# Section 4 – Guide to CAPITA Basefiles

**Date last modified: 14 September 2017**

This document outlines how to create the CAPITA basefiles and provides detail on the modules that create the basefiles.

# 4.1 How to create the CAPITA basefiles

## Introduction

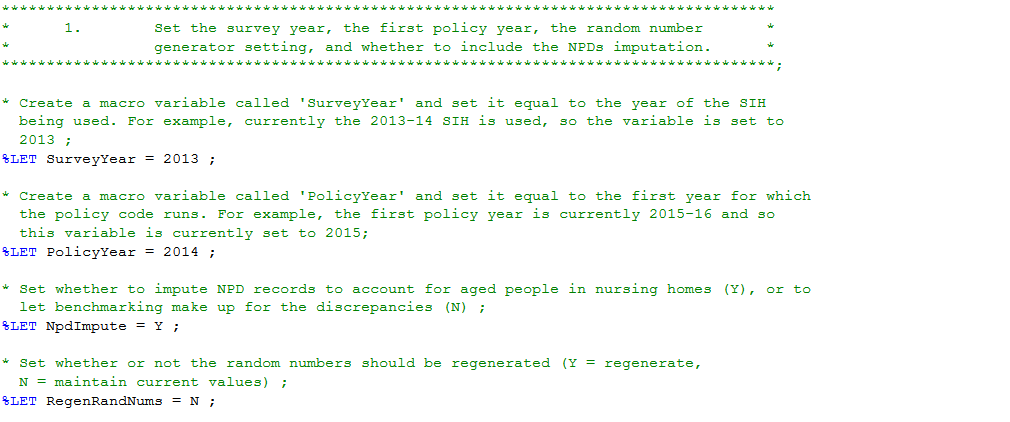
This section describes how the CAPITA basefiles are constructed and provides an overview of the modules used to create the basefiles.

## Constructing the basefiles

The central program for creating the CAPITA basefiles is the *BasefileCallingProgram*.

To create the basefiles:

* In Section 1 of the code:



**Figure 4.1: Section 1 of the *BasefileCallingProgram*.**

* + Specify the *SurveyYear*. For example, if the 2013-14 SIH is being used, this is set to 2013.
  + Specify the *PolicyYear.* This corresponds to the first financial year for which the model runs the policy code. The basefiles are then generated for this and the subsequent five financial years. For example, if CAPITA is being run for the financial years 2015-16 to 2020-21, then *PolicyYear* should be set to 2015.
  + Specify whether to impute Non-Private Dwelling (NPD) records. This should usually be set to Y, as this imputation creates replicate SIH records which aim to represent individuals living in NPDs which are not captured on the SIH datasets.
  + Specify whether or not the random number datasets should be (re)generated. The *RegenRandNums* macro variable will need to be set to Y when the basefiles are first created in order to create the random number datasets. For subsequent basefiles runs this should usually be set to N, to ensure that the random numbers do not change, unless there has been another significant update to the model which would require a reset of the imputations (for example, when a new SIH is being incorporated). Keeping the random numbers unchanged in most cases is important to ensure that variation from the random number process does not contribute to variations in model outcomes when conducting analysis.
* In Section 2 of the code, specify directories. The Library and Census directories define the locations of the SIH data (to form the main basefiles) and the Census data (which is used in the NPDs module) respectively. The other directories are explained in the code but for Treasury users of the model, once the main **CapitaDirectory** is defined, these directories should not require updating. Note in particular the **BFExport** directory, which specifies where the basefiles will be exported to on the network drive. As noted in Section 2, non-Treasury users would need to modify the directories in the model based on their own network drive.

Running this code will now create the CAPITA basefiles by sequentially calling each of the modules specified with %INCLUDE statements. Details of these modules are provided in Section 5.1. The basefiles are SAS datasets created in the form ‘Basefile*Year*’, with one basefile created for the survey year, and separate basefiles for each of the policy years.

## Key basefile inputs

The key basefile inputs will be updated as further releases of CAPITA are made available. Once these updates are made, re-running the *BasefileCallingProgram* (as described above) will generate the updated set of CAPITA basefiles.

### Survey of Income and Housing CURF

The *ReadSIH* module reads the variables from the Survey of Income and Housing (SIH) Confidentialised Unit Record File (CURF) required for use in CAPITA. The current version of CAPITA uses the 2013-14 SIH, which contains:

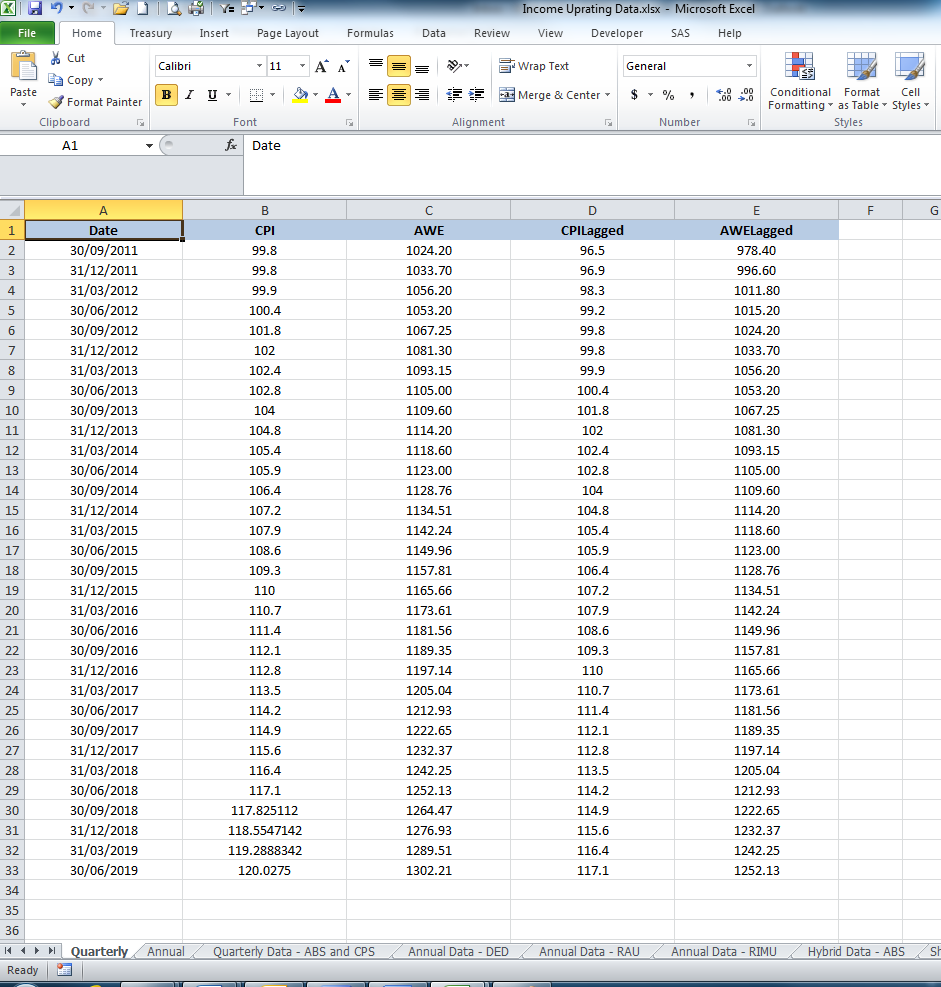
* 27,265 people on the person level dataset;
* 17,091 income units on the income unit level dataset;
* 14,162 households on the household level dataset.

Whenever new variables from the SIH are required for use in CAPITA, they need to be added to the variable lists in ReadSIH, with appropriate CAPITA variable names created for these new variables.

### Uprating Data

Income uprating is necessary because the various income sources earned by the household can be expected to be higher in future years than at the time when the SIH was conducted due to changes in various macroeconomic conditions such as price inflation, movements in interest rates, and movements in wages.

The data used to uprate the income items on the SIH is stored in the ‘Income Uprating Data’ spreadsheet in the ‘Uprating Data’ folder of the ‘Basefile Code’ folder, and it appears as follows:



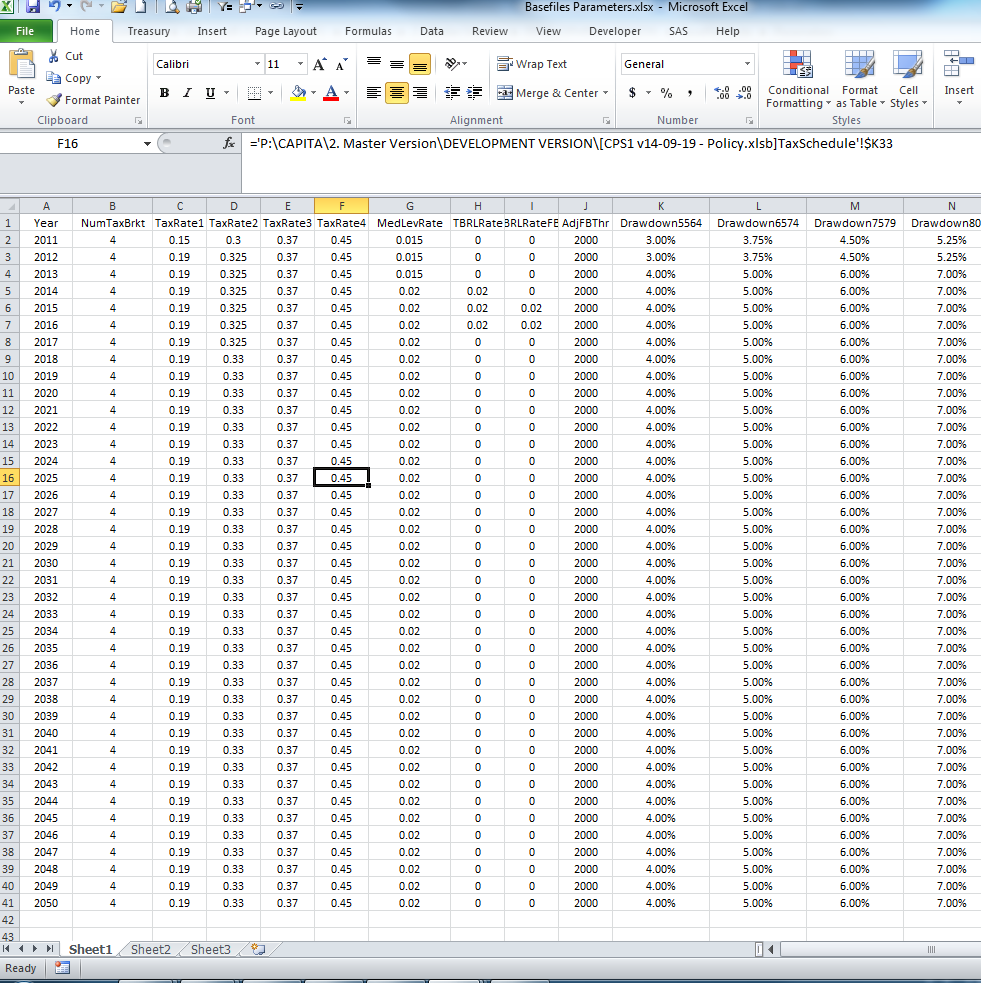
**Figure 4.2: The Income Uprating Data spreadsheet.**

This spreadsheet contains quarterly CPI and AWE series linked to the CPS (Common Parameter Spreadsheet). It also contains lagged series of CPI and AWE, which are used for uprating the previous financial year items on the SIH.

### Parameters

A small set of parameters is required in the basefile code, mostly for use in the imputation calculations.

The parameters are stored in the ‘Basefiles Parameters’ spreadsheet in the ‘Parameters’ folder of the ‘Basefile Code’ folder, and it appears as follows:

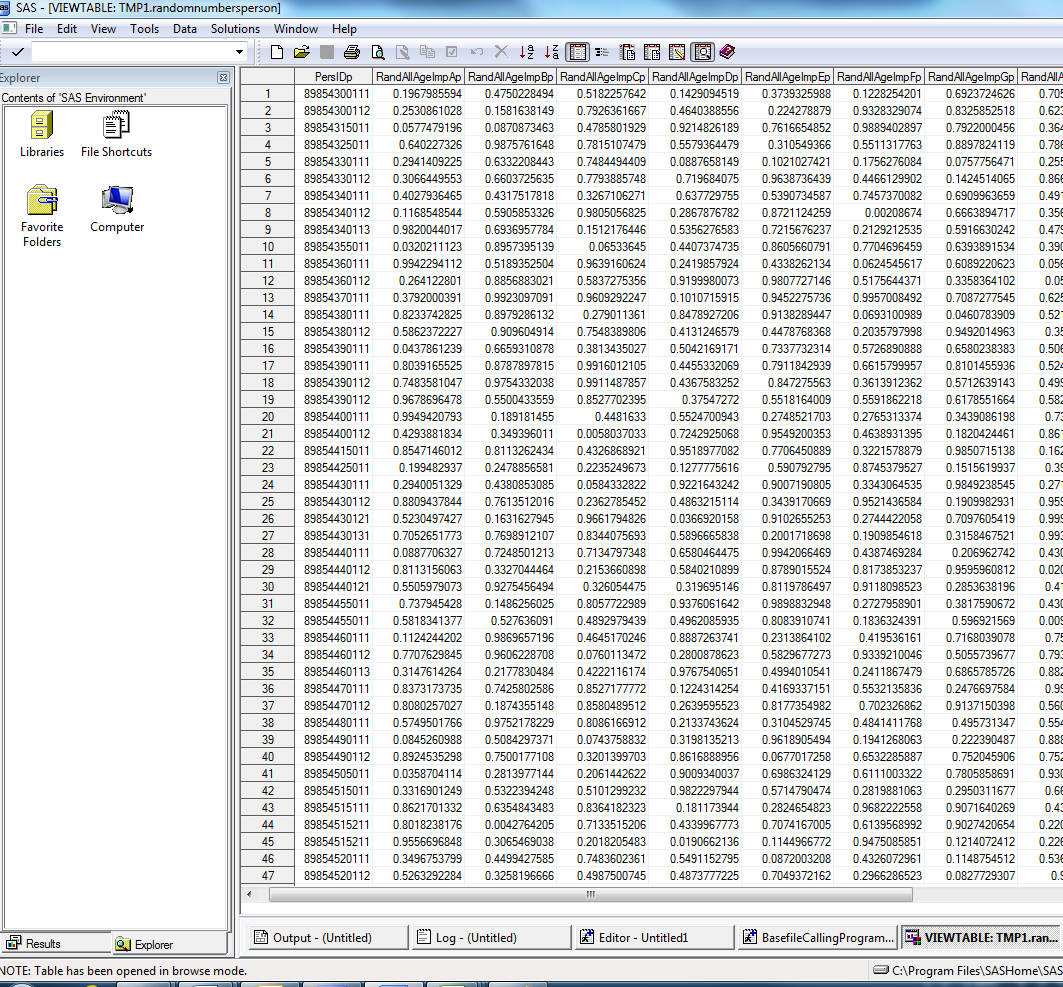


**Figure 4.3: The Basefiles Parameters spreadsheet.**

### Random Numbers

Random Numbers are generated for use in the imputations. As discussed above, the BasefileCallingProgram specifies whether or not the random numbers will be regenerated.

The random numbers datasets are stored in the ‘Random Numbers’ folder of the ‘Basefile Code’ folder, and, for example, the person level random numbers dataset appears as follows:



**Figure 4.4: The person level Random Numbers dataset.**

In the case of the person level dataset, each person in the basefiles is assigned a random number for each particular instance where random numbers are used in the imputations.

### Benchmarking Data

The benchmarking process makes use of demographic data (for example, population projections by age, labour force status, and education status) and administrative data (customer numbers by payment type) to re-weight the basefiles each year, to ensure that the sample remains representative of the actual population in that year. The re-weighting is performed using the **Gregwt** macro.

The ‘Benchmarks’ spreadsheet in the ‘Benchmarking’ folder of the ‘Basefile Code’ folder contains the benchmarking data. The spreadsheet appears as follows:

### 

**Figure 4.5: The Benchmarks spreadsheet.**

The spreadsheet contains a series of different sheets for each of the benchmarking datasets used.

### Imputations

The CAPITA policy modules require some variables which are not present on the SIH. Therefore, these variables need to be estimated, or ‘imputed’.

The imputations code and associated datasets are stored in the ‘Imputations’ folder in the ‘Basefile Code’ folder.

## The Basefile creation process

***Uprate person and household level datasets to each policy year***

***Survey Year datasets (taken separately)***

**Person&SurveyYear**

**Household&SurveyYear**

**Household**

**IU**

**Person**

***Merge Survey Year datasets to Income Unit level***

**BasefilesParameters**

**Uprate**

**Merge**

**BasefileMacros**

**Person&PolicyYear**

**Household&PolicyYear**

**ReadSIH**

**Basefile&SurveyYear**

**NPDs records**

***Merge uprated Policy Year datasets for Person and Household levels to Survey Year Income Unit level dataset***

**BasefilesInitialisation**

**VariableConstruct2**

**Merge (uprated P and H with survey year IU + VC2)**

**VariableConstruct1**

**VariableConstruct3**

**Basefile&SurveyYear**

**RandomNumbers**

**Imputations**

**Benchmarking**

**Basefile&PolicyYear**

# 4.2 Detailed guide to basefile code

The CAPITA basefile code consists of fourteen separate modules, as well as a number of individual modules for each of the imputations performed. These modules are called from a central program, *BasefileCallingProgram.sas*. This central program contains a number of settings which the user must specify before generating the basefiles – these settings were described in Section 4.1. Once these settings are specified, running the *BasefileCallingProgram* will create the basefiles for the survey year and for each of the policy years for which CAPITA is to be run.

This section walks the user through the *BasefileCallingProgram* sequentially. It is designed to be read in conjunction with the *BasefileCallingProgram* code, Section 4.1 of this document, the CAPITA variable register, and the ABS SIH data item listing (available on the ABS website).

Formatting in this document is used to indicate the following:

* *Variables*
* Macro variables
* **Macros**
* SAS COMMANDS

## Settings and Directories

Most of the settings and directories were discussed in Section 4.1.

The CapitaDirectory variable defines the CAPITA version folder currently being used. From there, the other directories are defined by beginning with CapitaDirectory and then adding specific folders or programs.

## BasefilesParameters

### Overview

The *BasefilesParameters* module defines a list of parameters required for the basefile modules and a macro called ImportParams for reading them into CAPITA. Note that these are only the parameters required for the basefiles, not the full set of parameters used by the policy modules – *RunCAPITA* reads in these separately. Also note that this macro is not actually called in *BasefilesParameters*, rather, it is called in the *Merge* module, to attach the parameters to the merged (income unit level) basefiles for each year.

### Define ParamList

The module first defines a macro variable ParamList, which contains a list of each of the names attached to the parameters in the parameter spreadsheet separated by the ‘-‘ delimiter. This list is used later in the *BasefileCallingProgram* code to drop each of the parameters after the basefiles have been formed to ensure that they do not clash with the parameters to be read in for the policy modules as part of *RunCAPITA*.

### Define the ImportParams macro

The module then defines the **ImportParams** macro which contains a PROC IMPORT step for reading in the parameters. This step extracts the appropriate year from the parameter spreadsheet and extracts the parameters in that particular row, and then outputs a dataset called ‘BasefilesParm*&Year*’.

## BasefileMacros

### Overview

The *BasefileMacros* module defines some of the key macros to be used in the basefile modules. As was also the case for the *BasefilesParameters* module described above, none of these macros are actually called in the *BasefileMacros* module.

### The KeepVar macro

This macro produces a list of variable names by extracting the variable names from a defined VarList. This macro is useful for avoiding the need to list variables manually in DROP and KEEP statements, and is used in the following modules.

### The Rename macro

This macro writes the following text to facilitate renaming of variables:

OldNameA = NewNameA

OldNameB = NewNameB

OldNameC = NewNameC

etc.

It does this by using a similar variable selection technique employed by the macros described above. Note that the ‘NameList’ input into this macro needs to be in the form:

OldNameA – NewNameA -

OldNameB – NewNameB -

OldNameC – NewNameC -

etc.

so that SAS can extract the old and new names through the use of the SCAN function.

### The RenameSuffix macro

This macro takes a list of variable names (‘NameList’), removes the final letter from the variable name, and replaces it with a specified ‘Suffix’. This macro is used in the *Merge* module to remove the ‘p’ suffix and then attach the appropriate person suffixes [r,s,1-4] to the variables from the person level datasets as they are being merged onto the income unit level dataset.

## ReadSIH

### Overview

This module reads in the ABS Survey of Income and Housing (SIH) Confidentialised Unit Record Files (CURF) from where they are stored on the network drive to the SAS work directory. There is a person level dataset, an income unit level dataset, and a household level dataset, and each of these are read in sequentially. Only the variables which are required for use in CAPITA are retained for the remainder of the basefiles creation process. Finally, the module renames the variables, from their SIH names to names which are consistent with CAPITA coding protocols.

### The CallSih macro

Inputs: InDataset – the filename of the SIH datasets on the network drive.

OutDataset – the name of the dataset containing the SIH data to be produced in the work folder.

This macro reads in the SIH data from the network drive to the SAS work folder. Note also that the LIBNAME ‘library’ was defined earlier in the *BasefileCallingProgram*, in the directory specification area. Finally, the FORMAT \_ALL\_ step is needed to remove the ABS formats from the SIH datasets.

### Reading in the SIH datasets

The module then uses the **CallSih**macro defined above to read in the SIH datasets. The person level dataset, income unit level dataset, and household level dataset are each read in.

### Create variable lists

The module then creates lists of variable names which specify the SIH names of the variables which will be required for use in the CAPITA basefiles and/or policy modules. To the left of the SIH name, and separated by a ‘-‘, the list also specifies the name to be assigned to each variable in CAPITA (see the CAPITA coding protocols for information on variable naming conventions). The lists are grouped by dataset level (person, income unit, or household) and by whether or not a suffix is required (the identifier variables are the only variables which do not require suffixes). Note that at this stage the person level suffixes are ‘p’. In the Merge module, this will be removed and replaced with [r,s,1-4] when the person level datasets are merged onto the income unit level dataset. Conversely, the income unit level and household level variables will retain their ‘u’ and ‘h’ suffixes through to the final basefiles.

**Important Note:** The variable lists define which variables will be retained from the SIH datasets. Whenever new variables from the SIH are needed for CAPITA, they will need to be added to these lists.

### Call KeepVar macro

The module then uses the **KeepVar** macro to write out the SIH names of all the variables being kept from each level of dataset.

### Create the Person, Income and Household datasets

The final step is to create datasets in the work directory called ‘Person*&SurveyYear*’, ‘Income*&SurveyYear*’, and ‘Household*&SurveyYear*’. For example, if the 2013-14 SIH is being used, these would be Person2013, Income2013 and Household2013. This is done by first using a KEEP statement to retain only those SIH variables which are contained in the lists created above. Then, the **Rename** macro is used to write text in the form:

SIHNameA = CapitaNameA

SIHNameB = CapitaNameB

SIHNameC = CapitaNameC

etc.

which is then used in a RENAME step to perform the renaming of the SIH variable names to CAPITA variable names.

## Non-Private Dwelling (NPD) imputation

**Overview**

The *NPDAged* module creates replicate records for aged people living in non-private dwellings (NPDs). Based on the 2011 Census, people residing in nursing homes and accommodation for the aged/retired (not self-contained) constitute about 46 per cent of the total NPD population. CAPITA has included the NPD aged people in its base dataset because many of them receive the Age Pension. They have been incorporated in the model through a process of matching and replication since the Census CURF data does not have the required detail for modelling tax and transfer payments. The process involves matching Census NPD observations with SIH observations based on age, income, sex and income unit type. The matching SIH observations are reweighted to Census equivalent weights and used as replicate records for NPD observations. They are incorporated in the CAPITA SIH data set with an NPD flag identifier. Census incomes have been uprated in order to match observations on the 2013-14 SIH. Census incomes are uprated by the mean growth in the age pension from 2011 to 2013.

The inclusion of replicate records for NPD student youth (those living in residential halls and colleges) in CAPITA was also investigated. Since the testing showed that they had negligible impact on the CAPITA results, they have not been included in the model.

**Define ScaleFactor**

Since the Census CURF includes observations that do not have income data so are not included in CAPITA, a ScaleFactor is calculated to account for them. The ScaleFactor is used in the calculation of the SIH weights of NPD aged by applying it to the replicate weights of observations that do have income data, on the assumption that the NPD aged that do not have income data have the same income distribution as those with income data. It is calculated in this module by grouping the observations into those with income data and those without income data. The ScaleFactor is calculated as the total combined Census weights of these two groups divided by the total Census weights of observations with income data.

**Create CensusAged dataset**

The CensusAged dataset is created from the Census dataset by selecting observations aged 65 and over residing in NPDs with income data. Since the age and income data in the Census are provided in categories, the age and sex variables correspond to categories not exact figures. These variables are converted into character variables so they can be concatenated and combined with the sex variable. A new variable *MatchId* is created for each observation by combining the age, income, and sex variables. For example, a person aged 65-69 years old, with weekly income between $200-$299, and who is Male will have the *MatchId* ‘65inc4M’ which corresponds to the age and income categories and gender of that person. Both singles and members of a couple in NPDs (but living separately) are mapped to SIH records that are lone person income units. Each Census observation is assigned a weight of 100 since they are derived from the 1% Census CURF.

**Create SihAged dataset**

The module then creates a SihAged dataset from the SIH dataset by selecting observations that have similar characteristics to the aged NPDs in the Census. These are people aged 65 and over, lone person income units not receiving wage, salary, business income or rent assistance. The SIH age and income data are put into equivalent categories to match those of the CensusAged dataset. A *MatchId* variable is then created for each observation by combining the variables corresponding to their age and income categories and gender.

As in the CensusAged dataset, a *MatchId* variable is also created by combining the age, income, and sex variables.

**Match the NpdAged dataset with the SihAged dataset**

The CensusAged dataset and the SihAged dataset are tabulated to get the total SIH weight and the total Census weight respectively for each *MatchID* in *CensusMatchIdSum* and *SihMatchIdSum*, respectively. The two datasets are then merged into a MatchIdMerge dataset based on *MatchId*. The resulting data set consists of Census observations that have matching SIH observations as well as unmatched Census observations (those that do not have matching SIH observations). SIH observations that do not have matching Census observations are dropped from the merged data set.

For unmatched Census observations in the merged data set, the closest *MatchId* from the matched observations is found for them. The weight of the unmatched observation is then added to the observation with the closest *MatchId*. For example, the weight of an unmatched observation with *MatchId* ‘65in12M’ is added to the Census observation with the closest *MatchID* (‘65in12F’), which as a result would have a new weight of 200 reflecting its original weight (100) and that of the unmatched observation (100). This process is done manually outside the code.

The result of the above process is then used to account for unmatched observations by creating a CensusSihMatch data set where the matched observations with the closest *MatchId*s are assigned their new weights reflecting the addition of unmatched observations. For each MatchId in the data set, a scalar variable is calculated as the ratio of the total Census weight to the total SIH weight.

**Create NpdAged dataset**

Finally, the NpdAged dataset is created by merging the CensusSihMatch and SihAged data sets based on *MatchId*. The weight of a replicate record is derived by multiplying its SIH weight by *Scalar*. SIH observations that do not have replicate weights (i.e. no Census match) are excluded from the NpdAged dataset. The final (pre-benchmarking) SIH weight for each replicate record is then calculated by multiplying the replicate weight by ScaleFactor, which was calculated at the beginning of the module to account for NPDs with no income data. To identify NPD replicate records, the NPD flag is set to 1. Finally, the NpdAged dataset variables are renamed to the CAPITA variable names prior to combining with the SIH data set.

It is proposed to use the same method whenever a new SIH CURF is released. For comparability, the Census income data category boundaries (hardcoded in the creation of the SihAged dataset) will need to be uprated to reflect income changes between the Census and SIH survey periods.

## BasefileInitialisation

### Overview

This module initialises all variables which are to be created in subsequent basefile modules.

### Define lists of variables to be initialised

The module first defines lists of variables that are to be initialised. Generally, any variable which is to be created in the basefile modules should be initialised here, with the exception of variables which are created after the Merge step (the variables created in *VariableConstruct2* and *VariableConstruct3*) because these modules construct variables at the income unit level, but here we are initialising variables at the person level.

The variable lists are grouped depending on dataset level (person, income unit and household) and whether they contain character or numeric variables. As was also the case for *ReadSih*, the variable lists need to contain the ‘p’, ‘u’ and ‘h’ suffixes because the variables are still being constructed at the separate dataset levels.

### Initialise the variables

#### The InitialiseNum macro

This macro initialises numeric variables which are to be created in the basefile modules. The input is ‘VarList’, which is the name of a list of variables separated by ‘-‘ delimiters. The macro uses the SCAN function to extract each variable from the ‘VarList’ and set its value to 0, which automatically initialises it as a numeric variable.

#### The InitialiseChar macro

This macro operates in a similar way to the **InitialiseNum** macro described above, except that two steps are required to initialise the variables as character variables once they are extracted from the ‘VarList’. First, the length of the variable is set to 15 characters, and then its value is set to blank. These two steps automatically initialise the variable as a character variable.

The variables listed in the variable lists are then initialised by calling the **InitialiseNum** and **InitialiseChar** macros.

## VariableConstruct1

### Overview

This module constructs some additional variables which are required for use in the CAPITA basefile and policy modules. It does not construct income unit level variables which require imputations – these are performed in the imputations modules. Also, it does not construct variables which require variables from more than one dataset level (this is done in VariableConstruct2, after the dataset levels have been merged) and it does not construct variables for any of the policy year basefiles (this is done in VariableConstruct3 in the loop which creates the basefiles for each of the policy years). Finally, any variables which require parameters for their construction are also created in VariableConstruct3.

### 1. Create additional variables required at the Person level

#### Identifier variables

The identifier variable created here is *FamPosp* and is created at the person level only, for use in the Allowances policy module in the calculation of Youth Allowance to indicate the position of a person in their family. It is necessary to create *FamPosp* because it combines information from the income unit position and relationship in the household variables from the SIH.

#### Demographic variables

A series of demographic variables are also constructed. These are simple reconfigurations of SIH variables to forms which are more convenient for use in CAPITA, and the code is self-explanatory (cross-reference with the ABS SIH data item listing and the CAPITA variable register). The most important demographic variables constructed in this module are gender, labour force status, whether the income unit is a couple, and whether the income unit is renting (note that the age variables need to be created as an imputation, which happens later).

#### Income variables

Calculations of income or expense related variables are the next step. Some of the calculations which are currently included are:

* Net share losses: These are estimated as the amount by which deductions against dividend income exceed the amount of dividend income actually earned, or as zero if deductions against dividend income are less than or equal to dividend income.
* Wage and Salary Income: If the reported wage and salary amount includes salary sacrificed amounts, then the code removes the salary sacrificed amount to construct the IncWageSWp variable.
* Frequency conversions: A number of frequency conversions are required to simplify calculations later in the CAPITA code.
* Total income from salary sacrificed and non-salary sacrificed benefits: The sum of these amounts from the SIH.
* Total income from salary sacrificed and non-salary sacrificed fringe benefits: Removes childcare, shares and superannuation from the above total, as these are not considered fringe benefits.
* Total interest income is the sum of interest income earned from various sources.
* Income from workers’ compensation is the sum of income from accident compensation and sickness insurance and income from regular workers’ compensation.
* Net income from rental income is the sum of residential property income and non-residential property income.
* Total net investment losses is the sum of net rental losses and net share losses.
* Deductible child maintenance for ATI purposes is set equal to maintenance income, meaning that maintenance income is used as a proxy.
* Service income is set equal to income from wages and salaries, meaning that service income is used as a proxy.
* Create a variable called ‘DataScopeType&psn’ which indicates whether the previous year data is available. A setting of 1 for FINSCOPE on the SIH is a sign of significant change within the household - recently arrived in Australia or there has been a change of marital status that may impact on the comparability of current year and previous year estimates.

#### Asset variables

Calculation of person level asset variables makes up the remainder of this step. Person level asset variables are aggregated up into four categories: real estate and business non-primary production assets, deemed financial assets, trusts and companies non-primary production assets, and other assets.

Asset information from the SIH is also required at the household level, so further calculations later in the module are necessary. To prepare for the household level step, the proportion of person level asset holdings within each household for each individual is calculated. This is to allow household-level assets to be allocated reasonably between people in each household.

### 2. Create additional variables required at the Income Unit level

#### NumIUu

NumIUu is the number of income units in the family. This can be calculated via the use of a PROC MEANS, using family ID as the classification variable. A MERGE step is then used to attach the NumIUu variable created out of the PROC MEANS step to the income unit level dataset.

### 3. Create additional variables required at the Household level

The variable *Stateh* is created as a character version of the numeric State variable on the SIH (which is given the CAPITA name *StateSh*). Note that the ACT and the NT are not separate categories on the SIH.

Finally, there are some asset variable calculations. Household level assets are aggregated up into the same categories as in the person level step. Then this information is merged back onto the person level dataset and added to the person level categories. These household level assets are allocated to individuals in proportion to their person level asset holdings within their household, as calculated earlier. If this proportion does not exist (i.e. the person holds no person level assets), household level assets are allocated evenly to individuals within a household.

## RandomNumbers

### Overview

This module generates random numbers for use in the imputation modules or the policy modules. It does this by constructing random numbers datasets for each of the person level and the income unit level datasets.

### Generation Process

Whenever the RegenRandNums macro variable is set to Y (recall that this is set in the *BasefileCallingProgram*), the random numbers datasets will be regenerated.

The code uses a DATA step on the person level dataset *purely so that it knows how many rows of random numbers to produce* (only the random numbers and the person identifier are kept at the end of this DATA step). The output dataset is stored in the specified RStore library on the network drive.

The random numbers are generated as variables which are named according to their use in either a specific imputation or their specific use in the policy modules. Using RANUNI, each value (i.e. row) of the variable is set equal to a random outcome from an independent standard uniform distribution. Finally, the resultant dataset is sorted based on the person identifier in preparation for later merging.

The code then performs the same process on the income unit level dataset for random numbers which will be used on the income unit level dataset in either the imputations or policy modules. The sorting is performed using the income unit level identifier.

### Merging onto the person and income unit level datasets

Once the datasets have been generated, the code then merges the random numbers datasets onto the person and income unit level datasets using the person or income unit level identifiers. Unlike the first half of the code, this section of the code runs regardless of the setting of RegenRandNums, since the random numbers are required regardless of whether they have been regenerated.

## Imputations

### Overview

A number of imputations are required to produce some additional variables which are required for the basefiles and/or policy modules of CAPITA. Some imputations are performed on the person level datasets and others are performed on the income level dataset. The outcome of each imputation is to add the imputed variables to the corresponding level of dataset. The imputations are called sequentially in the following order.

### All Age Imputation

#### Reason the imputation is required

The AGEBC variable on the person level dataset of the SIH (and renamed to AgeS[r,s,1-4] in CAPITA) contains the following age categories:

* Individual year of age from 15 to 24 years of age inclusive ;
* Five-year wide age ranges from ages 25 to 54 inclusive ;
* Individual year of age from 55 to 64 years of age inclusive ;
* Five-year wide age ranges from ages 65 to 79 inclusive ;
* One category for all individuals aged 80 years and over.

However, the CAPITA policy modules require the individual year of age for all people. Therefore, the variable ActualAge[r,s,1-4] must be imputed.

#### Information the imputation uses

Apart from the SIH, this imputation also uses Treasury population projections. Currently, these projections are based on ABS data[[1]](#footnote-1). Specifically, the data used are the distributions of ages from 80 to 100+ inclusive for both males and females, for the 2012 calendar year.

#### Methodology

The module contains three different types of imputations depending on the person’s age category:

* If the person is 80 years of age or over, the age distribution the population projections is used. The module uses the [inverse transformation method](http://en.wikipedia.org/wiki/Inverse_transform_sampling) to sample outcomes from this distribution and apply the age outcome to the person. Note that this method requires the use of the cumulative probability distribution function, which is a function that specifies the probability of the random variable (in this case, the age distribution) being less than the successive age increments. A macro called **DistImpute** is created to carry out these calculations.
* If the person falls into one of the five-year wide age ranges, they are randomly assigned an individual year of age within that age category. A macro called **FiveYearIntervals** is created to carry out these calculations.
* If the person falls into one of the individual year of age categories, their age is set equal to that age (a small calculation is required because the AGEBC variable on the SIH are categories, not the actual ages).

#### Explanation of code

The module begins by creating the **DistImpute** and **FiveYearIntervals** macros referred to above.

* The **DistImpute** macro serves as a generic macro which carries out imputations using the inverse transformation method when it is provided with arrays of cumulative distribution function probabilities and outcomes. The macro makes use of a DO loop which loops through each of the probabilities until the standard uniform outcome falls into one of the categories.
* The **FiveYearIntervals** macro loops through each of the five years in the specified age category and allocates people to each year randomly (i.e. with equal probabilities across the five years).

Next, the module carries out the DATA step required to conduct the imputation. The probability and outcome arrays for people 80 years and over are defined, and then the **DistImpute** macro is called. The **FiveYearIntervals** macro is then called nine times - once for each of the nine categories pertaining to the five-year age bands. Finally, the adjustment to convert categories to individual years of age is carried out for the remaining people.

#### Examples

Example 1: Demonstration of the inverse transformation method

As an example of the inverse transformation method, consider a random variable X which has the following probability distribution:

|  |  |
| --- | --- |
| X | Pr(X=x) |
| 0.7 | 0 |
| 0.1 | 1 |
| 0.2 | 2 |

To sample a random outcome from this distribution, first generate a random outcome from a standard uniform distribution (that is, a random number between 0 and 1). If the random uniform outcome is less than 0.7, the sampled value is set to 0. If the random uniform outcome is between 0.7 and 0.8, the sampled value is set to 1. If the random uniform outcome is between 0.8 and 1, the sampled value is set to 2. With a large number of repeated samples, around 70 per cent of outcomes will be 0, around 10 per cent will be 1 and around 20 per cent will be 2.

Example 2: Demonstration of the FiveYearIntervals macro

Consider a person aged between 30 and 34 years of age. Then Agep = 12 and so the DO loop in the ‘FiveYearIntervals’ macro is activated for this individual. Suppose that Random takes the value 0.72 for this person. Since 0.72 is not less than one fifth, the IF statement is not satisfied and so the Count variable gets increased by 1. Since 0.72 is not less than two fifths, the IF statement is not satisfied and the Count variable gets increased by 1 again. Since 0.72 is not less than three fifths, the IF statement is still not satisfied and the Count variable gets increased by 1 again. Since 0.72 is less than four fifths, the IF statement is satisfied, and the person’s age (ActualAgep) gets set equal to 30 (the lower bound of this age range) plus 3 (the current value of Count) = 33.

### Kid Age Imputation

#### Reason the imputation is required

The SIH contains the following variables at the income unit level:

* Number of dependent children aged 0 to 2 years in income unit
* Number of dependent children aged 3 to 4 years in income unit
* Number of dependent children aged 5 to 9 years in income unit
* Number of dependent children aged 10 to 14 years in income unit

However, these variables are each set to 2 when the number of children in the category is 2 or more. Since the policy modules require actual numbers of children, the actual number of children in each of these age ranges needs to be estimated.

In addition, the FTB module requires the number of kids in each income unit by individual year of age. After correcting for the issue described above, this imputation distributes the number of children in each of the above categories randomly to individual years of age, from **Kids0** to **Kids14**.

#### Information the imputation uses

This imputation does not use any data sources apart from the SIH.

#### Methodology

The imputation first calculates the difference between the total number of people in the income unit and the sum of the numbers of children in each of the age categories above. If this difference is greater than zero, the imputation adds the ‘excess’ children to age categories which contain 2 children (as these are the only categories which could have more children in them) randomly, according to the number of years in each of the categories.

The imputation then randomly allocates the kids in each of the categories to the individual years of age within those categories.

#### Explanation of code

The module begins by creating the variables to be imputed, which are the age categories above with ‘Imp’ suffixes added. It also initialises the individual year of age variables to zero.

The module then creates five arrays:

1. An array called **KidAgeRanges** containing the variable names for the kid age categories;

2. An array called **KidAgeRangesImp** containing the variable names for the imputed kid age categories;

3. An array called **KidIndYears** containing the variable names for the individual years of age;

4. An array called **LowerBound** containing the lower bounds of the year ranges in the categories;

5. An array called **UpperBound** containing the upper bounds of the year ranges in the categories.

Next, the module creates a variable called **ExcessKids** which calculates the difference between the total number of people in the income unit and the sum of the numbers of children in each of the age categories above.

The module then creates a ‘DO’ loop, which loops through each of the excess kids and allocates them to an age category.

* First a variable called **PotentialYears** is created. This variable is equal to the sum of the number of years in each of the age categories which are equal to 2. This is constructed by looping through each of the age categories and adding the length of each category when the category is equal to 2, where the length is calculated as the difference in the upper bounds of the age categories.
* The categories are then looped through again and **ActualYears** variables are created. Firstly, **ActualYears** is equal to the number of years in the first category which is 2, and if a uniform(0,1) covariate is less than the ratio of this number of years to **PotentialYears**, the excess kid is then allocated to the first category. If not, **PotentialYears** is then adjusted to the sum of the number of years in the first two categories which equal 2, and if the covariate is less than the new ratio, the excess kid is allocated to the second category. This loop continues until the excess kid has been allocated to a category.

Other variables which are created during this process are:

* **Count**: This variable is created to tell SAS when to stop looking through the categories to add the excess kid. It is initialised to zero at the beginning of each kid’s loop because it pertains to the category loop, not the kid loop.
* **Blank1** and **Blank2**: These variables are created purely as position holders in the age category variable name arrays so that they have the same number of elements as the upper bound and lower bound arrays. Each variable is set at zero.

Next, the module allocates the kids in each of the categories to individual years of age. It does this by looping through each of the age categories, and where it finds kids in a category, it then loops through the individual years in the category to allocate kids successively. The upper bounds are again used to calculate the denominators of the probability ranges, and the lower bounds are used as starting points to count through each of the individual years in the category.

#### Examples

Example 1: Demonstration of first step (i.e. correcting for topcoding)

Consider an income unit with these values:

**Kids0to2** = 0

**Kids3to4** = 2

**Kids5to9** = 1

**Kids10to14** = 2

**Adults15to64** = 2

**Adults65to99** = 0

**PersonsInIU** = 9

This means that there are two topcoded children, because the sum of each of the age ranges is 7 but there are a total of 9 people in the income unit. That is, **ExcessKids** = 2.

The DO loop for **PotentialYears** then proceeds as follows: **KidAgeRanges**{2} is zero, so nothing happens. **KidAgeRanges**{3} is 2, so **PotentialYears** is calculated as 0 + 4 – 2 = 2. **KidAgeRanges**{4} is 1, so nothing happens. **KidAgeRanges**{5} is 2, so **PotentialYears** is calculated as 2 + 14 – 9 = 7. Note that this is just the total number of years contained in each of the age categories which are 2.

For the first excess kid, assume that the value of **Random** is 0.35. The m loop then sets **ActualYears** = 0 + 4 – 2 = 2, and so the test for whether the first excess kid is added to **Kids3to4** or **Kids10to14** is **Random** < 2/7. Since 0.35 > 2/7, the IF statement is not satisfied and so the m loop moves to the next category and sets **ActualYears** = 2 + 14 – 9 = 7, and so the test becomes **Random** < 1, and so the IF statement is satisfied, meaning the kid gets added to **Kids10to14**.

For the second excess kid, the same process repeats, with a different value of **Random** being generated for this kid.

Example 2: Demonstration of second step (i.e. allocating to individual years)

Suppose that we need to allocate 1 kid in the 10 to 14 year age range to an individual year within that age category, and that we need to allocate 1 kid in the 3 to 4 year age range to an individual year within that age category.

For the kid in the 10 to 14 year age range, the value of **UpperBound**{n} - **UpperBound**{n-1} is 14 – 9 = 5. Since the value of **Years** is set to zero initially, the first test is **Random** < 1/5. Since **Count** is set to 1 initially, if this test holds, the kid is allocated to **KidIndYears**{**LowerBound**{n} + **Count**} = **KidIndYears**{10 + 1} = **KidIndYears**{11} = **Kid10**, which is the first year in the age range. If the test doesn’t hold, then the second test is **Random** < 2/5, and if this holds the kid is allocated to **Kid11**, and so on.

For the kid in the 3 to 4 year age range, the process is the same, however the condition will be **Random** < 1/2.

### Year of Arrival Imputation

The person-level SIH dataset contains the variable YOABC, which takes the values:

|  |
| --- |
| 0. Not applicable |
| 1. Born in Australia |
| 2. Arrived 1985 and before |
| 3. Arrived 1986-1995 |
| 4. Arrived 1996 to year of collection |
| 9. Not stated/inadequately described |

### The eligibility test for the Age Pension requires that the recipient has been an Australian resident for: (i) 10 continuous years including at least 5 years during their working life; (ii) 10 continuous years and has not received activity tested income support for a cumulative period of five years; or (iii) 15 continuous years. The previous policy was for 10 continuous years of residency. Given the small impact of policy change as at the Budget 2017-18, and the fact that individuals are likely to pass the other criteria, CAPITA makes a simplifying assumption for the purposes of this imputation that the residency requirement remains at 10 years.

### Due to this requirement, the individuals with a YOABC value of 4 must be separated into two groups using an imputation. The imputation uses a simple uniform estimation procedure based on the outcome of a standard uniform random variable and the number of years in category 4. The outcome is the *YearOfArrivalp* variable.

### Carer Dependants Imputation

Since CAPITA models the Carer Allowance (in the *Supplements* module), the policy code requires information on the number of people for which care is being provided. However, the SIH only contains information on the weekly amount of Carer Allowance received. Therefore, the number of people being cared for must be estimated based on these reported amounts. The estimation procedure consists of a comparison between:

* The total weighted amounts of Carer Allowance which would result from assigning either 0.5, 1, 2 or 3 carer dependants to the income unit according to particular threshold values of the weekly Carer Allowance amounts received; and
* The total weighted amounts of Carer Allowance actually received on the SIH.

Then, the particular threshold values are adjusted until the total cumulative errors made across the assigned number of carer dependants categories is minimised. The spreadsheet *Imputation Analysis.xlsx* in the Imputations folder of the Basefile Code folder contains these calculations.

The resulting estimated number of carer dependants (0.5, 1, 2 or 3) is stored as the *NumCareDepsp* variable.

### Workforce Independence Imputation

Workforce independence status determines whether or not a Youth Allowance recipients’ payment is affected by the parental income test. Generally speaking, a person is considered to be independent from their parents for social security purposes if they are over the age of independence (currently 22), are (or have been) married, have (or their partner has) a dependent child, or have shown themselves to be independent through workforce participation.

The SIH does not contain information on whether or not a person under the age of independence is workforce independent. Therefore, this needs to be imputed for those individuals.

A regression of workforce independence status against information available on the SIH is used to estimate whether or not the individual is workforce independent. The regression coefficients are estimated using data from the Household, Income and Labour Dynamics in Australian (HILDA) survey. Workforce independence status for the majority of individuals in the HILDA survey can be determined.

Data items used to create the regression relationship are contained in the *WorkforceIndependenceImp* code and the *CAPITA WFI imputation with history* code.

The imputation uses a logistic regression since the dependent variable (workforce independence status) is binary in nature. Implementing this approach provides each person on the basefile with a probability of being workforce independent.

A limitation of this approach is that it does not account for individuals who prove themselves to be workforce independent through a method which is not covered in the regression model. This methodology only includes the age, partner status, dependent child and self‑supporting through employment (full‑time and part‑time employment) paths to showing workforce independence. However, these are the most commonly used criteria in gaining workforce independence status.

### Franking Credits Imputation

#### Reason the imputation is required

Dividends paid to shareholders by Australian resident companies often carry ‘franking credits’ associated with the tax that the company has paid before distribution of dividends. Franking credits are recognised as refundable tax offsets by the ATO, and are also included in an individual’s assessable income. Therefore, CAPITA requires the franking credit amount in the Tax and Income 1 modules in the calculation of offset entitlements and income definitions. Whilst the SIH provides the amount of dividend income received (*including[[2]](#footnote-2)* the franking credit), it does not separate the franking credit amount from the actual dividend amount. This separation is not required for the assessable income calculations, but it is required for the tax offset calculations.

#### Information the imputation uses

Along with the dividend income amount contained on the SIH, this imputation utilises TaxStats data from 2012-13 to calculate an average amount of franked dividends out of total dividends. The data is also used to verify the approximation of the franking credit amount based on the value of franked dividends.

#### Methodology

The value of franking credits attached to a given value of dividends received can be approximated using the formula

,

where *F* is the value of franking credits, *D* is the total value of dividends received (excluding the franking credit), *f* is the proportion of the total dividends which are franked, and *t* is the corporate tax rate (30 per cent).However, the dividend income amount on the SIH *includes* the value of franking credits. Therefore, the SIH amount can be expressed as

,

where *S* is the amount of dividend income reported on the SIH. Rearranging this expression to give *D* in terms of *S* and *F,* substituting it into the first equation and solving for *F* gives

,

which provides the imputation formula which is used in the CAPITA basefiles. This formula requires a value for *f*, which is approximated using the TaxStats data referred to above. The data provides the following information on aggregate values of franked and unfranked dividends reported on personal income tax returns, along with aggregate franking credits.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **2006-07** | **2007-08** | **2008-09** | **2009-10** | **2010-11** | **2011-12** | **2012-13** |
| Unfranked dividends | 828,994,995 | 854,403,986 | 789,588,438 | 940,225,074 | 1,067,325,829 | 978,095,499 | 862,993,766 |
| Franked dividends | 18,865,462,410 | 19,880,049,359 | 20,232,654,778 | 18,548,022,079 | 20,697,126,192 | 20,404,712,580 | 20,810,116,253 |
| Franking credits | 8,073,905,501 | 8,509,972,026 | 8,662,624,383 | 7,939,115,606 | 8,863,897,688 | 8,738,105,920 | 8,909,777,437 |

From this data we calculate the average proportion of total dividends which are franked for 2013-14 as 95.66 per cent, and see that this average proportion has stayed fairly constant over the 2006-07 to 2012-13 financial years.

#### Explanation of code

The code employs the third formula above to approximate the value of franking credits, with *f* equal to 0.9566.

#### Testing

The data above also shows that multiplying the aggregate value of franked dividends by *t/(1-t)* , with *t* equal to the company tax rate of 30 per cent, gives a very close approximation of the aggregate value of franking credits received (this approximation differs from the actual value of franking credits reported by 0.1 per cent in the 2012-13 financial year), which justifies the use of the formulas above.

### Tax Deductions Imputation

#### Reason the imputation is required

An individual’s taxable income is calculated by subtracting allowable tax deductions from total assessable income. For this reason, it is necessary to impute tax deduction amounts for each record on the person level dataset as this information is not available in SIH.

#### Information the imputation uses

The tax deduction imputation utilises information from a 16 per cent sample of personal income tax returns for the 2011-12 income year. According to Taxation Statistics 2011-12, over 80% of those individuals who lodged a tax return have claimed deductions. Of the total value, the majority of tax deduction amounts are related to work‑related expenses, followed by personal super contributions, cost of managing tax affairs and deductible gifts and donations.

#### Methodology

The imputation approach adopted for CAPITA involves mapping tax deduction amounts from the personal income tax file to the person level dataset, for individuals with similar characteristics.

Recognising that the majority of individuals with tax deductions had claimed more than one type of deduction, individuals are disaggregated into the following categories:

1. with total deductions that included work-related expenses, but not personal super contributions (the median claim in 2011-12 was around $1,000);
2. with total deductions that included personal super contributions, i.e. the individual is not an employee (the median claim in 2011-12 was around $24,000);
3. with total deductions that did not include either work-related expenses or personal super contributions (the median claim in 2011-12 was around $300); and
4. no tax deductions.

The records in the tax file and person level dataset are grouped into combinations of the following characteristics:

* with or without salary income;
* with or without taxable government payments;
* with or without interest / dividend / trust income;
* age range; and
* total assessable income range.

For each group of individuals in the person level dataset, the likelihood that an individual belongs to one of the four deduction categories is calculated based on the number of occurrences for each category, as observed from the personal income tax file.

The tax deduction value allocated to an individual in the person level dataset is the geometric‑mean of the claim amounts corresponding to the individual’s characteristic group and deduction category. The approach utilises the geometric-mean, instead of the average value, because this is more representative of the value claimed by individuals within a characteristic group. The circumstances under which individuals incur deductible expenses can be varied and may not necessarily correlate with the information available on tax returns. For this reason, the range of deduction values for a group of individuals remains large even after allowing for the listed characteristics.

#### Explanation of code

The tax deduction imputation module reads into the model the set of probabilities and deduction values for all characteristic groups. The inverse cumulative distribution function is used to randomly determine a deduction category for each record within a characteristic group. A deduction value is then assigned based on the deduction category allocated to the record.

## Merge

### Overview

The merge module takes the separate person level, income unit level and household level datasets, and merges them together into a single income unit level dataset. It does this by first merging the household variables onto each income unit within each of the households. It then assigns appropriate suffixes to the variables in the person level dataset, either r, s, or 1-4, depending on whether the person is a reference person, a spouse, or a dependant. Then the person level variables are merged onto the row corresponding to the income unit to which they are attached, based on the income unit level identifier, which is contained on both the person and income unit level datasets.

### Calling the Merge macro

The *Merge* module defines a macro called **Merge** which is first called just after the *Merge* module has been included to create the merged income unit level basefile for the survey year. The **Merge** macro is then called again later in the **CreateBasefilesOutyears** macro in the *BasefileCallingProgram*, to merge the person, income unit and household level datasets for each policy year (i.e. after uprating is performed).

### The Merge macro

Input: ‘BasefileYear’, allows the macro to be called for either the survey year or any of the policy years, which is needed for the calling sequence described above.

The macro first sorts each of the datasets by the identifier variables to prepare for sorting.

Next, the household and income unit level datasets are merged together by household identifier, which means one or many income units are attached to each household (for example, if a household contains three income units, these will be three separate rows in the basefile, each with the same household identifier but with different income unit identifiers). The output dataset is called ‘Basefile*&BasefileYear*’. Finally, this is then sorted to prepare for merging onto the person level datasets.

The macro now creates separate person level datasets for the reference person (Personr), the spouse (Persons), and each of the dependants (Person1, Person2, Person3, and Person4).

* The reference person dataset contains all rows from the main person level dataset where the IUPos of the person is 1 (‘Reference person of income unit’);
* The spouse dataset contains all rows from the main person level dataset where the IUPos of the person is 2 (‘Partner of reference person of income unit’);
* The dependants datasets contains all rows from the main person level dataset where the IUPos of the person is 3 (‘Dependent child of reference person’), and either:
  + the IUType is 3 (the income unit is a lone parent with dependent children); or
  + the IUType is 1 (the income unit is a couple with dependent children).

Then, the code assigns the ordering of the dependents based on their person identifiers.

Next, the macro uses the RenameSuffix macro (defined in BasefileMacros) to generate RENAME statements to remove the ‘p’ suffix from each of the variables in each dataset, and then attach the appropriate person suffixes [r,s,1-4] to each variable within the corresponding dataset. For example, all the variable names in the ‘Persons’ dataset will have their p suffixes replaced with s suffixes.

All the separate person level datasets are now merged to create a dataset called ‘PersonsCombined*&BasefileYear*’, which then in turn is merged onto the ‘Basefile*&BasefileYear*’ dataset created earlier. Note that this merge statement is done differently depending on whether the merging is being carried out for the survey year basefile or for one of the policy year basefiles. If one of the policy year basefiles is being created, the merge also needs to include a dataset called ‘IUKeptVars’, which is created in *VariableConstruct2[[3]](#footnote-3).*

Next, for income units which have been created using the NPD imputation, the income unit weight is set equal to the person weight of the reference person. This is necessary because the SIH records are created as replicates of the person level SIH dataset, and so they do not contain an income unit weight variable.

Finally, the macro then calls the **ImportParams** macro (which was defined in *BasefilesParameters*) to import the parameters for the basefile year which is being created, and then attaches these parameters to the basefile for use in subsequent basefiles modules.

## VariableConstruct2

### Overview

This module constructs more additional variables which are required for use in the CAPITA basefile and policy modules. Unlike *VariableConstruct1*, which constructed variables for each dataset level separately, *VariableConstruct2* constructs variables which require variables from more than one dataset level, which can now be done because we have now created the merged income unit level basefile in the Merge step above.

### Structure

The variables are created using a DATA step on the survey year basefile.

The following variables are constructed in this module:

* Identifier variables are constructed for the household, family, income unit and person (HHID, FamID, IUID and PersID). Each refinement is constructed by adding the SIH identifier[[4]](#footnote-4) for that level to the cumulative identifier from the levels above. For example, within a household which has identifier 8985447, the second family of the household would be assigned the family identifier 8985447**02**, the first income unit within the family would be assigned the income unit identifier 898544702**1**, and the spouse within that income unit would be assigned the person identifier 8985447021**2**.
  + Note that the SUBSTR function must be used for the SihHID variable, as this SIH household identifier also contains letters, which must first be removed.
  + The INPUT function is also used, to convert the SihHID character variable into a numeric variable.

These identifier variables are also constructed at the income unit level and household level datasets, if applicable (for example, there is no person identifier on the income unit level dataset), to assist the merging of the dataset levels.

* *AgeYoungDepu* is the age of the youngest dependent in the income unit. This is calculated using a DO loop on an array of kid ages – for example, if the code finds that Kids3u is the first kid age variable which is positive, this will correspond to i = 4 and then *AgeYoungDepu* is set to 3 (setting i = 15 then ends the loop). Alternately, if there are no kids aged between 0 and 14, the code calculates the minimum of the ages of dependents 1-4.
* The *Kids0to15u* variable adds up the total number of children in the income unit which are 15 years of age or under. The statements (ActualAge1 = **15**)and so on are used as a shorter way of adding 1 if the statement is true (in this case, the first dependant in the income unit is 15 years of age) and 0 if the statement is false.
* The construction of the *TotalKidsu* variable uses a similar approach, with the statements (ActualAge1 > **15**) being used to check whether each dependant is over 15 years of age, adding 1 to the total if true, and 0 if false.
* The *SharerFlagu* variable indicates whether the person lives in share accommodation, and is derived based on the income unit type, the rental status of the income unit, and the family composition of the household to which the person belongs.
* The *PrivHlthInsu* variable indicates whether or not the income unit has private health insurance coverage. Note that the income unit is only considered to have private health insurance coverage if all individuals within the income unit have coverage. The ‘IF ActualAges NE **.**’ conditions are required to check whether the spouse and dependants exist before checking their private health insurance status.

The module then creates the dataset ‘IUKeptVars’ by only keeping the new variables which have been created in this module. As described above, this dataset then gets merged onto the policy year basefiles when they get created in the **Merge** macro call.

## PrepareForUprating

### Overview

This module prepares for uprating by importing the data to be used for uprating, specifying which variables will be uprated, and specifying which uprating methods will be used for each variable. It also defines the key macro to be used for performing the uprating, **UprateCommand.**

### Define the list of uprating series to be used

The module first defines the list of uprating series to be used. Currently, the model only uprates using either CPI or AWE so the list contains only these two series, as well as lagged versions of these series (for uprating the previous year income variables). Note that these names must correspond to the names given to the uprating series in the first row of the ‘Income Uprating Data’ spreadsheet.

### Import the uprated data

Next, the income uprating data is imported from the ‘Income Uprating Data’ spreadsheet using PROC IMPORT. Note that the location of the spreadsheet is defined by the macro variable ImportLocation, which is defined in the directory specification section of the *BasefileCallingProgram*. Currently, the data used for uprating (CPI and AWE) is available quarterly, and so only the data in the ‘Quarterly’ tab of the uprating data spreadsheet is being imported. Code which can be used for importing the ‘Annual’ tab is commented out for use if required.

### Create the SAS dataset containing the uprating factors

The module then creates the dataset ‘UpratingData’, which contains the uprating factors for each of the model years. This is done by first using a PROC MEANS to find the arithmetic average of each of the quarterly uprating series values across each financial year. Then the dataset is adjusted to calculate the uprating factors by dividing the current year series value by the survey year series value.

### Define the uprating lists

The module then defines the macro variables UpratingMethodsPerson and UpratingMethodsHousehold, which contain lists of the variables to be uprated and the uprating series by which they will be uprated, separated by ‘-‘ delimiters. Note that there is currently no UpratingMethodsIncome list, since there are no variables required on the income unit level SIH dataset which also require uprating, but this could be added in the same format as the other lists if required.

### Define the uprating macro

The module then defines a macro called **UprateCommand** which will be called in the *Uprate* module to actually perform the uprating. The macro uses the SCAN function to extract each variable and its corresponding uprating series from the uprating lists defined in the previous step. Next, the macro overwrites the variable to be equal to the product of itself and its uprating factor for the appropriate year.

## The CreateBasefilesOutyears macro

The next phase of the basefiles creation process in the *BasefileCallingProgram* is to loop through each of the policy years for which the basefiles are being created, and carry out the following steps:

* Uprate the income items on the survey year basefile to the required policy year. This is done by calling the **Uprate** macro, which is described below;
* Merge the uprated person and household level datasets onto the income unit level dataset to form the basefile for the required policy year. The operation of the **Merge** macro was described in the ‘Merge’ section above;
* Construct some additional variables which require parameters for the required policy year for their construction. This is done by calling the **VariableConstruct3** macro, which is described below. Note that this needs to be done after the **Merge** macro is called because these variables require the parameters that are attached at the end of that macro; and
* Drop the parameters which were attached to the basefile. This is necessary to ensure that the parameters which get assigned for the policy modules as part of RunCAPITA are attached properly.

## Uprate

### Overview

This module defines a macro called **Uprate** which is called as part of the **CreateBasefilesOutyears** macro described above. **Uprate** uses the ‘UpratingData’ dataset created in *PrepareForUprating*, and the uprating lists also defined in that module, to uprate the appropriate variables from the person and household level datasets.

### Structure

The macro first extracts the uprating factors for the required basefile year from the ‘UpratingData’ dataset, and then merges this onto the Person&SurveyYear dataset, whilst also renaming the dataset to Person&BasefileYear, which creates the person level dataset for the required policy year. Note that the \_N\_ is required because the datasets are not the same length – UpratingFactors is only one row, so the \_N\_ condition attaches this row to every row in the Person&SurveyYear dataset. Next, the **UprateCommand** macro is called to perform the uprating using the uprating lists defined in *PrepareForUprating*. Finally, the uprating factors are dropped from the dataset, by looping through the UpratingSeriesList defined in *PrepareForUprating* and constructing a DropSeries, which consists of only the names of the variables to be dropped, without the hyphen separators.

## VariableConstruct3

### Overview

This module constructs some additional variables which require parameters for the required policy year for their construction. The variables currently constructed in this module are:

* Taxable superannuation income and non-taxable superannuation income;
* Adjusted fringe benefits income; and
* An adjustment to the ages of individuals aged 65 on the SIH, to ensure that they continue to receive the age pension across the years of the model.

### Taxable and non-taxable superannuation income

The superannuation imputation calculates the amount of superannuation benefit that is assessable for taxation purposes. This amount is used to calculate assessable income and is also used to calculate the relevant amount of superannuation tax offset.

The superannuation imputation process for CAPITA requires several simplifying assumptions. Primarily, the total amount of superannuation benefit from the SIH is split into component amounts - Government and non-Government superannuation (as proxies for elements taxed in the fund and untaxed in the fund, respectively); each of these is then split into the Taxable Component and Taxfree Component; and finally, the assessable amount of superannuation benefit is calculated. The splits use calculated proportions based on various administrative data sources.

CAPITA includes certain simplifying assumptions, and removes the pre-83 crystallised amount to simplify the imputation.

#### Objectives of the Module

The objectives of the superannuation imputation are to:

* Determine the amount of superannuation benefit a person receives that is assessable, and the amount that is not assessable for taxation purposes;
* Determine the amount of assessable superannuation benefits that attract a tax offset;
* Ensure the super benefit amount is consistent with the statutory minimum drawdown amount.

A closely related issue is the uprating of super account balances. We need to uprate the account balances in order to calculate the minimum statutory drawdown requirement for allocated pensions. The super account balances are also included in the social security asset test for those over Age Pension age.

Superannuation benefits are part of a person’s private income, and can include amounts that are assessable and amounts that are non-assessable non-exempt (NANE). Assessable super benefits may also attract different amounts of tax offsets depending on the type of the benefit and the age of the person. The previous year taxable super benefit is included in previous year taxable income that is used for the Youth Allowance Parental Income Test.

For retirees superannuation benefits make up a significant share of their disposable income. Accordingly how it is modelled has a material impact on the disposable income of the person. It drives the amount of income that is subject to tax; income definitions such as Adjusted Taxable Income that are derived from taxable income; and the amount of tax offsets a person receives.

#### Data

The SIH has the following data related to the superannuation benefit:

* Income from superannuation, annuity, and private pension (ISUPERCP)
* Previous year income from superannuation, annuity, and private pension (ISUPERPP)
* Balance of account with Government super funds (VSUPGCP)
* Balance of account with non-Government super funds (VSUPNCP)

#### Imputation methodology

The amount of superannuation benefit undergoes the following imputation:

1. Split the source of superannuation income into Private, Government, or Mixed using account balance.
2. For Mixed source, split between Private and Government sources according to the proportion of the account balance from each source.
3. Make adjustment for statutory minimum drawdown requirements. The statutory minimum drawdown requirement applies to the account base pension (assumed to be Private source).
4. Split Private (after adjusting for statutory minimum drawdown) and Government super income into taxable and tax free components.
5. In the CAPITA basefile, the amount of the superannuation benefit is uprated using the CPI.

#### Assumptions

Due to limited data several assumptions are necessary. Some assumptions are sensible by assuming away those superannuation benefit types that only contribute a small amount to the total superannuation benefit. Some assumptions however require further analysis and more detailed imputation:

* General SIH assumptions
  + All income from superannuation is assumed to come from income streams. In reality some income comes from lump sums. The taxation of lump sums is different to the taxation of income streams.
  + The SIH reports the amount of superannuation as of the last known account balance. This is likely to be after the benefit is drawn down. CAPITA assumes the account balance reported is after the benefit (end of year).
* Private vs Government
  + All income from superannuation is either from Government or non-Government (Private) sources. **This split is done using an imputation.**
  + All Private from superannuation is assumed to be an allocated pension. (In reality private super can be annuities, or lump sums).
  + All allocated pensions are subject to drawdown requirements. (In reality accounts that commenced during the financial year are not subject to drawdown requirements)
  + Government superannuation account balances include the full accrued benefit (funded and unfunded). CSS and PSS hybrid fund member statements only report funded benefits (the member component and the productivity component, not the employer financed component), so the SIH may understate the amount of the true Government superannuation account balance.
* Tax assumptions
  + All Private superannuation is from a taxed source. These funds are currently the majority of super funds. The benefits consist of a taxable component with a taxed element, and a tax free component.
  + All income from Government funds are unfunded defined benefit funds. These funds have a large portion of their benefit as a taxable component with an untaxed element, with the rest being a tax free component. The taxable component is subject to different rates of tax and may attract different rates of tax offsets depending on the person’s age.

### Adjusted fringe benefits income

The previous section on income concepts explained the adjusted fringe benefit and reportable fringe benefit concepts. The SIH contains information on the amount of salary sacrificed and non‑salary sacrificed fringe benefits after tax. This amount is only reportable above the minimum reportable threshold.

To gross up the adjusted fringe benefit amount to obtain the reportable fringe benefit, the top marginal tax rate, the Medicare levy rate, and the Temporary Budget Repair levy (for 2014-15, 2015‑16 and 2016-17) are used.

### Adjustment to ages for age pension eligibility purposes

Because static microsimulation models leave the characteristics of the individuals in the underlying survey largely unchanged (only accounting for changes to demographics and incomes by reweighting and uprating), they can have difficulty reflecting changing policy environments. For example, a person who qualified for the Age Pension in the underlying survey, and so had stopped working, remains the same age in the estimated population for future years. However, in these later years people of this age may not have the option of receiving the Age Pension. If the new policy settings are implemented in the model without any adjustment, such people in the estimated future population will not be working or receiving any payments. This is not realistic because people would be expected to respond to the policy change (for example, by working longer or by testing their eligibility for a different payment).

The selected approach for correcting this issue in CAPITA is to increase the age of individuals aged 65 in the survey year to age 66, for the 2017 and 2018 basefiles only. This is because, for example, and assuming all people are born on 1 July for simplification, a person aged 65 in the 2017-18 model year would have an age pension eligibility age of 65.5, and hence they would be removed from the age pension if their age was not increased. The following table shows age pension eligibility ages for a person aged 65 in each of the tabulated years in the first row, and for a person aged 66 in each of the tabulated years in the second row.

**Age pension eligibility ages**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **2011-12** | **2016-17** | **2017-18** | **2018-19** | **2019-20** | **2020-21** | **2021-22** |
| **65 year old** | 65 | 65 | 65.5 | 65.5 | 66 | 66.5 | 66.5 |
| **66 year old** | 65 | 65 | 65 | 65.5 | 65.5 | 66 | 66.5 |

The table shows that, for the current version of CAPITA, the adjustment is only required for 65 year olds, and only for the 2017 and 2018 basefiles. However, this will need to be re-assessed for each SIH update and/or when the model is extended beyond the 2018-19 financial year.

## Benchmarking

### Overview

The benchmarking module is the final step in the basefile creation process. Together with income uprating, it statically ages the population by changing the **weights** of the relevant records to meet the various cohort statistics (i.e. **benchmarks**) subject to certain **constraints**. The module uses the GregWt.sas program to generate the weights. GregWt has been provided for open use in CAPITA by a third party.  No support or advice is available for this software.  This software does not conform to CAPITA coding protocols.

Each person observed on the SIH comes with an accompanying weight, which approximates the number of similar people that are estimated to exist within the Australian population. However, the SIH weights are based on the time the survey was conducted, and so new weights are required for each year to align the survey year with future population distributions. Furthermore, while the SIH has already been benchmarked broadly it may contain sampling errors at some levels of disaggregation. Accordingly, benchmarking helps to shape the population into its expected distributional form, for key population characteristics of interest, for each year the model runs.

#### The benchmarks

The benchmark categories can be broadly classified into demographic benchmarks (applicable to the whole population, e.g. age by sex) and administrative benchmarks (applicable to transfer payment recipients only, e.g. payment type). For the survey year and each year up to but excluding the current financial year, CAPITA uses actual available historical data for benchmarks. For subsequent years, the benchmarks are forecasts from Commonwealth Departments. Each benchmark category gives the number of people across a distribution by those characteristics, so each benchmark within a benchmark category is mutually exclusive to other benchmarks in that category. There may be overlapping of the population across benchmark categories.

CAPITA currently benchmarks to the number of people by selected population characteristics. Each benchmark within a benchmark category can be interpreted as the number of people in a cohort at each point in time over a financial year and so is unchanging over that period. It is calculated from the average of monthly point-in-time data over that year. While, in reality, the population is dynamic as people move in and out of a cohort, CAPITA applies benchmarking on an annual basis, which takes into account the trade-off between the complexity of conducting a shorter benchmarking period and the accuracy of the results. In any case, benchmarking can only be done at a point in time and hence requires the assumption that the number of people in each benchmark cohort remain constant over the benchmark period. In choosing the benchmark period, noting that significant population fluctuations generally occur across the span of a year or more, it is sensible to assume a constant population and a benchmark period of a financial year.

The current benchmarks were selected as appropriate for use of CAPITA as a general purpose model. It should not be taken as the optimum set of benchmarks for all types of analysis. Depending on the type of analysis, the model user should consider the appropriateness of the benchmarks used, and whether additional benchmarks should be included or existing one modified or removed. Some things the user should consider when selecting benchmarks include:

* Value added: How close does the modelled population already come to known aggregates, without including a benchmark? There is a trade-off between meeting a particular set of benchmarks and the broader unintended distortions to other partial distributions.
* Distortions: How much change does it cause to existing SIH distributions? The SIH is already benchmarked to broad aggregates and is generally representative of the population. While meeting one set of benchmarks, the changes can affect the partial distribution of other characteristics not benchmarked to.
* Coherence with other benchmarks: Does the benchmark pull the population distribution in the opposite direction to other benchmarks? It is important to be aware of any inconsistency either between a benchmark and the underlying SIH, or between multiple potential benchmarks, particularly where there are different assumptions underlying projections.
* Sample size: Are there enough records in a benchmark cohort? Where possible benchmark to wider categories and limit the number of benchmarks if they do not greatly improve the accuracy. Smaller, finer benchmarks coming from different sources increase the expected variance within weights and make non-convergence in GREGWT a real possibility.

All benchmarks currently used include the NPD population, with the exception of the number of households by State benchmark. The benchmarking has taken this into account and excluded NPDs from this benchmark.

#### The weights

The weight is a number attached to each income unit (that is each record), which represents the approximate number of income units with those characteristics. When performing analysis, the records should be weighted by this number. The SIH provides the weight for the survey year and the benchmarking module then updates this weight using selected benchmark data for each year the basefile is created.

The SIH weight defines the population distribution for the survey year and is used as the input weight for GregWt.sas to benchmark in the first year. The output weight becomes the weight for the basefile for that year. It is then grossed up for aggregate population changes and used as the input weight for GregWt in the following year. This process continues through to the final year. The pre‑weighting process, whereby weights are manually adjusted for aggregate population growth before benchmarking, assists in moving the input weights closer to the final weight and reduces the stress placed on GregWt.

The largest changes GregWt makes to the input weights usually occurs in the survey year. Changes in subsequent years are generally smaller as the input weights have been more closely aligned to previous benchmarks and benchmarks generally only change gradually over time.

#### The constraints

GREGWT uses a general regression method in order to generate weights by simultaneously hitting several population benchmarks, while restricting the weighting process using the user specified limits. These constraints ensure the results remain sensible after benchmarking by GregWt. The constraints include restricting the size of the movement of initial weights; the tolerance for how far the final weight is allowed to be from the benchmark; and the application of integrated weighting at the household unit level.

GregWt allows the user to specify the degree of movement that is allowed for an input weight, currently LOWER and UPPER is set to 50 per cent and 300 per cent of input weight. The limits are set to ensure the weights do not move too far from the starting weight. The user can also specify how far the final weight aggregate is allowed to be from the benchmark, currently set to 10 per cent, to ensure the benchmarks are met with an appropriate amount of tolerance for deviation. The integrated weighting ensures related units (e.g. income/family/household) are assigned the same weight.

### Benchmarking code

To ‘hit’ the benchmarks CAPITA uses GregWt.sas. GregWt.sas uses the general regression method to simultaneously hit all benchmarks at the same time. This is done separately for each basefile. The purpose of the benchmarking module is to feed the appropriate data into GregWt.sas to enable this process.

The code begins with %Initialise to specify:

* the years to benchmark to
* where the benchmark.xlsx (containing the benchmark data) is saved
* where the GregWt.sas is saved
* where the benchmarking module is located, and
* which benchmarks to use

%BenchIn then reads in all the benchmark data from benchmarks.xlsx.

The remaining code iterates through each benchmark year. RunCapita is run to generate the flags for receipt of transfer payments used in %CreatePreBenchmark that labels the relevant records that are to be benchmarked.

%GregWtWriter compiles the code which is input to GregWt, defining input such as the dataset to be benchmarked, the name of the output dataset containing the weights, and certain constraints. These are discussed further in the next section.

The remaining code in %RunBenchmarking merges the newly created weights onto the basefile.

#### The benchmark input spreadsheet

The benchmarks.xlsx contains all the benchmarks used in the benchmarking process. Each sheet contains the number of people in each benchmark for the benchmark category, and for each year from the survey year to the available projection years. Column 1 is the name of the benchmark category and the list of benchmarks in the category. Row 1 is the year of the benchmark.

#### Adding or removing benchmarks

The model user should consider the appropriateness of benchmarks they use for their analysis. The benchmarking module allows inclusion of new benchmarks, removal of existing ones, and using different benchmarks for different years.

##### Create or delete or modify benchmarks

The user can choose the list of benchmarks to use by defining the BenchList&year macro variable. At the same time these benchmarks need to exist in the benchmarks.xlsx, and the relevant records labelled in %CreatePreBenchmark.

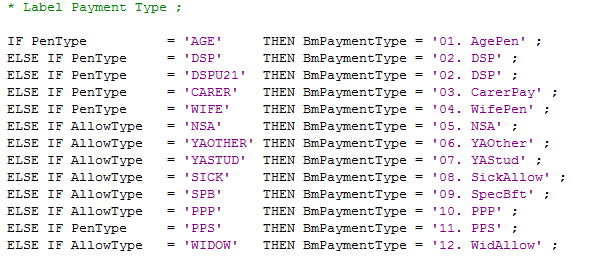
To delete existing benchmarks simply drop them from the BenchList&year. The benchmark.xlsx and the labelled records should also be changed accordingly.

##### Apply different benchmarks for different years

By default only one benchmark list is defined in %Initialise, which is applied for all benchmark years. The user is able to create a different list of benchmarks for each year. To include new or modified benchmarks follow the instructions above.

#### Interaction with the policy modules

To label the correct records for benchmarking, the module uses information from the basefile, and variables modelled in the policy modules such as PenType and AllowType. To label these modelled payments the benchmarking module calls RunCapita to run on the unweighted basefile and then adds benchmark flags based on the output. See figure below.



#### GregWt inputs

The main purpose of the benchmarking module is to prepare the basefile for GregWt, by feeding in the right input and specifying the right parameters for the GREGWT macro.

The dataset containing the units which we wish to reweight is specified using the parameter *UNITDSN*. This dataset must contain a variable which specifies starting weights for each of the units. This variable is specified under the parameter *INWEIGHT*. Note that as the algorithm minimises movement in weights it is desirable to have the starting weights as close to reality as possible. The output dataset will contain an observation for each unit containing the units’ starting and final weights, named as specified by the *WEIGHT* parameter. This dataset will be stored with the name and location specified by the *OUTDSN* parameter*.* Additional variables can be kept on this dataset by specifying them in the *ID* parameter.

The parameter *BnDSN* specifies the dataset which contains information on the nth benchmark. The *BnTOT* parameter specifies the variable on this dataset which contains the aggregate total to benchmark to. The default setting is to aggregate the sum of the weights to the benchmark. However, if the benchmark involves aggregating a different variable in the input dataset, this can be specified using *BnVAR*. Sometimes we will wish to benchmark distinct subsets of the population to different totals. For example, we may wish to benchmark by age groups. If this sub-setting is specified using a variable in the input dataset, this can be specified using the *BnCLASS* parameter*.*

The GREGWT macro allows the micro units to be grouped, and then ensures that each member of the group is given the same weight. If the groupings are specified by a variable in the input dataset then specifying this variable in the *GROUP* parameter achieves this. Benchmarking can also be performed at a group level using the *BnGROUP* parameter to specify the grouping for the nth benchmark; if this is not specified benchmarking is performed at the unit level. Multiple variables can be passed to the *GROUP* parameter. Note that the input dataset must be sorted by the grouping.

The *EPSILON* parameter specifies how closely the final weighted estimate must be to the benchmarks for the benchmarking to be considered successful. The benchmarking is completed if the ratio of the difference between the benchmark and the estimate and the benchmark itself is less than the epsilon value for all benchmarks.

The *UPPER* and *LOWER* parameters specify the allowable magnitude of the weights. This can be expressed as either an absolute value or a percentage of the starting weight. Equations in terms of variables on the input dataset are also permitted.

### Interpreting the benchmark output report

GREGWT produces three forms of output; a report on overall convergence, a benchmark report and an extreme unit report. This section of the note summarises this.

#### Report on Overall Convergence

An example of the convergence report is given in Table 5.1.1 below.

Table 5.1.1: Report on Overall Convergence

| **iters used** | **result code** | **!---- non- nils** | **unit nils** | **count -ve input** | **----! -ve final** | **!---- non- nils** | **group nils** | **count -ve input** | **----! -ve final** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **4** | **C** | 41113 | 0 | 0 | 0 | 22366 | 0 | 0 | 0 |
|  |  | **41113** | **0** | **0** | **0** | **22366** | **0** | **0** | **0** |

The “iters used” column reports the number of times that the algorithm had to repeat until the algorithm reached weight estimates that either met the constraints or could not be improved upon.

The “result code” returns a single letter which reports whether the benchmarking was successful. The letter ‘C’ represents a successful convergence. A letter ‘I’ represents the occurrence that the algorithm converged to a best estimate which did not meet the weight constraints specified. Note that in this case, the weights are truncated to meet the movement constraints, meaning the aggregates will no longer meet the benchmarks. The letter ‘N’ implies that the algorithm was unable to converge.

The remaining columns report on the sign of weights at both the unit and the group level. Reading across the columns, there are 41,113 units with non-zero weights, no units with zero weights, and no units with either negative input or final weights. At the group level, there were 22,366 different groups, all with positive final weights. (The remaining three columns report that there were no groups with zero or negative weights.)

If the algorithm successfully converges (that is, the result code is ‘C’), then the benchmarks have been met. If making minor adjustments to an existing process, which has previously yielded satisfactory results, then successful convergence is likely to mean that these new weights are also suitable. However, if the benchmarking is being run for a first time, or large changes have been made to either the underlying data or the benchmarking process, it may be worth looking at the extreme unit summary (which is discussed later).

If there is non-convergence of the algorithm (result code ‘N’) then the weights returned are unlikely to be at all appropriate. However, if the result code is ‘I’ then it may be the case that only be one or two benchmarks have failed to be obtained, and that the non-convergence may warrant adjustments to the benchmarking process. This can be ascertained by looking at the benchmark reports.

#### Benchmark Reports

An example of a benchmark report can be seen in Table 5.1.2 below. This is a benchmark report for one specific benchmark.

Table 5.1.2: Benchmark Report

| **EDSTATUS** | **count** | **bench -mark** | **vs** | **input weight** | **final weight** |
| --- | --- | --- | --- | --- | --- |
| **1** | 51132 | 1.89E7 | > | 1.639E7 | .. |
| **2** | 2231 | 850731 | > | 764615 | 850746 |
| **3** | 2529 | 888198 | > | 759549 | .. |

Each row contains the different classes of the variable specified in the *BnCLASS* statement (where *n* is the benchmark number).

The count variable lists the number of observations within each class. A reasonable rule of thumb to apply is that it is inappropriate to benchmark a subsample of the data on which making any inference would feel uncomfortable. Note that the value of such a number comes down to personal preference and may well differ between users.

The benchmark column gives the target benchmark for each class, and the input weight gives the initial weighted aggregate given by the starting weights. The ‘vs’ column indicates if there are large differences between the starting estimate and the target estimate. This is indicated using greater than/less than signs. Double signs indicate extremely large differences. The right most column gives the final weight. The two full stops, ‘..’ represent that the benchmark has been exactly achieved.

Recalling that the GREGWT algorithm works by minimising the distance between the starting and the finishing weight, large differences between the initial estimate and the benchmark are cause for concern. Forcing large changes in weights can potentially distort the population sample. If large differences exist then judgement needs to be applied. It may be the case that the sample is not representative of the population of interest, and an alternate data source should be considered for analysis. If there is a common relationship between all of the starting estimates and benchmarks (for example, above all of the starting weights are too low), a preliminary stage of reweighting may be useful. (For example, the user may wish to reweight the data to hit one benchmark using a simple ratio weighting methodology.)

These benchmark reports are also output in the form of SAS datasets. The report for the nth benchmark is output into a dataset called *BENOUTn*. The variable *\_count\_* gives the number of observations in each class, *\_best\_* gives the final estimates, and *\_iest\_* gives the starting estimate. If the benchmarking has met the criteria for the class then the variable *\_report\_* will take on the value ‘.’. This allows identification of the classes where convergence has not been possible. The *\_crit\_* variable returns the ratio of the difference between the benchmark and best estimate and the benchmark. Thus the larger this value the worse the estimate is.

As discussed previously, for the purpose of producing weights for a microsimulation model, it is often necessary to accept weights that do not perfectly meet the benchmarks, providing they are approximately accurate. The above code allows the specific benchmarks for which convergence has not been possible to be quickly identified. Once this has been done the user can judge whether the aggregates for categories where convergence has not occurred are appropriate. Often a compromise will need to be reached between hitting the benchmark and preventing large movements in weights for some units.

#### Extreme Unit Reports

The GREGWT macro also produces a dataset containing “extreme units”. These are units that have weights that:

* Are in the first or final percentile of the weights distribution;
* Have undergone relatively large changes in weight. That is units which lie in the in the first or final percentile of the distribution of the ratio of final and initial weights;
* Have weights which been truncated due to limits in weight movements, or;
* Have zero or negative weights.

This dataset, stored in the work library, is named *\_extout\_*.

The dataset includes all of the variables passed to the GREGWT macro specified in *BnVAR* parameters and the *ID* parameter in the GREGWT macro call. It also contains variables *\_finwt\_*, which give the final estimated weight, *\_regwt\_,* the weight estimated by the regression process prior to any truncation, *\_inwt\_*, the starting estimate and  *\_wtrat\_*, the ratio of the final and starting weights. There is also a variable called *severity*. If this variable has a large magnitude this indicates that the weight is “extreme” in both size and the change in weight. The sign of the severity variable indicates whether the weight is extreme in a small or large sense. (Negative severity indicates that the weight is relatively small.)

The existence of units in this dataset is not itself a problem. (There will always be points in the tails of distributions.) However, the dataset is useful for identifying issues that may exist with a reweighting. In particular, it is useful for identifying distortions occurring in the underlying data due to particular benchmarks. Comparing frequency tabulations for the *BnVAR* variables for the dataset containing all units and the *\_extout\_* dataset allows the user to identify any characteristics that are more prevalent amongst the extreme units relative to the entire sample.

Extreme units are likely to occur due to under or over representation of some characteristic in the sample relative to the population.

1. ABS publication ‘Demographic Statistics, Australia, June 2013’, Catalogue Number 3101.0, released 17 December 2013. [↑](#footnote-ref-1)
2. As per page 61 of the [ABS Survey of Income and Housing, User Guide, 2011-12](http://www.abs.gov.au/AUSSTATS/abs@.nsf/DetailsPage/6553.02011-12?OpenDocument). [↑](#footnote-ref-2)
3. Note that the Merge module is first called for the Survey Year, and so does not require the variables created in *VariableConstruct2* for the first call. [↑](#footnote-ref-3)
4. The SIH identifier variables take the following forms: SihHID contains 6 characters which just label the SIH that the variables have been collected in, followed by 7 numbers which uniquely identify the household. SihFID is a two digit number which specifies the number of the family within the household (note that this is not a continuous numeric scale – see the Variable Register for details). SihIUID is a single digit number which specifies the number of the income unit within the family. Finally, SihPID[r,s,1-4] is a single digit number which specifies the number of the person within the income unit. [↑](#footnote-ref-4)