East Dunbartonshire" = "westernscotland",
          "East Lothian" = "easternscotland",
          "East Renfrewshire" = "westernscotland",
          "City of Edinburgh" = "easternscotland",
          "Eilean Siar" = "westernscotland",
          "Falkirk" = "westernscotland",
          "Fife" = "easternscotland",
          "Glasgow City" = "westernscotland",
          "Highland" = "northernscotland",
          "Inverclyde" = "westernscotland",
          "Midlothian" = "easternscotland",
          "Moray" = "easternscotland",
          "North Ayrshire" = "westernscotland",
          "North Lanarkshire" = "westernscotland",
          "Orkney Islands" = "northernscotland",
          "Perth and Kinross" = "easternscotland",
          "Renfrewshire" = "westernscotland",
          "Scottish Borders" = "easternscotland",
          "Shetland Islands" = "northernscotland",
          "South Ayrshire" = "westernscotland",
          "South Lanarkshire" = "westernscotland",
          "Stirling" = "westernscotland",
          "West Dunbartonshire" = "westernscotland",
          "West Lothian" = "easternscotland")))
}
```

Create morphologytype

There is only a 2-fold definition of morphology type in the scottish survey. This has been compared to the english 4-fold definition used in the NHM to determine which values to map against

@param morphology - rururb from scottish survey

```
morphology.type <- function(morphology){
  as.factor(checked.revalue(
    morphology,
    c("Urban"="urban"
      ,"Rural"="townandfringe")))
}
```

Tenure

Generate tenure type

@param tenure - tenure type column from scottish survey

```
tenure.type <- function(tenure) {
    as.factor(checked.revalue(
        tenure,
        c("owner-occupier" = "owneroccupied",
          "private-rented" = "privaterented",
          "LA/other public" = "localauthority",
          "HA/co-op" = "housingassociation",
          "Unobtainable" = "owneroccupied")))
}
```

Building type

Convert a building type (combination of C1 and C2) into a built form type

@param buildingtype - see common.R building.type

```
builtform.type <- function(buildingtype) {
  as.factor(checked.revalue(
    buildingtype,c(
      "Corner / enclosed end" = "endterrace",
      "Detached" = "detached",
      "Enclosed mid" = "midterrace",
      "Mid-terrace" = "midterrace",
      "Mid-terrace with passage" = "midterrace",
      "End terrace" = "endterrace",
      "Semi-detached" = "semidetached",
      "4-in-a-block" = "purposebuiltlowriseflat",
      "Flat from conversion" = "convertedflat",
      "Tenement" = "purposebuiltlowriseflat",
      "Tower or slab" = "purposebuilthighriseflat")))
}
```

Building construction

Map N1E to ground floor type

@param N1E - N1_E ground/lowest level floor type column from Scottish survey

```
grndfloor.type <- function(N1E, C4) {
  floor <- as.factor(checked.revalue(
                       N1E,
                       c("Susp timber" = "suspendedtimber",
                         "Susp not timber" = "solid",
                         "Solid" = "solid",
                         "Not applicable" = "solid",
                         "Unobtainable" = "solid")))


  floor <- ifelse(C4 <= 1929 & N1E == "Unobtainable", "suspendedtimber",
                  ifelse(C4 <= 1929 & N1E == "Not applicable", "suspendedtimber",
                         levels(floor)[floor]))

  return(floor)

}
```

Create existence of loft

@param loft - M28 is Loft hatch present from scottish survey There is not specific variable identifying the presence of a loft, so the presence of a loft hatch is used as a proxy for identifying the presence of a loft

```
loft.existence <- function(loft, C1, C2){
  loftins <- as.factor(checked.revalue(
                         loft,
                         c("No" = "FALSE",
                           "yes, not draughtproofed" = "TRUE",
                           "yes, draughtproofed" = "TRUE",
                           "Eaves door" = "TRUE",
                           "Not applicable/no loft" = "FALSE",
                           "Unobtainable" = "FALSE")))

  builtformtype <- building.type(C1, C2)

  is.flat <- is.flat(builtformtype)
  is.house <- is.house(builtformtype)

  loftins <- ifelse(is.house & loft == "Not applicable/no loft", "FALSE",
                    ifelse(is.house & loft != "Not applicable/no loft", "TRUE",
                           levels(loftins)[loftins]))
  return(loftins)

}
```

Create existence of draught lobby

This uses the rules detailed in the RD-SAP methodology (SAP 9.91 Appendix S) that specifies that if the building type is a flat (i.e. low-rise, high-rise or converted) there is a draught lobby present, whereas if the building type is not a flat (i.e. a house) then there is no draught lobby present

@param draftlobby - thebuiltformtype created through building.type

```
draftlobby.existence <- function(draftlobby){
  as.factor(checked.revalue(
    draftlobby,
    c("purposebuiltlowriseflat" = "TRUE",
      "endterrace" = "FALSE",
      "detached"  = "FALSE",
      "midterrace" = "FALSE",
      "convertedflat" = "TRUE",
      "semidetached" = "FALSE",
      "purposebuilthighriseflat" = "TRUE")))
}
```

Create partlyownsroof

Assumed all records to have 'FALSE' values as there is no information available about this in the scottish survey

@param partroof - from uprn_new ensuring that the vector created is the correct length

```
partly.ownsroof <- function(partroof){
  partroof <- "FALSE"
}
```

Gas grid

Create ongasgrid

This is created from the description of which mains services are available for each case (electric and or gas). However if the main heating fuel is mains(gas) the case is also forced to be true for ongasgrid.

@param services - from L1 (What mains services?)

@param heatingfuel - from space heating, used to overide cases where the main heating system is mains gas, but survey suggests off the gas grid

```
on.gasgrid <- function(services,heatingfuel){
  services <- as.factor(checked.revalue(
    services,
    c("Electricity only" = "FALSE"
      ,"Electricity and gas" = "TRUE"
      ,"Gas only" = "TRUE"
      ,"No services" = "FALSE"
      ,"Unobtainable" = "FALSE"
    )))
  services[heatingfuel=="Gas (mains)"] <- "TRUE"
  return(services)
}
```

Living area fraction

Create livingareafaction

No information is available for the dimensions of the living space. The only information available to calculate the living area fraction is the number of rooms, which can be used in conjunction with RD-SAP Table S16 in section S9.2,
which specifies the living area fraction to used depending on the total number of habitable rooms in a dwelling. Table S16 is recreated and used below.

@param livingarea - number of rooms(J1) from the scottish survey

```
living.areafaction <- function(livingarea){
  #Cases with more than 14 rooms are converted to having 14 rooms (less than 10
  #cases). This does not include the values that indicate unobtainable (88) or
  #unknown (99)
  livingarea[livingarea >= 14 & livingarea != 88 & livingarea != 99] <- 14
  #The number of roomms are mapped against living area fractions.
  #Unobtainable and unknoum are mapped to the same value as for cases with 5 rooms
  #as the average number of rooms per house in the scottish survey was 5 rooms.
  livingarea <- checked.renum(livingarea,
                              data.frame(a = c(   1,   2,   3,   4,   5,
                                                  6,   7,   8,   9,  10,
                                                  11,  12,  13,  14, 88,
                                                  99)
                                         , b = c( 0.75,0.50,0.30,0.25,0.21,
                                                  0.18,0.16,0.14,0.13,0.12,
                                                  0.11,0.10,0.10,0.09,0.21,
                                                  0.21)))
}
```

Outside space (plots)

Create frontplotdepth

No available information in the Scottish survey: frontplotdepth,frontplotwidth, backplotdepth, backplotwidth

These can be used in the NHM when installing such measures as ground source head pumps and biomass biomass whereby these can only be installed where properties have sufficient outdoor space to install ground coils or a fuel store.

Therefore it has been assumed that in all cases frontplotwidth and frontplotdepth is 0m For cases that are not flats and not urban it has been assumed that backplotdepth is 5m and backplotwidth is 10m, thereby creating an external area of 50m2.

These assumptions are consistant with previous stock creations for Scotland. However, these values can be changed below as required.

@param morphology - the.morphologytype calculated in cases.R

@param builtform - the.builtformtype calculated in cases.R

```
front.plotdepth <- function(morphology,builtform){
  #A data frame is created with morphology and builtform. Then an additional column
  #called plotdepth is created and populated with 0. This column is then returned by
  #the function
  frontdepth <- data.frame(morphology,builtform)
  frontdepth$plotdepth <- 0
  return(frontdepth$plotdepth)
}
```

Create frontplotwidth

see frontplot depth

@param morphology - the.morphologytype calculated in cases.R

@param builtform - the.builtformtype calculated in cases.R

```
front.plotwidth <- function(morphology,builtform){
  frontwidth <- data.frame(morphology,builtform)
  frontwidth$plotwidth <- 0
  return(frontwidth$plotwidth)
}
```

Create backplotdepth

see frontplot depth

@param morphology - the.morphologytype calculated in cases.R

@param builtform - the.builtformtype calculated in cases.R

```
back.plotdepth <- function(morphology,builtform){
  #A data frame is created with morphology and builtform. Then an additional column
  #called plotdepth is created and populated with 0.For cases where the morphology is
  #not urban or any flats then this is updated to 5.This column is then returned by
  #the function
  backdepth <- data.frame(morphology,builtform)
  backdepth$plotdepth <- 0
```

```
  backdepth <- within(backdepth,
                      plotdepth[morphology != "urban" &
                                  (builtform != "purposebuiltlowriseflat" &
                                  builtform != "purposebuilthighriseflat" &
                                  builtform != "convertedflat")] <- 5)
  return(backdepth$plotdepth)
}
```

Create backplotwidth

see frontplot depth

@param morphology - the.morphologytype calculated in cases.R

@param builtform - the.builtformtype calculated in cases.R

```
back.plotwidth <- function(morphology,builtform){
  backwidth <- data.frame(morphology,builtform)
  backwidth$plotwidth <- 0
  backwidth <- within(backwidth,
                      plotwidth[morphology != "urban" &
                                  (builtform != "purposebuiltlowriseflat" &
                                     builtform != "purposebuilthighriseflat" &
                                     builtform != "convertedflat")] <- 10)
  return(backwidth$plotwidth)
}
```

Create hasaccesstooutsidespace

This is a flag based on the plotdepths and plotwidths. It is made using the same logic that was used to create back.plotdepth as any cases where this is true will also have access to outside space.

UPDATE 27.08.2015: In the absence of any data, default assumption that dwellings/households have access to outdoor space

@param morphology - the.morphologytype calculated in cases.R

@param builtform - the.builtformtype calculated in cases.R

```
has.accesstooutsidespace <- function(morphology,builtform){
  outside <- data.frame(morphology,builtform)
  outside$space <- "TRUE"
  #outside <- within(outside,
  #                  space[morphology != "urban" &
  #                                  (builtform != "purposebuiltlowriseflat" &
  #                                     builtform != "purposebuilthighriseflat" &
  #                                     builtform != "convertedflat")] <- "TRUE")
  return(outside$space)
}
```

Roofs

The following section of code is generated in roofs.R

Make and save roofs data

Create a .csv file using the function make.roofs, which creates a dataframe containing a complete set of populated variables for the roofs.csv stock file.

The roofs stock file contains four variables with information on the roof structure , the roof covering material, roof construction type and the level of insulation in the roof/loft (depending on type of roof is present).

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the extension .csv

```
save.roofs <- function(shcs, output) {
  write.csv(make.roofs(shcs), file=output, row.names=FALSE)
}
```

Make the dataframe that contains all the information required for the roofs.csv file

The dataframe is constructed and returned (as it is the last thing in the function that is assigned)

@param shcs - the scottish survey data

```
make.roofs <- function(shcs) {
  the.coveringtype <- covering.type(shcs$Q21)
  the.structuretype <- structure.type(shcs$Q19,
                                      building.type(shcs$C1, shcs$C2))
  the.constructiontype <- construction.type(the.structuretype,
                                            the.coveringtype)
  the.insulationthickness <- insulation.thickness(shcs$loftins, shcs$M26)

  data.frame(aacode = shcs$uprn_new
             ,coveringtype = the.coveringtype
             ,insulationthickness = the.insulationthickness
             ,contructiontype = the.constructiontype
             ,structuretype = the.structuretype
  )
}
```

Roof type

Map covering type

This is entirely produced from Q21 in the SHCS. - 'asbestos' has no cognate in the NHM, and so is taken to be 'asphalt' - missing data is taken to be 'tile'

@param Q21 - column Q21 (principle roof cover) from the scottish survey

```
covering.type <- function(Q21) {
    ## preconditions
    as.factor(checked.revalue(
        Q21,
        c("Slates" = "slates",
          "Tiles" = "tile",
          "Felt" = "felt",
          "Asphalt" = "asphalt",
          "Asbestos" = "asphalt",
          "Metal" = "metal",
          "Other" = "thatch",
          "Not applicable" = "tile",
          "Unobtainable" = "tile")))
}
```

Roof structure type

The principal roof type column is used to map against roof types for the NHM, the building type is used in a set of rules to fill in missing values

@param Q19 - column Q19 (Principal roof type) from the scottish survey

@param buildingtype - a combined building type (see common.R, building_type)

```
structure.type <- function(Q19, buildingtype) {
  #If Q19 is unobtainable, a default is chosen based on the building type:
      # flats are assigned a flat roof whilst all other dwellings a pitched roof
  Q19.with.defaults <- as.factor(ifelse(Q19 == "Unobtainable",
                                        ifelse(is.flat(buildingtype),
                                               "Flat",
                                               "Pitched"),
                                        as.character(Q19)))
  #Having filled in the unobtainable values, then recode to the NHM convention.
  #Note: mandard is a standardised spelling mistake in the NHM
  checked.revalue(Q19.with.defaults,
                  c("Pitched" = "pitched",
                    "Flat" = "flat",
                    "Mono" = "flat",
                    "Mansard" = "mandard", #sic
                    "Half mansard" = "mandard"))
}
```

Infer roof construction type

The roof constuction type is inferred from the structure type and the covering type as described in the function below.

@param structuretype - per the structure.type function above

@param coveringtype - per the covering.type function above.

```
construction.type <- function(structuretype, coveringtype) {
  ifelse(
    structuretype == "pitched" | structuretype == "mandard",
    "pitchedslateortiles",
    ifelse(coveringtype == "thatch",
           "thatched",
           "flat"))
}
```

Roof insulation

Map insulation thickness

This is produced using 'loftins', except where that is invalid, in which case a value is taken from 'M26.' 'loftins' is a derived variable and M26 is the raw survey information. 'M26' values are only used when 'loftins' is missing data. The following logic is applied to values in the stock data: - none, flat roof unmeasured, not applicable, and unobtainable are all taken to be 0 - 300mm and above is taken to be 300mm.

@param loftins - column loftins (Roof/loft insulation + imputed) from the scottish survey

@param M26 - column M26 from the scottish survey

```
insulation.thickness <- function(loftins, M26) {
  #Recode the loftins column
  from.loftins <- checked.revalue(
    loftins,
    c("none" = "0",
      "flat roof unmeasured" = NA,
      "Not Applicable" = NA,
      "Unobtainable" = NA,
      "12mm"  = "0",  "25mm"  = "25",
      "50mm"  = "50",  "75mm"  = "75",
      "100mm" = "100", "150mm" = "150",
      "200mm" = "200", "250mm" = "250",
      "300mm and over" = "300"))

  #Overwrite any NA values in the result with M26 values
  #M26 has a slightly different coding to loftins.
  from.m26 <-
    checked.revalue(
      M26,
      c("None" = "0",
        "12mm"  = "0",  "25mm"  = "25",
        "50mm"  = "50",  "75mm"  = "75",
        "100mm" = "100", "150mm" = "150",
        "200m" = "200",  "250mm" = "250",
        "300mm or more" = "300",
        "Flat roof" = NA,
        "Not applicable" = NA,
        "Unobtainable" = NA))

  #The two columns are combined using the loftins, unless it is an NA in which
  #case M26 is used.
  #Both columns are converted to characters first to make it easy to combine
  from.both <- ifelse(is.na(from.loftins),
                      as.character(from.m26),
                      as.character(from.loftins))

  #Any remaining NAs are turned into zeros
  from.both[is.na(from.both)] <- "0"

  #The results from combining both columns are then turned into a numeric
  as.numeric(from.both)
}
```

Lighting

The following section of code is generated in lighting.R

Make and save lighting data

Create a .csv file using the function make.lighting, which creates a dataframe containing a complete set of populated variables for the lighting.csv stock file.

The lighting dataframe contains only one variable - the fraction of lighting in the dwelling which is low energy lighting.

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the extension .csv

```
save.lighting <- function(shcs, output) {
  write.csv(make.lighting(shcs), file=output, row.names=FALSE, na ="NULL")
}
```

Make the dataframe that contains all the information required for the lighting.csv file

The dataframe is constructed and returned (as it is the last thing in the function that is assigned)

@param shcs - the scottish survey data

```
make.lighting <- function(shcs) {
  the.lightingfraction <- lighting.fraction(shcs$L18)

  data.frame(aacode = shcs$uprn_new,
             fraction = the.lightingfraction)
}
```

Create lighting fraction

Create the lighting fraction

@param lighting - column L18 from the SHCS. This is the tenths of lighting which is low energy lighting. This is divided by 10 to get a fraction (i.e. value 0-1.0) For missing data, the low energy lighting is assummed to be 0.

```
lighting.fraction <- function(lighting){
  #Values of not applicable or unknown are turned into 0
  lighting <- checked.renum(lighting,
                            data.frame(a = c(88,99), b = c(0,0)))
  lighting <- lighting/10
}
```

Occupants

The following section of code is generated in occupants.R

Make and save occupants data

Create a .csv file using the function make.occupants, which creates a dataframe containing a complete set of populated variables for the occupants.csv stock file.

The information on occupants covers income, chief income earner's age, chief income earner's working hours, details on whether anyone in the dwelling is long term sick or disabled, whether anyone in the household is one benefits and how long the current inhabitants have lived in the dwelling. Not all this information is required and some variables can be left NULL/blank.

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the extension .csv

```
save.occupants <- function(shcs, output) {
  write.csv(make.occupants(shcs), file=output, row.names=FALSE, na = "")
}
```

Make the dataframe that contains all the information required for the occupants.csv file

The dataframe is constructed and returned (as it is the last thing in the function that is assigned)

@param shcs - the scottish survey data

```
make.occupants <- function(shcs) {
  the.datemovedin <- date.movedin(shcs$uprn_new)
  the.hasdisabledorsickoccupant <- has.disabledorsickoccupant(shcs$health)
  the.hasoccupantonbenefits <- has.occupantonbenefits(shcs)
  the.workinghours <- working.hours(shcs$uprn_new)

  data.frame(aacode = shcs$uprn_new
             ,chiefincomeearnersage = shcs$hihage
             ,datemovedin = the.datemovedin
             ,hasdisabledorsickoccupant = the.hasdisabledorsickoccupant
             ,hasoccupantonbenefits = the.hasoccupantonbenefits
             ,householdincomebeforetax = shcs$annhhinc
             ,workinghours = the.workinghours
  )
}
```

Occupant information

Make hasdisabledorsickoccupant

This is mapped directly from a column in the scottish survey

@param occupant - health (any long term sick/disabled in household) from scottish survey

```
has.disabledorsickoccupant <- function(occupant){
  as.factor(checked.revalue(
    occupant,
    c("No" = "FALSE"
      ,"Yes" = "TRUE"
      ,"Unobtainable" = "NULL")))
}
```

Make datemovedin

There is no information in the scottish survey about the date moved in or length of residency. This column cannot be blank and so is set as zero.

This variable uses Unix time - a system for describing instants in time, defined as the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970. It is used widely in Unix-like and many other operating systems and file formats.

Hence, by setting this variable to 0, all lengths of residency are state as being since 01/01/1970. However, this variable and the information do not form a core part of the stock data or how the stock is handle in the NHM and this will not have any direct impact on running scenarios in the model.

@param moving - uprn_new from the scottish survey is used so that that the vector created is of the correct length

```
date.movedin <- function(moving){
  moving <- as.data.frame(moving)
  moving[1] <- 0
  return(moving$moving)
}
```

Make workinghours

There is no information in the scottish survey about the working hours of the occupants. This column cannot be blank and so is set as "N".

@param working - uprn_new from the scottish survey is used so that that the vector created is of the correct length

```
working.hours <- function(working){
  working <- as.data.frame(working)
  working[1] <- "N"
  return(working$working)
}
```

Benefits information

Make hasoccupantsonbenefits

Information on benefits is recorded across a large number of columns - each type of benefit is recorded in a seperate column and are specific to the year in which the survey was carried out.

Benefits 2011: Note - jobseekers (income based) and jobseekers (contribution based) are not included as no survey entries have been recorded and their inclusion introduces invalid entries to the data.

@param benefits - the whole of the scottish survey data (benefit data is spread across a large number of columns)

```
has.occupantonbenefits <- function(benefits){
  #Benefits of interest from 2011 year are checked, if occupant recieved any of the
  #benefits listed then they are flagged as "TRUE" or having an occupant on benefits
  benefits$benefit2011<-
    ifelse( benefits$hh57_01_2011 == "Income support"
            | benefits$hh57_02_2011 == "Working Tax Credit (WTC)"
            | benefits$hh57_03_2011 == "Child tax Credit (CTC)"
            | benefits$hh57_06_2011 == "Housing Benefit"
            | benefits$hh57_08_2011 == "Child Benefit"
            | benefits$hh57_12_2011 == "Pension Credit"
            | benefits$hh58_01_2011 ==
              "Incapacity Benefit (formerly Invalidity Benefit)"
            | benefits$hh58_02_2011 == "Disability Living Allowance Care component"
            | benefits$hh58_03_2011 ==
              "Disability Living Allowance Mobility Component"
            | benefits$hh58_06_2011 == "Severe Disablement Allowance"
            | benefits$hh58_09_2011 ==
              "A disability premium with your Income Support/Housing Benefi"
            | benefits$hh58_10_2011 == "Attendance Allowance"
            ,"TRUE"
            ,"FALSE")

  #Benefits of interest from 2012 year are checked, if occupant recieved any of the
  #benefits listed then they are flagged as "TRUE" or having an occupant on benefits
  benefits$benefit2012<-
    ifelse( benefits$hh57_01_2012 == "A - Income Support"
            | benefits$hh57_02_2012 == "B - Employment and Support Allowance"
            | benefits$hh57_03_2012 == "C - Working Tax Credit (WTC)"
            | benefits$hh57_04_2012 == "D - Child Tax Credit (CTC)"
            | benefits$hh57_05_2012 ==
              "E - Jobseeker's Allowance (JSA) - Income Based"
            | benefits$hh57_06_2012 ==
              "F - Jobseeker's Allowance (JSA) - Contribution Based"
            | benefits$hh57_07_2012 == "G - Housing Benefit"
            | benefits$hh57_08_2012 == "H - Council Tax Benefit"
            | benefits$hh57_10_2012 == "J - Child Benefit"
            | benefits$hh57_14_2012 == "N - Pension Credit"
            | benefits$hh58_01_2012 ==
              "A - Incapacity Benefit (formerly Invalidity Benefit)"
            | benefits$hh58_02_2012 ==
              "B - Disability Living Allowance Care component"
            | benefits$hh58_03_2012 ==
```

```
              "C - Disability Living Allowance Mobility Component"
            | benefits$hh58_06_2012 == "F - Severe Disablement Allowance"
            | benefits$hh58_09_2012 ==
              "I - Disability premium with your Income Support/Housing Benefit"
            | benefits$hh58_10_2012 == "J - Attendance Allowance"
            ,"TRUE"
            ,"FALSE")

  #Benefits of interest from 2013 year are checked, if occupant recieved any of the
  #benefits listed then they are flagged as "TRUE" or having an occupant on benefits
  benefits$benefit2013<-
    ifelse( benefits$hh57_01_2013 == "A - Income Support"
            | benefits$hh57_02_2013 == "B - Employment and support allowance"
            | benefits$hh57_03_2013 == "C - Working Tax Credit (WTC)"
            | benefits$hh57_04_2013 == "D - Child Tax Credit (CTC)"
            | benefits$hh57_05_2013 ==
              "E - Jobseeker's Allowance (JSA) - Income Based"
            | benefits$hh57_06_2013 ==
              "F - Jobseeker's Allowance (JSA) - Contribution Based"
            | benefits$hh57_07_2013 == "H - Housing Benefit"
            | benefits$hh57_08_2013 == "I - Council Tax Benefit"
            | benefits$hh57_10_2013 == "K - Child Benefit"
            | benefits$hh57_14_2013 == "O - Pension Credit"
            | benefits$hh57_20_2013 == "G - Universal Credit"
            | benefits$hh58_01_2013 ==
              "A - Incapacity Benefit (formerly Invalidity Benefit)"
            | benefits$hh58_02_2013 ==
              "B - Disability Living Allowance Care Component"
            | benefits$hh58_03_2013 ==
              "C - Disability Living Allowance Mobility Component"
            | benefits$hh58_06_2013 == "H - Severe Disablement benefit"
            | benefits$hh58_09_2013 ==
              "K - Disability premium with Income Support/Housing Benefit"
            | benefits$hh58_10_2013 == "L - Attendance allowance"
            | benefits$hh58_19_2013 ==
              "D - Personal Independence Payment Mobility Component"
            | benefits$hh58_20_2013 ==
              "E - Personal Independence Payment Daily Living Component"
            ,"TRUE"
            ,"FALSE")

  #Amalgamate the three years into one piece of information - if the occupant
  #recieved benefits in the year they were interviewed, hasoccupantsonbenefits is
  #set to "TRUE".
  benefits$benefit[benefits$benefit2011 == "TRUE"
                   | benefits$benefit2012 == "TRUE"
                   | benefits$benefit2013 == "TRUE"] <- "TRUE"
  benefits$benefit[benefits$benefit2011 == "FALSE"
                   | benefits$benefit2012 == "FALSE"
                   | benefits$benefit2013 == "FALSE"] <- "FALSE"

  return(benefits$benefit)
}
```

Spaceheating

The following section of code is generated in spaceheating.R

Make and save spaceheating data

Create a .csv file using the function make.spaceheating, which creates a dataframe containing a complete set of populated variables for the space-heating.csv stock file.

The space-heating.csv file contains a set of information on the heating systems, including fuel type, efficencies and heating control systems. Storage heaters and storage combi require additional information if they are present in a dwelling, otherwise these columns do not need to be populated

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the extension .csv

```
save.spaceheating <- function(shcs, output, path.to.output) {
  write.csv(make.spaceheating(shcs,path.to.output), file=output, row.names=FALSE, na ="NULL")
}
```

Make a space-heating DTO from the SHCS

The survey does not contain all data required to create the space-heating dto; two additional sources of information are required: The sedbuk output file (created for the Scottish stock) and the look-up tables created from SAP 2012 space heating system tables. The can be found in the SAP 2012 documentation in 'Table 4a: Heating systems (space and water)' and 'Table 4b: Seasonal efficiency for gas and oil boilers'.

Two variables - boiler manufacturer and model in the SHCS allow boiler matching to occur to the latest sedbuk database. This returns a series of information about gas and oil boilers, including the type of boiler, whether it is condensing, if it is a storage combi, what the storage volume is, and the fuel type. In some instances the fuel type matched to the boiler description will be different to fuel type described in the survey. Any information taken from SEDBUK is deemed to be correct.

A series of other variable describing the heating system are available in the SHCS these are used to create a series of lookup tables in csv format which can be editted by the user. These assign the information available in SAP tables 4a and 4b where data matches are found.

teh sedbuk matching is performed first, and then all records which have not been successfully matched through sedbuk are matched to the SAP heating tables.

These sources of information are combined using the create.heating function to make the spaceheating dataframe.

@param shcs - the scottish survey data

```
make.spaceheating <- function(shcs, path.to.output) {

  #The spaceheating dataframe is first made. This is a combination of the scottish
  #survey data and information taken from sedbuk output and the heating table lookups
  spaceheating <- create.heating(shcs,path.to.output)

  the.basicefficiency <- basic.efficiency(spaceheating$basicefficiency)
  the.chpfraction <- chp.fraction(spaceheating$M5)
  the.communitychargingusagebased <- community.chargingusagebased(spaceheating$M5)
  the.electrictariff <- electric.tariff(spaceheating$L2,spaceheating$M2)
  the.fluetype <- flue.type(spaceheating$fluetype)
```

```
  the.winterefficiency <- winter.efficiency(spaceheating$winterefficiency)
  the.summerefficiency <- summer.efficiency(spaceheating$summerefficiency)
  the.iscondensing <- is.condensing(spaceheating$condensing)
  the.heatingsystemcontroltypes <- heating.systemcontroltypes(spaceheating$M21
                                                              ,spaceheating$M2)
  the.installationyear <- installation.year(spaceheating$M6
                                            ,spaceheating$installationyear)
  the.mainheatingfuel <- space.heatingfuel(spaceheating$checkedfueltype)
  the.secondaryheatingsystemtype <- secondary.heatingsystemtype(spaceheating$M21a
                                                                ,spaceheating$M22)
  the.spaceheatingsystemtype <-
    space.heatingsystemtype(spaceheating$spaceheatingsystemtype)
  the.storageheatercontroltype <- storage.heatercontroltype(spaceheating$M21
                                                            ,spaceheating$M2)
  the.storageheatertype <- storage.heatertype(spaceheating$M16
                                              ,spaceheating$spaceheatingsystemtype)

  data.frame(aacode = spaceheating$uprn_new
             ,basicefficiency = the.basicefficiency
             ,chpfraction = the.chpfraction
             ,communitychargingusagebased = the.communitychargingusagebased
             ,electrictariff = the.electrictariff
             ,fluetype = the.fluetype
             ,summerefficiency = the.summerefficiency
             ,winterefficiency = the.winterefficiency
             ,iscondensing = the.iscondensing
             ,heatingsystemcontroltypes = the.heatingsystemcontroltypes
             ,installationyear = the.installationyear
             ,mainheatingfuel = the.mainheatingfuel
             ,secondaryheatingsystemtype = the.secondaryheatingsystemtype
             ,spaceheatingsystemtype = the.spaceheatingsystemtype
             ,isstoragecombicylinderfactoryinsulated = "NULL"
             ,storagecombicylinderinsulationthickness =
               spaceheating$storeinsulationthickness
             ,isstoragecombicylinderthemostatpresent = "NULL"
             ,storagecombicylindervolume = spaceheating$storeboilervolume
             ,storagecombisolarvolume = spaceheating$storesolarvolume
             ,storageheatercontroltype = the.storageheatercontroltype
             ,storageheatertype = the.storageheatertype
  )
}
```

Create spaceheating

This function creates a table of space heating information for each house case.

It requires the sedbuk-output.csv and lup-####.csv files to be in the same folder as the scottish stock.

The sedbuk matching is performed using a java programme built by CSE. For the scotland stock conversion the sedbuk matching has been previously performed to produce an output table (sedbuk-output.csv) which can be matched on urpn_new

Following that process, the SAP lookup tables are then used to catch the remaining cases.

@param shcs - a dataframe containing all scottish survey data

```
create.heating <- function(shcs,path.to.output){
  #import sedbuk heating matches
  sedbuk <- sedbuk.heating(path.to.output)
  sedbuk.id <-data.frame(uprn_new=sedbuk$uprn_new)
  #match remaining cases against look up tables
  matched.cases <- heating.matching(shcs,path.to.output,sedbuk.id,
                                    "lup-boiler-4b.csv",c("M2","M5","M6","M7","M8"))
  matched.cases <- rbind(matched.cases,heating.matching(shcs,path.to.output
                      ,rbind(sedbuk.id,data.frame(uprn_new=matched.cases$uprn_new)),
                      "lup-electric-boiler.csv",c("M2","M5","M9","M13")))
  matched.cases <- rbind(matched.cases,heating.matching(shcs,path.to.output
                       ,rbind(sedbuk.id,data.frame(uprn_new=matched.cases$uprn_new)),
                       "lup-gas-room-heater.csv",c("M2","M5","M14","M6")))
  matched.cases <- rbind(matched.cases,heating.matching(shcs,path.to.output
                       ,rbind(sedbuk.id,data.frame(uprn_new=matched.cases$uprn_new)),
                       "lup-heat-pump.csv",c("M2","M5","M11")))
  matched.cases <- rbind(matched.cases,heating.matching(shcs,path.to.output
                       ,rbind(sedbuk.id,data.frame(uprn_new=matched.cases$uprn_new)),
                       "lup-solid-fuel-boiler.csv",c("M2","M5","M9","M15")))
  matched.cases <- rbind(matched.cases,heating.matching(shcs,path.to.output
                       ,rbind(sedbuk.id,data.frame(uprn_new=matched.cases$uprn_new)),
                       "lup-storage-heater.csv",c("M2","M16")))
  matched.cases <- rbind(matched.cases,heating.matching(shcs,path.to.output
                       ,rbind(sedbuk.id,data.frame(uprn_new=matched.cases$uprn_new)),
                       "lup-warm-air.csv",c("M2","M5","M12","M6","M8")))
  matched.cases <- rbind(matched.cases,heating.matching(shcs,path.to.output
                        ,rbind(sedbuk.id,data.frame(uprn_new=matched.cases$uprn_new)),
                        "lup-oil-room-heater.csv",c("M2","M5","M15","M6")))
  matched.cases <- rbind(matched.cases,heating.matching(shcs,path.to.output
                        ,rbind(sedbuk.id,data.frame(uprn_new=matched.cases$uprn_new)),
                        "lup-other-room-heater.csv",c("M2","M5","M15")))
  matched.cases <- rbind(matched.cases,heating.matching(shcs,path.to.output
                         ,rbind(sedbuk.id,data.frame(uprn_new=matched.cases$uprn_new)),
                         "lup-community-heating.csv",c("M2","M5")))

  #combine all spaceheating with the main table
  matched.sedbuk <- data.frame(uprn_new=sedbuk$uprn_new
                               ,basicefficiency=sedbuk$annualefficiency
                               ,winterefficiency=sedbuk$winterefficiency
                               ,summerefficiency=sedbuk$summerefficiency
                               ,spaceheatingsystemtype=sedbuk$boilertype
                               ,fluetype=sedbuk$fluetype
```

```
                                ,checkedfueltype=sedbuk$fueltype
                                ,condensing=sedbuk$condensing
                                ,installationyear=NA
  )

  matched.cases <-data.frame(uprn_new=matched.cases$uprn_new
                             ,basicefficiency=matched.cases$basicefficiency
                             ,winterefficiency=matched.cases$winterefficiency
                             ,summerefficiency=matched.cases$summerefficiency
                             ,spaceheatingsystemtype=
                               matched.cases$spaceheatingsystemtype
                             ,fluetype=matched.cases$fluetype
                             ,checkedfueltype=matched.cases$mainheatingfuel
                             ,condensing=as.factor(matched.cases$iscondensing)
                             ,installationyear=matched.cases$installationyear
  )

  matched.heating <- rbind(matched.sedbuk,matched.cases)
  sedbuk<-subset(sedbuk,select=c("uprn_new","storeboilervolume"
                                 ,"storesolarvolume","storeinsulationthickness"))
  matched.heating <- join(matched.heating,sedbuk,by="uprn_new")
  heating <- join(shcs,matched.heating,by="uprn_new")
  return(heating)
}
```

Spaceheating import files

This function imports data from the sedbuk-output (sedbuk-output.csv) file.

This file must be available in the same folder as the scottish stock. If a different file name is use then it must be corrected in the creation of the sedbuk data command below.

@param none

```
sedbuk.heating <- function(path.to.output){
  sedbuk <- read.csv(file.path(path.to.output,"sedbuk-output.csv"), header=TRUE)
  colnames(sedbuk)[names(sedbuk) == "aacode"] <- "uprn_new"
  sedbuk<-subset(sedbuk,select=c(-manufacturer,-brand,-model,-qualifier))
  return(sedbuk)
}
```

And once matching with sedbuyk data is complete, this function imports data from the SAP 4a and 4b look up tables...

These file must be available in the same folder as the scottish stock

@param shcs - a dataframe containing all survey data

@param matched.id - a vector containing the uprn_new/aacodes of all houses for which space-heating information has yet to be matched

@param name.to.heatinglookup - a character string which contains the name of the look-up file including it's extention (.csv)

@param columns.to.match - a character list of the column names that appear at the start of each look-up file (i.e c("M2","M5"))

```
heating.matching <- function(shcs,path.to.output,matched.id
                             ,name.to.heatinglookup,columns.to.match){
  boiler <- read.csv(file.path(path.to.output,name.to.heatinglookup), header=TRUE)
  if (("installationyear" %in% colnames(boiler))==FALSE){
    boiler$installationyear <- NA
  }
  if (("winterefficiency" %in% colnames(boiler))==FALSE){
    boiler$winterefficiency <- NA
  }
  if (("summerefficiency" %in% colnames(boiler))==FALSE){
    boiler$summerefficiency <- NA
  }

  unmatched <- subset(shcs,!shcs$uprn_new %in% matched.id$uprn_new
                      ,select=c("uprn_new",columns.to.match))
  unmatched<-join(unmatched,boiler,by=columns.to.match)
  matched<-subset(unmatched,is.na(unmatched$Heating.System.Type)==FALSE
                  ,select=c(uprn_new, winterefficiency, summerefficiency
                            ,basicefficiency, spaceheatingsystemtype, fluetype
                            ,mainheatingfuel, iscondensing,installationyear))
   return(matched)
}
```

Efficiency

This function turns the basic efficiency from a percentage to a decimal as required for the dto files

@param efficiency - a vector containing the basic efficiency of spaceheating

```
basic.efficiency <- function(efficiency){
  efficiency = efficiency/100
  return(efficiency)
}
```

This function turns the winter efficiency from a percentage to a decimal as required for the dto files

@param efficiency - a vector containing the winter efficiency of spaceheating

```
winter.efficiency <- function(efficiency){
  efficiency = efficiency/100
  return(efficiency)
}
```

This function turns the summer efficiency from a percentage to a decimal as required for the dto files

@param efficiency - a vector containing the summer efficiency of spaceheating

```
summer.efficiency <- function(efficiency){
  efficiency = efficiency/100
  return(efficiency)
}
```

Heating system type

Relabel the spaceheatingsystemtype to the correct formats for the NHM stock import

@param heatingsystem - a vector containing the heating system type from sedbuk and look-up data files

```
space.heatingsystemtype <- function(heatingsystem){
  heatingsystem <- as.factor(checked.revalue(
    heatingsystem,
    c("CPSU" = "cpsu"
      ,"INSTANT_COMBI" = "combi"
      ,"REGULAR" = "standard"
      ,"STORAGE_COMBI" = "storage_combi"
      ,"back_boiler" = "back_boiler"
      ,"combi" = "combi"
      ,"cpsu" = "cpsu"
      ,"standard" = "standard"
      ,"room_heater" = "room_heater"
      ,"storage_heater" = "storage_heater"
      ,"warm_air" = "warm_air"
      ,"community_heating_with_chp" = "community_heating_with_chp"
      ,"community_heating_without_chp" = "community_heating_without_chp"
      ,"ground_source_heat_pump" = "ground_source_heat_pump"
      ,"air_source_heat_pump" = "air_source_heat_pump"
    )))
  return(heatingsystem)
}
```

This function relabels elements of a vector to the correct versions required for the dto files

NOte: community_heat is not a fuel type and it is assumed that for the significant majority of community heating systems, mains_gas is the heating fuel.

@param fuel - a vector containing the main heating fuel type associated with the space heater

```
space.heatingfuel <- function(fuel){
  fuel <- as.factor(checked.revalue(
    fuel,
    c("MAINS_GAS" = "mains_gas"
      ,"mains_gas" = "mains_gas"
      ,"OIL" = "oil"
      ,"oil" = "oil"
      ,"bottled_lpg" = "bottled_lpg"
      ,"bulk_lpg" = "bulk_lpg"
      ,"biomass_pellets" = "biomass_pellets"
      ,"biomass_wood" = "biomass_wood"
      ,"biomass_woodchip" = "biomass_woodchip"
      ,"house_coal" = "house_coal"
      ,"electricity" = "electricity"
      ,"community_heat" = "mains_gas"
    )))
  return(fuel)
}
```

This function re-labels elements of a vector to the correct versions required for the dto files

@param flue - a vector containing the flue type associated with the space heater

```
flue.type <- function(flue){
  flue <- as.factor(checked.revalue(
    flue,
    c( "BalancedFlue" = "balancedflue"
       ,"FanAssistedBalancedFlue" = "fanassistedbalancedflue"
       ,"OpenFlue" = "openflue"
       ,"balancedflue" = "balancedflue"
       ,"fanassistedbalancedflue" = "fanassistedbalancedflue"
       ,"notapplicable" = "notapplicable"
       ,"openflue" = "openflue"
       ,"chimney" = "chimney"
    )))
    return(flue)
}
```

This function relables elements of a vector to the correct versions required for the dto files

@param condensing - a vector containing a flag to indicate wether each boiler is condensing or not, information from sedbuk output and look-up files

```
is.condensing <- function(condensing){
  condensing <- as.factor(checked.revalue(
    condensing,
    c("TRUE" = "true"
      ,"FALSE" = "false"
      ,"true" = "true"
      ,"false" = "false"
    )))
  return(condensing)
}
```

This function assigns an installation year

The installation year is assigned for the main space heating system from the information in the look-up files as a priority, if this is not available then installation year is determined from the bands provided in the survey

@param heatyear a - vector containing the band in which the main space heating system was installed from the survey (M6)

@param checkedheatyear - a vector containing the year in which the main space heating system was installed from the look-up files

```
installation.year <- function(heatyear,checkedheatyear){
  heatyear <- as.factor(checked.revalue(
    as.factor(heatyear),
    c("1998+" = "2005"
      ,"pre 1998" = "1995"
      ,"old system" = "1985"
      ,"Other" = "9999"
      ,"Not applicable" = "9999"
      ,"Unobtainable" = "9999"
    )))
  heatyear<-as.numeric(levels(heatyear))[heatyear]
```

```
  heatyear[heatyear==9999]<-NA
  checkedheatyear<-ifelse(is.na(checkedheatyear)==FALSE,checkedheatyear,heatyear)
  return(checkedheatyear)
}
```

Secondary heating system

Function creates the secondary heating system type - first a check is done on existance of secondary heating system, then if a secondary system exists, the data describing secondary heating system type and fuel is used to assigned a secondary heater type from the room heating information in SAP table 4a.

@param secondarysystem - a vector indicating existance of a secondary heating system taken from the survey (M21a) @param roomheaters - a vector indicating the type of room heater that exists taken from the survey (M22)

```
secondary.heatingsystemtype <- function(secondarysystem,roomheaters){
  secondarysystem<-as.factor(ifelse(secondarysystem=="Yes"
                                    ,as.character(roomheaters),
                                    as.character(secondarysystem)))
  secondarysystem <- as.factor(checked.revalue(
    secondarysystem,
    c("Closed solid fuel fire" = "gas_fire"
      ,"Elec room heaters" = "electric_room_heaters"
      ,"Gas,coal effect fire" = "gas_coal_effect_fire"
      ,"No" = "no_secondary_system"
      ,"No other room heaters" = "no_secondary_system"
      ,"Not applicable" = "no_secondary_system"
      ,"Open solid fuel fire" = "open_fire"
      ,"Other" = "electric_room_heaters"
      ,"Post 1980 gas fire" = "gas_fire_flueless"
      ,"Pre 1980 gas fire" = "gas_fire_open_flue"
      ,"Unobtainable" = "not_known"
    )))
  secondarysystem[is.na(secondarysystem)==TRUE] <- "no_secondary_system"
  return(secondarysystem)
}
```

Heating system controls

This function sets the heating system control types to the correct format for the dto files and ensures that the stock cannot have any system control types that are not possible options for the main space-heating system.

in the dto, the heating control variable is a list, separated by semi-colons, of all heating control systems present in the dwelling.

@param controls - a vector of controls from the survey

@param heatingsystem - a vector of heating system types from the survey

```
heating.systemcontroltypes <- function(controls,heatingsystem){
  #Storage heating controls are placed in a separate column and are left blank
  #in this function
  controls <- as.factor(checked.revalue(
    controls,
    c("No controls" = ""
      ,"Programmer only" = "programmer"
      ,"Room stat only" = "roomthermostat"
      ,"Programmer & room stat" = "programmer;roomthermostat"
      ,"Programmer, room stat & TRV" =
        "programmer;roomthermostat;thermostaticradiatorvalve"
      ,"Programmer & TRV" = "programmer;thermostaticradiatorvalve"
      ,"Boiler manager" = "boilerenergymanager" #boiler
      ,"TRV only" = "thermostaticradiatorvalve"
      ,"appliance stat" = "appliancethermostat" #room heaters
      ,"appliance stat & prog" = "programmer;appliancethermostat"
      ,"manual charge control" = ""
      ,"auto charge control" = ""
      ,"More than 1 stat" = "roomthermostat"
      ,"Time / temp zone control" = "timetemperaturezonecontrol"
      ,"Programmer, 2 stats" = "programmer;roomthermostat"
      ,"Other" = ""
      ,"No Controls" = ""
      ,"Unobtainable"  = ""
    )))
  #consistency checks
  #Storage heating controls are stored in a separate column
  #(storageheatercontroltype)
  controls[heatingsystem=="Storage heating"]<-""
  #Only room heaters can have appliance thermostats
  controls[(heatingsystem!="Room heater" &
              heatingsystem != "Room heater (bb no rads)") &
              (controls=="appliancethermostat" |
                 controls == "programmer;appliancethermostat")]<- ""
  #only boilers can have these types of controls
  controls[heatingsystem!="Boiler" & controls=="boilerenergymanager"]<- ""

  return(controls)
}
```

Storage heating

Function creates the controls for storage heaters

This function sets the storage heating system control types to the correct format for the dto files and ensures that stock cannot have any system control types that are not possible options for the storage heating systems.

@param controls - a vector of controls from the survey

@param heatingsystem - a vector of heating system types from the survey

```
storage.heatercontroltype <- function(controls,heatingsystem){
  controls <- as.factor(checked.revalue(
    controls,
    c("No controls" = "NULL"
      ,"Programmer only" = "NULL"
      ,"Room stat only" = "NULL"
      ,"Programmer & room stat" = "NULL"
      ,"Programmer, room stat & TRV" = "NULL"
      ,"Programmer & TRV" = "NULL"
      ,"Boiler manager" = "NULL"
      ,"TRV only" = "NULL"
      ,"appliance stat" = "NULL"
      ,"appliance stat & prog" = "NULL"
      ,"manual charge control" = "manualchargecontrol"
      ,"auto charge control" = "automaticchargecontrol"
      ,"More than 1 stat" = "NULL"
      ,"Time / temp zone control" = "NULL"
      ,"Programmer, 2 stats" = "NULL"
      ,"Other" = "NULL"
      ,"No Controls" = "NULL"
      ,"Unobtainable"  = "NULL"
    )))
  #only storage heaters control types are recorded in this column, set to null in all
  #other cases.
  controls[heatingsystem!="Storage heating"]<-"NULL"
  return(controls)
}
```

Map the type of storage heater

Function creates the storage heater type by re-labeling storageheater to required values for the NHM stock import, it then ensures that only heating systems of the type storage heater have a storage heater type.

@param storageheater - vector containing information about the type of storage heater taken from the survey (M16)

```
storage.heatertype <- function(storageheater,heatingsystem){
  storageheater <- as.factor(checked.revalue(
    storageheater,
    c("New style" = "slimline"
      ,"Fan Assisted" = "fan"
      ,"Old Style" = "oldlargevolume"
      ,"Integrated storage / direct" = "integrateddirectacting"
      ,"Under Floor" = "oldlargevolume" #override as not an option in the NHM
      ,"Not applicable" = "NULL"
```

```
      ,"Unobtainable" = "NULL"
    )))
  storageheater<-as.factor(ifelse(heatingsystem=="storage_heater"
                                  ,as.character(storageheater)
                                  ,"NULL"))
  return(storageheater)
}
```

Community heating

This function sets the chpfraction of chp heating systems. When unknown, the default value for the fraction of heat optained from a CHP system is 0.35.

@param chp a vector containing the spaceheating fuel type from the survey

```
chp.fraction <- function(chp){
  chp <- as.factor(checked.revalue(
    chp,
    c("Gas (mains)" = "NULL"
      ,"Bulk LPG" = "NULL"
      ,"Bottled gas" = "NULL"
      ,"Oil" = "NULL"
      ,"House coal" = "NULL"
      ,"Smokeless fuel" = "NULL"
      ,"Antracite nuts and grain" = "NULL"
      ,"Wood chips" = "NULL"
      ,"Wood logs" = "NULL"
      ,"Wood pellets" = "NULL"
      ,"Peak electric" = "NULL"
      ,"Off-peak electric" = "NULL"
      ,"Communal heating, no CHP" = "NULL"
      ,"Communal heating, with CHP" = "0.35"
      ,"Biogas" = "NULL"
      ,"Dual fuel" = "NULL"
      , "Other" = "NULL"
      ,"Not applicable" = "NULL"
      ,"Unobtainable" = "NULL"
    )))
  return(chp)
}
```

This function sets communitychargingusagebased to true if a communal system. When unknown, it is assumed that charging for community systems is usage based.

The default value

@param communal a vector containing the spaceheating fuel type from the survey

```
community.chargingusagebased <- function(communal){
  communal <- as.factor(checked.revalue(
    communal,
    c("Gas (mains)" = "NULL"
      ,"Bulk LPG" = "NULL"
      ,"Bottled gas" = "NULL"
      ,"Oil" = "NULL"
      ,"House coal" = "NULL"
      ,"Smokeless fuel" = "NULL"
      ,"Antracite nuts and grain" = "NULL"
      ,"Wood chips" = "NULL"
      ,"Wood logs" = "NULL"
      ,"Wood pellets" = "NULL"
      ,"Peak electric" = "NULL"
      ,"Off-peak electric" = "NULL"
```

```
      ,"Communal heating, no CHP" = "true"
      ,"Communal heating, with CHP" = "true"
      ,"Biogas" = "NULL"
      ,"Dual fuel" = "NULL"
      , "Other" = "NULL"
      ,"Not applicable" = "NULL"
      ,"Unobtainable" = "NULL"
    )))
  return(communal)
}
```

Waterheating

The following section of code is generated in waterheating.R

Make and save waterheating data

Create a .csv file using the function make.waterheating, which creates a dataframe containing a complete set of populated variables for the water-heating.csv stock file.

The water-heating.csv file contains a set of information on the hot water heating systems, including fuel type and efficencies. However, a significant majority of dwellings will have their hot water provide by their space heating systems (e.g. mains gas combi boilers or standard boilers with a water tank). In these instances, much less information is required in the water-heating.csv and data that describes the efficiency or fuel used by the water heating system will be taken from the space heating system for that case.

Otherwise the water heating data in 'Table 4a: Heating systems (space and water)' in the SAP 2012 documentation is used to allocate heating system types and efficiencies where a dwelling has separate space heating and water heating systems.

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the extension .csv

@param path.to.input - the path to the folder where the survey data and lookup tables are placed

@param path.to.outputs - the path to the output folder

```
save.waterheating <- function(shcs, output, path.to.input, path.to.output) {
  write.csv(make.waterheating(shcs,path.to.input,path.to.output)
            , file=output, row.names=FALSE, na ="NULL")
}
```

Make the dataframe that contains all the information required for the waterheating.csv file

The dataframe is constructed and returned (as it is the last thing in the function that is assigned)

@param shcs - the scottish survey data

@param path.to.input - the path to the folder where the survey data and lookup tables are placed

@param path.to.outputs - the path to the output folder

```
make.waterheating <- function(shcs,path.to.input,path.to.output) {
  #First the waterheating dataframe is created from information contained in the
  #scottish survey, the spaceheating.csv and the waterheating look up table
  waterheating <- create.waterheating(shcs,path.to.output,path.to.input)

  the.basicefficiency <- wh.basic.efficiency(waterheating$basicefficiency
                                             ,waterheating$spaceheatingbasicefficiency
                                             ,waterheating$withcentralheating)
  the.chpfraction <- wh.chp.fraction(waterheating$M18
                                     ,waterheating$withcentralheating)
  the.communitychargingusagebased <-
    wh.community.chargingusagebased(waterheating$M18,waterheating$withcentralheating)
  the.electrictariff <- electric.tariff(waterheating$L2,waterheating$M2)
  the.fluetype <- wh.flue.type(waterheating$fluetype)
  #No information available about summer or winter efficiency waterheating
```

```
the.summerefficiency <- rep("NULL", length(waterheating$uprn_new))
the.winterefficiency <- rep("NULL", length(waterheating$uprn_new))
the.cylinderfactoryinsulated <-
  cylinder.factoryinsulated(waterheating$M30,waterheating$waterheatingsystemtype
                            ,waterheating$immersionheatertype
                            ,waterheating$spaceheatingsystemtype)
the.cylinderinsulationthickness <-
  cylinder.insulationthickness(waterheating$M31
                               ,waterheating$waterheatingsystemtype
                               ,waterheating$immersionheatertype
                               ,waterheating$spaceheatingsystemtype)
the.cylinderthermostatpresent <-
  cylinder.thermostatpresent(waterheating$M32
                             ,waterheating$waterheatingsystemtype
                             ,waterheating$immersionheatertype
                             ,waterheating$spaceheatingsystemtype)
the.cylindervolume <- cylinder.volume(waterheating$M29
                                      ,waterheating$waterheatingsystemtype
                                      ,waterheating$immersionheatertype
                                      ,waterheating$spaceheatingsystemtype)
#No information available about installation year of waterheating
the.installationyear <- rep("NULL", length(waterheating$uprn_new))
the.mainheatingfuel <- wh.main.heatingfuel(waterheating$M18
                                           ,waterheating$spaceheatingmainfuel
                                           ,waterheating$withcentralheating)
the.solarhotwaterpresent <- solar.hotwaterpresent(waterheating$D9)
the.solarstoreincylinder <- solar.storeincylinder(the.solarhotwaterpresent
                                                  ,the.cylindervolume)
the.solarstorevolume <- solar.storevolume(the.solarhotwaterpresent
                                          ,the.cylindervolume)
the.withcentralheating <- with.centralheating(waterheating$withcentralheating)

data.frame(aacode = waterheating$uprn_new
           ,basicefficiency = the.basicefficiency
           ,chpfraction = the.chpfraction
           ,communitychargingusagebased = the.communitychargingusagebased
           ,electrictariff = the.electrictariff
           ,fluetype = the.fluetype
           ,summerefficiency = the.summerefficiency
           ,winterefficiency = the.winterefficiency
           ,cylinderfactoryinsulated = the.cylinderfactoryinsulated
           ,cylinderinsulationthickness = the.cylinderinsulationthickness
           ,cylinderthermostatpresent = the.cylinderthermostatpresent
           ,cylindervolume = the.cylindervolume
           ,immersionheatertype = waterheating$immersionheatertype
           ,installationyear = the.installationyear
           ,mainheatingfuel = the.mainheatingfuel
           ,solarhotwaterpresent = the.solarhotwaterpresent
           ,solarstoreincylinder = the.solarstoreincylinder
           ,solarstorevolume = the.solarstorevolume
           ,waterheatingsystemtype = waterheating$waterheatingsystemtype
           ,withcentralheating = the.withcentralheating
)
```

```
}
```

Create waterheating

The dataframe is created by importing the spaceheating file that was created for the stock and matching that with waterheating systems from the waterheating look up table, which has been created using a subset of information contained in the 'Table 4a: Heating systems (space and water)' in the SAP 2012 documentation.

'lup-water-heating.csv' represents this information and is stored with the other lookup table csv files for heating systems.

@param shcs - the scottish survey data

@param path.to.input - the path to the folder where the survey data and lookup tables are placed

@param path.to.outputs - the path to the output folder

```
create.waterheating <- function(shcs,path.to.output,path.to.input){
  #import the spaceheating data that was created by spaceheating.R
  spaceheating <- read.csv(file.path(path.to.output,"space-heating.csv")
                           , header=TRUE)
  #Rename the columns to allow the join onto the scottish survey without overwriting
  #required variables
  spaceheating <- data.frame(uprn_new = spaceheating$aacode
                             ,spaceheatingsystemtype =
                               spaceheating$spaceheatingsystemtype
                             ,spaceheatingbasicefficiency =
                               spaceheating$basicefficiency
                             ,spaceheatingmainfuel = spaceheating$mainheatingfuel
                             ,withcentralheating = 0)
  #Join the spaceheating dataframe to the scottish survey
  spaceheating <- join(spaceheating,shcs,by="uprn_new")
  #If the waterheating is flagged as from mains heating and the main heating type
  #is not a room heater or a storage heater then with central heating is set to 1.
  spaceheating$withcentralheating[spaceheating$M17=="Mains heating" &
                          (spaceheating$spaceheatingsystemtype != "storage_heater" &
                          spaceheating$spaceheatingsystemtype != "room_heater")] <- 1
  #If the main heating system is combi or cpsu it is assumed that waterheating comes
  #from the main heating system and the withcentralheating is set to 1.
  spaceheating$withcentralheating[spaceheating$spaceheatingsystemtype == "combi" |
                                    spaceheating$spaceheatingsystemtype == "cpsu"] <- 1
  #The waterheating look up file is imported
  waterheating <- read.csv(file.path(path.to.input,"lup-water-heating.csv")
                           , header=TRUE,na.strings = "")
  #Ensures that only stock where withcentralheating is false
  #have waterheating matched
  waterheating$withcentralheating <- 0
  matched.waterheating <- join(spaceheating,waterheating
                               ,by=c("M17","M18","withcentralheating"))
  return(matched.waterheating)
}
```

Waterheating systems

With central heating

Values are mapped against true and false for the NHM, converting 1 to TRUE and 0 to FALSE (as allocated above)

@param withcentralheating - flag created during create.waterheating value is 0 if there is a separate water heating system to the spaceheating system

```
with.centralheating <- function(withcentralheating){
  withcentralheating <- as.factor(checked.revalue(
    as.factor(withcentralheating)
    ,c("0" = "FALSE"
       ,"1" = "TRUE"
    )))
  return(withcentralheating)
}
```

Main heating fuel (for water systems)

The water heating fuel is determined from the information cotained in the water heating fuel type variable in the SHCS unless the hot water is produced by the main central heating.

NOte: community_heat is not a fuel type and it is assumed that for the significant majority of community heating systems, mains_gas is the heating fuel.

@param wh.fuel - the heating fuel type of waterheating which comes from the lookup table

@param sh.fuel - the heating fuel type from the spaceheating file

@param ch - flag indicating if the waterheating is provided by the spaceheating 1 indicates waterheating is provided by spaceheating

```
wh.main.heatingfuel <- function(wh.fuel, sh.fuel, ch){
  #waterheating fuel column is changed to the correct values for the NHM.
  wh.fuel<-as.factor(checked.revalue(
    wh.fuel,
    c("Gas (mains)" = "mains_gas"
      ,"Bulk LPG"  = "bulk_lpg"
      ,"Bottled gas" = "bottled_lpg"
      ,"Oil" = "oil"
      ,"House coal" = "house_coal"
      ,"Smokeless fuel" = "house_coal"
      ,"Antracite nuts and grain" = "house_coal"
      ,"Wood chips" = "biomass_woodchip"
      ,"Wood logs" = "biomass_wood"
      ,"Wood pellets" = "biomass_pellets"
      ,"Peak electric" = "electricity"
      ,"Off-peak electric" = "electricity"
      ,"Communal heating, no CHP"  = "mains_gas"
      ,"Communal heating, with CHP" = "mains_gas"
      ,"Biogas" = "NULL"
      ,"Dual fuel" = "house_coal"
      ,"Other"  = "NULL"
      ,"Not applicable" = "NULL"
      ,"Unobtainable" = "NULL"
```

```
    )))
  #The two columns are then combined
  all.fuel <- ifelse(ch==1,"",levels(wh.fuel)[wh.fuel])
  return(all.fuel)
}
```

Basic efficiency

Basic efficiency is made from the basic efficiency of waterheating unless the heating comes from central heating

@param wh.efficiency - the efficiency of waterheating which comes from the lookup table

@param sh.efficiency - the efficiency of spaceheating which comes from spaceheating file created by the spaceheating.R script

@param ch - flag indicating if the waterheating is provided by the spaceheating 1 indicates waterheating is provided by spaceheating

```
wh.basic.efficiency <- function(wh.efficiency, sh.efficiency, ch){
  all.efficiency <- ifelse(ch==1,0,(wh.efficiency/100))
  return(all.efficiency)
}
```

Flue type

Flue type of the waterheating system is mapped to the correct values

@param flue - the flue type of the waterheating system from the waterheating lookup table

```
wh.flue.type <- function(flue){
    as.factor(checked.revalue(
    flue
    ,c("notapplicable" = "notapplicable"
       ,"openflue"="openflue"
    )))
  return(flue)
}
```

Cylinder information

Cylinder factory insulated

Values are mapped against true and false for the NHM and checked for consistency

@param factory - M30 instalation type for hot water cylinder from SHCS

@param w.heating - water heating system type

@param immersion - type of immersion heater (single/duel/null)

@param s.heating - spaceheating system type

```
cylinder.factoryinsulated <- function(factory,w.heating,immersion,s.heating){
  factory <- as.factor(checked.revalue(
    factory,
    c("Sprayed" = "TRUE"
      ,"Jacket" = "FALSE"
      ,"Encapsulated" = "TRUE"
      ,"Both" =  "TRUE"
      ,"No Insulation" = "FALSE"
      ,"No hw storage" = "NULL"
      ,"Unobtainable"  = "FALSE")))
  # These system types should not have cylinders
  factory[s.heating == "cpsu" | s.heating == "combi" | w.heating == "multipoint"
          | w.heating == "singlepoint"] <- "NULL"
  #These system types should all have cylinders,
  #if no information assume no factory insulation
  factory[factory == "NULL" & (w.heating == "back_boiler"
                               | w.heating == "standard_boiler"
                               | immersion == "dual_coil"
                               | immersion == "single_coil")] <- "FALSE"
  return(factory)
}
```

Cylinder insulation thickness

Values are mapped against true and false for the NHM and checked for consistency

@param thickness - M31 cylinder insulation thickness from scottish survey

@param w.heating - water heating system type

@param immersion - type of immersion heater (single/duel/null)

@param s.heating - spaceheating system type

```
cylinder.insulationthickness <- function(thickness,w.heating,immersion,s.heating){
  thickness <- checked.renum(thickness,
                             data.frame(a = c(888,999), b = c(NA,NA)))
  # These system types should not have cylinders
  thickness[s.heating == "cpsu" | s.heating == "combi" | w.heating == "multipoint"
            | w.heating == "singlepoint"] <- NA
  #These system types should all have cylinders,
  #if no information assume 0 insulation thickness
  thickness[is.na(thickness) == "TRUE" & (w.heating == "back_boiler"
                                          | w.heating == "standard_boiler"
                                          | immersion == "dual_coil"
```

```
                                          | immersion == "single_coil")] <- 0
  return(thickness)
}
```

Cylinder thermostat present

Values are mapped against true and false for the NHM and checked for consistency

@param thermostat - M32 cylinder thermostat present from scottish survey

@param w.heating - water heating system type

@param immersion - type of immersion heater (single/duel/null)

@param s.heating - spaceheating system type

```
cylinder.thermostatpresent <- function(thermostat,w.heating,immersion,s.heating){
  thermostat <- as.factor(checked.revalue(
    thermostat
    ,c("Yes" = "TRUE"
       ,"No" = "FALSE"
       ,"Not applicable" = "NULL"
       ,"Unobtainable" = "FALSE")))
  # These system types should not have cylinders
  thermostat[s.heating == "cpsu" | s.heating == "combi"
             | w.heating == "multipoint" | w.heating == "singlepoint"] <- "NULL"
  #These system types should all have cylinders, if no information assume
  #no thermostat
  thermostat[thermostat == "NULL" & (w.heating == "back_boiler"
                                     | w.heating == "standard_boiler"
                                     | immersion == "dual_coil"
                                     | immersion == "single_coil")] <- "FALSE"
  return(thermostat)
}
```

cylinder volume

Values are mapped against true and false for the NHM and checked for consistency, using information contained in the RD SAP documentation on typical sizes of cylinders.

@param volume - M29 cylinder volume from SHCS

@param w.heating - water heating system type

@param immersion - type of immersion heater (single/duel/null)

@param s.heating - spaceheating system type

```
cylinder.volume <- function(volume,w.heating,immersion,s.heating){
  volume <-as.factor(checked.revalue(
    volume
    ,c("Small (<90 L)" = "80" ##SAP 2009 table 2a value is 80
       ,"Normal (90-130 L)" = "110"
       ,"Medium (130-170 L)" = "140" ##SAP 2009 table 2a value is 140
       ,"Large (> 170 L)" = "210"
       ,"No hw storage" = "NULL"
       ,"Unobtainable" = "110" ##Assume normal size
    )))
```

```
  # These system types should not have cylinders
  volume[s.heating == "cpsu" | s.heating == "combi" | w.heating == "multipoint"
         | w.heating == "singlepoint"] <- "NULL"
  #These system types should all have cylinders, if they do not then assume
  #normal size
    volume[volume == "NULL" & (w.heating == "back_boiler"
                               | w.heating == "standard_boiler"
                               | immersion == "dual_coil"
                               | immersion == "single_coil")] <- "110"
  return(volume)
}
```

Community heating

Set the chpfraction of chp heating systems

When unknown, the default value for the fraction of heat optained from a CHP system is 0.35.

@param wh.chp a vector containing the waterheating fuel type

@param ch - flag indicating if the waterheating is provided by the spaceheating 1 indicates waterheating is provided by spaceheating

```
wh.chp.fraction <- function(wh.chp,ch){
  wh.chp <- as.factor(checked.revalue(
    wh.chp,
    c("Gas (mains)" = "NULL"
      ,"Bulk LPG" = "NULL"
      ,"Bottled gas" = "NULL"
      ,"Oil" = "NULL"
      ,"House coal" = "NULL"
      ,"Smokeless fuel" = "NULL"
      ,"Antracite nuts and grain" = "NULL"
      ,"Wood chips" = "NULL"
      ,"Wood logs" = "NULL"
      ,"Wood pellets" = "NULL"
      ,"Peak electric" = "NULL"
      ,"Off-peak electric" = "NULL"
      ,"Communal heating, no CHP" = "NULL"
      ,"Communal heating, with CHP" = "0.35"
      ,"Biogas" = "NULL"
      ,"Dual fuel" = "NULL"
      , "Other" = "NULL"
      ,"Not applicable" = "NULL"
      ,"Unobtainable" = "NULL"
    )))
  wh.chp[ch==1] <- "NULL"
  return(wh.chp)
}
```

Community charging usage based

This function sets 'communitychargingusagebased' to TRUE if a communal system. When unknown, it is assumed that charging for community systems is usage based. #' @param wh.communal a vector containing the spaceheating fuel type from the survey

@param ch - flag indicating if the waterheating is provided by the spaceheating 1 indicates waterheating is provided by spaceheating

```
wh.community.chargingusagebased<- function(wh.communal,ch){
  wh.communal <- as.factor(checked.revalue(
    wh.communal,
    c("Gas (mains)" = "NULL"
      ,"Bulk LPG" = "NULL"
      ,"Bottled gas" = "NULL"
      ,"Oil" = "NULL"
      ,"House coal" = "NULL"
      ,"Smokeless fuel" = "NULL"
```

```
      ,"Antracite nuts and grain" = "NULL"
      ,"Wood chips" = "NULL"
      ,"Wood logs" = "NULL"
      ,"Wood pellets" = "NULL"
      ,"Peak electric" = "NULL"
      ,"Off-peak electric" = "NULL"
      ,"Communal heating, no CHP" = "true"
      ,"Communal heating, with CHP" = "true"
      ,"Biogas" = "NULL"
      ,"Dual fuel" = "NULL"
      , "Other" = "NULL"
      ,"Not applicable" = "NULL"
      ,"Unobtainable" = "NULL"
    )))
  wh.communal[ch==1] <- "NULL"
  return(wh.communal)
}
```

Solar hot water heating

Existance of solar hot water

If an area bigger than zero of solar hot water panels exist assume that case has solar hot water

@param solar - D9 (% roof with solar installed) from the scottish survey

```
solar.hotwaterpresent <- function(solar){
  #88: not applicable and 99: unobtainable
  solar[solar == 88 | solar == 99] <- NA
  solar[solar > 0] <- "TRUE"
  solar[is.na(solar)=="TRUE"] <- "FALSE"
    return(solar)
}
```

Solar stored in cylinder

If solar hot water exists and there is a cylinder for hot water then it is assumed that all hot water is stored in the same cylinder as no other information is present in the scottish survey

@param solar - the.solarhotwaterpresent from the solar.howaterpresent function

@param volume - the volume of the hotwater cyclinder

```
solar.storeincylinder <- function(solar,volume){
  solar <- ifelse(solar=="TRUE" & volume != "NULL","TRUE","FALSE")
  return(solar)
}
```

Solar stored volume

If solar hot water exists and there is a cylinder for hot water then it is assumed that half the cylinder volume is used for storing hot water from the solar thermal system. If there is no cylinder for the hot water system it is assumed that a tank of volume 75 has been installed for the solar thermal system.

@param solar - the.solarhotwaterpresent from the solar.howaterpresent function

@param volume - the volume of the hotwater cyclinder

```
solar.storevolume <- function(solar,volume){
  solar <- ifelse(solar=="TRUE",as.numeric(levels(volume)[volume])/2,0)
  solar[is.na(solar)=="TRUE"] <- 75
  return(solar)
}
```

Storeys

The following section of code is generated in storeys.R

Make and save cases data

Create a .csv file using the function make.storeys, which creates a dataframe containing a complete set of populated variables for the storeys.csv stock file.

The storeys file contains a set of information on each habitable storey in each dwelling, including the location of each storey and its' dimensions.

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the extension .csv

```
save.storeys <- function(shcs, output) {
  write.csv(make.storeys(shcs), file=output, row.names=FALSE,na = "NULL")
}
```

Make the dataframe that contains all the information required for the storeys.csv file

The dataframe is constructed and returned (as it is the last thing in the function that is assigned)

@param shcs - the scottish survey data

```
make.storeys <- function(shcs) {
  #First all possible storeys are made (1 to 5) from the survey, if a floor does
  #not exist then all the information about that storey will be zero. These storeys
  #are then filtered and storeys are assigned the actual name of the storey.
  #Any additional storeys (i.e. basements) are added.
  #Finally the story height is made by adding 0.25 to the ceiling height of all
  #floors (except basement and ground floor) following SAP methodology.
  storeys <- make.polypointsandceilingheights(shcs)
  storeys <- assigning.polypointsandceilingheights(shcs,storeys)
  storeys <- make.storeyheight(storeys)

  storeys <- data.frame(aacode = storeys$uprn_new
              ,ceilingheight = storeys$ceilingheight
              ,type = storeys$type
              ,polygonxpoints = storeys$polygonxpoints
              ,polygonypoints = storeys$polygonypoints
              ,polypoints = storeys$polypoints
              ,storyheight = storeys$storyheight)

   #additional removal of anyfloors that don;t exist
   storeys <- subset(storeys, ceilingheight != 0, drop = FALSE)

  return(storeys)
}
```

Create storeys geometry

Function creates storeys geometries

All geometry (polygon points and ceiling heights) are created for all storeys available in the scottish survey. There is floor area and ceiling height available for the 'ground floor/lowest floor', 'first floor', 'second floor', 'third floor and above', 'room in roof' and 'extension 1' and 'extension 2'. These are used to create the coordinates of a polygon which describes the floor plan of each storey.

There is no information available in the survey about the location of extension 1 or extension 2. It has been assumed that extension 1 is always located at the front of the building at level one, as this is always an exposed unattached face. Extension 2, if it exists in a dwelling, has also been located on at the front of the building but at level two unless there isn't a level two, in which case it has been located at the back of level one (all buildings to which this applies ~200 have unattached back elevations).

@param shcs - the scottish survey data

```
make.polypointsandceilingheights <- function(shcs){
  #Select columns of interest (area and height for all floor areas)
  storeys <- subset(shcs,select = c(uprn_new
                                    ,N1_A,N1_C
                                    ,N2_A,N2_C
                                    ,N3_A,N3_C
                                    ,N4_A,N4_C
                                    ,N5_A,N5_C
                                    ,N6_A,N6_C
                                    ,N7_A,N7_C))
  storeys <- gather(storeys,key,value,-uprn_new)
  storeys <- separate(storeys,variable,into = c("floor","dimension"),sep="_")
  storeys <- spread(storeys,dimension,value)

  #If data is not applicable or unobtainable then set dimensions to zero
  storeys$A[storeys$A== 888 | storeys$A == 999 | is.na(storeys$A) == "TRUE"] <- 0
  storeys$C[storeys$C == 8.8 | storeys$C == 9.9| is.na(storeys$C) == "TRUE"] <- 0

  #To calculate the perimeter from the area the ratio of length to width must be set
  #This value has been based on common ratios of length to width in houses (i.e the
  #length of a dwelling is 50% greater than the width, this is a commonly used ratio)
  dimension.ratio <- 1.5
  storeys$mainlength <- sqrt(dimension.ratio*storeys$A)
  storeys$mainwidth <- sqrt(storeys$A/dimension.ratio)

  #The dataframe is reorganised to match extension 1 with level 1 and extension 2
  #with level 2 as no data is available as to which floor each extension is on
  floors <- subset(storeys, floor != "N6" & floor != "N7",select=c(uprn_new,floor
                                                                   ,C,mainlength
                                                                   ,mainwidth))
  colnames(floors)[names(floors) == "C"] <- "mainheight"
  floors$floor <- as.factor(checked.revalue(
    as.factor(floors$floor),c("N1" = "1"
                              ,"N2" = "2"
                              ,"N3" = "3"
                              ,"N4" = "4"
                              ,"N5" = "5")))
  extension <- subset(storeys, floor == "N6" | floor == "N7",select=c(uprn_new,floor
```

```
                                                                ,C,mainlength
                                                                ,mainwidth))
extension$floor <- as.factor(checked.revalue(
  as.factor(extension$floor),c("N6" = "1"
                               ,"N7" = "2")))
colnames(extension)[names(extension) == "mainlength"] <- "frontextlength"
colnames(extension)[names(extension) == "mainwidth"] <- "frontextwidth"
colnames(extension)[names(extension) == "C"] <- "frontextheight"
storeys <- join(floors,extension,by=c("uprn_new","floor"))
storeys[is.na(storeys)] <- 0

#Some (~200 cases) have an extension 2 but no level 2, these cases are assumed to
#have a second extension at the rear of the house (as all dwellings this applies
#to are dwellings with an unattached back face)
backextension <- subset(storeys,storeys$frontextheight > 0
                        & storeys$mainheight == 0,
                        select = c(uprn_new, floor, frontextlength, frontextwidth
                                   , frontextheight))
colnames(backextension)[names(backextension) == "frontextlength"] <-
  "backextlength"
colnames(backextension)[names(backextension) == "frontextwidth"] <-
  "backextwidth"
colnames(backextension)[names(backextension) == "frontextheight"] <-
  "backextheight"
backextension$floor <- 1

storeys$frontextlength[storeys$frontextheight > 0 & storeys$mainheight == 0] <- 0
storeys$frontextwidth[storeys$frontextheight > 0 & storeys$mainheight == 0] <- 0
storeys$frontextheight[storeys$frontextheight > 0 & storeys$mainheight == 0] <- 0

storeys <- join(storeys,backextension, by = c("uprn_new","floor"))
storeys[is.na(storeys)] <- 0

#Convert units of dimensions from m into mm (multiply by 100)
storeys$mainwidth <- storeys$mainwidth  * 100
storeys$mainlength <- storeys$mainlength  * 100
storeys$frontextwidth <- storeys$frontextwidth  * 100
storeys$frontextlength <- storeys$frontextlength  * 100
storeys$backextwidth <- storeys$backextwidth  * 100
storeys$backextlength <- storeys$backextlength  * 100

#Polypoints
#polypoints are a series of corresponding x and y coordinates on a cartesian plane
#that detail the layout of each storey, and 'draw' the shape of each storey.
#Dwellings with no externsions have just five sets polypoints as the storey is a
#rectagular shape (points 1 and 5 have the same locations in order to complete the
#shape); dwellings with one extension have seven sets of polypoints and dwellings
#with two extensions on the same storey have nine sets of polypoints.
#(dimensions in mm).
  storeys$polygonxpoints <- paste("{",round(0,0)
                                  ,",",round(0,0)
                                  ,",",round(storeys$backextwidth,0)
                                  ,",",round(storeys$backextwidth,0)
```

```
                                    ,",",round(storeys$mainwidth,0)
                                    ,",",round(storeys$mainwidth,0)
                                    ,",",round(storeys$mainwidth
                                              -storeys$frontextwidth,0)
                                    ,",",round(storeys$mainwidth
                                              -storeys$frontextwidth,0)
                                    ,",",round(0,0)
                                    ,"}",sep="")


  storeys$polygonypoints <- paste("{",round(0,0)
                                    ,",",round(storeys$mainlength
                                              +storeys$backextlength,0)
                                    ,",",round(storeys$mainlength
                                              +storeys$backextlength,0)
                                    ,",",round(storeys$mainlength,0)
                                    ,",",round(storeys$mainlength,0)
                                    ,",",round(-storeys$frontextlength,0)
                                    ,",",round(-storeys$frontextlength,0)
                                    ,",",round(0,0)
                                    ,",",round(0,0)
                                    ,"}",sep="")
  storeys$polypoints <- 9
  storeys$ceilingheight <- storeys$mainheight

  storeys <- subset(storeys,select=c(uprn_new,floor,polygonxpoints
                                     ,polygonypoints,polypoints,ceilingheight))

  return (storeys)
}
```

Assign storeys geometry

Function assigns the storeys geometry created previously, the correct label for the type of storey (i.e. level 1 becomes ground for houses, but may become any of ground, higher or top_floor for flat).

In addition, if column J4 in the scottish survey (rooms in basement) indicates the existence of a basement, one is created by replicating the floor plan of the ground floor. If column J2 (habitable rooms excluding room in roof) indicates that there are storeys higher than the second floor the level 3 and above floor plan is replicated as required.

@param shcs - the scottish survey data

@param storeys - the storey geometry created by the make.polypointsandceilingheights function

```
assigning.polypointsandceilingheights <- function(shcs,storeys){

  #Remove floors that don't physically exist
  floortype <- subset(storeys,ceilingheight != 0.0,select=c(uprn_new
                                                            ,floor,ceilingheight))
  floortype <- spread(floortype,floor,ceilingheight)

  additional.info <- subset(shcs,select = c(uprn_new,J2,J4,E5,E6,C2))
  floortype <- join(floortype,additional.info,by="uprn_new")

  #If level 5 and not flat then room in roof
  floortype$lvl.5[is.na(floortype[6]) == FALSE
                  & floortype$C2 == "Not flat"] <- "room_in_roof"
  #if level 5 and flat then top floor
  floortype$lvl.5[is.na(floortype[6]) == FALSE
                  & floortype$C2 != "Not flat"] <- "top_floor"

  # if level 1 and either not flat or exposed floor then ground
  floortype$lvl.1[is.na(floortype[2]) == FALSE
                  & (floortype$C2 == "Not flat" | (floortype$E5 == "Ground floor"
                                      |floortype$E5 == "Exposed")) ] <- "ground"
  # if level 1 and flat and exposed roof but no room in roof then top floor
  floortype$lvl.1[is.na(floortype[2]) == FALSE & is.na(floortype$lvl.1) == TRUE
                  & floortype$C2 != "Not flat" & floortype$J2 <= 1
                  & floortype$lvl.5 != "top_floor"
                  & (floortype$E6 == "Pitched roof" |
                       floortype$E6 == "Flat roof")] <- "top_floor"
  #if level 1 exists and flats but not yet filled in then first floor
  floortype$lvl.1[is.na(floortype[2]) == FALSE & is.na(floortype$lvl.1) == TRUE
                  & floortype$C2 != "Not flat"] <- "first_floor"

  #if level 2 exists and not a flat then first floor
  floortype$lvl.2[is.na(floortype[3]) == FALSE
                  & floortype$C2 == "Not flat"] <- "first_floor"
  #if level 2 exists, flat and exposed roof and no room in roof then top floor
  floortype$lvl.2[is.na(floortype[3]) == FALSE
                  & floortype$C2 != "Not flat"
                  & (floortype$E6 == "Pitched roof" | floortype$E6 == "Flat roof")
                  & is.na(floortype$lvl.5) == TRUE] <- "top_floor"
  #if level 2 exists, flat and not yet filled in then higher_floor
  floortype$lvl.2[is.na(floortype[3]) == FALSE & is.na(floortype$lvl.2) == TRUE
```

```
                & floortype$C2 != "Not flat"] <- "higher_floor"

  #If level 3 and 4 exist then assign correct floor type
  #No flats in the survey have more than 2 floors
  floortype$lvl.3[is.na(floortype[4]) == FALSE
                & floortype$C2 == "Not flat"] <- "second_floor"
  #3 cases with 4 floors, no cases with more floors
  floortype$lvl.4[is.na(floortype[4]) == FALSE
                & floortype$C2 == "Not flat"] <- "higher_floor"

  #Transform dataframe to correct format, join to the storeys dataframe
  #and remove any floors that don't exist
  floortype <- subset(floortype,select = c(uprn_new,J4,lvl.1,lvl.2,lvl.3
                                           ,lvl.4,lvl.5))
  floortype <- gather(floortype,key,type,-uprn_new, -J4)
  floortype <- separate(floortype,variable,into=c("key","floor"))
  names(floortype) [5] <- "type"

  floortype <- subset(floortype,select = c(uprn_new,floor,type,J4))
  floortype <- join(floortype,storeys,by=c("uprn_new","floor"))
  floortype <- subset(floortype,is.na(floortype$type)==FALSE)

  #If there are rooms in the basement then a basement is created from the
  #ground floor plan
  basements <- subset(floortype,floortype$type == "ground"
                      & floortype$J4 >= 1 & floortype$J4 <88)
  basements$floor <- 0
  basements$type <- "basement"

  #Basements are then joined into the floortype again
  floortype <- rbind(floortype,basements)

  return(floortype)
}
```

Storey height

Make storey height

Storey heights are made by adding 0.25 to the ceiling height, unless the storey is the ground or basement, as the lowest storey height will also be the ceiling height. 0.25m is the RD SAP dimension given for the thickness of internal ceilings/ floors to add to additional storeys (see Appendix S RD SAP 9.91 section S3.5).

@param storeys - the storey geometry dataframe created from the scottish survey data in the previous two functions

```
make.storeyheight <- function(storeys){
  storeys$storyheight <- storeys$ceilingheight
  storeys$storyheight[storeys$type != "ground" & storeys$type != "basement"] <-
    storeys$ceilingheight[storeys$type != "ground" & storeys$type != "basement"] + 0.25
  return(storeys)
}
```

Ventilation

The following section of code is generated in ventilation.R

Make and save ventilation data

Create a .csv file using the function make.ventilation, which creates a dataframe containing a complete set of populated variables for the ventilation.csv stock file.

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the extension .csv

```
save.ventilation <- function(shcs, output) {
  write.csv(make.ventilation(shcs), file=output, row.names=FALSE)
}
```

Make the dataframe that contains all the information required for the ventilation.csv file

The dataframe is constructed and returned (as it is the last thing in the function that is assigned)

@param shcs - the scottish survey data

```
make.ventilation <- function(shcs) {
  #Most columns are not used in the NHM but cannot be left blank. They are created as
  #vectors containing all 0 or "natural" of the same length as the number of cases
  the.chimneysmainheating <- rep(0, length(shcs$uprn_new))
  the.chimneysother <- rep(0, length(shcs$uprn_new))
  the.chimneyssecondaryheating <- rep(0, length(shcs$uprn_new))
  the.intermittentfans <- rep(0, length(shcs$uprn_new))
  the.passivevents <- rep(0, length(shcs$uprn_new))
  the.ventilationsystem <- rep("natural", length(shcs$uprn_new))
  #Column is used in the NHM, however as no information is available it is
  #based on the extent of double glazing -  this is consistant with previous stock
  #creation and the option to apply as described below.
  the.windowsanddoorsdraughtstrippedproportion <-
    windows.anddoorsdraughtstrippedproportion(shcs$Q47)

  data.frame(aacode = shcs$uprn_new
             ,chimneysmainheating = the.chimneysmainheating
             ,chimneysother = the.chimneysother
             ,chimneyssecondaryheating = the.chimneyssecondaryheating
             ,intermittentfans = the.intermittentfans
             ,passivevents = the.passivevents
             ,ventilationsystem = the.ventilationsystem
             ,windowsanddoorsdraughtstrippedproportion =
               the.windowsanddoorsdraughtstrippedproportion
  )
}
```

Draught stripped proportion

Infer the proportion of window and door draught stripped from the proportion of double glazing

There is no information available in the scottish survey and this value is inferred from the proportion of double glazing as described in the RD SAP methodlogy. See S8.1:

```
#    if the state of the draught proofing cannot be determined then take triple,
#    double or secondary glazed as being draught proofed, and single glazed windows
#    and doors as not draught stripped
```

@param draughtproof - the percentage of double glazing from the scottish survey

```
windows.anddoorsdraughtstrippedproportion<-function(draughtproof){
  draughtproof <- draughtproof/10
  return(draughtproof)
}
```

Elevations

The following section of code is generated in elevations.R

Make and save elevations data

Create a .csv file using the function make.elevations, which creates a dataframe containing a complete set of populated variables for the elevations.csv stock file.

The elevations file contains four rows for each dwelling case, one for each elevation of a dwelling (front, back, left and right). The information in the survey is used to determine which of these elevations are attached to party walls, the construction types of the unattached walls and levels of insulation on each elevation. Also included in data on types of window (glazing and frame type) and what proportion of each elevation is windows, and how many and the type of doors present in each elevation.

Each elevation can have only one wall type and can only have full or no insulation but multiple types of insulation are allowed (i.e. an external wall on an elevation could have both external and internal wall insulation present).

Attachments and proportion of windows are expressed in tenths.

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the extension .csv

```
save.elevations <- function(shcs, output) {
  write.csv(make.elevations(shcs), file=output, row.names=FALSE,na = "")
}
```

Make the dataframe that contains all the information required for the elevations.csv file

The dataframe is constructed and returned (as it is the last thing in the function that is assigned)

@param shcs - the scottish survey data

```
make.elevations <- function(shcs) {
  #The tenths attached for each wall (front,left,back,right) are created.
  elevations <- make.tenthsattached(shcs)

  #The construction type of the primary and secondary walls are created, followed by
  #creation of the existance of wall insulation (cavity,internal,external).
  #These wall types and insulations are then assigned to each of the elevations based
  #on the physical presence of an external wall calculated from the tenthsattached
  #columns.
  elevations <- make.externalwallconstructiontype(elevations)
  elevations <- make.wallinsulation(elevations)
  elevations <- assigning.walltypeandinsulation(elevations)

  #The tenths openings are calculated in the make.tenthsopening function
  elevations <- make.tenthsopening(elevations)

  #The type of window frame for each case are made and these are then assigned to
  #each elevation. The number of each type of doorframe on each elevation are then
  #assigned.
  elevations <- make.windowframes(elevations)
  elevations <- assigning.windowframes(elevations)
```

```
  elevations <- assigning.doorframes(elevations)

  #The dataframe created by the functions above is then changed into the correct
  #format. The relevent columns are selected and then turned into a long dataframe
  #The key is then split into a description of the value in that row and the
  #elevation to which the value belongs. The dataframe is then changed from this long
  #format into a wide format.
  elevations <- select(elevations,uprn_new,contains("front."),contains("left."),
                       contains("back."),contains("right."))
  elevations <- gather(elevations,key,value,-uprn_new)
  elevations <- separate(elevations,variable,into=c("elevationtype","key"),sep="\\.")
  elevations <- spread(elevations,key,value)

  #creates a dataframe containing column names (word before the = sign) and a vector
  #to fill that column (vector after the = sign)
  #Due to the characters (: and ,) that appear in the column names for doorframes the
  #command check.names = FALSE is included to prevent the format of these names being
  #altered
  elevations <- data.frame(aacode = elevations$uprn_new
                  ,cavitywallinsulation = elevations$cavitywallinsulation
                  ,`doorframe:wood,doortype:solid` =
                    elevations$`doorframe:wood,doortype:solid`
                  ,`doorframe:wood,doortype:glazed` =
                    elevations$`doorframe:wood,doortype:glazed`
                  ,`doorframe:metal,doortype:solid` =
                    elevations$`doorframe:metal,doortype:solid`
                  ,`doorframe:metal,doortype:glazed` =
                    elevations$`doorframe:metal,doortype:glazed`
                  ,`doorframe:upvc,doortype:solid` =
                    elevations$`doorframe:upvc,doortype:solid`
                  ,`doorframe:upvc,doortype:glazed` =
                    elevations$`doorframe:upvc,doortype:glazed`
                  ,doubleglazedwindowframe = elevations$doubleglazedwindowframe
                  ,elevationtype = elevations$elevationtype
                  ,externalwallinsulation = elevations$externalwallinsulation
                  ,externalwallconstructiontype = elevations$externalwallconstructiontype
                  ,internalinsulation = elevations$internalinsulation
                  ,percentagedoubleglazed = elevations$percentagedoubleglazed
                  ,singleglazedwindowframe = elevations$singleglazedwindowframe
                  ,tenthsattached = elevations$tenthsattached
                  ,tenthsopening = elevations$tenthsopening
                  ,tenthspartywall = elevations$tenthsattached
                  ,check.names = FALSE
                  )
  #correct insulation values where no external wall exists
#   elevations$cavitywallinsulation <- ifelse(elevations$tenthsattached == 10 &
#                                         is.na(elevations$cavitywallinsulation), "",
#                                         elevations$cavitywallinsulation)
#   elevations$externalwallinsulation <- ifelse(elevations$tenthsattached == 10 &
#                                         is.na(elevations$externalwallinsulation), "",
#                                         elevations$externalwallinsulation)
#   elevations$internalinsulation <- ifelse(elevations$tenthsattached == 10 &
#                                         is.na(elevations$internalinsulation), "",
```

```
#                                           elevations$internalinsulation)
  return(elevations)
}
```

Tenths attached

Create the tenths attached for all elevations

There is limited information available in the scottish survey to create this. Therefore it is created based on assumptions about the house from it's house type and for flats from the variable E7 which contains the number of walls exposed It is assumed that the front of the house or flat is always exposed and then for flats the walls are assigned clockwise (i.e. front,left,back,right). A flat with three exposed walls would have the front, left and back walls exposed and the right wall would be attached.

Mid-terraced dwellings have the front and back exposed by the left and right face attached, semi-detached and end-terrace houses have only the right face attached and detached houses have no faces attached

@param stock - this is the whole of the scottish survey

```
make.tenthsattached <- function(stock){
  #combine housetype information with information about wall exposure in flats using
  #the building type function
  stock$buildinginfo <- building.type(stock$C1,stock$E7)

  stock$front.tenthsattached <- as.factor(checked.revalue(
    stock$buildinginfo,c(
      "Detached" = 0
      ,"End terrace" = 0
      ,"Semi-detached" = 0
      ,"Mid-terrace with passage" = 0
      ,"Mid-terrace" = 0
      ,"Corner / enclosed end" = 0
      ,"Enclosed mid" = 0
      ,"1 wall exposed" = 0
      ,"1 to 2 walls exposed" = 0
      ,"2 walls exposed" = 0
      ,"2 to 3 walls exposed" = 0
      ,"3 walls exposed" = 0
      ,"3 to 4 walls exposed" = 0
      ,"4 walls exposed" = 0
    )))

  stock$left.tenthsattached <- as.factor(checked.revalue(
    stock$buildinginfo,c(
      "Detached" = 0
      ,"End terrace" = 0
      ,"Semi-detached" = 0
      ,"Mid-terrace with passage" = 0
      ,"Mid-terrace" = 10
      ,"Corner / enclosed end" = 0
      ,"Enclosed mid" = 10
      ,"1 wall exposed" = 10
      ,"1 to 2 walls exposed" = 10
      ,"2 walls exposed" = 10
      ,"2 to 3 walls exposed" = 5
      ,"3 walls exposed" = 0
      ,"3 to 4 walls exposed" = 0
      ,"4 walls exposed" = 0)))
```

```
  stock$back.tenthsattached <- as.factor(checked.revalue(
    stock$buildinginfo,c(
      "Detached" = 0
      ,"End terrace" = 0
      ,"Semi-detached" = 0
      ,"Mid-terrace with passage" = 0
      ,"Mid-terrace" = 0
      ,"Corner / enclosed end" = 10
      ,"Enclosed mid" = 10
      ,"1 wall exposed" = 10
      ,"1 to 2 walls exposed" = 5
      ,"2 walls exposed" = 0
      ,"2 to 3 walls exposed" = 0
      ,"3 walls exposed" = 0
      ,"3 to 4 walls exposed" = 0
      ,"4 walls exposed" = 0)))

  stock$right.tenthsattached <- as.factor(checked.revalue(
    stock$buildinginfo,c(
      "Detached" = 0
      ,"End terrace" = 10
      ,"Semi-detached" = 10
      ,"Mid-terrace with passage" = 10
      ,"Mid-terrace" = 10
      ,"Corner / enclosed end" = 10
      ,"Enclosed mid" = 10
      ,"1 wall exposed" = 10
      ,"1 to 2 walls exposed" = 10
      ,"2 walls exposed" = 10
      ,"2 to 3 walls exposed" = 10
      ,"3 walls exposed" = 10
      ,"3 to 4 walls exposed" = 5
      ,"4 walls exposed" = 0
    )))
  return(stock)
}
```

Wall construction and insulation

Create the primary and secondary wall types and calculate the number walls that should be made out of the primary and secondary wall types

The wall type information is stored across several values of the survey data, this is combined where necessary to create a vector which is then mapped to possible wall types for the NHM.

The stock data details two wall types: primary and secondary as well as providing information on the coverage (in tenths) of the primary wall. Where a secondary wall type exists, the function below determines the proportion of exposed walls to allocate to the details (construction and insulation) described in the primary wall variables. The secondary wall details are then used to assign information to the remaining faces of each case/dwelling.

Any cavity construction type that is built with blockwork or brick is allocated as a 'cavity' wall, which in the NHM covers all masonry cavities. Other cavity walls constructed of different materials are assigned accordingly and as shown below.

So to calculate the number of primary walls that should exist, the number of unattached walls is combined with the proportion of walls that are primary walls.

@param stock - this is the whole of elevations created by other functions in this script, which is comprised of the scottish survey with additional columns

```
make.externalwallconstructiontype <- function(stock){
  #Combine secondary wall types recorded across several columns into one column -
  #Q2 is required for wall types of brick or blockwork, Q5 is required for stone
  #walls
  stock$primarywall <- ifelse(stock$Q3 != "Brick" & stock$Q3 != "Blockwork"
                              ,levels(stock$Q3)[stock$Q3]
                              ,paste(stock$Q3,stock$Q2,sep="."))
  stock$primarywall <- as.factor(ifelse(stock$primarywall != "Stone"
                                        ,stock$primarywall
                                        ,paste(stock$primarywall,stock$Q5,sep=".")))
  stock$primarywall <- as.factor(checked.revalue(
    stock$primarywall,c(
      "Blockwork.Cavity" = "cavity"
      ,"Blockwork.Other" = "solidbrick"
      ,"Blockwork.Solid" = "solidbrick"
      ,"Brick.Cavity" = "cavity"
      ,"Brick.Other" = "solidbrick"
      ,"Brick.Solid" = "solidbrick"
      ,"Clay / earth" = "cob"
      ,"Concrete" = "systembuild"
      ,"Metal" = "metalframe"
      ,"Other" = "systembuild"
      ,"Stone.Granite" = "graniteorwhinstone"
      ,"Stone.Not stone" = "graniteorwhinstone"
      ,"Stone.Other" = "graniteorwhinstone"
      ,"Stone.Sandstone" = "sandstone"
      ,"Stone.Whin" = "graniteorwhinstone"
      ,"Timber" = "timberframe"
    )))


  #Combine secondary wall types recorded across several columns into one column -
```

```
  #Q11 is required for wall types of brick or blockwork, Q14 is required for stone
  #walls
  stock$secondarywall <- ifelse(stock$Q12 != "Brick" & stock$Q12 != "Blockwork"
                                ,levels(stock$Q12)[stock$Q12]
                                ,paste(stock$Q12,stock$Q11,sep="."))
  stock$secondarywall <- as.factor(ifelse(stock$secondarywall != "Stone"
                                          ,stock$secondarywall
                                          ,paste(stock$secondarywall,stock$Q14
                                                 ,sep=".")))
  #One case has an unobtainable secondary wall type, this has been set to sandstone
  #this is because it is the most common construction type and material in the survey
  stock$secondarywall <- as.factor(checked.revalue(
    stock$secondarywall,c(
      "Blockwork.Cavity" = "cavity"
      ,"Blockwork.Other" = "solidbrick"
      ,"Blockwork.Solid" = "solidbrick"
      ,"Brick.Cavity" = "cavity"
      ,"Brick.Other" = "solidbrick"
      ,"Brick.Solid" = "solidbrick"
      ,"Brick.Unobtainable" = "solidbrick"
      ,"Clay / earth" = "cob"
      ,"Concrete" = "systembuild"
      ,"Metal" = "metalframe"
      ,"Other" = "systembuild"
      ,"Stone.Granite" = "graniteorwhinstone"
      ,"Stone.Not stone" = "graniteorwhinstone"
      ,"Stone.Other" = "graniteorwhinstone"
      ,"Stone.Sandstone" = "sandstone"
      ,"Stone.Unobtainable" = "graniteorwhinstone"
      ,"Stone.Whin" = "graniteorwhinstone"
      ,"Timber" = "timberframe"
      ,"Unobtainable" = "sandstone"
    )))

  # This final part of the function calculates the number of elevations of the house
  # which are of primary wall type, rounded to the nearest whole number and assigns
  # it to 'numprimarywalls' (Note: each elevation can only be assigned one wall type,
  # hence the majhority of the front elevations are assigned the primary wall details
  stock$numprimarywalls <-
    round2(((40-
               (as.numeric(levels(stock$front.tenthsattached)[stock$front.tenthsattached])
                +as.numeric(levels(stock$left.tenthsattached)[stock$left.tenthsattached])
                +as.numeric(levels(stock$back.tenthsattached)[stock$back.tenthsattached])
                +as.numeric(levels(stock$right.tenthsattached)[stock$right.tenthsattached])))
             *stock$Q10)/100,0)

  return(stock)
}
```

Create the existance of cavity wall insulation, internal insulation and external insulation

This is created for both the primary and secondary walls and some logic is included to read just the survey data where inconsistent (i.e a solid wall with cavity wall insulation)

@param stock - this is the whole of elevations created by other functions in this script, which is comprised of the scottish survey with additional columns

```
make.wallinsulation <- function(stock){
  stock$primary.cavityinsulation <- as.factor(checked.revalue(
    stock$Q8,c(
      "None" = "FALSE"
      ,"cavity" = "TRUE"
      ,"Internal"  = "FALSE"
      ,"External" = "FALSE"
      ,"cavity and internal" = "TRUE"
      ,"cavity and external" = "TRUE"
      ,"internal and external" = "FALSE"
      ,"Not applicable" = "FALSE"
      ,"Unobtainable"  = "FALSE"
    )))
  #Only cavity,timberframe,metalframe and systembuild wall types can have cavity wall
  #insulation, so if PRIMARY WALL cases exist where other wall types have cavity wall
  #insulation, cavity wall insualtion is set to FALSE
  stock$primary.cavityinsulation[stock$primary.cavityinsulation == "TRUE" &
                                   stock$primarywall != "cavity" &
                                   stock$primarywall != "timberframe"
                                 & stock$primarywall != "metalframe" &
                                   stock$primarywall != "systembuild" ] <- "FALSE"
  stock$secondary.cavityinsulation <- as.factor(checked.revalue(
    stock$Q17,c(
      "None" = "FALSE"
      ,"cavity" = "TRUE"
      ,"Internal"  = "FALSE"
      ,"External" = "FALSE"
      ,"cavity and internal" = "TRUE"
      ,"cavity and external" = "TRUE"
      ,"internal and external" = "FALSE"
      ,"Not applicable" = "FALSE"
      ,"Unobtainable"  = "FALSE"
    )))
  #Only cavity,timberframe,metalframe and systembuild wall types can have cavity wall
  #insulation, so if SECONDARY WALL cases exist where other wall types have cavity
  #wall insulation, cavity wall insualtion is set to FALSE
  stock$secondary.cavityinsulation[stock$secondary.cavityinsulation == "TRUE" &
                                     stock$secondarywall != "cavity" &
                                     stock$secondarywall != "timberframe"
                                   & stock$secondarywall != "metalframe" &
                                     stock$secondarywall != "systembuild" ] <- "FALSE"
  stock$primary.externalinsulation <- as.factor(checked.revalue(
    stock$Q8,c(
      "None" = "FALSE"
      ,"cavity" = "FALSE"
      ,"Internal"  = "FALSE"
      ,"External" = "TRUE"
```

```
      ,"cavity and internal" = "FALSE"
      ,"cavity and external" = "TRUE"
      ,"internal and external" = "TRUE"
      ,"Not applicable" = "FALSE"
      ,"Unobtainable"  = "FALSE"
    )))
  stock$secondary.externalinsulation <- as.factor(checked.revalue(
    stock$Q17,c(
      "None" = "FALSE"
      ,"cavity" = "FALSE"
      ,"Internal"  = "FALSE"
      ,"External" = "TRUE"
      ,"cavity and internal" = "FALSE"
      ,"cavity and external" = "TRUE"
      ,"internal and external" = "TRUE"
      ,"Not applicable" = "FALSE"
      ,"Unobtainable"  = "FALSE"
    )))
  stock$primary.internalinsulation <- as.factor(checked.revalue(
    stock$Q8,c(
      "None" = "FALSE"
      ,"cavity" = "FALSE"
      ,"Internal"  = "TRUE"
      ,"External" = "FALSE"
      ,"cavity and internal" = "TRUE"
      ,"cavity and external" = "FALSE"
      ,"internal and external" = "TRUE"
      ,"Not applicable" = "FALSE"
      ,"Unobtainable"  = "FALSE"
    )))
  stock$secondary.internalinsulation <- as.factor(checked.revalue(
    stock$Q17,c(
      "None" = "FALSE"
      ,"cavity" = "FALSE"
      ,"Internal"  = "TRUE"
      ,"External" = "FALSE"
      ,"cavity and internal" = "TRUE"
      ,"cavity and external" = "FALSE"
      ,"internal and external" = "TRUE"
      ,"Not applicable" = "FALSE"
      ,"Unobtainable"  = "FALSE"
    )))
  return(stock)
}
```

Each elevation is assigned the correct wall type and insulation based on the tenths attached and the number of walls that are made from the primary wall type

Walls are assigned in the following order: front, left, back, right

@param stock - this is the whole of elevations created by other functions in this script, which is comprised of the scottish survey with additional columns

```
assigning.walltypeandinsulation <- function(stock){

  #columns are turned into correct format for next section
  stock$usedwalls <- 0
  stock$front.tenthsattached <-
    as.numeric(levels(stock$front.tenthsattached)[stock$front.tenthsattached])
  stock$left.tenthsattached <-
    as.numeric(levels(stock$left.tenthsattached)[stock$left.tenthsattached])
  stock$back.tenthsattached <-
    as.numeric(levels(stock$back.tenthsattached)[stock$back.tenthsattached])
  stock$right.tenthsattached <-
    as.numeric(levels(stock$right.tenthsattached)[stock$right.tenthsattached])
  stock$primarywall <-
    as.character(levels(stock$primarywall)[stock$primarywall])
  stock$secondarywall <-
    as.character(levels(stock$secondarywall)[stock$secondarywall])

  #Assign wall types and insulation to each elevation: if wall is an external wall
  #indicated by tenths attached being less than 10 and the number of walls that have
  #already been assigned as the primary wall type is less than the total number of
  #primary walls then primary wall type is assigned, otherwise secondary wall type
  #is assigned.
  #if wall is not an external wall then no wall type is assigned.
  #Order of wall assignment: front, left, back, right
  #Walls are assigned using the wall.typeandinsulation function
  stock <- wall.typeandinsulation(stock,"front")
  stock <- wall.typeandinsulation(stock,"left")
  stock <- wall.typeandinsulation(stock,"back")
  stock <- wall.typeandinsulation(stock,"right")
  return(stock)
}
```

Each elevation is assigned the correct wall type and insulation based on the tenths attached and the number of walls that are made from the primary wall type

A test is carried out to see if the wall exists and if all the primary walls have been used up, this is used to assign the wall type, cavitywall, internal and external insulation details

@param stock - this is the whole of elevations created by other functions in this script, which is comprised of the scottish survey with additional columns

@param elevation - a string which contains the type of elevation, i.e. “front”

```
wall.typeandinsulation <- function(stock,elevation){
  #test to indicate if the primary or secondary wall type should be assigned.

  tenths.attached.column <- paste(elevation,"tenthsattached",sep=".")
  has.exposed.wall.in.elevation <-  stock[,tenths.attached.column] < 10

  prim.test <- has.exposed.wall.in.elevation & stock$usedwalls < stock$numprimarywalls
  sec.test  <- has.exposed.wall.in.elevation & stock$usedwalls >= stock$numprimarywalls

  stock[prim.test, paste(elevation,"externalwallconstructiontype",sep=".")] <-
    stock$primarywall[prim.test]
  stock[sec.test,paste(elevation,"externalwallconstructiontype",sep=".")] <-
    stock$secondarywall[sec.test]

  stock[prim.test,paste(elevation,"cavitywallinsulation",sep=".")] <-
    stock$primary.cavityinsulation[prim.test]
  stock[sec.test,paste(elevation,"cavitywallinsulation",sep=".")] <-
    stock$secondary.cavityinsulation[sec.test]

  stock[prim.test,paste(elevation,"externalwallinsulation",sep=".")] <-
    stock$primary.externalinsulation[prim.test]
  stock[sec.test,paste(elevation,"externalwallinsulation",sep=".")] <-
    stock$secondary.externalinsulation[sec.test]

  stock[prim.test,paste(elevation,"internalinsulation",sep=".")] <-
    stock$primary.internalinsulation[prim.test]
  stock[sec.test,paste(elevation,"internalinsulation",sep=".")] <-
    stock$secondary.internalinsulation[sec.test]
  #The number of primary walls used is increased by one each time the function is
  # called if the prim.test is true
  stock$usedwalls[prim.test] <- (stock$usedwalls[prim.test] + 1)
  return(stock)
}
```

Tenths opening

Create the tenths opening for all elevations

There is limited information available in the scottish survey to create this. It has been created by dividing the window area as calculated using the calculations detailed in the RD-SAP methodology (SAP 9.91 Appendix S Table S4: Windowbarea (m?)) by the total wall area calculated by multipling the external wall perimeter by the height for all storeys.

@param stock - this is the whole of elevations created by other functions in this script, which is comprised of the scottish survey with additional columns

```
make.tenthsopening <- function(stock){
  #Set all floors sizes that do not exist to a floor size of 0 for summation
  #These should not be altered
  stock$N1_A[stock$N1_A == 888] <- 0
  stock$N2_A[stock$N2_A == 888] <- 0
  stock$N2_A[stock$N2_A == 999] <- 0
  stock$N3_A[stock$N3_A == 888] <- 0
  stock$N4_A[stock$N4_A == 888] <- 0
  stock$N5_A[stock$N5_A == 888] <- 0
  stock$N6_A[stock$N6_A == 888] <- 0
  stock$N7_A[stock$N7_A == 888] <- 0

  #Set all heights that do not exist to a heights of 0 for summation
  #These should not be altered

  stock$N1_C[stock$N1_C == 8.8 | is.na(stock$N1_C) == "TRUE"] <- 0
  stock$N2_C[stock$N2_C == 8.8 | stock$N2_C == 9.9 |is.na(stock$N2_C) == "TRUE"] <- 0
  stock$N3_C[stock$N3_C == 8.8 | stock$N3_C == 9.9 |is.na(stock$N3_C) == "TRUE"] <- 0
  stock$N4_C[stock$N4_C == 8.8 | stock$N4_C == 9.9 |is.na(stock$N4_C) == "TRUE"] <- 0
  stock$N5_C[stock$N5_C == 8.8 | stock$N5_C == 9.9 |is.na(stock$N5_C) == "TRUE"] <- 0
  stock$N6_C[stock$N6_C == 8.8 | stock$N6_C == 9.9 |is.na(stock$N6_C) == "TRUE"] <- 0
  stock$N7_C[stock$N7_C == 8.8 | stock$N7_C == 9.9 |is.na(stock$N7_C) == "TRUE"] <- 0

  #Set all external perimeters that do not exist external perimeters of 0 for
  #summation
  #These should not be altered
  stock$N1_D[stock$N1_D == 888 | is.na(stock$N1_D) == "TRUE"] <- 0
  stock$N2_D[stock$N2_D == 888 | stock$N2_D == 999 |is.na(stock$N2_D) == "TRUE"] <- 0
  stock$N3_D[stock$N3_D == 888 | stock$N3_D == 999 |is.na(stock$N3_D) == "TRUE"] <- 0
  stock$N4_D[stock$N4_D == 888 | stock$N4_D == 999 |is.na(stock$N4_D) == "TRUE"] <- 0
  stock$N6_D[stock$N6_D == 888 | stock$N6_D == 999 |is.na(stock$N6_D) == "TRUE"] <- 0
  stock$N7_D[stock$N7_D == 888 | stock$N7_D == 999 |is.na(stock$N7_D) == "TRUE"] <- 0

  #Calculate the number of floors that the dimensions from the level3 and higher
  #columns need to be replicated for.
  #This is set so that if the number of floors (excluding room in roof) is 3, then
  #the number of times that the level 3 and higher columns need to be replicated is
  #once.
  #If the number of times that the level 3 and higher columns need to be replicated
  #is less than zero (i.e for a 1 story house) then set this flag to 0.
  stock$levelthreefloors <- stock$J2 - 2
  stock$levelthreefloors[stock$levelthreefloors < 0] <- 0
```

```
#Calculate total floor area using the floor area for each level, the room in roof
#and extension 1 and extension 2. Level 3 floor areas are multiplied by the number
#of times that level 3 and above information needs to be replicated
stock$totalfloorarea <- stock$N1_A+stock$N2_A+
  stock$N3_A+(stock$N4_A*stock$levelthreefloors) +
  stock$N5_A+stock$N6_A+stock$N7_A


#The next section calculates the window area for each case based on the
#calculations detailed in SAP 9.91 Appendix S Table S4: Windowbarea (m?) - there
#are a number of calculations based on the building type (flat or house), the age
#of the property, and the total floor area.

#Test if case is flat - TRUE, if case is not flat (i.e. house) - FALSE
flat.test <- levels(stock$C1)[stock$C1] =="Not house"
#Window areas of houses
stock$windowarea[!flat.test & (levels(stock$M1)[stock$M1]=="Pre 1919" |
                                 levels(stock$M1)[stock$M1]=="1919 - 1929" |
                                 levels(stock$M1)[stock$M1]=="1930 - 1949")] <-
  0.1220*stock$totalfloorarea[!flat.test &
                                (levels(stock$M1)[stock$M1]=="Pre 1919" |
                                 levels(stock$M1)[stock$M1]=="1919 - 1929" |
                                 levels(stock$M1)[stock$M1]=="1930 - 1949")]+6.875
stock$windowarea[!flat.test & levels(stock$M1)[stock$M1]=="1950 - 1964"] <-
  0.1294*stock$totalfloorarea[!flat.test &
                                levels(stock$M1)[stock$M1]=="1950 - 1964"]+5.515
stock$windowarea[!flat.test & levels(stock$M1)[stock$M1]=="1965 - 1975"] <-
  0.1239*stock$totalfloorarea[!flat.test &
                                levels(stock$M1)[stock$M1]=="1965 - 1975"]+7.332
stock$windowarea[!flat.test & levels(stock$M1)[stock$M1]=="1976 - 1983"] <-
  0.1252*stock$totalfloorarea[!flat.test &
                                levels(stock$M1)[stock$M1]=="1976 - 1983"]+5.520
stock$windowarea[!flat.test & levels(stock$M1)[stock$M1]=="1984 - 1991"] <-
  0.1356*stock$totalfloorarea[!flat.test &
                                levels(stock$M1)[stock$M1]=="1984 - 1991"]+5.242
stock$windowarea[!flat.test & levels(stock$M1)[stock$M1]=="1992 - 1996"] <-
  0.0948*stock$totalfloorarea[!flat.test &
                                levels(stock$M1)[stock$M1]=="1992 - 1996"]+6.534
stock$windowarea[!flat.test & levels(stock$M1)[stock$M1]=="1999 - 2002"] <-
  0.1382*stock$totalfloorarea[!flat.test &
                                levels(stock$M1)[stock$M1]=="1999 - 2002"]-0.027
stock$windowarea[!flat.test & (levels(stock$M1)[stock$M1]=="2003 - 2007" |
                                 levels(stock$M1)[stock$M1]=="2008 onwards")] <-
  0.1435*stock$totalfloorarea[!flat.test &
                                (levels(stock$M1)[stock$M1]=="2003 - 2007" |
                                levels(stock$M1)[stock$M1]=="2008 onwards")]-0.403
#Window areas of flats
stock$windowarea[flat.test & (levels(stock$M1)[stock$M1]=="Pre 1919" |
                                levels(stock$M1)[stock$M1]=="1919 - 1929" |
                                levels(stock$M1)[stock$M1]=="1930 - 1949")] <-
  0.0801*stock$totalfloorarea[flat.test
                              & (levels(stock$M1)[stock$M1]=="Pre 1919" |
                                 levels(stock$M1)[stock$M1]=="1919 - 1929" |
```

```
                                levels(stock$M1)[stock$M1]=="1930 - 1949")]+5.580
stock$windowarea[flat.test & levels(stock$M1)[stock$M1]=="1950 - 1964"] <-
  0.0341*stock$totalfloorarea[flat.test
                              & levels(stock$M1)[stock$M1]=="1950 - 1964"]+8.562
stock$windowarea[flat.test & levels(stock$M1)[stock$M1]=="1965 - 1975"] <-
  0.0717*stock$totalfloorarea[flat.test
                              & levels(stock$M1)[stock$M1]=="1965 - 1975"]+6.560
stock$windowarea[flat.test & levels(stock$M1)[stock$M1]=="1976 - 1983"] <-
  0.1199*stock$totalfloorarea[flat.test
                              & levels(stock$M1)[stock$M1]=="1976 - 1983"]+1.975
stock$windowarea[flat.test & levels(stock$M1)[stock$M1]=="1984 - 1991"] <-
  0.0510*stock$totalfloorarea[flat.test
                              & levels(stock$M1)[stock$M1]=="1984 - 1991"]+4.554
stock$windowarea[flat.test & levels(stock$M1)[stock$M1]=="1992 - 1996"] <-
  0.0813*stock$totalfloorarea[flat.test
                              & levels(stock$M1)[stock$M1]=="1992 - 1996"]+3.744
stock$windowarea[flat.test & levels(stock$M1)[stock$M1]=="1999 - 2002"] <-
  0.1148*stock$totalfloorarea[flat.test
                              & levels(stock$M1)[stock$M1]=="1999 - 2002"]+0.392
stock$windowarea[flat.test & (levels(stock$M1)[stock$M1]=="2003 - 2007" |
                                levels(stock$M1)[stock$M1]=="2008 onwards")] <-
  0.1148*stock$totalfloorarea[flat.test
                              & (levels(stock$M1)[stock$M1]=="2003 - 2007" |
                                levels(stock$M1)[stock$M1]=="2008 onwards")]+0.392

#Calculate the total wall area using all floors apart from the room in roof
stock$wallarea <- (stock$N1_C*stock$N1_D) +
  (stock$N2_C*stock$N2_D) +
  (stock$N3_C*stock$N3_D) +
  (stock$N4_C*stock$N4_D*stock$levelthreefloors) +
  (stock$N6_C*stock$N6_D) +
  (stock$N7_C*stock$N7_D)
#Calculate the total wall area for the few cases which are all flats where the
#only level is a room in roof
stock$wallarea[stock$wallarea == 0 & stock$E7 == "1 wall exposed"] <-
  stock$N5_C[stock$wallarea == 0 & stock$E7 == "1 wall exposed"] *
  (sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "1 wall exposed"]/1.5))

stock$wallarea[stock$wallarea == 0 & stock$E7 == "1 to 2 walls exposed"] <-
  stock$N5_C[stock$wallarea == 0 & stock$E7 == "1 to 2 walls exposed"] *
  (((sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "1 to 2 walls exposed"]
          *1.5))*0.5)+
  (sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "1 to 2 walls exposed"]/1.5)))

stock$wallarea[stock$wallarea == 0 & stock$E7 == "2 walls exposed"] <-
  stock$N5_C[stock$wallarea == 0 & stock$E7 == "2 walls exposed"] *
  (sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "2 walls exposed"]*1.5)+
     sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "2 walls exposed"]/1.5))

stock$wallarea[stock$wallarea == 0 & stock$E7 == "2 to 3 walls exposed"] <-
  stock$N5_C[stock$wallarea == 0 & stock$E7 == "2 to 3 walls exposed"] *
  (sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "2 to 3 walls exposed"]*1.5)+
  ((sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "2 to 3 walls exposed"]/1.5))
```

```
    *1.5))

  stock$wallarea[stock$wallarea == 0 & stock$E7 == "3 walls exposed"] <-
    stock$N5_C[stock$wallarea == 0 & stock$E7 == "3 walls exposed"] *
    (sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "3 walls exposed"]*1.5)+
    ((sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "3 walls exposed"]/1.5))*2))

  stock$wallarea[stock$wallarea == 0 & stock$E7 == "3 to 4 walls exposed"] <-
    stock$N5_C[stock$wallarea == 0 & stock$E7 == "3 to 4 walls exposed"] *
    (((sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "3 to 4 walls exposed"]*1.5))
      *1.5)+
    ((sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "3 to 4 walls exposed"]/1.5))
     *2))

  stock$wallarea[stock$wallarea == 0 & stock$E7 == "4 walls exposed"] <-
    stock$N5_C[stock$wallarea == 0 & stock$E7 == "4 walls exposed"] *
    (((sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "4 walls exposed"]*1.5))*2)+
       ((sqrt(stock$N5_A[stock$wallarea == 0 & stock$E7 == "4 walls exposed"]/1.5))
        *2))

  #Calculate the total openings of each case by dividing the window area by the wall
  #area, multiplying by 10 then rounding to turn into tenths as required by the NHM.
  #Cases with areas less than 1 are set to 1 tenth
  #Cases with areas higher than 7 (2 cases with greater than 10 tenths) are set to
  #7 tenths
  stock$totalopenings <- signif((stock$windowarea/stock$wallarea)*10,1)
  stock$totalopenings[stock$totalopenings < 1] <- 1
  stock$totalopenings[stock$totalopenings > 7] <- 7

  #These tenths are then assigned to each elevation using a calculation based on the
  #amount of tenths unattached (i.e external wall available for windows)
  stock$front.tenthsopening <-
    round((stock$totalopenings*(10-stock$front.tenthsattached)/10),0)
  stock$left.tenthsopening <-
    round((stock$totalopenings*(10-stock$left.tenthsattached)/10),0)
  stock$back.tenthsopening <-
    round((stock$totalopenings*(10-stock$back.tenthsattached)/10),0)
  stock$right.tenthsopening <-
    round((stock$totalopenings*(10-stock$right.tenthsattached)/10),0)
  return(stock)
}
```

Windows

Create the types of double glazing and single glazing window frame and the percentage of double glazing

The single glazing window frame is set to the same type as the type of double glazing as no information about single glazing window types is available in the SHCS stock variables

@param stock - this is the whole of elevations created by other functions in this script, which is comprised of the scottish survey with additional columns

```
make.windowframes <- function(stock){
  stock$doubleglazed <- as.factor(checked.revalue(
    stock$Q45,c(
      "Wood" = "wood"
      ,"Metal (therm break)" = "metal"
      ,"Metal (no therm break)" = "metal"
      ,"UPVC" = "upvc"
      ,"Not applicable" = "NULL"
      ,"Unobtainable" = "NULL"
    )))
  #Double glazing percentage is turned from tenths (as presented in the SHCS) into a
  #percentage
  stock$percentagedoubleglazing <- stock$Q47*10

  stock$singleglazed <- as.factor(checked.revalue(
    stock$Q45,c(
      "Wood" = "wood"
      ,"Metal (therm break)" = "metal"
      ,"Metal (no therm break)" = "metal"
      ,"UPVC" = "upvc"
      ,"Not applicable" = "NULL"
      ,"Unobtainable" = "NULL"
    )))

  #if the double glazing area is 100% then single glazing percentage is set to NULL
  stock$singleglazed[stock$percentagedoubleglazing == 100] <- "NULL"

  return(stock)
}
```

Each elevation is assigned window frame type and the percentage of double glazing No information is available about the window area on different faces of the dwelling so each elevation has the same double glazing percentage as the overall dwellings

Calls the window.frames function

@param stock - this is the whole of elevations created by other functions in this script, which is comprised of the scottish survey with additional columns

```
assigning.windowframes <- function(stock){
  #columns in correct format for next section
  stock <- window.frames(stock,"front")
  stock <- window.frames(stock,"left")
  stock <- window.frames(stock,"back")
  stock <- window.frames(stock,"right")
  return(stock)
}
```

An elevation is assigned window frames and percentage of double glazing

This is independent of if the elevation is attached or not as required by the NHM

@param stock - this is the whole of elevations created by other functions in this script, which is comprised of the scottish survey with additional columns

@param elevation - a string which contains the type of elevation, i.e. “front”

```
window.frames <- function(stock,elevation){
  #If opening is present on elevation then assign window types and percentage
  #of double glazed
  stock[,paste(elevation,"doubleglazedwindowframe",sep=".")] <- stock$doubleglazed
  stock[,paste(elevation,"singleglazedwindowframe",sep=".")] <- stock$singleglazed
  stock[,paste(elevation,"percentagedoubleglazed",sep=".")] <-
    stock$percentagedoubleglazing
  return(stock)
}
```

Door frames

Each elevation is assigned door frames

Calls the door.frames function

@param stock - this is the whole of elevations created by other functions in this script, which is comprised of the scottish survey with additional columns

```
assigning.doorframes <- function(stock){
  stock <- door.frames(stock,"front")
  stock <- door.frames(stock,"left")
  stock <- door.frames(stock,"back")
  stock <- door.frames(stock,"right")
  return(stock)
}
```

An elevation is assigned external door frames

An elevation can only be assigned an external door if there is some tenthsopening on that elevation.

There is no data in the scottish survey about doors. It has been assumed that the type of door frame is the same material as the windows and all doors are solid. It has been assumed that doors only exist on either the front or if applicable the back (for houses with an exposed back wall, so for example, enclosed terraces are not assigned a second external door in the back wall).

@param stock - this is the whole of elevations created by other functions in this script, which is comprised of the scottish survey with additional columns

@param elevation - a string which contains the type of elevation, i.e. “front”

```
door.frames <- function(stock,elevation){
  #If opening is present on an elevation and the elevation is back and the case is
  #a house or the elevation is the front and not a towerblock unless all 4 walls are
  #exposed then a door can be placed on the elevation
  door.test <- ((stock[,paste(elevation,"tenthsopening",sep=".")] > 0) &
                  ((elevation == "back" & stock$C1 != "Not house") |
                   (elevation == "front" & (stock$C2 != "Tower or slab" |
                                              stock$buildinginfo == "4 walls exposed"))))

  #Set all the doorframe types to 0 for all cases
  stock[,paste(elevation,"doorframe:wood,doortype:glazed",sep=".")] <- 0
  stock[,paste(elevation,"doorframe:metal,doortype:glazed",sep=".")] <- 0
  stock[,paste(elevation,"doorframe:upvc,doortype:glazed",sep=".")] <- 0
  stock[,paste(elevation,"doorframe:wood,doortype:solid",sep=".")] <- 0
  stock[,paste(elevation,"doorframe:metal,doortype:solid",sep=".")] <- 0
  stock[,paste(elevation,"doorframe:upvc,doortype:solid",sep=".")] <- 0

  #If the door.test is true (i.e. there can be a door on elevation) then one door is
  #assigned to the elevation of a material to match the double glazing.
  stock[(door.test == "TRUE" & stock$doubleglazed == "wood"),
        paste(elevation,"doorframe:wood,doortype:solid",sep=".")] <- 1
  stock[(door.test == "TRUE" & stock$doubleglazed == "metal"),
        paste(elevation,"doorframe:metal,doortype:solid",sep=".")] <- 1
  stock[(door.test == "TRUE" & stock$doubleglazed == "upvc"),
        paste(elevation,"doorframe:upvc,doortype:solid",sep=".")] <- 1
```

```
  return(stock)
}
```

Additional properties

The following section of code is generated in additional-properties.R

Make and save additional-properties data

Create a .csv file using the function make.additionalproperties, which creates a dataframe containing a complete set of populated variables for the additional-prope rties.csv stock file.

This file must be present in the stock import DTO package but can be empty save for the aacode field containing the survey code. However, additional field can be added to the file that can be called from scenarios in the NHM.

Currently, the code here populates the additional-properties.csv file with seven fields. These are SHCS raw variables and a corresponding renamed and recoded variable to match variables and levels contained in the ehs. These recoded variables are findisty (from M3), hhcompx(from hhtype), dblglaz4 (from Q47) and wallinsx from five SHCS varieable describing the primary and secondary wall types. (these are not currently included in the file.)

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the extension .csv

```
save.additionalproperties <- function(shcs, output) {
  #creates a completed .csv file from:
  #write.csv(dataframe containing values and column names, file name, without row id)
  write.csv(make.additionalproperties(shcs), file=output, row.names=FALSE, na="")
}
```

Make the dataframe that contains all the information required for the additional- properties.csv file

The dataframe is constructed and returned (as it is the last thing in the function that is assigned)

Additional columns can be added to the data frame by adding additional column names and vectors to fill that column (see help(data.frame))

@param shcs - the scottish survey data

```
make.additionalproperties <- function(shcs) {
  #creates a dataframe containing column names (word before the = sign) and a vector
  #to fill that column (vector after the = sign)
    shcs.findisty <- get.findisty(shcs$M3)
    shcs.hhcompx <- get.hhcompx(shcs$hhtype)
    shcs.dblglaz4 <- get.dblglaz4(shcs$Q47)
    shcs.wallinsx <- get.wallinsx(shcs$Q2, shcs$Q8, shcs$Q10, shcs$Q11, shcs$Q17)
    shcs.Felcav <- get.Felcav(shcs$Q8, shcs$Q10, shcs$Q17)
    shcs.Felext <- get.Felext(shcs$Q8, shcs$Q10, shcs$Q17)
    shcs.Felpv <- get.Felpv(shcs$D8)
    shcs.Felsol <- get.Felsol(shcs$D9)

  data.frame(aacode = shcs$uprn_new,
             Farnatur = NA,
             Felorien = NA,
             Felcavff = NA,
             Felcavlf = NA,
             Felcavrf = NA,
```

```
            Felcavbf = NA,
            Felextff = NA,
            Felextlf = NA,
            Felextrf = NA,
            Felextbf = NA,
            Felpvff = NA,
            Felpvlf = NA,
            Felpvrf = NA,
            Felpvbf = NA,
            Felsolff = NA,
            Felsollf = NA,
            Felsolrf = NA,
            Felsolbf = NA,
            Felcav_shcs = shcs.Felcav,
            Felext_shcs = shcs.Felext,
            Felpv_shcs = shcs.Felpv,
            Felsol_shcs = shcs.Felsol,
            Findisty = shcs.findisty,
            dblglaz4 = shcs.dblglaz4,
            NRmsEHS = shcs$J1,
            NRms2a = NA,
            NRms4 = NA,
            NRms5 = NA,
            hhcompx = shcs.hhcompx,
            imd1010 = NA,
            wallinsx = shcs.wallinsx,
            Felroofp = NA,
            AWEligible = "MISSING",
            CERTpriority = NA,
            WFG_preApr11 = NA,
            sap09 = shcs$SAP2009_BRE)
}
```

Map M3 to findisty; determine what distribution method is used by the main heating system

```
get.findisty <- function(M3) {
  as.factor(checked.revalue(
    M3,
    c(
      "Radiators" = "Radiators",
      "Appliance" = "Question Not Applicable",
      "Under-floor" = "Underfloor",
      "Ceiling" = "Question Not Applicable",
      "Other" = "Question Not Applicable",
      "No wet system" = "Question Not Applicable",
      "Unobtainable" = "Unknown")))
}
```

```
get.hhcompx <- function(hhtype) {
  as.factor(checked.revalue(
    hhtype,
    c(
      "Single adult" = "one person under 60",
```

```
      "Small adult" = "couple, no dependent child(ren) under 60",
      "Single parent" = "lone parent with dependent child(ren)",
      "Small family" = "couple with dependent child(ren)",
      "Large family" = "couple with dependent child(ren)",
      "Large adult" = "other multi-person households",
      "Older smaller" = "couple, no dependent child(ren) aged 60 or over",
      "Single pensioner" = "one person aged 60 or over"
    )))
}
```

@return a factor, with levels “no double glazing” “less than half” “more than half” “entire house”

```
get.dblglaz4 <- function(Q47) {
  dblglaz4 <- cut(Q47, c(0, 1, 4, 9, 10, 100),
      labels = c(
      "no double glazing",
      "less than half",
      "more than half",
      "entire house",
      "unknown"))
  dblglaz4[is.na(dblglaz4)] <- "unknown"
  dblglaz4[dblglaz4 == "unknown"] <- "no double glazing"
  droplevels(dblglaz4)
}
```

‘wallinsx’ variable creation: Q2 - Primary external wall construction Q8 - Insulation added to prim ext walls? Q10 - Extent of prim external wall Q11 - Sec external wall construction Q17 - Insulation added to sec ext walls? @return a factor, with levels “cavity insulated” “cavity uninsulated” “other”

```
get.wallinsx <- function(Q2, Q8, Q10, Q11, Q17) {
  # a column, which is true if we want the primary wall
  primary.wall <- Q10 >= 5

  construction <- ifelse(primary.wall, levels(Q2)[Q2], levels(Q11)[Q11])
  insulation   <- ifelse(primary.wall, levels(Q8)[Q8], levels(Q17)[Q17])

  ifelse(construction == "Cavity",
         # is it insulated?
         ifelse(insulation %in% "cavity",
                "cavity with insulation", "cavity uninsulated"),
         "other")
}

get.Felcav <- function(Q8, Q10, Q17) {
  primary.cwi <- ifelse(Q8 %in% "cavity", TRUE, FALSE)
  primary.Felcav <- ifelse(primary.cwi == TRUE, Q10, 0)
  secondary.cwi <- ifelse(Q17 %in% "cavity", TRUE, FALSE)
  secondary.Felcav <- ifelse(secondary.cwi == TRUE, 10 - Q10, 0)
  Felcav = primary.Felcav + secondary.Felcav
  Felcav <- ifelse(Felcav > 0, "Yes", "No")
}

get.Felext <- function(Q8, Q10, Q17) {
```

```
  primary.ext <- ifelse(Q8 %in% "External", TRUE, FALSE)
  primary.Felext <- ifelse(primary.ext == TRUE, Q10, 0)
  secondary.ext <- ifelse(Q17 %in% "External", TRUE, FALSE)
  secondary.Felext <- ifelse(secondary.ext == TRUE, 10 - Q10, 0)
  Felext = primary.Felext + secondary.Felext
  Felext <- ifelse(Felext > 0, "Yes", "No")
}

get.Felpv <- function(D8) {
  ifelse(D8 > 0 & D8 <= 10, "Yes", "No")
}

get.Felsol <- function(D9) {
  ifelse(D9 > 0 & D9 <= 10, "Yes", "No")
}
```

Additional DTO import files

The following section of code is generated in DTO-import-files.R

The additional files required for the stock import are created.

Make and save IStockImportMetadata

Create a .csv file that contains the correct data which is created in the function make.IStockImportMetadataDTO

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the .csv

```
save.IStockImportMetadataDTO <- function(shcs, output) {
  write.csv(make.IStockImportMetadataDTO(shcs), file=output, row.names=FALSE)
}
```

Make a IStockImportMetadataDTO from the SHCS

@param shcs - the scottish survey data

```
make.IStockImportMetadataDTO <- function(shcs) {
  data.frame(
              Date  = "[new_date]",
              DescriptionByUser = "[description_update]",
              SourceName    = "[sourceName_update]",
              SourceVersion = "2012",
              StockImporterVersion  = "[stockImportVersion_update]",
              UserName =    "[ImporterUserName]")
}
```

Make and save IImportLog

Create a .csv file that contains the correct data which is created in the function make.IImportLogDTO

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the .csv

```
save.IImportLogDTO <- function(shcs, output) {
  write.csv(make.IImportLogDTO(shcs), file=output, row.names=FALSE)
}
```

Make a IImportLog from the SHCS

@param shcs - the scottish survey data

```
make.IImportLogDTO <- function(shcs) {
subset(data.frame(
              Aacode    = "",
              Messages  = ""), Aacode == "", DROP = TRUE)
}
```

Make and save metadata

Create a .csv file that contains the correct data which is created in the function make.metadata

@param shcs - the scottish survey data

@param output - the path to the output file including the required file name and the .csv

```
save.metadata <- function(shcs, output) {
  write.csv(make.metadata(shcs), file=output, row.names=FALSE)
}
```

Make a metadata from the SHCS

@param shcs - the scottish survey data

```
make.metadata <- function(shcs) {
subset(data.frame(
              type  = "",
              DTO   = ""), type == "", DROP = TRUE)
}
```