England survey stock conversion

Centre for Sustainable Energy

Tue May 10 16:55:27 2016

Contents

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("cases.R",chdir = T)
source("stories.R",chdir = T)
source("elevations.R",chdir = T)
source("occupants.R",chdir = T)
source("roofs.R",chdir = T)
source("lighting.R",chdir = T)
source("spaceheating.R", chdir = T)
source("waterheating.R", chdir = T)
source("ventilation.R", chdir = T)
source("additional-properties.R", chdir = T)
source("DTO-import-files.R", chdir = T)
```

The main entry point for making english data

@param path.to.ehcs - where the EHS SPSS file is

@param path.to.output - a directory to put the output files in

```
make.stock <- function(path.to.ehcs, path.to.output) {
  print(paste("Loading", path.to.ehcs))

  #' Read in required properties from survey files into a single wide data-frame
  allEntries <- merge.all.sav.files(path.to.ehcs)

  #' The following are a special case, where there are more than one enty per aacode
  #' and adding to all entries table is not necessary.
  doorEntries <- read.spss(file.path(path.to.ehcs, "physical/doors.sav"),
                           to.data.frame = TRUE, reencode='utf-8')
  peopleEntries <- read.spss(file.path(path.to.ehcs, "interview/people.sav"),
                             to.data.frame = TRUE, reencode='utf-8')
  roomsEntries <- read.spss(file.path(path.to.ehcs, "physical/introoms.sav"),
                            to.data.frame = TRUE, reencode='utf-8')

  #' Construct DTO's
  casesDTO <- cases.make(allEntries)
  elevationsDTO <- elevations.make(allEntries, doorEntries)
  occupantsDTO <- occupants.make(allEntries, peopleEntries)
  roofsDTO <- roofs.make(allEntries)
  lightingDTO <- lighting.make(roomsEntries)
```

```
  spaceHeatingDTO <- space.heating.make(allEntries, path.to.ehcs, path.to.output)
  waterHeatingDTO <- make.waterheating(allEntries, spaceHeatingDTO, path.to.ehcs)
  ventilationDTO <- make.ventilation(allEntries, elevationsDTO)
  additionalpropertiesDTO <- make.additionalproperties(allEntries)
  IImportLogDTO <- make.eng_IImportLogDTO(allEntries)
  IStockImportMetadataDTO <- make.eng_IStockImportMetadataDTO(allEntries)
  metadata <- make.eng_metadata(allEntries)

  #' Output DTO's
  save.dto(casesDTO, file.path(path.to.output, "cases.csv"))
  save.dto(elevationsDTO, file.path(path.to.output, "elevations.csv"))
  save.dto(occupantsDTO, file.path(path.to.output, "occupants.csv"))
  save.dto(roofsDTO, file.path(path.to.output, "roofs.csv"))
  save.dto(lightingDTO, file.path(path.to.output, "lighting.csv"))
  save.dto(spaceHeatingDTO, file.path(path.to.output, "space-heating.csv"))
  save.dto(waterHeatingDTO, file.path(path.to.output, "water-heating.csv"))
  save.dto(ventilationDTO, file.path(path.to.output, "ventilation.csv"))
  save.dto(additionalpropertiesDTO, file.path(path.to.output,
                                              "additional-properties.csv"))
  save.eng_IImportLogDTO(IImportLogDTO, file.path(path.to.output, "IImportLogDTO.csv"))
  save.eng_IStockImportMetadataDTO(IStockImportMetadataDTO,
                            file.path(path.to.output, "IStockImportMetadataDTO.csv"))
  save.eng_metadata(metadata, file.path(path.to.output, "metadata.csv"))

  #' Just do stories on their own as they have a separate bit of code.
  scale.storeys <- if (exists("option.ehs.storeys.scale")) option.ehs.storeys.scale
                   else FALSE

  generate.all.storeys(path.to.ehcs, file.path(path.to.output, "storeys.csv"), scale.storeys)
}

merge.all.sav.files <- function(path.to.ehcs){
  #' We use general.sav as our base-line of all cases available
  allEntries <- read.spss(file.path(path.to.ehcs, "derived/general.sav"),
                          to.data.frame = TRUE, reencode='utf-8')

  #' Now merge all other spss files that should have just one entry for each house case
  toMerge <- Reduce(function(a, b){
    join(a,b, by = "aacode")
  },Map(function(name){
    read.spss(file.path(path.to.ehcs, name), to.data.frame = TRUE, reencode='utf-8')
  }, c("physical/firstimp_physical.sav",
       "physical/shape.sav",
       "physical/interior.sav",
       "derived/physical.sav",
       "physical/around.sav",
       "physical/services.sav",
       "physical/flatdets.sav",
       "derived/interview.sav",
       "interview/rooms.sav",
       "physical/elevate.sav",
       "fuel_poverty/fuel_poverty_dataset.sav",
       "fuel_poverty/fuel-poverty-dataset-supplementary-variables.sav")))
```

```
  merged <- join(allEntries,
                 toMerge,
                 by = "aacode")

  #' Dimensions sav file uses different case for Aacode column so we need to-do a
  #' different merge
  allEntries <- merge(merged,
                      read.spss(file.path(path.to.ehcs, "derived/dimensions.sav"),
                                to.data.frame = TRUE, reencode='utf-8'),
                      all.x = TRUE,
                      by.x = "aacode", by.y = "Aacode")


  #' supplementary data on AW flag provided by DECC via BRE. data is in csv format so
  #' different merge
  allEntries <- merge(allEntries,
                      read.csv(file.path(path.to.ehcs, "fuel_poverty/2012-aw-flag.csv")),
                      all.x = TRUE,
                      by = "aacode")


  # Create room summary and merge with allEntries data.frame
  introoms <- read.spss(file.path(path.to.ehcs, "physical/introoms.sav"),
                        to.data.frame = TRUE, reencode='utf-8')
  case.room.summary <- summarise.rooms(introoms)
  allEntries <- merge(allEntries, case.room.summary, all.x = TRUE, by = "aacode")

  return(allEntries)
}
```

Constructs summary information from all room files for a specific house case

```
summarise.rooms <- function(introoms){
  summary <- cast(introoms, aacode  ~ type, value = "Finflrsf")
  names(summary) <- c("aacode",
                      "livingRoomHasSolidFloor",
                      "kitchenHasSolidFloor",
                      "bedroomHasSolidFloor",
                      "bathroomHasSolidFloor",
                      "circulationHasSolidFloor")
  return(summary)
}

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

Cases

The following section of code is generated in cases.R

Make 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 allEntries - This is a dataframe containing a a data-frame derived from one or more survey spss(sav) files and their variables. Survey files which contain more than one entry per house case are handled seperatley and should be specified in the following parameters.

@assumption - If not a house then assume it has a draught lobby

@param allEntries - combined sav files from the EHS

```
cases.make <- function(allEntries) {
  # Create a variable which contains a count of all survey case
  numOfCases = length(allEntries$aacode)

  # Calculates the dimensions for a house case, see  calc.plot.dimensions function
  # for more detail, where allEntries$[variable name] is a merged spss data.frame and
  # variable name is the name from ths spss files
  plot.dimensions <- calc.plot.dimensions(
      widthOfPlot = allEntries$Fexwidth,
      doesFrontPlotExist = allEntries$Fexplotf,
      doesBackPlotExist = allEntries$Fexplotr,
      depthOfBackPlot = allEntries$Fexp2fdp,
      depthOfFrontPlot =  allEntries$Fexp1fdp)

  occupants <- occupant.counts(allEntries$aacode, allEntries$hhsizex,
                               allEntries$ndepchild)

  living.area.data <- cal.livingarea.data(allEntries$Finrooms,allEntries$Finlivwi,
                                          allEntries$Finlivde,allEntries$FloorArea)

  # Construct a data-frame for each house case
  cases <- data.frame(
            aacode = allEntries$aacode,
            adults = occupants$adults,
            backplotdepth = plot.dimensions$backplotDepth,
            backplotwidth = plot.dimensions$backplotWidth,
            buildyear = allEntries$fodconac,
            builtformtype = builtform.type(allEntries$dwtypenx),
            children = occupants$children,
            dwellingcaseweight = allEntries$aagpd1112,
            grndfloortype = groundfloor.construction.type(allEntries$kitchenHasSolidFloor,
                                                  allEntries$livingRoomHasSolidFloor),
            frontplotdepth = plot.dimensions$frontplotDepth,
            frontplotwidth = plot.dimensions$frontplotDepth,
            hasaccesstooutsidespace = has.access.to.outside.space(allEntries$Fexpltyp,
```

```
                                                    plot.dimensions$backplotArea,
                                                    plot.dimensions$frontplotArea),
            hasdraftlobby = (is.a.house(allEntries$dwtype8x) == FALSE),
            hasloft = has.loft(allEntries$Flitypes, allEntries$Flithick),
            householdcaseweight = household.weight(allEntries$aagph1112),
            livingareafaction = living.area.data$livingAreaFaction,
            morphologytype = morphology.type.lookup(allEntries$rumorph),
            numofhabitalrooms = living.area.data$numHabitalRooms,
            numofbedrooms = count.num.of.bedrooms(allEntries),
            ongasgrid = ifelse(is.na(allEntries$Fingasms), FALSE,
                               ifelse(allEntries$Fingasms == "Yes", TRUE, FALSE)),
            partlyownsroof = ifelse(is.na(allEntries$Owntype), FALSE,
                                    owns.part.of.roof(allEntries$Owntype)),
            regiontype = region.type(allEntries$GorEHS),
            tenuretype = tenure.type(allEntries$tenure8x))

  print(paste("Cases DTO complete; number of records: ", nrow(cases)))
  return(cases)

}
```

Rurality

Determine the rurality of each dwelling by recoding an existing variable

```
morphology.type.lookup <- function(rumorph){
  as.factor(checked.revalue(
    rumorph,c(
      "hamlets and isolated dwellings" = "hamletsandisolateddwellings"
      ,"village" = "village"
      ,"town and fringe"= "townandfringe"
      ,"urban > 10k" = "urban"
      )))
}
```

Presence of a loft

If Loft type is not NA or not loft OR insulation thickness greate than 0 then has a loft

@param FLITYPES - Services/Loftype @param FLITHICK - Approx loft insulation thickness

```
has.loft <- function(FLITYPES, FLITHICK) {
  hasNoLoft <- is.na(FLITYPES) | FLITYPES == "No loft - flat or very shallow pitch"
  noInsulation <- is.na(FLITHICK) | FLITHICK == "Don t know thickness" | FLITHICK ==
                    "No insulation" | FLITHICK == 0

  return(hasNoLoft == FALSE | noInsulation == FALSE)
}
```

Tenure

Determine the tenure of the dwelling by recoding an existing variable

```
tenure.type <- function(tenure8x){
  as.factor(checked.revalue(
    tenure8x,c(
      "owner occupied - occupied" = "OwnerOccupied",
      "private rented - occupied" = "PrivateRented",
      "local authority - occupied" = "LocalAuthority",
      "RSL - occupied" = "HousingAssociation",
      "owner occupied - vacant" = "OwnerOccupied",
      "private rented - vacant" = "PrivateRented",
      "local authority - vacant" = "LocalAuthority",
      "RSL - vacant" = "HousingAssociation"
      )))
}

region.type <- function(GorEHS){
  as.factor(checked.revalue(
    GorEHS,c(
      "North West" = "northwest",
      "North East" = "northeast",
      "Yorkshire and the Humber" = "yorkshireandhumber",
      "East Midlands" = "eastmidlands",
      "West Midlands" = "westmidlands",
      "East" = "eastofengland",
      "London" = "london",
      "South East" = "southeast",
      "South West" = "southwest")))
}
```

Number of bedrooms

Determined from the introoms file, which has a main bedroom variable and then a uses the addition room variables to determine which of them are additional bedrooms

```
count.num.of.bedrooms <- function(allEntries){
  numOfBedrooms = 0;

  numOfBedrooms <- ifelse(allEntries$Finbedex == "Yes", numOfBedrooms + 1, 0)
  numOfBedrooms <- ifelse(is.a.bedroom(allEntries$Finex1ex, allEntries$Finex1fu),
                          numOfBedrooms + 1, numOfBedrooms)
  numOfBedrooms <- ifelse(is.a.bedroom(allEntries$Finex2ex, allEntries$Finex2fu),
                          numOfBedrooms + 1, numOfBedrooms)
  numOfBedrooms <- ifelse(is.a.bedroom(allEntries$Finex3ex, allEntries$Finex3fu),
                          numOfBedrooms + 1, numOfBedrooms)
  numOfBedrooms <- ifelse(is.a.bedroom(allEntries$Finex4ex, allEntries$Finex4fu),
                          numOfBedrooms + 1, numOfBedrooms)
  numOfBedrooms <- ifelse(is.a.bedroom(allEntries$Finex5ex, allEntries$Finex5fu),
                          numOfBedrooms + 1, numOfBedrooms)
  numOfBedrooms <- ifelse(is.a.bedroom(allEntries$Finex6ex, allEntries$Finex6fu),
                          numOfBedrooms + 1, numOfBedrooms)
  numOfBedrooms <- ifelse(is.a.bedroom(allEntries$Finex7ex, allEntries$Finex7fu),
                          numOfBedrooms + 1, numOfBedrooms)
```

```
  return (numOfBedrooms)
}
```

Specific function to determine whether the addition rooms are bedrooms

@param room.function

```
is.a.bedroom <- function(does.room.exist, room.function){
  isBedroom <- ifelse(does.room.exist == "Yes", TRUE, FALSE)
  isBedroom <- ifelse(isBedroom, is.na(room.function) == FALSE, FALSE)
  isBedroom <- ifelse(isBedroom, ifelse(room.function == "Twin bedroom" |
                              room.function == "Single bedroom", TRUE, FALSE), FALSE)

  return(isBedroom)
}
```

Household weight

Returns the dwelling weight, if this is NA the function returns -9

@param aagph1112 - DwellWeight_PairedCases

```
household.weight <- function(aagph1112){
  return (ifelse(is.na(aagph1112), -9, aagph1112))
}
```

Access to outside space

Returns true only if type of plot is not No private plot or shared plot and either front or back plots have an area greated than 0.

@param Fexpltyp type of plot @param backPlotArea @param frontPlotArea

```
has.access.to.outside.space <- function(Fexpltyp, backPlotArea, frontPlotArea){
  okPlotType <- !Fexpltyp == "No private plot or shared plot"
  return(okPlotType & (backPlotArea > 0 | frontPlotArea > 0))
}
```

Stub method just returns string of TODO

@param length - The number of rows to return

```
to.do <- function(length){
  rep("TODO", length(length))
}
```

External area dimensions

Calculates, if available the dimensions of the front and back, first uses values indicating existence of front and back plot, if these are caluclated as true width and depth and then derived from widthOfPlot and depthOfBackPlot.

@param widthOfPlot - contains value from which width of plot can be derived

@param doesFrontPlotExist - contains a YES/NO/NA value

@param doesBackPlotExist - contains a YES/NO/NA value

@param depthOfBackPlot - contains value from which width of back plot can be derived

@param depthOfFrontPlot - contains value from which depth of front plot can be derived

```
calc.plot.dimensions <- function(widthOfPlot, doesFrontPlotExist, doesBackPlotExist,
                                 depthOfBackPlot, depthOfFrontPlot){
  #Frontplot dimensions

  #Backplot dimensions
  doesBackPlotExist <- ifelse(is.na(doesBackPlotExist), FALSE,
                              ifelse(doesBackPlotExist == "Yes", TRUE, FALSE))
  backplotWidth <- ifelse(doesBackPlotExist,
                          ifelse(is.na(widthOfPlot), 0, widthOfPlot), 0)
  backplotDepth <- ifelse(is.na(depthOfBackPlot),0,depthOfBackPlot)

  #Frontplot dimensions
  doesFrontPlotExist <- is.na(doesFrontPlotExist) == FALSE & doesFrontPlotExist == "Yes"
  frontplotWidth <- ifelse(doesFrontPlotExist,
                           ifelse(is.na(widthOfPlot), 0, widthOfPlot), 0)
  frontplotDepth <- ifelse(is.na(depthOfFrontPlot),0,depthOfFrontPlot)

  data.frame(
    backplotWidth <- ifelse(doesBackPlotExist,backplotWidth, 0),
    backplotDepth <- ifelse(doesBackPlotExist,backplotDepth, 0),
    backplotArea <- ifelse(doesBackPlotExist, backplotDepth * backplotWidth, 0),

    frontplotWidth <- ifelse(doesFrontPlotExist,frontplotWidth, 0),
    frontplotDepth <- ifelse(doesFrontPlotExist,frontplotDepth, 0),
    frontplotArea <- ifelse(doesFrontPlotExist, frontplotDepth * frontplotWidth, 0))
}
```

Occupants

Calculates the number of adults and chidren

Returns - data.frame with two columns, adult and children

```
occupant.counts <- function(aacode, numOfPeopleInHouse, numOfDepdententChildrenInTheHouse){
  calc_numOfPeopleInHouse <- ifelse(is.na(numOfPeopleInHouse),0,numOfPeopleInHouse)
  calc_children <- ifelse(is.na(numOfDepdententChildrenInTheHouse),
                          0,numOfDepdententChildrenInTheHouse)

  calc_adults <- calc_numOfPeopleInHouse - calc_children
  calc_adults <-  ifelse(calc_adults < 0, 0, calc_adults)
  calc_adults <-  ifelse(calc_adults < 0, 0, calc_adults)

  calc_adults <-  ifelse(calc_adults == 0 & calc_children > 0 , NA, calc_adults)

  calc_adults <-  ifelse(calc_adults == 0, "NULL", calc_adults)
```

```
  calc_adults <-  ifelse(calc_adults == "", "NULL", calc_adults)
  calc_adults <-  ifelse(is.na(calc_adults), "NULL", calc_adults)
  calc_children <- ifelse(calc_adults == "NULL" & calc_children == 0, "NULL", calc_children)

  data.frame(aacode, adults = calc_adults, children = calc_children)
}
```

Builtform type

Simple look-up table which maps an EHCS survey value for dwelling type to one which can be used by the NHM stock import

@param dwellingType - String representing dwelling type from the survey

```
builtform.type <-function(dwellingType){
  as.factor(checked.revalue(
    dwellingType,c(
      "converted flat" = "ConvertedFlat",
      "semi detached" = "SemiDetached",
      "purpose built flat, low rise" = "PurposeBuiltLowRiseFlat",
      "purpose built flat, high rise" = "PurposeBuiltHighRiseFlat",
      "detached" = "Detached",
      "bungalow" = "Bungalow",
      "end terrace" = "EndTerrace",
      "mid terrace" = "MidTerrace")))}
```

Roof ownership

```
owns.part.of.roof <- function(OWNTYPE){
  return (as.character(levels(roof.look.up(OWNTYPE))[roof.look.up(OWNTYPE)]))
}
```

Determine the ownership of the dwelling in order to determine whether the roof is owned by the occupants

```
roof.look.up <- function(OWNTYPE){
  as.factor(checked.revalue(
    OWNTYPE,c(
      "freeholder of house" = TRUE,
      "leaseholder - no share of FH" = FALSE,
      "leaseholder owning FH collectively" = TRUE,
      "leaseholder owning FH of whole bldg" = TRUE,
      "freeholder of flat - owning FH of flat only" = FALSE,
      "commonholder (property built as CH)" = TRUE,
      "commonholder (property converted to CH)" = TRUE)))
}
```

Groundfloor construction type

If either hasSolidKitchenFloor or hasSolidLivingRoomFloor have the value of "NO" then we assume the floor construction type is solid. This methodology came from CARS, we could do a more detailed job using floor levels etc... as we have model for each story in the house.

```
groundfloor.construction.type <- function(hasSolidKitchenFloor, hasSolidLivingRoomFloor){
      floorType <- ifelse(is.na(hasSolidKitchenFloor), "solid",
                   ifelse(hasSolidKitchenFloor == "Yes", "solid", "suspendedtimber"))
      floorType <- ifelse(floorType == "solid",
                          ifelse(
                            is.na(hasSolidLivingRoomFloor),
                            "solid",
                            ifelse(hasSolidLivingRoomFloor == "Yes",
                                   "solid","suspendedtimber")),
                          "suspendedtimber")

      return (floorType)
}
```

Living area fraction

Returns a data frame with the following values: livingArea - this is (livingRoomWidth * livingRoomDepth) numHabitalRooms - uses supplied value as is numOfBedrooms - always returns zero TODO: don't know why?

@param numOfHabitalRooms - FINROOMS (interior) @param livingRoomWidth - FINLIVWI (interior) in meters @param livingRoomDepth - FINLIVDE (interior) in meters @param totalFloorArea - FLOORARE (dimensions) in meters

```
cal.livingarea.data <- function(numOfHabitalRooms, livingRoomWidth,
                                livingRoomDepth, totalFloorArea){


  calc_livingRoomWidth <- ifelse(is.na(livingRoomWidth),0, livingRoomWidth)
  calc_livingRoomWidth <- ifelse(calc_livingRoomWidth == 88.8, 0, calc_livingRoomWidth)

  calc_livingRoomDepth <- ifelse(is.na(livingRoomDepth),0, livingRoomDepth)
  calc_livingRoomDepth <- ifelse(calc_livingRoomDepth == 88.8, 0, calc_livingRoomDepth)

  calc_livingAreaFaction <- (livingRoomWidth * livingRoomDepth)/ totalFloorArea

  calc_livingAreaFaction <- ifelse(totalFloorArea <= 0, 0,
                           (calc_livingRoomWidth * calc_livingRoomDepth)/ totalFloorArea)

  data <- data.frame(
    livingAreaFaction = calc_livingAreaFaction,
    numHabitalRooms = ifelse(is.na(numOfHabitalRooms), 0, numOfHabitalRooms),
    numOfBedrooms = 0)

  return (data)
}
```

Access to outdoor space

If front or back plots are not zero in size and plot type is Private or Shared then returns true, otherwise false.

@param typeOfPlot - String @param frontPlotArea - Numeric - Area in m2 @param backPlotArea - Numberic - Area in m2

```
accces.to.outside.space <- function(typeOfPlot, frontPlotArea, backPlotArea){
    #' Check either back or front plot exists
    # backPlotExists = plot.dimensions$backplotDepth > 0 & plot.dimensions$backplotWidth > 0
    # frontPlotExists = plot.dimensions$backplotDepth > 0 & plot.dimensions$backplotWidth > 0

    #' If plot is private or shared then return true
}
```

Simple look up table based on floor location type of either ground or basement

@param level - As in the EHCS this is one of bb, BB, gg or GG

```
floor.level <- function(level){
  as.factor(checked.revalue(
    level,c(
      "gg" = "GROUND",
      "GG" = "GROUND",
      "bb" = "BASEMENT",
      "BB" = "BASEMENT"
      )))
}
```

Roofs

The following section of code is generated in roofs.R

Make 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 - combined sav files data.frame from the EHS data

```
roofs.make <- function(allEntries){
  roofs <- data.frame(
    aacode = allEntries$aacode,
    insulationthickness = loft.insulation.lookup(allEntries$loftinsx),
    coveringtype = roof.covering.type.lookup(allEntries$typercov),
    structuretype = roof.structure.type.lookup(allEntries$typerstr),
    contructiontype = roof.construction.type(allEntries$typercov, allEntries$typerstr)
  )

  print(paste("roofs DTO complete; number of records: ", nrow(roofs)))

  return (roofs)
}
```

Roof structure type

Assign each dwelling case a roof structure

@assumption - NHM doesn't have mixed type, pitched roofs are most common in the survey so map to this

```
roof.structure.type.lookup <- function(typerstr){
  as.factor(checked.revalue(
    typerstr,c(
      "mixed types" = "Pitched",
      "pitched" = "Pitched",
      "mansard" = "Mandard",
      "flat" = "Flat",
      "chalet" = "Chalet"
      )))
}

roof.structure <- function(typerstr){
  return (as.character(
    levels(roof.structure.lookup(typerstr))[roof.structure.lookup(typerstr)]))
}

roof.structure.lookup <- function(typerstr){
  as.factor(checked.revalue(
    typerstr,c(
```

```
      "chalet" = "PitchedSlateOrTiles",
      "pitched" = "PitchedSlateOrTiles",
      "mixed types" = "PitchedSlateOrTiles",
      "flat" = "Flat",
      "mansard" = "PitchedSlateOrTiles"
    )))
}
```

Roof covering type

Assign the type of roofing material (e.g. tiles, slates or other material) to each dwelling case

@asummption - Where covering type is mixed types assume is actually Tile

```
roof.covering.type.lookup <- function(typercov){
  as.factor(checked.revalue(
    typercov,c(
      "mixed types" = "Tile",
      "natural slate/stone/shingle" = "Slates",
      "man made slate" = "Slates",
      "clay tile" = "Tile",
      "concrete tile" = "Tile",
      "asphalt" = "Asphalt",
      "felt" = "Felt",
      "glass/metal/laminate" = "Metal",
      "thatch" = "Thatch"
    )))
}
```

Roof construction type

@param typercov - predominantTypeOfRoofCovering @param typerstr - predominantTypeOfRoofStucture

```
roof.construction.type <- function(typercov, typerstr){
  constructionType <- ifelse(typercov == "thatch", "Thatched", NA)
  constructionType <- ifelse(is.na(constructionType), roof.structure(typerstr),
                             constructionType)

  return (constructionType)
}
```

Loft insulation thickness

Loft insulation for properties withouta loft or heat loss roof can be left blank or set to 0.

```
loft.insulation <- function(loftinsx){
    return (as.numeric(levels(
      loft.insulation.lookup(loftinsx))[loft.insulation.lookup(loftinsx)]))
}

loft.insulation.lookup <- function(loftinsx){
```

```
#     from.Flithick <- as.factor(checked.revalue(
#                         Flithick,c(
#                           "25mm" = 25,
#                           "50mm" = 50,
#                           "75mm" = 75,
#                           "100mm" = 100,
#                           "125mm" = 125,
#                           "150mm" = 150,
#                           "200mm" = 200,
#                           "250mm" = 250,
#                           "300mm" = 300,
#                           ">300mm" = 350,
#                           "No insulation" = 0,
#                           "Don t know thickness" = 0,
#                           "------------" = 0
#                           )))
    #override previous logic with loftinxs variable from derived physical file
    from.loftinsx <- ifelse(loftinsx < 12, 0,
                            ifelse(loftinsx == 12, 0,
                            ifelse(loftinsx <= 25, 25,
                            ifelse(loftinsx <= 50, 50,
                            ifelse(loftinsx < 90, 75,
                            ifelse(loftinsx < 113, 100,
                            ifelse(loftinsx < 138, 125,
                            ifelse(loftinsx < 175, 150,
                            ifelse(loftinsx < 225, 200,
                            ifelse(loftinsx < 275, 250,
                            ifelse(loftinsx < 325, 300,
                            ifelse(loftinsx >= 325, 350,
                            0))))))))))))
    ifelse(is.na(from.loftinsx), 0, from.loftinsx)
}
```

Lighting

The following section of code is generated in lighting.R

Make 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 introoms - the the introoms sav file data frame

```
lighting.make <- function(introoms){

  roomsWithWeights <- data.table(
    aacode = introoms$aacode,
    type = introoms$type,
    hasLowEnergyLights = ifelse(is.na(introoms$Finhtglg), FALSE,
                                ifelse(introoms$Finhtglg == "Yes", TRUE, FALSE)),
    weight = room.weighting(introoms$type)
  )

  # Create sum weights for each room in house case
  dominator <- roomsWithWeights[, j=(sum(weight)), by = aacode]
  setnames(dominator, c("aacode", "dominator"))

  # Create sum weight for each room in a house case that has low energy lights
  roomsWithWeights <- subset(roomsWithWeights, hasLowEnergyLights == TRUE)
  numerator <- roomsWithWeights[, j=(sum(weight)), by = aacode]
  setnames(numerator, c("aacode", "numerator"))

  # Merge the weights
  merged <- join(dominator, numerator, by = "aacode")

  # Compile the results
  lights <- data.frame(
    aacode = merged$aacode,
    fraction = ifelse(is.na(merged$numerator/merged$dominator),0,
                      merged$numerator/merged$dominator)
  )

  print(paste("lighting DTO complete; number of records: ", nrow(lights)))

  return(lights)
}
```

Determine number of fittings in each room

Apply weighting of rooms in order to determine the number of lightbulbs found in each room.

```
room.weighting <- function(TYPE){
  return (as.numeric(levels(room.weighting.lookup(TYPE))[room.weighting.lookup(TYPE)]))
}
```

Gets the weighting for a room to be used in the Low Energy Lighting calculation, as specified in Bredem 8 section 4.2.

@param TYPE - Room Type

```
room.weighting.lookup <- function(TYPE){
  as.factor(checked.revalue(
    TYPE,c(
      "Living room" = 2,
      "Kitchen" = 2,
      "Bedroom" =  1,
      "Bathroom" = 1,
      "Circulation" = 2
      )))}
```

Occupants

The following section of code is generated in occupants.R

Make 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 allEntries

@param peopleEntries

```
occupants.make <- function(allEntries, peopleEntries){
  surveyYear <- 2012
  dateMovedIn <- ifelse(is.na(allEntries$lenres), NA, surveyYear - allEntries$lenres)
  dateMovedIn <- ifelse(is.na(dateMovedIn), "NULL",
                        as.POSIXct(paste(c(dateMovedIn),c("01"),c("01"),sep = "-"),
                                   format = "%Y-%m-%d", tz = "UTC"))

  allOccupants <- data.frame(
    aacode = allEntries$aacode,
    chiefincomeearnersage = ifelse(is.na(allEntries$agehrpx), "", allEntries$agehrpx),
    householdincomebeforetax = allEntries$fpfullinc,
    hasoccupantonbenefits = ifelse(is.na(allEntries$hhvulx), "",
                                   ifelse(allEntries$hhvulx == "yes", TRUE, FALSE)),
    hasdisabledorsickoccupant = ifelse(is.na(allEntries$hhltsick), "",
                                       ifelse(allEntries$hhltsick == "yes", TRUE, FALSE)),
    datemovedin = dateMovedIn
  )

  #Add Working Hours
  hrpPersons <- subset(peopleEntries, peopleEntries$PERSNO == peopleEntries$HRP)
  allHours <- data.frame(
    aacode = hrpPersons$aacode,
    workinghours = hrpPersons$NoOfHrsR
  )

  allOccupants <- join(x = allOccupants, y = allHours, by = c("aacode"), type = c("left"))

  print(paste("occupants DTO complete; number of records: ", nrow(allOccupants)))

  return (allOccupants)
}
```

Spaceheating

The following section of code is generated in spaceheating.R

Make 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 allEntries - combined sav files data.frame from the EHS data

```
space.heating.make <- function(allEntries, path.to.ehcs, path.to.output){
  use.sedbuk <- if (exists("option.ehs.spaceheating.sedbuk")) option.ehs.spaceheating.sedbuk
                else TRUE
  if (use.sedbuk) {
      run.sedbuk <- if (exists("option.ehs.spaceheating.sedbuk.run")) option.ehs.spaceheating.sedbuk.ru
                    else FALSE

      # Run sedbuk matching java code if needed.
      if (run.sedbuk) {
          sedbukLookUp <- sedbuk.look.up.table(allEntries)
          write.csv(sedbukLookUp, file=file.path(path.to.ehcs, "r_sedbuk.csv"), row.names=FALSE)

          jarFileLocation <- paste(file.path(getwd(), "ehcs", "boilermatcher-all-1.0.0.jar"))
          fileLocations <- paste(file.path(path.to.ehcs, "r_sedbuk.csv"),
                                 file.path(path.to.output, "sedbuk_matches.csv"))
          javaCommand <- paste("java -jar", jarFileLocation, fileLocations)
          print(paste("Executing java sedbuk look-up with command: ",javaCommand))
          system(command = javaCommand)
      }

      ## Read in the created sebuk match results
      sedbuk <- read.csv(file.path(path.to.output, "sedbuk_matches.csv"))
  }

  ## match remaining cases against look up tables - we loop over the lookup tables
  ## specified in option.ehs.spaceheating.lookups.

  unmatched.cases <- if (use.sedbuk) subset(allEntries, !(aacode %in% sedbuk$aacode))
                     else allEntries

  ## supply a default set of lookups if not defined elsewhere (e.g. by main function)
  lookups <- if (exists("option.ehs.spaceheating.lookups"))
                 option.ehs.spaceheating.lookups
             else
                 c("lup-boiler-4b.csv",
                   "electric-boiler-lup.csv",
                   "gas-room-heater-lup.csv",
                   "oil-room-heater-lup.csv",
                   "other-room-heater-lup.csv",
```

```
                  "solid-fuel-boiler-lups.csv",
                  "storage-heater-lup.csv",
                  "warm-air-lup.csv",
                  "Finchbcd-lup.csv",
                  "space-heating-missing-data-lup.csv")


for (lup in lookups) {
    new.matched.cases <- heating.matching(
        path.to.ehcs,
        unmatched.cases,
        lup)
    unmatched.cases <- subset(unmatched.cases, !(aacode %in% new.matched.cases$aacode))
    matched.cases <- if (exists("matched.cases")) rbind(matched.cases, new.matched.cases)
                     else new.matched.cases
}

#combine all spaceheating with the main table
                                        #gather information for the sedbuk matched data into a data fra
if (use.sedbuk) {
    matched.sedbuk <- data.frame(
        aacode=sedbuk$aacode
       ,basicefficiency=sedbuk$annualefficiency
       ,winterefficiency=sedbuk$winterefficiency
       ,summerefficiency=sedbuk$summerefficiency
       ,spaceheatingsystemtype=sedbuk.boiler.to.spaceheating.type(sedbuk$boilertype)
       ,fluetype=sedbuk$fluetype
       ,mainheatingfuel=sedbuk$fueltype
       ,iscondensing=sedbuk$condensing
                                        #,matchSource=rep("sedbuk", length(sedbuk$aacode))
       ,isstoragecombicylinderthemostatpresent=rep("NULL", nrow(sedbuk))
       ,isstoragecombicylinderfactoryinsulated=rep("NULL", nrow(sedbuk))
       ,storageheatertype=rep("", nrow(sedbuk))
    )
}
#gather information for the SAP table 4a/4b and Fincbcd matched data into a second
#data frame of a similar structure
matched.cases <-data.frame(
             aacode=matched.cases$aacode
            ,basicefficiency=matched.cases$basicefficiency
            ,winterefficiency=matched.cases$winterefficiency
            ,summerefficiency=matched.cases$summerefficiency
            ,spaceheatingsystemtype=tolower(matched.cases$spaceheatingsystemtype)
            ,fluetype=tolower(matched.cases$fluetype)
            ,mainheatingfuel=tolower(matched.cases$mainheatingfuel)
            ,iscondensing=as.factor(matched.cases$iscondensing)
            #,matchSource=rep("lookup", length(matched.cases$aacode))
            ,isstoragecombicylinderthemostatpresent=rep("NULL", nrow(matched.cases))
            ,isstoragecombicylinderfactoryinsulated=rep("NULL", nrow(matched.cases))
            ,storageheatertype=tolower(matched.cases$storageheatertype)
)

if (use.sedbuk) {
```

```
    matched.heating <- rbind(matched.sedbuk,matched.cases)
    ##collect additional info on storage combi boilers from the sedbuk data (storage
    ##combis can only be identified through sedbuk information)
    sedbuk<-subset(sedbuk,select=c("aacode","storeboilervolume"
                                   ,"storesolarvolume","storeinsulationthickness"))
    ##change storage combi boiler variables to match DTO nomenclature
    setnames(sedbuk, c("aacode", "storagecombicylindervolume",
                       "storagecombisolarvolume","storagecombicylinderinsulationthickness"))
    ## join additional storage combi boilers to the combined heating system match data
    matched.heating <- join(matched.heating,sedbuk,by="aacode")
} else {
    matched.heating <- matched.cases
    matched.heating$storagecombicylindervolume <- NA
    matched.heating$storagecombisolarvolume <- NA
    matched.heating$storagecombicylinderinsulationthickness <- NA
}

##additional information for other heating system types, plus boiler installation
##year and secondary heating information (using additional functions defined below)
allHeatingSysteams <- data.table(
    aacode = allEntries$aacode
   ,electrictariff = electricity.tariff(allEntries$Finmhfue)
   ,heatingsystemcontroltypes = heating.control.types(allEntries$Finchtim,
                                                      allEntries$Finchrom,
                                                      allEntries$Finchtrv,
                                                      allEntries$Finchtzc,
                                                      allEntries$Finchdst)
   ,installationyear = installation.year(rep("NOT", length(allEntries$aacode)),
                                         allEntries$Finwhxag,
                                         allEntries$Finchbag,
                                         allEntries$fodconac)
   ,secondaryheatingsystemtype = secondary.heating(allEntries$Finoheat,
                                                   allEntries$Finohtyp)
 )

#add additional data to main matched.heating table
matched.heating <- join(matched.heating,allHeatingSysteams,by="aacode")

#Add details on community heating
allHeatingSysteams <- data.table(
  aacode = matched.heating$aacode
  ,chpfraction = ifelse(matched.heating$spaceheatingsystemtype ==
                          "community_heating_with_chp",as.character(0.35), "")
  ,communitychargingusagebased =
    ifelse(matched.heating$spaceheatingsystemtype == "community_heating_with_chp" |
      matched.heating$spaceheatingsystemtype == "community_heating_without_chp", TRUE,
      FALSE)
)
matched.heating <- join(matched.heating,allHeatingSysteams,by="aacode")

#Add storage heater control types
storageHeaterSurveyVars <- subset(allEntries,select = c(aacode, Finchctc, Finchacc))
allHeatingSysteams <- join(matched.heating, storageHeaterSurveyVars, by = "aacode")
```

```
  matched.heating <- join(matched.heating,
                          storage.heater.control.type(allHeatingSysteams),by="aacode")

  #convert heating efficiencies to proportions from percentages
  matched.heating$basicefficiency <- matched.heating$basicefficiency / 100
  matched.heating$winterefficiency <- matched.heating$winterefficiency / 100
  matched.heating$summerefficiency <- matched.heating$summerefficiency / 100

  #print success message
  print(paste("spaceheating DTO complete; number of records: ", nrow(matched.heating)))
  ## Return the compiled data
  return (matched.heating)
}
```

Sedbuk boilers: rename

Converts the boiler type returned from the sedbuk match to a space heating systemt typr

@param boilerType - one of CPSU | INSTANT_COMBI | STORAGE_COMBI | REGULAR

```
sedbuk.boiler.to.spaceheating.type <- function(boilerType){
  as.factor(checked.revalue(
    boilerType,c(
      "CPSU" = "cpsu"
      ,INSTANT_COMBI = "combi"
      ,STORAGE_COMBI = "storage_combi"
      ,REGULAR =  "standard")))}
```

Heating system installation year

@param spaceHeatingType @param backBoilerAge - Finwhxag @param boilerAge - Finchbag @param buildYear - fodconac

```
installation.year <- function(spaceHeatingType, backBoilerAge, boilerAge, buildYear){
  isBackBoiler <- spaceHeatingType == "BACK_BOILER" | spaceHeatingType ==
                                                      "BACK_BOILER_NO_CENTRAL_HEATING"

  calcBoilerAge <- ifelse(isBackBoiler, age.or.buildYear(backBoilerAge, buildYear),
                                                                        "NULL")
  calcBoilerAge <- ifelse(calcBoilerAge == "NULL",
                          age.or.buildYear(boilerAge, buildYear), calcBoilerAge)

  return (calcBoilerAge)
}
```

Returns either the survey year 2012 - age of boiler or if age of boiler is 88 or na then build year of the property

@param age - of boiler @param buildYear - of house case

```
age.or.buildYear <- function(age, buildYear){
  boilerAge <- ifelse(is.na(age) | age == 88, buildYear, 2012 - age)
  return(boilerAge)
}
```

Electricity tariff

@param Finmhfue - main heating system fuel

```
electricity.tariff <- function(Finmhfue){
  as.factor(checked.revalue(
    Finmhfue,c(
      "Gas - Mains" = "NULL",
      "Gas - Bulk/LPG" = "NULL",
      "Gas - bottled" = "NULL",
      "Oil" = "NULL",
      "Solid fuel - coal" = "NULL",
      "Solid fuel - smokeless fuel" = "NULL",
      "Solid fuel - anthracite" = "NULL",
      "Solid fuel - wood" = "NULL",
      "Electricity - standard" = "FLAT_RATE",
      "Electricity - 7hr tariff" = "ECONOMY_7",
      "Electricity - 10hr tariff" = "ECONOMY_10",
      "Electricity - 24hr tariff" = "TWENTYFOUR_HOUR_HEATING",
      "Communal - CHP/Waste heat" = "NULL",
      "Communal - from boiler" = "NULL"
    )))}
```

Heating system controls

Returns a delimeted list (";") of control types present in house case

@param Finchtim - Progammer present @param Finchrom - Room Thermostat present @param Finchtrv - TRV present @param Finchtzc - Time temperature zone control present @param Finchdst - Delaye start thermostat

```
heating.control.types <- function(Finchtim, Finchrom, Finchtrv, Finchtzc, Finchdst){
  is.yes <- function(X, label) ifelse(is.na(X),"", ifelse(X == "Yes", label, ""))
  remove.blanks <- function(X)
          str_replace_all(
            str_replace_all(
              str_replace_all(
                str_replace_all(X, ";;;" ,";"), ";;", ";"), "^;", ""), ";$", "")
  controls <-
    remove.blanks(
    paste(
      sep=";",
      is.yes(Finchtim, "programmer"),
      is.yes(Finchrom, "room_thermostat"),
      is.yes(Finchtrv, "thermostaticradiatorvalve"),
      is.yes(Finchtzc, "timetemperaturezonecontrol"),
      is.yes(Finchdst, "delayed_start_thermostat")
      )
    )


  return (controls)
}
```

Storage heater controls

With only heating systems that are storage heaters return a data frame where each row has a value for storageheatercontroltype

@param allHeatingSysteams - data.frame of columns: spaceheatingsystemtype | Finchacc | Finchctc

```
storage.heater.control.type <- function(allHeatingSysteams){
  onlyStorageHeaters <- subset(allHeatingSysteams,
                               spaceheatingsystemtype == "storage_heater")
  celectChargeControl <- subset(onlyStorageHeaters, Finchctc == "Yes")

  others <- subset(onlyStorageHeaters,
                   !onlyStorageHeaters$aacode %in% celectChargeControl$aacode)
  automaticChargeControl <- subset(others, Finchacc == "Yes")
  manualChargeControl <- subset(others,
                                !others$aacode %in% automaticChargeControl$aacode)

  celectChargeControl <- data.table(
    aacode = celectChargeControl$aacode,
    storageheatercontroltype = rep("CELECT_CHARGE_CONTROL",
                                   length(celectChargeControl$aacode)))

  automaticChargeControl <- data.table(
    aacode = automaticChargeControl$aacode,
    storageheatercontroltype = rep("AUTOMATIC_CHARGE_CONTROL",
                                   length(automaticChargeControl$aacode)))

  manualChargeControl <- data.table(
    aacode = manualChargeControl$aacode,
    storageheatercontroltype = rep("MANUAL_CHARGE_CONTROL",
                                   length(manualChargeControl$aacode)))

  storageHeaterControls <- rbind(celectChargeControl, automaticChargeControl)
  storageHeaterControls <- rbind(storageHeaterControls, manualChargeControl)

  return (storageHeaterControls)
}
```

Storage combination boiler information

If the boiler type is STORAGE_COMBI then Sets cylinder and boiler store volumes including solar store and insulatin thickness of combi cylinder. Otherwise set's values to null.

@param

```
storage.combi <- function(sedbuk.match){
  isStorage <- matches$boilertype == "STORAGE_COMBI"

  details <- data.table(
    aacode =  matches$aacode
    ,storagecombicylindervolume = ifelse(isStorage, matches$storeboilervolume, "NULL")
    ,storagecombisolarvolume = ifelse(isStorage, matches$storesolarvolume, "NULL")
    ,storagecombicylinderinsulationthickness = ifelse(isStorage,
```

```
                                                    matches$storeinsulationthickness)
    ,isstoragecombicylinderthemostatpresent = "NULL"
    ,isstoragecombicylinderfactoryinsulated = "NULL"
  )

  return (details)
}
```

Secondary heating systems

Returns secondary heating system type

@param FINOHEAT - Is secondary heating type present @param FINOHTYP - Other Heating Type of System

```
secondary.heating <- function(FINOHEAT, FINOHTYP){
  isPresent = ifelse(FINOHEAT == "Yes", TRUE, FALSE)

  return (secondaryheatingsystemtype = ifelse(isPresent,
                                              secondary.heating.type(FINOHTYP),
                                              "NO_SECONDARY_SYSTEM"))
}
```

Returns a character string describing the type of secondary heating

@note Does not accept NA values @paran FINOHTYP - Other Heating Type of System

```
secondary.heating.type <- function(FINOHTYP){
  type <- (as.character(levels(
    secondary.heating.type.lookup(FINOHTYP))[secondary.heating.type.lookup(FINOHTYP)]))
  return(type)
}
```

Simple conversion from Survey value to NHM value for secondary heating type @param FINOHTYP - Other Heating Type of System

```
secondary.heating.type.lookup <- function(FINOHTYP){
  as.factor(checked.revalue(
    FINOHTYP,c(
      "Mains gas - open flue" = "GAS_FIRE_OPEN_FLUE"
      ,"Mains gas - balances flue" = "GAS_FIRE"
      ,"Mains gas - fan assisted"  = "GAS_FIRE"
      ,"Mains gas - condensing" =  "GAS_FIRE"
      ,"Mains gas - live effect - sealed to chim" = "GAS_COAL_EFFECT_FIRE"
      ,"Mains gas - live effect - fan assisted f" = "GAS_COAL_EFFECT_FIRE"
      ,"Mains gas - decorative - open to chimney" = "GAS_COAL_EFFECT_FIRE"
      ,"Mains gas - flueless" = "GAS_FIRE_FLUELESS"
      ,"Mains gas - unknown" = "GAS_FIRE_FLUELESS"
      ,"LPG - fixed heaters" = "GAS_FIRE"
      ,"Electric heaters - panel/convector or ra" = "ELECTRIC_ROOM_HEATERS"
      ,"Electric heaters - portable" = "ELECTRIC_ROOM_HEATERS"
      ,"Electric heaters - individual storage he" = "ELECTRIC_ROOM_HEATERS"
      ,"Solid fuel heaters - open fire" = "OPEN_FIRE"
```

```
      ,"Solid fuel heaters - stove/space heater" = "OPEN_FIRE"
      ,"Paraffin - portable heaters" = "GAS_FIRE"
      ,"Other" = "GAS_FIRE"
    )))
}
```

Sedbuk boiler matching

Pre-processing of variables

Creates a data.table that has the required inputs needed to search the sedbuk index for a boiler match

@return data.table with columns aacode, make, model, fuel, flue, type

```
sedbuk.look.up.table <- function(allEntries){
  look.up.table <- data.table(
    aacode = allEntries$aacode
    ,make =  as.character(allEntries$Finchbma)
    ,model = as.character(allEntries$Finchbmo)
    ,fuel =  sedbuk.fuelType(allEntries$Finmhfue)
    ,flue = rep("", length(allEntries$aacode))
    ,type = sedbuk.boiler.type(allEntries$Finmhboi)
  )

  return (look.up.table)
}
```

Rename of boiler systems returned from sedbuk matching

type - REGULAR | INSTANT_COMBI | STORAGE_COMBI | CPSU | UNKNOWN |

```
sedbuk.boiler.type <- function(Finmhboi){
  as.factor(checked.revalue(
    Finmhboi,c(
      "Standard" = "REGULAR",
      "Back boiler" = "UNKNOWN",
      "Combination" = "INSTANT_COMBI",
      "Condensing" = "REGULAR",
      "Condensing combi" = "INSTANT_COMBI",
      "Combined primary storage unit" = "CPSU",
      "No boiler" = "EMPTY",
      "Unknown" = "UNKNOWN"
    )))
}
```

Main heating system fuel type

Converts from EHS fuel type to a variable the sedbuk boiler match recoginises, all other fuel types are set to NULL

@param Finmhfue - main heating system fuel

```
sedbuk.fuelType <- function(Finmhfue){
  as.factor(checked.revalue(
    Finmhfue,c(
      "Gas - Mains" = "MAINS_GAS",
      "Gas - Bulk/LPG" = "BULK_LPG",
      "Gas - bottled" = "BULK_LPG",
      "Oil" = "OIL",
      "Solid fuel - coal" = "NULL",
      "Solid fuel - smokeless fuel" = "NULL",
      "Solid fuel - anthracite" = "NULL",
      "Solid fuel - wood" = "NULL",
      "Electricity - standard" = "NULL",
      "Electricity - 7hr tariff" = "NULL",
      "Electricity - 10hr tariff" = "NULL",
      "Electricity - 24hr tariff" = "NULL",
      "Communal - CHP/Waste heat" = "NULL",
      "Communal - from boiler" = "NULL"
    )))
}

nhm.fuelType <- function(Finmhfue){
  as.factor(checked.revalue(
    Finmhfue,c(
      "Gas - Mains" = "MAINS_GAS",
      "Gas - Bulk/LPG" = "BULK_LPG",
      "Gas - bottled" = "BOTTLED_LPG",
      "Oil" = "OIL",
      "Solid fuel - coal" = "HOUSE_COAL",
      "Solid fuel - smokeless fuel" = "HOUSE_COAL",
      "Solid fuel - anthracite" = "HOUSE_COAL",
      "Solid fuel - wood" = "BIOMASS_WOOD",
      "Electricity - standard" = "ELECTRICITY",
      "Electricity - 7hr tariff" = "ELECTRICITY",
      "Electricity - 10hr tariff" = "ELECTRICITY",
      "Electricity - 24hr tariff" = "ELECTRICITY",
      "Communal - CHP/Waste heat" = "MAINS_GAS",
      "Communal - from boiler" = "MAINS_GAS"
    )))
}
```

Boiler matching: SAP lookups

This function imports data from heating look up tables

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

@param path.to.ehcs - the path to the ehcs data (used to find boiler lookup tables)

@param unmatched - the wide SPSS data frame containing only those cases which we have left to match

@param name.to.heatinglookup - the filename of the lookup table to use. this file should have the matching columns on the left

```
heating.matching <- function(path.to.ehcs,
                             unmatched,
                             name.to.heatinglookup) {
    boiler <- read.csv(file.path(path.to.ehcs,"boilerLookUps", name.to.heatinglookup),
                       header=TRUE)

    ## we match on the intersection of columns in the lookup table
    ## and columns in the raw data.
    match.columns <- colnames(boiler)
    match.columns <- match.columns[match.columns %in% colnames(unmatched)]

    ## boiler table may be missing some specifics, in which case we need some NAs
    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
    }

    ## reduce the unmatched data down to only the columns we need (makes the join faster)
    unmatched <- unmatched[, c("aacode", match.columns)]

    ## do the join - we want an inner join (i.e. only rows that are in both sets)
    matched <- join(unmatched,boiler, by=match.columns, type="inner")

    ## finally the results - we want only to keep certain columns
    matched[, c("aacode", "winterefficiency", "summerefficiency"
                ,"basicefficiency", "spaceheatingsystemtype", "fluetype"
                ,"mainheatingfuel", "iscondensing", "installationyear"
                ,"storageheatertype")]
}
```

Waterheating

The following section of code is generated in waterheating.R

Make 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 allEntries - combined sav files data.frame from the EHS data

@param - spaceHeatingDTO - the space heating dto file created in a previous script

@param - path.to.ehcs - the file path to the ehcs survey data

```
make.waterheating <- function(allEntries, spaceHeatingDTO, path.to.ehcs){
  # Make entries for those that should be with central heating
  waterHeatingWithCentralHeating <- data.table(subset(allEntries
                                                      , watersys == "with central heating"))
  allWaterHeating <- data.table(
    aacode = waterHeatingWithCentralHeating$aacode
    ,withcentralheating = rep(TRUE, length(waterHeatingWithCentralHeating$aacode))
    ,basicefficiency = rep(0, nrow(waterHeatingWithCentralHeating))
  )

  # Create efficiencies and system type information for those that don't use Central
  # Heating
  withCentral.id <-data.frame(aacode=allWaterHeating$aacode)
  matched.cases <- water.system.matching(allEntries,path.to.ehcs,withCentral.id,
                        "water-heating-singlepoint-lup.csv",c("watersys","Finwspty"))
  matched.cases <- rbind(matched.cases,water.system.matching(allEntries,path.to.ehcs
                    ,rbind(withCentral.id,data.frame(aacode=matched.cases$aacode)),
                    "water-heating-single-immrsn-lup.csv",c("watersys","Finwsity")))
  matched.cases <- rbind(matched.cases,water.system.matching(allEntries,path.to.ehcs
                     ,rbind(withCentral.id,data.frame(aacode=matched.cases$aacode)),
                     "water-heating-community-lup.csv",c("watersys","Finwhlty")))
  matched.cases <- rbind(matched.cases,water.system.matching(allEntries,path.to.ehcs
                     ,rbind(withCentral.id,data.frame(aacode=matched.cases$aacode)),
                     "water-heating-other-boiler-lup.csv",c("watersys","Finwhoty")))
  matched.cases <- rbind(matched.cases,water.system.matching(allEntries,path.to.ehcs
                     ,rbind(withCentral.id,data.frame(aacode=matched.cases$aacode)),
                     "water-heating-multipoint-lup.csv",c("watersys","Finwmpty")))
  matched.cases <- rbind(matched.cases,water.system.matching(allEntries,path.to.ehcs
                     ,rbind(withCentral.id,data.frame(aacode=matched.cases$aacode)),
                     "water-heating-double-immrsn-lup.csv",c("watersys","Finwdity")))
```

```
matched.cases <- rbind(matched.cases,water.system.matching(allEntries,path.to.ehcs
                   ,rbind(withCentral.id,data.frame(aacode=matched.cases$aacode)),
                   "water-heating-comm-boiler-lup.csv",c("watersys","Finwhlty")))
matched.cases <- rbind(matched.cases,water.system.matching(allEntries,path.to.ehcs
                   ,rbind(withCentral.id,data.frame(aacode=matched.cases$aacode)),
                   "water-heating-back-boiler-lup.csv",c("watersys","Finwhxty")))
matched.cases <- rbind(matched.cases,water.system.matching(allEntries,path.to.ehcs
                    ,rbind(withCentral.id,data.frame(aacode=matched.cases$aacode)),
                    "water-heating-missing-data-lup.csv",c("watersys")))
# Set flag for with central heating to false for all these cases
matched.cases$withcentralheating <- rep(FALSE, nrow(matched.cases))
allWaterHeating <- rbind(allWaterHeating, matched.cases, fill = TRUE)

# Add Cylinders
cylinderProperties <- cylinder.properties(allEntries)
allWaterHeating <- join(allWaterHeating, cylinderProperties, by = "aacode")

# Add Immersion Heaters
immersionHeaters <- immersion.heater(allEntries)
allWaterHeating <- join(allWaterHeating, immersionHeaters, by = "aacode")

# Solar Hot Water
hasSolarHotWater <- subset(allEntries, Finwotfu == 15)

noSolarHotWater <- subset(allWaterHeating
                          , !allWaterHeating$aacode %in% hasSolarHotWater$aacode)
noSolarHotWater <- data.table(
  aacode = noSolarHotWater$aacode
  ,solarhotwaterpresent = rep(FALSE, nrow(noSolarHotWater))
  ,solarstoreincylinder = rep(FALSE, nrow(noSolarHotWater))
  ,solarstorevolume =  rep(as.factor(0), nrow(noSolarHotWater))
)

# Uses main hot water cylinder
solarstoreincylinder <- subset(allWaterHeating,
allWaterHeating$aacode %in% hasSolarHotWater$aacode & !allWaterHeating$volume == 0)
solarstoreincylinder <- data.table(
  aacode = solarstoreincylinder$aacode
  ,solarhotwaterpresent = rep(TRUE, nrow(solarstoreincylinder))
  ,solarstoreincylinder = rep(TRUE, length(solarstoreincylinder$aacode))
  ,solarstorevolume =  solarstoreincylinder$cylindervolume
)

# Uses additional hot water cylinder - TODO Slight descrepency between num of has
# SolarHotWater and sum of solarstoreincylinder and solarStoreInAdditionalCylinder
solarStoreInAdditionalCylinder <- subset(allWaterHeating,
                       allWaterHeating$aacode %in% hasSolarHotWater$aacode
                       & !allWaterHeating$aacode %in% solarstoreincylinder$aacode)

solarStoreInAdditionalCylinder <- data.table(
  aacode = solarStoreInAdditionalCylinder$aacode
  ,solarhotwaterpresent = rep(TRUE, nrow(solarStoreInAdditionalCylinder))
  ,solarstoreincylinder = rep(FALSE, length(solarStoreInAdditionalCylinder$aacode))
```

```
    ,solarstorevolume =  rep(as.factor(75),
                             length(solarStoreInAdditionalCylinder$aacode))
  )

  solarStoreCylinders <- rbind(noSolarHotWater, solarstoreincylinder,
                               solarStoreInAdditionalCylinder)
  allWaterHeating <- join(allWaterHeating, solarStoreCylinders, by = "aacode")

  #Add final columsn for unknown values
  allWaterHeating$installationyear <- rep("", nrow(allWaterHeating))
  allWaterHeating$chpfraction <- rep("", nrow(allWaterHeating))
  allWaterHeating$communitychargingusagebased <- rep("", nrow(allWaterHeating))
  allWaterHeating$electrictariff <- rep("", nrow(allWaterHeating))

  #Return the compiled data
  allWaterHeating$cylindervolume <- revalue(allWaterHeating$cylindervolume,
                                            c("0" = ""))

  allWaterHeating$basicefficiency <- allWaterHeating$basicefficiency / 100
  allWaterHeating$winterefficiency <- allWaterHeating$winterefficiency / 100
  allWaterHeating$summerefficiency <- allWaterHeating$summerefficiency / 100

  print(paste("water heating DTO complete; number of records: ", nrow(allWaterHeating)))
  return (allWaterHeating)
}
```

Immersion heater system type

```
immersion.heater <- function(allEntries){
  DUAL_COIL <- subset(allEntries, allEntries$Finwdipr == "Yes")
  DUAL_COIL <- data.table(
     aacode = DUAL_COIL$aacode,
     immersionheatertype = "DUAL_COIL"
    )

  # There are some entries where both DUAL_COIL and SINGLE_COIL is yes so exclude
  # those from SINGLE_COIL
  SINGLE_COIL <- subset(allEntries,
                        allEntries$Finwsipr == "Yes" & allEntries$Finwdipr == "No")
  SINGLE_COIL <- data.table(
    aacode = SINGLE_COIL$aacode,
    immersionheatertype = "SINGLE_COIL"
  )

  heaters <- rbind(SINGLE_COIL, DUAL_COIL)

  return(heaters)
}
```

Hot water cylinders

Hot water cylinder volume

If Cylinder is present and has volume then return converted volume If cylinder is present but volume is unknown the set as default 110 otherwise set volume to 0

@param allEntries - combined sav files data.frame from the EHS data

```
cylinder.properties <- function(allEntries){
  knownCylinderVolumes <- subset(allEntries, is.na(Finwhsiz) == FALSE, c(aacode,
                                                                         Finwhsiz))

  knownCylinderVolumes <- data.table(
    aacode = knownCylinderVolumes$aacode
    ,cylindervolume = cylinder.volume(knownCylinderVolumes$Finwhsiz)
    )

  unKnownCylinderVolumes <- subset(allEntries,
                                   !allEntries$aacode %in% knownCylinderVolumes$aacode
                                     & allEntries$Finwhcyl == "Yes")
  unKnownCylinderVolumes <- data.table(
    aacode = unKnownCylinderVolumes$aacode
    ,cylindervolume = as.factor(110)
  )

  cylinderProperties <- rbind(knownCylinderVolumes, unKnownCylinderVolumes)

  noCylinderInformation <- subset(allEntries,
                                  !allEntries$aacode %in% cylinderProperties$aacode)
  noCylinderInformation <- data.table(
    aacode = noCylinderInformation$aacode
    ,cylindervolume = as.factor(0)
  )

  cylinderProperties<- rbind(cylinderProperties, noCylinderInformation)
  cylinderProperties$cylindervolume < revalue(cylinderProperties$cylindervolume,
                                                                      c("0" = ""))

  #InsulatiOn cylinder thickness and type
  isFactoryInsulated <- subset(allEntries, allEntries$Finwhins == "Foam")
  hasThermostat <- subset(allEntries, allEntries$Finwhthe == "Yes")

  cylinderThickness <- data.table(
    aacode = allEntries$aacode
    ,cylinderinsulationthickness = cylinder.thickness(allEntries$Finwhmms)
    ,cylinderfactoryinsulated = allEntries$aacode %in% isFactoryInsulated$aacode
    ,cylinderthermostatpresent = allEntries$aacode %in% hasThermostat$aacode
  )

  cylinderProperties <- join(cylinderProperties, cylinderThickness, by = "aacode")

  return(cylinderProperties)
}
```

Hot water cylinder insulation thickness

Converts survey string value for cylinder insulation thickness into a numeric factor

@param Finwhmms - cylinder insulation thickness

```
cylinder.thickness <- function(Finwhmms){
  as.factor(checked.revalue(
    Finwhmms,c(
      "0" = 0,
      "12.5mm" = 12.5,
      "25 mm" = 25,
      "38 mm" = 38,
      "50 mm" = 50,
      "80 mm" = 80,
      "100 mm" = 100,
      "150 mm" = 150
    )))}
```

Hot water cylinder volume

```
cylinder.volume <- function(Finwhsiz){
  as.factor(checked.revalue(
    Finwhsiz,c(
      "450 x 900mm (110 l)" = 110,
      "450 x 1050mm (140 l)" = 140,
      "450 x 1500mm (210 l)" = 210,
      "450 X 1650mm (245 l)" = 245
    )))}
```

Matching independant hot water systems

This function uses a lookup table to determine the hot water heating system where the survey specifies that it is not from mains heating (variable = watersys)

hot water heating system is obtain from a lookup table created using EHS variables and data contained in SAP table 4a tables on heating and hot water systems

```
water.system.matching <- function(allEntries,path.to.ehcs,
                                  matched.id,
                                  name.to.heatinglookup,
                                  columns.to.match) {
  boiler <- read.csv(file.path(path.to.ehcs,"boilerLookUps", name.to.heatinglookup),
                     header=TRUE)

  unmatched <- subset(allEntries,!allEntries$aacode %in% matched.id$aacode
                      ,select=c("aacode",columns.to.match))

  unmatched<-join(unmatched,boiler,by=columns.to.match)

  matched<-subset(unmatched,is.na(unmatched$Heating.System)==FALSE
                  ,select=c(aacode, winterefficiency, summerefficiency
```

```
                            ,basicefficiency, waterheatingsystemtype, fluetype
                            ,mainheatingfuel))

  return(matched)
}
```

Storeys

The following section of code is generated in stories.R

Functions for making polygons

Each storey that is produced has a floorplan, which is defined by a polygon. Ultimately, the polygon is represented as a list of x-coordinates and a list of y-coordinates, but as an intermediate form we use a 2xN matrix where the first column is called xs and the second ys. This is easier to work with. All floorplans in the EHS are composed of one or two rectangles; the first rectangle is the "main module" and the second is the "additional module". The additional module is joined to the main module in one of 12 positions. These 12 positions can be produced by rotations and mirrorings of two canonical positions, which is how it is done here: ## Flip a polygon in the x-axis.

For example, if you had a floor with an additional module on the back elevation, and you wanted one on the front elevation instead, you could flip it on the x-axis (this is the back-to-front axis).

```
poly.flipx <- function(a) {
    xs <- a[, 'xs']
    cbind(xs=-xs + max(xs),
          ys=a[, 'ys'])
}
```

Flip a polygon in the y-axis

Analogous to poly.flipx above, but for y.

```
poly.flipy <- function(a) {
    ys <- a[, 'ys']
    cbind(xs=a[, 'xs'],
          ys=-ys + max(ys))
}
```

Reflect a polygon in y=x

If you had a floor with a module on the front or back elevation, and you wanted it on the bottom or top elevation, you could reflect it in y=x; this is like rotating it 90 degrees clockwise.

```
poly.mirror <- function(a) cbind(xs=a[, 'ys'],ys=a[, 'xs'])
```

Make a rectangular polygon

This makes a polygon with 4 points, having the given depth (x-dimension) and width (y-dimension)

The polygon starts at 0,0

@param md the depth (x extent) @param mw the width (y extent)

```
poly.rect <- function(md, mw)
    cbind(xs=c(0, md, md, 0),
          ys=c(0, 0,  mw, mw))
```

Make a polygon for a house with an additional module on the left of the front elevation.

This is one of the two canonical types of two-module house. It's a bit like an L-shape:

```
      MD                AD
+-----------------+--------+
|                 |        |
```

MW | M | A | AW | +——–+ | | +——————+

@param md the main module's depth @param mw the main module's width. @param ad the additional module's depth @param aw the additional module's width If this equals MW, the result is just a rectangle.

```
poly.leftmodule <- function(md, mw, ad, aw)
{
    if      (ad == 0 | aw == 0) {poly.rect(md, mw)} # module is empty
    else if (md == 0 | mw == 0) {poly.rect(ad, aw)} # main is empty
    else if (aw == mw)          {poly.rect(md + ad, mw)} # module is as wide as main
    else                        {cbind(xs=c(0, md + ad, md + ad, md, md, 0 ),
                                       ys=c(0, 0      , aw     , aw, mw, mw))}
}
```

Make a polygon for a house with an additional module in the middle of the front elevation.

This is the other canonical type of two module house; it's a bit like a T-shape: MD +——————+ | | AD | +——-+ M| | A | AW W| M | | | +——-+ | | +——————+

The horizontal centre line of the additional module is the horizontal centre line of the main module

@param md main module depth @param mw main width @param ad additional depth @param aw additional width. If this equals MW, the result is just a rectangle.

```
poly.midmodule <- function(md, mw, ad, aw)
{
    if      (ad == 0 | aw == 0) {poly.rect(md, mw)} # module is empty
    else if (md == 0 | mw == 0) {poly.rect(ad, aw)} # main is empty
    else if (aw == mw)          {poly.rect(md + ad, mw)} # module is as wide as main
    else                        {cbind(xs=c(0, md, md           , md + ad      , md + ad    , md
                                       ys=c(0, 0 , (mw - aw) / 2, (mw - aw) / 2, (mw + aw)/2, (mw + aw),
}
```

Make a polygon for an arbitrary EHS house

Create a polygon describing a floor, based on the main an additional module dimensions, and the additional module location.

This is done by flipping and rotating either poly.leftmodule or poly.midmodule. Note that whenever a rotation is used (poly.mirror), the depth and width arguments to the polygon that's been mirrored are swapped. This is because after the rotation, depth is width and vice-versa.

@param md main module depth @param mw main module width @param ad additional module depth @param aw additional module width @param a.location additional module location

```
floor.polygon <- function(md, mw,
                          ad, aw,
                          a.location)
    switch(as.character(a.location),
           "Back elevation: Left"    = poly.leftmodule(md, mw, ad, aw),
           "Back elevation: Centre"  = poly.midmodule(md, mw, ad, aw),
           "Back elevation: Right"   = poly.flipy(poly.leftmodule(md, mw, ad, aw)),
           "Front elevation: Left"   = poly.flipx(poly.leftmodule(md, mw, ad, aw)),
           "Front elevation: Centre" = poly.flipx(poly.midmodule(md, mw, ad, aw)),
           "Front elevation: Right"  = poly.flipx(poly.flipy(poly.leftmodule(md, mw, ad, aw))),
           "Left elevation: Back"    = poly.flipx(poly.mirror(poly.flipx(poly.leftmodule(mw, md, aw, ad
           "Left elevation: Centre"  = poly.flipx(poly.mirror(poly.flipx(poly.midmodule(mw, md, aw, ad)
           "Left elevation: Front"   = poly.mirror(poly.flipx(poly.leftmodule(mw, md, aw, ad))),
           "Right elevation: Back"   = poly.flipx(poly.mirror(poly.leftmodule(mw, md, aw, ad))),
           "Right elevation: Centre" = poly.mirror(poly.midmodule(mw, md, aw, ad)),
           "Right elevation: Front"  = poly.mirror(poly.leftmodule(mw, md, aw, ad)),
           "No additional part"      = poly.rect(md, mw),
           "Unknown"                 = poly.rect(md, mw),
           poly.rect(md, mw))
```

Produce an output string from a vector of coordinates

This is simple enough; coordinate.string(c(1,2,3)) is "{ 1, 2, 3 }"

```
coordinate.string <- function(vec) paste("{", paste(round(vec), collapse=","),"}")
```

Make a DTO storey type from a floor number

When building the storeys, we use a number to keep track of what floor we are on. The NHM wants to see a floor type, so we code the floor index up as a type.

The convention is -1 for basement and 0 for ground.

```
floor.number.to.type <- function(n) {
    if (n == -1) {
        "BASEMENT"
    } else if (n == 0) {
        "GROUND"
    } else if (n == 1) {
        "FIRST_FLOOR"
    } else if (n == 2) {
        "SECOND_FLOOR"
    } else {
        "HIGHER_FLOOR"
    }
}
```

Make a single floor (just the polygon and type)

This function produces a single floor of a flat or a house, in a DTO form in a data frame. It also throws in the area, just in case.

@param floor - which floor we are looking at. This is always a number from 1 to N, representing which floor this is out of the total we are making. It's not which actual floor of the building we are on. @param use.flatdets - if TRUE, we will try and find the dimensions in the flatdets SPSS file. Otherwise, they come from the physical dimensions file. @param frame - A frame containing all the SPSS variables about one single house. Variables should have lowercase names.

```
make.one.floor <- function(floor, # floor number, from 1 to n
                           use.flatdets,
                           start.floor, # position of bottom floor (-1 for basement, 0 for 0, etc.)
                           frame) {
    if (use.flatdets) { # in a flat, we often use these variables:
        if (floor == 1) {
            maindepth <- frame$fdfmaind
            mainwidth <- frame$fdfmainw
            adddepth <- 0
            addwidth <- 0
        } else {
            maindepth <- frame$fdfnextd
            mainwidth <- frame$fdfnextw
            adddepth <- 0
            addwidth <- 0
        }
    } else { # in a house, we use the module information
        maindepth <- 0
        mainwidth <- 0
        adddepth <- 0
        addwidth <- 0

        ## we use the highest numbered set of floor data
        ## which (a) is good for this floor (i.e. not too high)
        ## and   (b) has a defined main module size.

        if (floor > 0 &
            !is.na(frame$fdhmdep1) &
            !is.na(frame$fdhmwid1)   ) {
            maindepth <- frame$fdhmdep1
            mainwidth <- frame$fdhmwid1
            adddepth  <- frame$fdhadep1
            addwidth  <- frame$fdhawid1
        }

        if (floor > 1  &
            !is.na(frame$fdhmdep2) &
            !is.na(frame$fdhmwid2)   ) {
            maindepth <- frame$fdhmdep2
            mainwidth <- frame$fdhmwid2
            adddepth  <- frame$fdhadep2
            addwidth  <- frame$fdhawid2
        }

        if (floor > 2 &
            !is.na(frame$fdhmdep3) &
            !is.na(frame$fdhmwid3)   ) {
            maindepth <- frame$fdhmdep3
```

```
            mainwidth <- frame$fdhmwid3
            adddepth  <- frame$fdhadep3
            addwidth  <- frame$fdhawid3
        }
    }

    ## cook up the polygon for the floor (times 100 as we are in cm)
    poly <- floor.polygon(maindepth*100, mainwidth*100,
                          adddepth*100, addwidth*100,
                          ## this the additional module location -
                          ## if the additional module data is missing we supply
                          ## "", which causes no additional module to be built.
                          ## otherwise fshaddit.
                          if (use.flatdets | is.na(adddepth) | is.na(addwidth)) ""
                          else frame$fshaddit)

    polyarea <- areapl(poly)

    data.frame(area=polyarea,
               type=floor.number.to.type(floor - 1 + start.floor),
               polygonxpoints=coordinate.string(poly[, 'xs']),
               polygonypoints=coordinate.string(poly[, 'ys']),
               polypoints=length(poly[, 1]))
}
```

Should we start with a basement storey?

This is true if basement is 'yes', or if fdhmlev1 is bb or -1, or dwtype3x contains 'basement'

```
is.basement <- function(frame)
    frame$basement == "yes" | # sometimes the basement field is yes
        tolower(frame$fdhmlev1) %in% c("bb", "-1") | ## sometimes there is bb in the coding
        grepl("basement", frame$dwtype3x, ignore.case=TRUE) # and sometimes it says basement in the nam
```

Now we're getting somewhere: make the DTO data for one house

@param frame - a data frame containing the required variables and a single row which is just one aacode.

@param floors.total - the total number of floors to make @param start.floor - the index of the starting floor; -1 for basement, 0 for ground, 1 for first...

@return a data frame containing several rows, one for each storey of the house

```
one.house.storeys <- function(frame, floors.total, start.floor) {
    ## this is just saying: for each integer from 1 to floors.total
    ## call make.one.floor(N, FALSE, start.floor, frame)
    ## and stick the results together in one frame.
    result <- do.call(rbind,
                      lapply(1:floors.total, make.one.floor,
                             FALSE, start.floor, frame))

    ## we need the top floor area in case there is a room in roof
```

```
    ## as the room in roof is modelled as a simple square on the top
    top.area <- result$area[length(result$area)]

    ## now we kill that columns as we don't want it to come out in the
    ## DTO.
    result$area <- NULL

    ## make a storey for the room in roof, if there is one!
    if ( tolower(frame$attic) == "yes" ) {
        ## room in roof area is top floor area / 2
        side <- sqrt(top.area / 2)
        poly <- poly.rect(side, side)
        ## could be a little neater, but there we go
        result <- rbind(result,
                        data.frame(
                            type="ROOM_IN_ROOF",
                            polygonxpoints=coordinate.string(poly[, 'xs']),
                            polygonypoints=coordinate.string(poly[, 'ys']),
                            polypoints=length(poly[, 1])))
    }

    ## and we are done
    result
}
```

The action hots up: make the storeys for a flat

This is the analog of one.house.storeys but for flats.

@param frame - a data frame with one row in, which is the SPSS data for one flat. @return a data frame for the storeys in that flat

```
one.flat.storeys <- function(frame) {
    ## the fdffloor variable tells us how many floors are in the flat.
    ## sometimes it's NA, in which case we go for 1 floor
    floors <- ifelse(is.na(frame$fdffloor), 1, frame$fdffloor)

    ## How we determine the bottom floor's type:
    if (is.basement(frame)) {
        start.floor <- -1
    } else if (tolower(frame$finlopos) == "ground floor flat") {
        start.floor <- 0
    } else {
        start.floor <- 1
    }

    ## An EHS oddity: some flats are described by the flatdets file,
    ## and the module information is for the building containing all
    ## the flats. Other flats are described by the module information;
    ## fdfsamed is "yes" when we should use the module information,
    ## but in certain cases this says "yes" but the module information
    ## is NA and the flatdets information is not. C'est la vie.
    if (tolower(frame$fdfsamed) == "no" & !is.na(frame$fdfmainw)) {
```

```
        ## So in this case we make a floor for each floor, telling
        ## make.one.floor to use flatdets information (the TRUE below).
        result <- do.call(rbind, lapply(1:floors,
                                        make.one.floor,
                                        TRUE, start.floor, frame))

        result$area <- NULL
        result
    } else {
        ## Otherwise we just treat it like a house (the module data is the same)
        one.house.storeys(frame, floors, start.floor)
    }
}
```

The inverse of coordinate.string

```
parse.coordinate.string <-
    function(ps) (gsub("[^0-9,-\\.]", "", as.character(ps)) %>%
                  strsplit(",", fixed=TRUE) %>%
                  unlist() %>%
                  as.numeric()) / 100
```

Compute the area of a polygon expressed in DTO format

```
coordinate.string.area <- function(xs, ys)
    mapply(function(xs, ys)
        areapl(cbind(parse.coordinate.string(xs),
                     parse.coordinate.string(ys))),
        xs, ys)
```

Re-scale the coordinates of a polygon

@param xs - the coordinates, written in DTO form (i.e. a string) @param factor - a scaling factor @return a new DTO form coordinate list

```
rescale.coordinate.string <- function(xs, factor)
    mapply(function(xs, factor) coordinate.string(parse.coordinate.string(xs) * 100 * factor),
           xs, factor)
```

Re-scale an entire building

@param result - the kind of frame made within one.building.storeys @param target - the target total area for the building @return a copy of result with the polygons scaled to have the target area

```
scale.building.storeys <- function(result, target) {
    area <- sum(with(result, coordinate.string.area(polygonxpoints, polygonypoints)))
    scaling.factor <- sqrt(target / area)
```

```
    mutate(result,
           polygonxpoints = rescale.coordinate.string(polygonxpoints, scaling.factor),
           polygonxpoints = rescale.coordinate.string(polygonxpoints, scaling.factor))
}
```

What you've all been waiting for: make the storeys for a house or a flat

This makes the final DTO rows we need for any row in the EHS,

@param frame - a data frame containing all the spss variables for a single case, lowercased column names.
@param scale - if true, scale total area to floorarea in frame

```
one.building.storeys <- function(frame, scale) {
    ## Find out if we are making a house
    if (grepl("house", frame$dwtype8x)) {
        ## so if we are making a house, we need to know how many storeys and
        ## whether there's a basement
        if (is.basement(frame)) {
            start.floor <- -1
            floors.total <- frame$storeyx + 1
        } else {
            start.floor <- 0
            floors.total <- frame$storeyx
        }

        result <- one.house.storeys(frame, floors.total, start.floor)
    } else {
        result <- one.flat.storeys(frame)
    }

    ## so now result is done, except it needs the ceiling and storey heights.

    ## set ceiling heights with mean ceiling height, for most cases
    result$ceilingheight <- mean(c(frame$finlivcl,
                                   frame$finbedcl,
                                   frame$finkitcl,
                                   frame$finbatcl),
                                 na.rm=TRUE)

    ## set exterior height to ceiling height plus a bit
    ## to start with it's at least the ceiling height + 0.25
    result$storyheight <- result$ceilingheight + 0.25

    ## except on the first floor
    result$storyheight[[1]] <- result$ceilingheight[[1]]

    ## and room in roof floor is also special.
    result$storyheight[result$type == "ROOM_IN_ROOF"] <- 2.45
    result$ceilingheight[result$type == "ROOM_IN_ROOF"] <- 2.45

    ## This is a technical detail: because this result is for a single
    ## house, and we are going to glue all the results for many houses
    ## together, we need to ensure that the text columns are actually
```

```
    ## text rather than factors. Gluing factor columns together
    ## produces weird results, because they are really integers coding
    ## for the factor levels and those will be different for each
    ## house, blah blah.

    result$polygonxpoints <- as.character(result$polygonxpoints)
    result$polygonypoints <- as.character(result$polygonypoints)
    result$type <- as.character(result$type)

    if (scale) {
        result <- scale.building.storeys(result, frame$floorarea)
    }

    result
}
```

The main method

This makes all the storeys for the whole EHS in one go.

@param base.directory - where to read the SPSS files from @param output.file - where to write the storeys.csv file to. @param scale - if TRUE, scale the total floor area for each house to derived/dimensions/FloorArea

```
generate.all.storeys <- function(base.directory, output.file, scale) {
    read.input <- function(p) {
        r <- read.spss(file.path(base.directory, p),
                       to.data.frame=TRUE,
                       reencode='utf-8')
        colnames(r) <- tolower(colnames(r))
        r$caseno <- NULL
        r
    }

    join.aacode <- function(a, b) merge(a, b, by="aacode")

    big.data <- Reduce(join.aacode,
                       list(
                           read.input("derived/dimensions.sav"),
                           read.input("derived/physical.sav"),
                           read.input("physical/shape.sav"),
                           read.input("physical/flatdets.sav"),
                           read.input("physical/interior.sav"),
                           read.input("physical/services.sav")))

    ## for speed, retain only the variables we are actually using
    ## and make a data.table rather than a data.frame.
    big.data <- big.data[, c(
        "aacode",   "attic",    "basement", "dwtype3x", "dwtype8x",
        "fdffloor", "fdfmaind", "fdfmainw", "fdfnextd", "fdfnextw",
        "fdfsamed", "fdhadep1", "fdhadep2", "fdhadep3", "fdhawid1",
        "fdhawid2", "fdhawid3", "fdhmdep1", "fdhmdep2", "fdhmdep3",
        "fdhmlev1", "fdhmwid1", "fdhmwid2", "fdhmwid3", "finbatcl",
        "finbedcl", "finkitcl", "finlivcl", "fshaddit", "finlopos",
```

```
        "storeyx",  "floorarea"
    )]

    big.data <- data.table(big.data)

    setkey(big.data, "aacode")

    ## this is the bit where we say, split the table by aacode, and then
    ## run one.building.storeys for each subset, and then join them
    ## all together again. For more on this, read the data.table manual
    all.result <- big.data[, one.building.storeys(.SD, scale), by=aacode]

    write.csv(all.result, output.file, row.names=FALSE)
}
```

Ventilation

The following section of code is generated in ventilation.R

Make ventilation data

Create a dataframe containing a complete set of populated variables for the ventilation.csv stock file.

@param allEntries - combined sav files data.frame from the EHS data

```
make.ventilation <- function(allEntries, elevationsDTO){

  #calculate the percentage draught stripped from the proportion of double glazed
  #windows
  draftStipped <- proportion.draft.stripped(elevationsDTO)

  #NHM import used heating systems to determine, chimneys, vents and fans so we zero
  # these in the ventilations file - they are no longer used.
  allVentilation <- data.table(
    aacode = allEntries$aacode
    ,chimneysmainheating = rep(0, nrow(allEntries))
    ,chimneyssecondaryheating = rep(0, nrow(allEntries))
    ,chimneysother = rep(0, nrow(allEntries))
    ,intermittentfans = rep(0, nrow(allEntries))
    ,passivevents = rep(0, nrow(allEntries))
    ,ventilationsystem = rep("natural", nrow(allEntries))
    )

  allVentilation <- join(allVentilation, draftStipped, by = "aacode")
  print(paste("ventilation DTO complete; number of records: ", nrow(allVentilation)))

  return(allVentilation)
}
```

Proportion of dwelling draught proofing

Calculates proportion of Windows and Doors that are draft stripped as the mean of percentage double glazed elevations (in line with RD SAP assumptions).

@param elevationDTO - data.frame containing aacode and percentageWindowDblGlazed for each elevation

```
proportion.draft.stripped <- function(elevationsDTO){

  elevationDTOTable <- data.table(elevationsDTO)
  meanPercentDblGlazed <- elevationDTOTable[, j=list(mean(percentagedoubleglazed)),
                                            by = aacode]
  draftStipped <- data.table(
    aacode = meanPercentDblGlazed$aacode
    ,windowsanddoorsdraughtstrippedproportion = meanPercentDblGlazed$V1 / 100
  )

  return (draftStipped)
}
```

Elevations

The following section of code is generated in elevations.R

Make 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 allEntries - combined sav files data.frame from the EHS data

```
elevations.make <- function(allEntries, doorEntries) {
  allElevations <- build.empty.elevations(allEntries)

  elevationDetail <- calc.elevation.detail(allEntries)
  wallConstruction <- build.external.walls(allEntries)
  doors <- build.doors(doorEntries)

  allElevations <- Reduce(function (a,b) {
    merge(a,b, by = c("aacode","elevationtype"), all.x = TRUE)
  }, list(allElevations, elevationDetail, wallConstruction, doors))

  #Fix attached/party walls
  #if walls are 100% attached then they should have no external wall type
  fixedWallConstructionType <- data.table(
    aacode = allElevations$aacode
    ,elevationtype = allElevations$elevationtype
    ,externalwallconstructiontype = ifelse(allElevations$tenthsattached < 10,
                                    allElevations$externalwallconstructiontype, NA)
    ,cavitywallinsulation = ifelse(allElevations$tenthsattached < 10,
                                   allElevations$cavitywallinsulation, FALSE)
    ,externalwallinsulation = ifelse(allElevations$tenthsattached < 10,
                                     allElevations$externalwallinsulation, FALSE)
  )

  allElevations <- subset(allElevations, select = c(-externalwallconstructiontype,
                                    -cavitywallinsulation, -externalwallinsulation))
  allElevations <- merge(allElevations, fixedWallConstructionType,
                         by = c("aacode","elevationtype"))

  #Put window data onto every elevation
  allElevations <- merge(allElevations, build.windows(allEntries),
                         by = "aacode", all.x = TRUE)
```

```
  print(paste("elevations DTO complete; number of records: ", nrow(allElevations)))

  return(allElevations)
}
```

Window glazing

Determines the window glazing proportion in the dwelling and constructs a window data.frame

```
build.windows <- function(allEntries){

  percentageDblGlazed <- as.numeric(levels(
    percentage.glazed(allEntries$dblglaz4))[percentage.glazed(allEntries$dblglaz4)])
  predominantWindowType <- as.character(levels(
    window.frame.type(allEntries$typewin))[window.frame.type(allEntries$typewin)])

  windows <- data.frame(
    aacode = allEntries$aacode,
    percentagedoubleglazed = percentageDblGlazed,
    doubleglazedwindowframe = ifelse(percentageDblGlazed > 0,
                                                    predominantWindowType, "NULL"),
    singleglazedwindowframe = ifelse(percentageDblGlazed < 100,
                                                    predominantWindowType, "NULL")
  )

  return (windows)
}
```

Window frame

Converts the SPSS variable for window frame type into a value the NHM import understands

@assumption SPSS value of "Mixed types" defaults to Wood

@param typewin

```
window.frame.type <- function(typewin){
  as.factor(checked.revalue(
    typewin,c(
      "double-glazed- metal" =  "Metal",
      "double-glazed- UPVC" = "uPVC",
      "double-glazed- wood" = "Wood",
      "single-glazed- metal" = "Metal",
      "single-glazed- UPVC" = "uPVC",
      "single-glazed- wood sash" = "Wood",
      "single-glazed- wood casement" = "Wood",
      "mixed types" = "Wood"
    )))}

percentage.glazed <- function(dblglaz4){
  as.factor(checked.revalue(
    dblglaz4,c(
```

```
      "no double glazing" =  0,
      "less than half" = 25,
      "more than half" = 75,
      "entire house" = 100
    )))}
```

Doors

Builds doors in front and back elevations, for left and right elevations doors of all types are always zero as we have no information on these from the survey.

```
build.doors <- function(doorEntries){
  # Summaries the number of front doors of different frame types
  frontDoors_summmary <- cast(doorEntries, aacode ~ type, value = "Fexdf1no")

  frontElevations <- data.frame(
    aacode = frontDoors_summmary$aacode,
    elevationtype = rep("FRONT", length(frontDoors_summmary$aacode)),
    "doorframe:wood,doortype:solid" = frontDoors_summmary$Wood,
    "doorframe:wood,doortype:glazed" = rep(0, length(frontDoors_summmary$aacode)),
    "doorframe:metal,doortype:solid" =  frontDoors_summmary$Metal,
    "doorframe:metal,doortype:glazed" = rep(0, length(frontDoors_summmary$aacode)),
    "doorframe:upvc,doortype:solid" = frontDoors_summmary$UPVC,
    "doorframe:upvc,doortype:glazed"= rep(0, length(frontDoors_summmary$aacode)),
    check.names = FALSE
  )

  # Summaries the number of back doors of different frame types
  backDoors_summmary <- cast(doorEntries, aacode ~ type, value = "Fexdf2no")

  backElevations <- data.frame(
    aacode = backDoors_summmary$aacode,
    elevationtype = rep("BACK", length(backDoors_summmary$aacode)),
    "doorframe:wood,doortype:solid" = backDoors_summmary$Wood,
    "doorframe:wood,doortype:glazed" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:metal,doortype:solid" = backDoors_summmary$Metal,
    "doorframe:metal,doortype:glazed" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:upvc,doortype:solid" = backDoors_summmary$UPVC,
    "doorframe:upvc,doortype:glazed"= rep(0, length(backDoors_summmary$aacode)),
    check.names = FALSE
  )

# Note not a typo, we have no data for left and right elevations so just add zeros
# and use the length of the back_door summary for the number of rows to populate
  leftElevations <- data.frame(
    aacode = backDoors_summmary$aacode,
    elevationtype = rep("LEFT", length(backDoors_summmary$aacode)),
    "doorframe:wood,doortype:solid" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:wood,doortype:glazed" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:metal,doortype:solid" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:metal,doortype:glazed" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:upvc,doortype:solid" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:upvc,doortype:glazed"= rep(0, length(backDoors_summmary$aacode)),
```

```
    check.names = FALSE
  )

  rightElevations <- data.frame(
    aacode = backDoors_summmary$aacode,
    elevationtype = rep("RIGHT", length(backDoors_summmary$aacode)),
    "doorframe:wood,doortype:solid" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:wood,doortype:glazed" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:metal,doortype:solid" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:metal,doortype:glazed" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:upvc,doortype:solid" = rep(0, length(backDoors_summmary$aacode)),
    "doorframe:upvc,doortype:glazed"= rep(0, length(backDoors_summmary$aacode)),
    check.names = FALSE
  )


  allDoors <- rbind(frontElevations, backElevations, leftElevations, rightElevations)
  return(allDoors)
}
```

External wall construction type and insulation

Builds external walls and insulation, assumes all walls have the same construction type, derived from typewstr2

@assumption internalinsulation - Always FALSE, is dealt with later on

@param Felcavff - cavitywallinsulation(Yes or No) @param Felextff - externalwallinsulation(Yes or No) @param typewstr2 - predominant wall construction type

```
build.external.walls <- function(allEntries) {

  wallConstructionType <- wall.structure.type(allEntries$typewstr2)

  can.have.cwi <- wallConstructionType %in% c("Cavity", "SystemBuild", "TimberFrame")

  frontElevations <- data.frame(
    aacode = allEntries$aacode,
    elevationtype = rep("FRONT", length(allEntries$aacode)),
    externalwallconstructiontype = wallConstructionType,
    cavitywallinsulation = is.na(allEntries$Felcavff) == FALSE &
                                        allEntries$Felcavff == "Yes" & can.have.cwi,
    internalinsulation = rep(FALSE, length(allEntries$aacode)),
    externalwallinsulation = is.na(allEntries$Felextff) == FALSE &
                                                      allEntries$Felextff == "Yes"
  )

  backElevations <- data.frame(
    aacode = allEntries$aacode,
    elevationtype = rep("BACK", length(allEntries$aacode)),
    externalwallconstructiontype = wallConstructionType,
    cavitywallinsulation = is.na(allEntries$Felcavbf) == FALSE &
                                       allEntries$Felcavbf == "Yes" & can.have.cwi,
    internalinsulation = rep(FALSE, length(allEntries$aacode)),
    externalwallinsulation = is.na(allEntries$Felextbf) == FALSE &
```

```
                                                        allEntries$Felextbf == "Yes"
  )

  leftElevations <- data.frame(
    aacode = allEntries$aacode,
    elevationtype = rep("LEFT", length(allEntries$aacode)),
    externalwallconstructiontype = wallConstructionType,
    cavitywallinsulation = is.na(allEntries$Felcavlf) == FALSE &
                                          allEntries$Felcavlf == "Yes" & can.have.cwi,
    internalinsulation = rep(FALSE, length(allEntries$aacode)),
    externalwallinsulation = is.na(allEntries$Felextlf) == FALSE &
                                                        allEntries$Felextlf == "Yes"
  )

  rightElevations <- data.frame(
    aacode = allEntries$aacode,
    elevationtype = rep("RIGHT", length(allEntries$aacode)),
    externalwallconstructiontype = wallConstructionType,
    cavitywallinsulation = is.na(allEntries$Felcavrf) == FALSE
                                        & allEntries$Felcavrf == "Yes" & can.have.cwi,
    internalinsulation = rep(FALSE, length(allEntries$aacode)),
    externalwallinsulation = is.na(allEntries$Felextrf) ==
                                                  FALSE & allEntries$Felextrf == "Yes"
  )

  mergedElevations <- rbind(frontElevations, backElevations, leftElevations, rightElevations)
  mergedElevations$externalwallconstructiontype <-
                          as.character(mergedElevations$externalwallconstructiontype)

  return(mergedElevations)
}
```

Door frame type

Determines the door frame type from the doors sav file in the EHS.

```
doors.frame.type <- function(type){
  as.factor(checked.revalue(
    type,c(
      "Wood" = "Wood",
      "UPVC" = "uPVC",
      "Metal" = "Metal")))}

wall.structure.type <- function(typewstr2){
    return (as.character(
      levels(wall.structure.type.lookup(typewstr2))[wall.structure.type.lookup(typewstr2)]))
}
```

Predominant wall structure type

@param typewstr2 - Predominant wall structure type @assumption Mixed Types are assumed to be Granite-OrWhinstone - Sure we can do better

```
wall.structure.type.lookup <-function(typewstr2){
  as.factor(checked.revalue(
    typewstr2,c(
      "mixed types" = "GraniteOrWhinstone",
      "masonry cavity" = "Cavity",
      "masonry single leaf" = "SolidBrick",
      "9 inch solid" = "GraniteOrWhinstone",
      "greater than 9 inch solid" = "Sandstone",
      "in situ concrete" = "SystemBuild",
      "concrete panels" = "SystemBuild",
      "timber/metal/plastic panels" = "TimberFrame")))}
```

Wall attachments

Calculates tenths attached, opening and partywall for each elevation

@param aacode @param Felfenfw - tenths opening front wall @param Felfenbw - tenths opening back wall @param Felfenlw - tenths opening left wall @param Felfenrw - tenths opening right wall

@note - There are 113 Case where a house has an NA value for back tenths attached @note - There are 3947 Case where a house has an NA value for left tenths attached @note - There are 4015 Case where a house has an NA value for right tenths attached

@assumption - If tenths attached for an elevation in house or flat is NA set to 10 @assumption - If tenths opening for an elevation or house is NA set to 0

```
calc.elevation.detail <- function(allEntries){

  isAHouse <- is.a.house(allEntries$dwtype8x)

  frontTenthsAttached <- ifelse(is.a.house(allEntries$dwtype8x) == TRUE,
                          ifelse(is.na(allEntries$Fvwtenff), 10, allEntries$Fvwtenff),
                          10 - allEntries$Fdffrooa)
  backTenthsAttached <- ifelse(is.a.house(allEntries$dwtype8x) == TRUE,
                        ifelse(is.na(allEntries$Fvwtenbf),10, allEntries$Fvwtenbf),
                        10 - allEntries$Fdfbckoa)
  leftTenthsAttached <- ifelse(is.a.house(allEntries$dwtype8x) == TRUE,
                        ifelse(is.na(allEntries$Fvwtenlf),10, allEntries$Fvwtenlf),
                        10 - allEntries$Fdfrigoa)
  rightTenthsAttached <- ifelse(is.a.house(allEntries$dwtype8x) == TRUE,
                          ifelse(is.na(allEntries$Fvwtenrf),10, allEntries$Fvwtenrf),
                          10 - allEntries$Fdffrooa)

  frontElevations <- data.frame(
    aacode = allEntries$aacode,
    elevationtype = rep("FRONT", length(allEntries$aacode)),
    tenthsattached = frontTenthsAttached,
    tenthsopening = ifelse(is.na(allEntries$Felfenfw), 0, allEntries$Felfenfw),
    tenthspartywall = ifelse(isAHouse == TRUE, frontTenthsAttached, allEntries$Fdffroof)
  )

  backElevations <- data.frame(
    aacode = allEntries$aacode,
    elevationtype = rep("BACK", length(allEntries$aacode)),
```

```
    tenthsattached = backTenthsAttached,
    tenthsopening = ifelse(is.na(allEntries$Felfenbw), 0, allEntries$Felfenbw),
    tenthspartywall = ifelse(isAHouse == TRUE, backTenthsAttached, allEntries$Fdfbckof)
  )

  leftElevations <- data.frame(
    aacode = allEntries$aacode,
    elevationtype = rep("LEFT", length(allEntries$aacode)),
    tenthsattached = leftTenthsAttached,
    tenthsopening = ifelse(is.na(allEntries$Felfenlw), 0, allEntries$Felfenlw),
    tenthspartywall = ifelse(isAHouse == TRUE, leftTenthsAttached, allEntries$Fdflftof)
  )

  rightElevations <- data.frame(
    aacode = allEntries$aacode,
    elevationtype = rep("RIGHT", length(allEntries$aacode)),
    tenthsattached = rightTenthsAttached,
    tenthsopening = ifelse(is.na(allEntries$Felfenrw), 0, allEntries$Felfenrw),
    tenthspartywall = ifelse(isAHouse == TRUE, rightTenthsAttached, allEntries$Fdfrigof)
  )

  mergedElevations <- rbind(frontElevations, backElevations, leftElevations, rightElevations)

  return(mergedElevations)
}
```

Build elevations

Constructs the basic structure of the elevations file

```
build.empty.elevations <- function(allEntries){
  elevations <- data.frame(elevationtype = c("FRONT","BACK","LEFT","RIGHT"))
  allElevations <- data.frame(aacode = allEntries$aacode)
  allElevations <- merge(allElevations, elevations)

  return(allElevations)
}
```

Additional properties

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

Make 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.

Here a selection of variables of interest are selected in to a data.frame that is saved as a csv file as part of the make.england function.

@param allEntries - the merged collection of all SPSS files used in the stock conversion process.

```
make.additionalproperties <- function(allEntries) {
  #creates a dataframe called additional.properties containing column names (word
  #before the = sign) and a vector to fill that column (vector after the = sign)
  additional.properties <- data.table(aacode = allEntries$aacode,
                                      Farnatur = allEntries$Farnatur,
                                      Felorien = allEntries$felorien,
                                      Felcavff = allEntries$Felcavff,
                                      Felcavlf = allEntries$Felcavlf,
                                      Felcavrf = allEntries$Felcavrf,
                                      Felcavbf = allEntries$Felcavbf,
                                      Felextff = allEntries$Felextff,
                                      Felextlf = allEntries$Felextlf,
                                      Felextrf = allEntries$Felextrf,
                                      Felextbf = allEntries$Felextbf,
                                      Felpvff = allEntries$Felpvff,
                                      Felpvlf = allEntries$Felpvlf,
                                      Felpvrf = allEntries$Felpvrf,
                                      Felpvbf = allEntries$Felpvbf,
                                      Felsolff = allEntries$Felsolff,
                                      Felsollf = allEntries$Felsollf,
                                      Felsolrf = allEntries$Felsolrf,
                                      Felsolbf = allEntries$Felsolbf,
                                      Felcav_shcs = NA,
                                      Felext_shcs = NA,
                                      Felpv_shcs = NA,
                                      Felsol_shcs = NA,
                                      Findisty = allEntries$Findisty,
                                      dblglaz4 = allEntries$dblglaz4,
                                      NRmsEHS = allEntries$NRmsEHS,
                                      NRms2a = allEntries$NRms2a,
                                      NRms4 = allEntries$NRms4,
                                      NRms5 = allEntries$NRms5,
                                      hhcompx = allEntries$hhcompx,
                                      imd1010 = allEntries$Imd1010,
                                      wallinsx = allEntries$wallinsx,
                                      Felroofp = allEntries$felroofp,
```

```
                                    AWEligible = allEntries$AWEligible,
                                    CERTpriority = allEntries$CERTpriority,
                                    WFG_preApr11 = allEntries$WFG_preApr11,
                                    sap09 = allEntries$sap09
                                   )
  print(paste("additional-properties DTO complete; number of records: ",
              nrow(additional.properties)))
  return(additional.properties)
}
```