Mobile Data Collection using CSPro

CSPro version 7.1

International Programs
Population Division
U.S. Census Bureau

4600 Silver Hill Road
Washington, DC 20233

cspro@lists.census.gov

Table of Contents

Table of Contents 1
Session 1: Overview of CSPro, Dictionary and Forms 2
Session 2: Skips and Prefills 12
Session 3: Consistency Checks 18
Session 4: More Logic in Data Entry, Rosters, Subscripts and Loops 25
Session 5: CAPI Features 31
Session 6: Lookup files, Navigation & System Control 41
Session 7: Multimedia & GPS 52
Session 8: Menu Programs 59
Session 9: Synchronization 71
Session 10: Batch Edit and Export 80

Session 1: Overview of CSPro, Dictionary and Forms

At the end of this lesson participants will be able to:

- Identify different CSPro modules and tools and their roles in the survey workflow
- Create a simple data entry application including dictionary and forms
- Run a data entry application on Windows
- Run a data entry application on mobile and retrieve the data entered
- View data using DataViewer

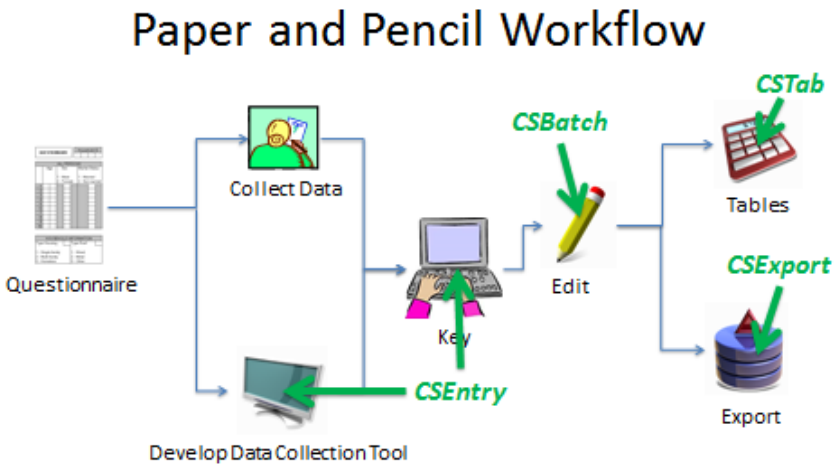
CSPro Overview

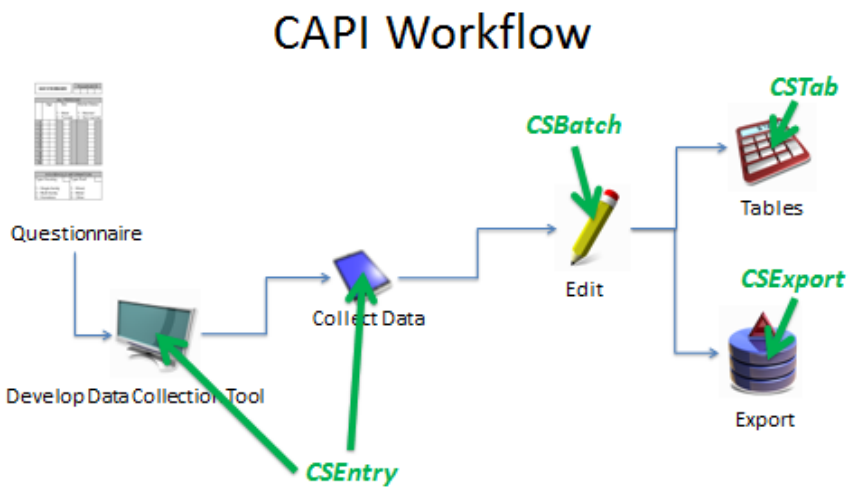
CSPro is a suite of software tools for census and survey data processing that includes modules for data collection, editing, tabulation, and dissemination.

CSPro has a long history. It was first released in 2000 and has been used in over 100 countries worldwide. It has been used for censuses all over the world as well as for many large and complex household surveys including the Demographic and Health Survey (USAID), Multiple Indicator Cluster Survey (UNICEF) and Living Standards Measurement Study (World Bank). The first Android version was released in 2014. CSPro Android has already been used in production for household surveys and population censuses in multiple countries.

CSPro is free software developed by the US Census Bureau and funded by USAID. The Census Bureau provides free email customer support. You can send questions to cspro@lists.census.gov. There is also a user forum where you can post questions and see answers to questions posted by other users at http://www.csprousers.org/forum/.

CSPro can be used for both the traditional PAPI (pencil and paper interview) workflow as well as the computer aided personal interview (CAPI) workflow. In this workshop, we will focus on data collection in CSPro using a CAPI workflow.





Required Hardware and Software

To run the examples in this workshop you will need the following:

- Windows PC with CSPro Version 7.1 or later
- Android tablet with CSEntry Version 7.1 or later

Using CSPro Android

Before we learn how to build data entry applications in CSPro, let's try doing some data collection to get comfortable with the system.

Group Exercise

Split into groups of 3-4 people and use the provided tablets to interview each other using the "Getting to Know You" application. Interview each member of the group so that we have data for all workshop participants. When you are done, tap the sync button (🔄) to upload your results to the server.

Creating a Data Entry Application

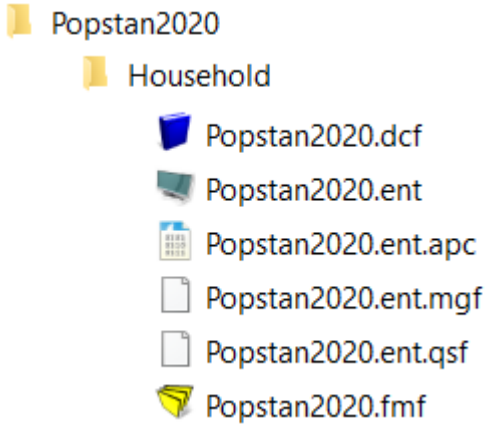
For most of the examples in this workshop, we will be creating a data entry application based on the Popstan 2020 Census questionnaire included with the workshop materials. With CAPI applications, a paper questionnaire alone is not sufficient to define a data entry application. We also a need a specification document that describes consistency checks, skip patterns, text fills, error messages and other aspects of the interactive application that are not defined on a paper questionnaire. Take a moment to review both the questionnaire and the accompanying specification document.

While CSPro data entry programs can be run on Android and Windows[1] tablets and phones, to develop a survey or census application in CSPro you will use the CSPro Designer which runs on Windows PCs. When you launch CSPro Designer you are given the choice of "Data Entry Application" for key from paper (PAPI) and "CAPI Data Entry Application" for electronic data collection using phones/tablets/laptops. The differences are:

- System controlled (tightly controlled path)
- CAPI question text
- Extended controls (radio buttons, checkboxes, date picker, …)

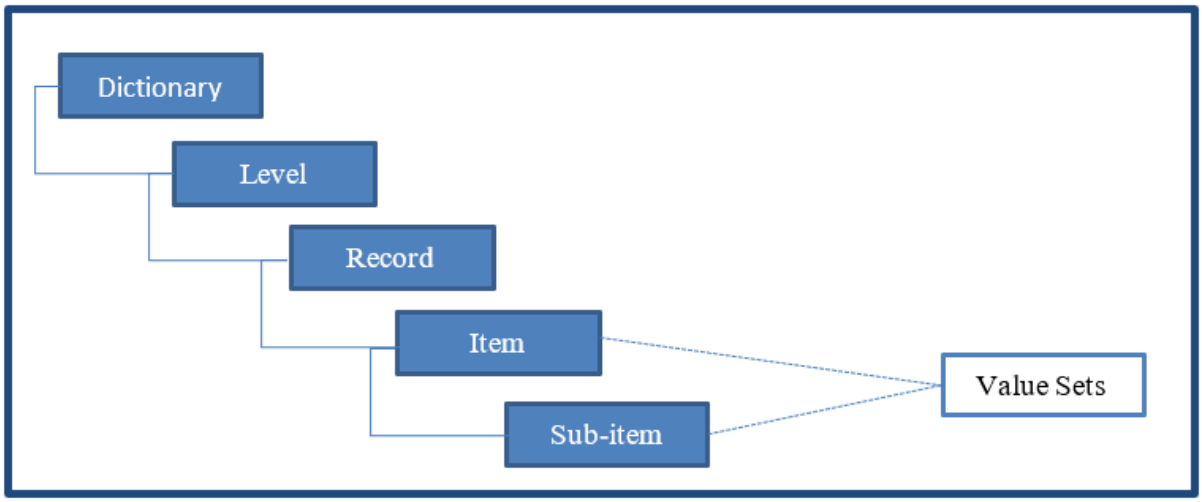
[1] The mobile version of CSEntry for Windows runs only on phones and tablets running Windows 10 or later

Since we are creating a CAPI application we will choose “CAPI Data Entry Application”. We will name the application “Popstan2020” and we will use the same name for the dictionary. Since we will eventually add other applications such as the listing questionnaire and the menu, we will create the following folder structure:



The Data Dictionary

The first step in creating the application is to define the data dictionary. The data dictionary lists all the data items and possible responses that will be in the application and organizes them in records and levels. The dictionary has the following hierarchy:



Before defining the record and items we first create ID-items. ID-items uniquely define each case (each questionnaire). Usually these are geographic codes.

What are the ID-items for the example questionnaire?

Province, District, Enumeration Area, Area Type, and Household Number.

Why not include GPS, interview date, start and end time since they are in the same section of the questionnaire? Because they are not part of the codes needed to uniquely identify the questionnaire.

Add the id-items to the dictionary

- Province (numeric, length 1)
- District (numeric, length 2)
- Enumeration area (numeric, length 3)
- Area type (numeric, length 1)
- Household number (numeric, length 3)

Properties of dictionary items:

- **Label:** text displayed to interviewers (not the full question text, we will see where to put that in a later lesson)
- **Name:** used to refer to variable in CSPro logic (interviewers will never see this)
- **Start:** column number of first digit (character) of variable
- **Len:** number of characters/digits used to store variable
- **Data type:** alpha for names and other text, numeric for numbers and coded responses
- **Item type:** whether variable is item or sub-item
- **Occ:** number of occurrences for variables that are repeated in record
- **Dec:** for numbers with fractional parts, number of digits after decimal point
- **Dec char:** whether or not to include decimal point in decimal numbers
- **Zero fill:** for numeric values; whether to add zeros or blanks to left of number when number of digits in value is less than the length of the variable. Using this will avoid problems when dealing with subitems. We can enable zero fill by default on the options menu.

> Tip: Note that you can toggle showing names or labels in the dictionary tree on the left side of the screen using the View menu. You can also select "Append names to labels in tree" to show both at the same time.

What are the records used in our survey?

Record Type	Record Name	Section(s) of the Questionnaire
A	INTERVIEW_REC	A. Non id-items (A6-A10)
B	PERSON_REC	B. Demographics, C. Education, D. Fertility,
E	DEATHS_REC	E. Deaths of Household Members in the Past 5 years
F	HOUSING_REC	F. Housing Characteristics
G	POSSESSIONS_REC	G. Household Possessions

Note that we could have separate records for education and fertility but instead we will combine them with the person record. This will simplify analysis later on since we will not have to link the records together. Later on, we will see that even with the records combined we can still put education and fertility into separate rosters on our forms.

Let's start by just creating the housing and person records.

Properties of records:

- Label/name: same as for variables.
- Type Value: to distinguish between different records in the data file.
- Required: whether or not the record must be entered for the questionnaire to be complete.
- Max: for multiply occurring records the maximum number of occurrences allowed. Generally 1 for singly occurring records (like housing) and a larger number for repeating records (like 50 for household members). Note that CSPro doesn't allocate space in the data file for occurrences that are not used so it is better to err on the side of caution and allow extra occurrences.

What are the properties of the housing record?

- Type value: F (can use anything but nice to use something meaningful like section letter)
- Required: no (we will not collect section F for vacant/refusal)
- Max: 1

for the person record?

- Type value: B
- Required: no (we can have empty households)
- Max: 30 (questionnaire has limit of 10 but no penalty for adding a few extra just in case)

Now add some fields to the person record:

- Person number (numeric length 2)[2]
- Name (alpha length 30)
- Relationship (numeric length 1)
- Sex (numeric length 1)
- Age (numeric length 3)

And to the housing record:

- Number of rooms (numeric length 2)
- Type of main dwelling (numeric length 1)

A note on variable naming

Different people have different styles of naming dictionary variables. Some use a descriptive name such as "PLACE_OF_BIRTH" others prefer to use the question number such as "B07" and others prefer a combination such as "B-7_PLACE_OF_BIRTH". Whichever approach you choose just make sure that it will be easy for users of your application and your data to understand. Will everyone working on the logic for your application know what B07 is?

For each of our variables we need to add the possible responses (value sets). The value set lists all valid responses along with their corresponding labels for coded variables. Without a value set, the interviewer can enter any value (except blank) but with a value set they are limited to the options defined in the value set. Without a values set, users can even enter negative numbers. For this reason, it is good practice to use a value set for all numeric variables.

Define the value sets for some of our variables based on the response codes on the questionnaire:

- Area type (1- Urban, 2- Peri-urban, 3- rural)
- Province (copy/paste from Excel)
- Person number (range: 1-30)
- Name (no value set)
- Sex (1- Male, 2-Female)
- Relationship (see questionnaire)
- Age (use a range: 0-120, plus don't know code 999)
- Number of rooms (use generate value set to generate the codes 1-20)
- Type of main dwelling (use codes from questionnaire)

[2]The line number is not needed in CSPro itself as there are ways to determine the row number using logic, however, when exporting the data to other packages it is often useful to have it. We will see later how to fill this in automatically during data entry.

Dictionary Macros

There are some useful functions for working with dictionaries that you can access by right-clicking on the dictionary in the tree on the left side of the screen and choosing "Dictionary Macros". In particular you can copy/paste all value sets or all item names/labels from the dictionary to/from Excel. This can be used to create codebooks to share with people who do not have access to CSPro. It can also be used to do bulk modifications on dictionary items such as renumbering values in value sets or adding prefixes to item names.

Forms

Before we can enter data, we need to create data entry forms. To start, click on the yellow stack of forms on the toolbar. To follow the look of the paper questionnaire we will create one form for each page of the paper questionnaire.

Create a form for section A: Identification. Drag and drop the id-items onto the form. Note that we can drag and drop individual items or entire records. By right clicking on the form in the forms tree on left side of the screen we can change the label and name of the form. Let's make the label "A: Identification" and make the name "IDENTIFICATION_FORM".

Create a form for section B: Demographics. Drag drop the items from the person record. Let's give the form label "B: Demographics" and name "DEMOGRAPHICS_FORM". Note that when we drop the record we have the option to put the items in a roster or a repeating form. If we drop the items on the household identification form, we can only roster since the household identification isn't repeated. For our example let's use a roster.

When we create the rosters, CSPro automatically gives them a name that ends in "000", for example "PERSON000". You can see this in the forms tree on the left side of the screen. We can change this by right clicking on the roster in the forms tree and choosing properties. Let's name our roster "DEMOGRAPHICS_ROSTER".

Create a form for section: F: Housing Characteristics. Drag and drop the items from the housing record.
For paper and pencil surveys, we would spend a lot of time on the layout, adding additional labels and frames to make the form look exactly like the questionnaire. However, when rendered on Android, the form is rendered one question at a time so making the form look like the paper form is not as important.

Running the Application on Windows

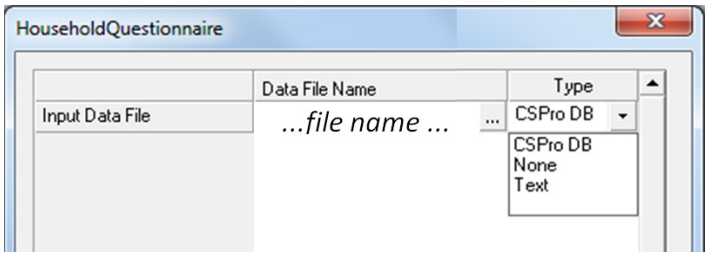
Click on the green light icon on the toolbar to run the data entry application on your PC. Enter the name of the data file. For data files CSPro uses the file extension ".csdb". Let's name our data file "Household.csdb".

Data File Types

By default CSPro stores data in a CSPro database file with the extension .csdb. This file not only contains the data itself but it also contains the notes, the index, the partial save status and metadata used for data synchronization. CSPro database files can be viewed by double clicking on them or using the DataViewer tool from the tools menu.

The csdb extension is new in CSPro 7.0. In earlier versions of CSPro, the notes, index and partial save were stored in extra files that accompanied the data file itself. Unlike the text file used by earlier versions of CSPro, this is a binary file that cannot be viewed using TextViewer. While it is still possible to use text files in CSPro 7.0, it is highly recommended to use CSPro DB instead. There are new features such as smart sync and case labels that are not supported using text files.

When you launch a CSPro data entry application you can select the data file name and the type of data file that you want to use:



- **CSPro DB:** This is a CSPro Data Base File. It contains the data and metadata.
- **None:** There is no associated data file. This is useful when building menus.
- **Text:** This is a UTF8 Text file. This is what CSPro has used in previous versions and is provided for compatability.

Controlling the size of the roster

Our program at this point has no way to stop entering household members unless we fill in all 50 rows of the roster. To limit the number of rows in the roster to the number of actual household members we can have the interviewer enter the number of household members and use that to set the size of the roster. Add a new variable, “number of household members”, to control the size of the roster. Which record should we add it to? It can’t be on the person record since we don’t want it to repeat so let’s add it to the household characteristics record but put it on the person (section B) form. Right click on the roster, choose properties and set this new field as the occurrence control field.
Try running the application again. Unfortunately, when we get to the person form we go into the roster instead of the “number of household members” field. By default, CSEntry goes in the order in which the fields were dropped on the form. We can change the order by dragging the fields in the forms tree. Move the “number of household members” field up in the tree so that it comes before the roster. Now the roster control field should work. In the next lesson we will use logic to exit the roster without the roster control field.

Running the Application on Mobile

To put the application on a phone/tablet we need to do the following:

1. Publish the .pen file (File menu -> Publish Entry Application).
2. Connect the tablet to the PC using a USB cable. The tablet should show up like a USB drive.
3. Copy the files Popstan2020.pen and Popstan2020.pff into the CSEntry directory on the tablet.
4. Start CSEntry on the tablet.
5. Enter the data (note the differences between Mobile and Windows).
6. When you are done, connect the tablet back to the computer and copy the file Household.csdb from the tablet back to the PC. One some tablets (Nexus 7 for example) the first time the data file is created it may not show up when connected to the PC. If this is the case, reboot the tablet and it will.

When we copied the application to the tablet we copied the pen and the pff file. The pff file contains various parameters about how to launch the data entry application including the data file to use. You can modify the pff file by right clicking on it in Windows Explorer and choosing “Edit”. This allows you to modify the name of the data file, force the application to start in add or modify mode and to lock various parts of the user interface.

On Android, the list of available applications on the device is constructed by finding all the pff files in the CSEntry directory and subdirectories so a pen file without a pff will not show up in the list.

Partial Save and Case Tree

It can be a pain when testing a question that is on the third form of a survey to have to reenter all of the data up to the question you are testing. We can enable partial save under the data entry options so that we can exit data entry, modify the application, and come back right to where we left off. While we are in the data entry options we can also enable the case tree on both Windows and Android to make it easier to navigate around the questionnaire while we are testing. The case tree is enabled by default on Android since Android only shows one question at a time but it is off by default on Windows. Note that on mobile phones, since there is not enough space to display the case tree and the questions at the same time so you need to tap on the big green “CS” in the top left corner of the screen to bring up the case tree.

In addition to the data entry options, there are other useful settings in the "Data Source Options" dialog. For example, here you can enable automatic partial save at regular intervals. This is important for long interviews where if there is a problem with the software or device part way through the interview you could lose data if the interview has not been partially saved.

Modifying the dictionary after collecting data

Add the variable **F02 number of bedrooms** to the dictionary and form right before F03 type of main dwelling. Don’t forget to modify the order in the form tree so that it is asked before F03. Run the application again and modify one of the existing cases. Why is the number of household members blank? Once data has been entered, you should avoid adding variables in the middle of a record. It is fine to add to the end of the record but adding in the middle invalidates the start positions of the existing variables. If you must add a variable in the middle of a record, you can use the reformat data tool to adjust old data files to match the new dictionary.

Group exercise

Add a new record to the dictionary for section E of the questionnaire (deaths). Name this record "Deaths". How many occurrences should it have? Don't include E01 and E02 in the new record as they do not have the same number of occurrences as the other variables in this section. Instead, add them to the housing record. Create a new form for section E and add the fields onto it to create a roster. Use E02 as an occurrence control field to the roster to limit the number of rows to the number of deaths in the household. Test the application on both Windows and on Android.

Subitems

Let's add the **Date of Birth (B06)** to the application. In order to be able to look at both the date as an 8-digit number and look at the day, month and year individually we can create an item with subitems. Subitems are items that are made up of a subset of the digits of their parent item. Add the item for interview date and the following subitems:

- year of birth (length 4)
- month of birth (length 2)
- day of birth (length 2)

We are putting year first then month and day because this format will work better with other CSPro features that we will see later. Click on **Layout** in the toolbar to ensure that the item and subitem overlap. Add the subitems to the form, add the value sets for each subitem and test the application. Note that when we add the subitems to the form we do not need keep the same order that we have in the dictionary. On the form we can put the day, month then the year.

Linked Value Sets

Note that questions **B07 (place of birth)** and **B08 (residence 1 year ago)** both have the same value set. We could simply copy and paste the value set into both items, however, this would leave us with two copies of the same value set. If later on we change one of them and forget to change the other, then they will be out of sync. This could cause consistency problems later on. Instead, we can create the value set for **B07** and then paste it as a linked value set into **B08**. With a linked value set, CSPro only stores one copy of the value set that is shared between both variables. This way if you edit the value set, the change is reflected in both items.

Multiply Occurring Items

Let's add questions **F04** and **F05** on number of housing units to the application. **F04** is just a yes/no question. **F05** asks the number of units of each of five different types of housing unit. We could make five different variables in the dictionary: HOUSING_UNITS_ROUND_HUT, HOUSING_UNITS_DETACHED, HOUSING_UNITS_SEMI_DETACHED etc… To simplify our dictionary, we can instead make a single variable in the dictionary, HOUSING_UNITS, with five occurrences: one for each type of housing unit. When we drag this variable onto the housing characteristics form we will get a roster with 5 rows.

Occurrence Labels

With this approach, our form does not show the housing unit types but we can fix that by using occurrence labels. Select the housing units variable in the dictionary and choose "Occurrence Labels…" from the Edit menu. Add the names of the five types of housing units in the grid that comes up. Note that you can copy from Excel and paste into this dialog. Now when we drag the variable to the form the roster shows the type of housing unit for each row.

Checkboxes

We could also use a multiply occurring item for question **B10**, disabilities, but that can be implemented more easily using checkboxes. Checkboxes offer a friendly interface for multiple response questions by presenting a single screen with a checkbox for each option rather than presenting the options one by one.

In CSPro multiple response questions are implemented as alpha variables whose length is the same as the number of options that can be selected at the same time. The value set has a value for each option which is usually a single letter. The resulting value is a string containing the values for each of the selected items.

For disabilities we will use the following value set:

Visual	A
Hearing	B
Speech	C
Physical	D
Mental	E
Self-care	F

If the interviewer checks the boxes for Visual, Physical and Mental the value for the variable will be "ADE". We will see later how can convert the alpha value into a series of yes/no values to simplify analysis.

Capture Types

If you right click on the disabilities field on the form and choose "Field Properties…" you will see that the capture type is set to check box. There are other capture types:

- Text Box: keyboard data entry
- Radio button: choose one option from many with radio button for each option
- Drop Down: choose one option from many without radio buttons
- Combo Box: combination of keyboard input and drop down (same as drop down on Windows)
- Check Box: choose multiple responses

When you drag an item on the form, CSPro sets the capture type based on the value set for that item. If there is no value set when you drop the item, the capture type will be set to text box. You can always change the capture using the Field Properties dialog.

Date Fields

Let's add the interview date to the identification section. Which record should it go on? It could go on any singly occurring record such as housing but let's create a new record called INTERVIEW_REC to hold the section A items that are not part of the id-items and add it there. We can add an eight-digit item for the interview date. We can also add sub-items for the year, month and day of the interview. Drag the date item onto the form. When dragging don't use the sub-items, just the items. Now change the capture type for the interview date to be Date and set the date format to be YYYYMMDD to match the order of the sub-items year, month and day.

Filling in Names First

Currently the interviewer has to fill in all the demographic information for the first person before moving on to the next person in the household. In practice, it is simpler to have them fill in just the name, age, sex and relationship of all the household members first and then fill in the remaining details only after all the basic information has been entered. To do this we need to pull just the **PERSON_NUMBER**, **NAME**, **SEX**, **RELATIONSHIP**, **AGE** and **DATE_OF_BIRTH** into a separate roster. Delete them from the existing **DEMOGRAPHICS_ROSTER**, create a new form before the **DEMOGRAPHICS_FORM** called **HOUSEHOLD_MEMBERS_FORM** and drop **PERSON_NUMBER** onto it to make a new roster. Rename this roster "**HOUSEHOLD_MEMBERS_ROSTER**". Drop **NAME** from the dictionary on top of the **HOUSEHOLD_MEMBERS_ROSTER** to add it to the roster. Note that when you drop an item from a repeating record on top of an existing roster it gets added to that roster but if you drop it outside the roster it creates a new roster. Drop the **SEX**, **RELATIONSHIP**, **AGE** and **DATE_OF_BIRTH** variables onto the new roster. Remove the **NUMBER_OF_HOUSEHOLD_MEMBERS** field from the **DEMOGRAPHICS_FORM** and drop it onto the **HOUSEHOLD_MEMBERS_FORM**. In the forms tree, move **NUMBER_OF_HOUSEHOLD** members so that it comes before the **HOUSEHOLD_MEMBERS_ROSTER**. Set the occurrence control field for **HOUSEHOLD_MEMBERS_ROSTER** to be **NUMBER_OF_HOUSEHOLD_MEMBERS**.

Tip: When dragging items onto a form with an existing roster drop the items inside the roster or you will end up with a second roster for the new item.

Exercises

1. Add the remaining fields from the housing section of the questionnaire to the dictionary and the housing form (F06 through F13). Make sure to add the appropriate value sets.
2. Add the remaining fields from the demographics section (B) of the questionnaire to the dictionary and to the demographics form. Make sure to add the appropriate value sets. For Language(B15) use checkboxes.
3. Add the fields for section C, Education, to the dictionary. Add them to the person record. Create a new form for section C and drop the education items onto to it to create a roster. Set the occurrence control field for the roster to the number of household members.
4. Add the start and end times of the interview to the identification form after the interview date. Use subitems for the hours and minutes. Add appropriate value sets.
5. Add a new record and form for section G, household possessions. Since G01 (quantity and value) repeat, put these items in their own repeating record but do not include G02. This roster should include the possession code (1-10), quantity and value per unit. Set the occurrence labels in the roster for G01 to the names of the possessions. Make G02 a singly occurring checkbox field and put it in the housing record since it does not repeat. The value set for G02 should have the possession names as labels with codes “A”, “B”, “C”…

Make sure to test your application on both Windows and Android.

Session 2: Skips and Prefills

At the end of this lesson participants will be able to:

- Use the commands `skip` and `ask if` to skip fields
- Use `if then else` statements to implement skip patterns
- End rosters from logic using `endgroup`
- Automatically fill in fields using logic

Skips

In question **F04** (other housing units), we need to skip over question **F05** (numbers of housing units) if they answer "no".

To skip to a field, we need to use logic. To add logic to a CSPro application click on the logic button  on the toolbar to switch to the logic view. Here we can type in CSPro logic to control the program. Logic is added to a PROC (procedure) usually associated with a field on the form. To go to the PROC for a field in the logic view, just click on the field in the form tree on the left while in logic view. We will add our skip to the PROC of field **OTHER_HOUSING_UNITS** so click on **OTHER_HOUSING_UNITS** in the form tree.

To skip to a field, we use the command `skip`. In order to skip only when the answer is no, we need to combine the skip with an `if` statement as follows:

```
PROC OTHER_HOUSING_UNITS

if OTHER_HOUSING_UNITS = 2 then
    skip to TENURE;
endif
```

The `if` statement only executes the code between the `then` and the `endif` if the condition (`OTHER_HOUSING_UNITS = 2`) is true, otherwise it goes straight to the first command after the `endif`.

This will skip directly to TENURE for individuals who answer “no” to OTHER_HOUSING_UNITS, skipping over the HOUSING_UNITS roster.

In addition to “=” to check if two values are equal, you can use the following operators:

Operation	Symbol
Equal to	=
Not equal to	<>
Less than	<
Less than or equal to	<=
Greater than	>
Greater than or equal to	>=

For example, to skip over **B08**, residence one year ago, for those less than one year of age we could add the following logic to the postproc of the preceding question:

```
PROC PLACE_OF_BIRTH
if AGE < 1 then
    skip to MOTHER_ALIVE;
endif
```

Note that once we have skipped the field **RESIDENCE_ONE_YEAR_AGO**, we can’t get to it by clicking on it or going back from the following field. CSEntry keeps track of the fact that this field was skipped and won’t let us back in until we change the value of **AGE**. This is an important difference between system controlled and operator controlled data entry mode.

We should add a comment to other readers of our logic so that they will understand what we are trying to do. This will not only help others understand our logic but it will help us when come back to it after a few months.

```
PROC PLACE_OF_BIRTH
// Do not ask residence one year ago for those less than one year old
if AGE < 1 then
    skip to MOTHER_ALIVE;
endif
```

Everything on a line after // is considered a comment and is ignored by CSPro. You can also use { } for multiline comments. It is good practice to use plenty of comments to document your logic.

A note on coding style

When writing if statements, your code will be more readable and easier for others to understand if you indent the statements between the then and the endif. It is also helpful to use consistent capitalization. We recommend all uppercase for dictionary variables like **NUMBER_OF_HOUSEHOLD_MEMBERS** and all lowercase for CSPro keywords like `if`, `then` and `endif`. Finally, you should use comments as much as possible to help others, and your future self, better understand your code.

Ask if

An alternative to the skip is the statement `ask if`. It skips the current question if the condition is NOT met. For example, to use `ask if` in order to only ask the question **RESIDENCE_ONE_YEAR_AGO** (**B08**) for those aged 1 and over we can use the following logic:

```
PROC RESIDENCE_ONE_YEAR_AGO
preproc
ask if AGE >= 1
```

The `ask if` statement skips over the question unless the condition `AGE >= 1` is true. In other words, if the age is less than 1, the question will be skipped. Note that when changing from `skip` to `ask if`, the condition was negated by switching from "<" to ">=".

The line `preproc` tells CSPro to execute this logic *before* asking the question. Our previous examples did not specify `preproc`, so by default the logic was executed in the `postproc`. Postproc logic is executed *after* the data are entered. Preproc logic is executed before the data are entered. For each variable, you can specify a preproc, a postproc or both. `Ask if` will always be used in the `preproc` of the field being skipped while `skip` can either be in the preproc of the field being skipped or in the postproc of the preceding field.

Group exercise

Implement the skip pattern for questions **F10** (has a toilet) and **F11** (type of toilet). If the answer to question F10, have toilet, is "no", skip question **F11**, type of toilet. You can use either `skip` or `ask if`. Make sure to add a comment to your code and to format your code using correct indentation and capitalization.

Compound Conditions

As another example, we will implement the skip pattern for question **C01**. We need to skip over **C02**, highest level of education, if school attendance is "never attended" (1) or "don't know" (9). We can implement this with two if statements:

```
PROC ATTENDED_SCHOOL

// Skip over highest level education for those who never attended school
if ATTENDED_SCHOOL = 1 then
    skip to LITERATE;
endif;

// Skip over highest level education for those whose attendance is "don't know"
if ATTENDED_SCHOOL = 9 then
    skip to LITERATE;
endif;
```

Note that in CSPro, logic statements must be separated by semicolons (;). This tells CSPro when one command ends and the next begins. Forgetting to put the semicolon at the end of a statement is a very common error that new users make. If you forget the semicolon, usually CSPro will tell you that that a semicolon was expected, although sometimes it gets confused and gives you a less informative error message.
We can simplify this code by combining the two if statements using the "or" operator to create a compound condition.

```
PROC ATTENDED_SCHOOL

// Skip over highest level education for those whose attendance is "never attended"
// or "don't know"
if ATTENDED_SCHOOL = 1 or ATTENDED_SCHOOL = 9 then
    skip to LITERATE;
endif;
```

In addition to the "or" operator you can also make compound conditions using the "and" operator. A compound statement linked by "or" is true if either one of the conditions are true but a compound statement using "and" is only true if both conditions are true.

In addition to “or”, CSPro supports the following logical operators:

Operation	**Keyword**
Negate an expression	not
True if both expressions are true	and
True if either expression is true	or

You can also use compound conditions with `ask if`. For example we only want to ask question **B14**, age at first marriage, for those who responded "married" or "widowed" or "divorced" for marital status.

```
PROC AGE_AT_FIRST_MARRIAGE
preproc
// Only ask age at first marriage for those whose marital status indicates
// they are/were married
ask if MARITAL_STATUS = 2 or MARITAL_STATUS = 3 or MARITAL_STATUS = 4;
```

An “or” expression is true when either the left or right expressions are true so the question will only be asked if the MARITAL_STATUS is either 2, 3 or 4.

We can make our expression even simpler using the "in" operator which checks if a value is in a range:

```
PROC AGE_AT_FIRST_MARRIAGE
preproc
// Only ask age at first marriage for those whose marital status indicates
// they are/were married
ask if MARITAL_STATUS in 2:4;
```

Other (specify) fields

A common pattern in CSPro is to use the `ask if` statement to implement other (specify) fields. Let's implement the other (specify) field for question **F08**, roofing material. First we add a new alpha field to the dictionary to capture the "other" value. We can call it **ROOFING_MATERIAL_OTHER** and drop it onto the form so that it comes right after **ROOFING_MATERIAL**. We only want to ask for the "other" value if they selected "other" (code 5) for ROOFING_MATERIAL, otherwise we want to skip it. We can do that using ask if in the preproc for ROOFING_MATERIAL_OTHER.

```
PROC ROOFING_MATERIAL_OTHER
preproc

// ask this question only if "Other (specify)" is picked in roofing material.
ask if ROOFING_MATERIAL = 5;
```

Skip to next

Now let's add the skip between sign language (**B12**) and marital status (**B13**) to skip to the next person for household members under 10 years old. In this case, what do we skip to? We can't skip to **B01** (line number) since that will try to send us backwards to the line number for the current row. Instead, we use `skip to next` which automatically skips to the first field in the next row of the roster. We will put this skip in the preproc of **MARITAL_STATUS**. It could go in the postproc of the previous field, however, later on we will will skip the **SIGN_LANGUAGE** question entirely for those who do not have difficulty hearing, in which case we will skip over the skip.

```
PROC MARITAL_STATUS
preproc
// Skip to next person for household members under 10 years of age
if AGE < 10 then
    skip to next;
endif;
```

Let's add the skip in section C, education, for those under 3 years of age. We are only supposed to fill in the education section for those aged three years and above. We can use `skip to next` to skip over the education questions if the age is less than three. We need to do this in the preproc of **C01**, the first field in the education roster, **ATTENDED_SCHOOL**.

Note that since this field already has a `postproc` we need to add the `preproc` first and then add the word "`postproc`" so that the original `postproc` logic does not become part of the `preproc`.

```
PROC ATTENDED_SCHOOL
preproc

// Skip education for household members under 3 years of age
if AGE < 3 then
    skip to next;
endif;

postproc
// Skip highest grade if never attended or don't know
if ATTENDED_SCHOOL = 1 or ATTENDED_SCHOOL = 9 then
    skip to LITERATE;
endif;
```

Prefills

Let's fill in the line number automatically so that the interviewer doesn't have to enter it. We can do this in the preproc of **PERSON_NUMBER**:

```
PROC PERSON_NUMBER
preproc

// Fill in line number automatically
PERSON_NUMBER = curocc();
```

The CSPro function curocc() gives us the current occurrence number of a repeating record or item. In other words, it gives the row number of the roster we are currently on.

To avoid having to hit enter to move through this field we can use the statement noinput which accepts the assigned answer and automatically moves to the next field as if the interviewer had used the enter key.

```
PROC PERSON_NUMBER
preproc

// Fill in line number automatically
PERSON_NUMBER = curocc();
noinput;
```

Alternatively, we can make the field uneditable by checking the protected box in the field properties. With noinput it is still possible to go back and modify the field, but protected fields cannot be modified except from logic. Be aware that if you do not set the value of a protected field in logic *before* the interviewer gets to the field, CSEntry will give you a fatal error.

Group Exercise

In question **B10**, mother line number, automatically fill in the code 88 (deceased) if the response to **B09**, mother alive, is "no". You may be tempted to use the skip statement for this but you should use the noinput statement instead.

Endgroup

Instead of using the occurrence control field in the roster, we could ask the user if they want to terminate the roster. To do that, we first add a new field to the dictionary and it to the roster. Let's call it **MORE_PEOPLE** and give it the value set Yes - 1, No – 2. We will put it at the end of the names roster. If the interviewer picks "no" then we use the command endgroup which terminates entry of the current roster or repeating form.

```
PROC MORE_PEOPLE

// Exit roster when no more people in household
if MORE_PEOPLE = 2 then
    endgroup;
endif;
```

With this, we no longer need to use the occurrence control field of the roster. However, since the other rosters (demographics and education) still rely on the variable **NUMBER_OF_HOUSEHOLD_MEMBERS** as a roster control field we need to set its value after completing the names roster. We can make it a protected field, move it after the names roster and set it using logic:

```
PROC NUMBER_OF_HOUSEHOLD_MEMBERS
preproc
// Set number of household members to size of person roster.
// It is used as roster occurrence control field for later rosters.
NUMBER_OF_HOUSEHOLD_MEMBERS = totocc(HOUSEHOLD_MEMBERS_ROSTER);
```

This uses the function totocc() which gives the total occurrences of a repeating record or item. In other words, it gives us the total number of rows of a roster.

Endlevel

There is also the endlevel command which is similar to endgroup but skips the rest of the current questionnaire (except for two level applications when in a second level node where it terminates the current node).

Skipping Based on Checkboxes

We only want to ask question **B12**, sign language, if the household member is hearing disabled. We want to skip question **B12** if hearing disabled is not checked in question **B11**. How can we tell in logic if hearing disabled is checked? Hearing disabled is option B in the value set so we need to determine if the set of all checked options contains the letter B.
In CSPro logic we can use the function `pos()` which returns the position of one string within another. For example `pos("C", "CSPro")` will be one, `pos("P", "CSPro")` will be three and `pos("o", "CSPro")` will be five. If the search string is not found, `pos()` will return zero. In our case, the letter B is not always at the same position in the string. If the interviewer chooses just option B then B will be the first character in the string but if the interviewer chooses A and B then the result will be "AB" and B will be in position two. However, all we care about is whether or not B is in the result string at all. For this we can check the result of `pos()`. If the result of `pos()` is nonzero then B is in the string and if the result is zero, B is not in the string.

Using `pos()` the logic for skipping sign language if hearing is not checked is:

```
PROC SIGN_LANGUAGE

preproc
//ask this question only if hearing disabled (option B) is checked
ask if pos("B", DISABILITIES) > 0;
```

Exercises

1. Skip question **F07**, monthly rent, if dwelling is not rented in **F06**.
2. In section E, deaths, implement the skip pattern for question **E01**, any deaths.
3. In section E, deaths, implement the skip pattern for question **E09**, died while pregnant.
4. In section E, deaths, skip questions **E09** and **E10** for household members who are NOT women aged 12-50.
5. In section E, use logic to pre-fill the line number field **E03**. Make sure that the line number is protected.
6. Add an additional question to **F13** to determine if the respondent wants to give the distance to water in minutes or kilometers. If they choose kilometers then skip the distance in minutes question otherwise skip the distance in km question.
7. Add the other(specify) field and skip pattern for **F08**, wall material.
8. Add the other(specify) field for **B19**, languages spoken.
9. In question **G01**, household possessions, skip the value field if the quantity is zero.
10. Add variables and a form for section D: Fertility. Add the variables to the person record but create a new form with its own roster just like we did for section C. Don't worry about the last question on the form that displays the total births. We will cover that in a later lesson. Add the skip patterns for the fertility section. Don't forget to skip the entire section for males and females NOT between 12 and 50 years old.

Session 3: Consistency Checks

At the end of this lesson participants will be able to:

- Use the command `errmsg` to display messages to the interviewer
- Use the errmsg with `select` keyword
- Use `if then else` statements to implement consistency checks across multiple variables
- Use the command `warning` to implement "soft" checks
- Create and run test plans for consistency checks
- Implement consistency checks on dates
- Declare and use logic variables

Consistency Checks with Two Variables

Up to now, we have been able to limit the responses for each variable to only a set of valid responses using value sets. What if we want to check for consistency between two different variables?

For example, let's prevent the interviewer from entering more bedrooms than there are rooms in the house. Click on the **NUMBER_OF_BEDROOMS** field in the forms tree and then click on Logic in the toolbar to open the logic editor. Add the following code in the procedure for **NUMBER_OF_BEDROOMS**:

```
PROC NUMBER_OF_BEDROOMS

// Ensure that number of bedrooms is not more than number of rooms.
if NUMBER_OF_BEDROOMS > NUMBER_OF_ROOMS then
    errmsg("Number of bedrooms cannot exceed number of rooms");
    reenter;
endif;
```

The `if` statement only executes the code between the `then` and the `endif` if the condition (`NUMBER_OF_BEDROOMS > NUMBER_OF_ROOMS`) is true. The `errmsg` statement will display a message to the user. The reenter statement forces the interviewer to stay in the current field and does not allow them to advance to the next field.

Note that we put our logic in the proc for **NUMBER_OF_BEDROOMS** and not in the proc for **NUMBER_OF_ROOMS**. This is because when we are in the proc for **NUMBER_OF_ROOMS**, the value for **NUMBER_OF_BEDROOMS** has not yet been entered. When creating consistency checks we always put the logic for the check in the proc for the last field involved in the check.

Let's take a look at another example, the edit specifications call for the following control:

• If **RELATIONSHIP** is spouse, then **MARITAL_STATUS** should not be coded 1 (never married), 2 (divorced) or 3 (widowed).

Which proc should we put this check in? Since **MARITAL_STATUS** comes after **RELATIONSHIP** we will put it in the proc for **MARITAL_STATUS**.

```
PROC MARITAL_STATUS

// Ensure that spouse is not single, divorced or widowed
if (MARITAL_STATUS = 1 or MARITAL_STATUS = 3 or MARITAL_STATUS = 4)
   and RELATIONSHIP = 2 then
    errmsg("Marital status of spouse cannot be single, divorced or widowed");
    reenter;
endif;
```

Note that when combining multiple "and" and "or" expressions, "and" expressions are evaluated first and then "or" expressions are evaluated which can sometimes lead to unexpected results. You can use parentheses to force the order of evaluation you want like we do above. What happens to the above expression without the parentheses if the relationship is not 2 and the marital status is 3?

Of course there are a couple of simpler ways to write this check without using "or". We can use the not equals operator (“<>”):

```
// Ensure that spouse is not single
if MARITAL_STATUS <> 2 and RELATIONSHIP = 2 then
    errmsg("Marital status of spouse cannot be single, divorced or widowed");
    reenter;
endif;
```

Or we can use the keyword "in":

```
// Ensure that spouse is not single
if MARITAL_STATUS in 1,3,4 and RELATIONSHIP = 2 then
    errmsg("Marital status of spouse cannot be single, divorced or widowed");
    reenter;
endif;
```

The in operator will be true if the value matches any of the numbers in the comma separated list. It also supports ranges by separating the start and end of the range with a colon (:). For example `BEDROOMS in 1:4` will be true if **BEDROOMS** is 1,2,3 or 4.

Better Error Messages

We can improve our error message by adding parameters to the string. Let’s go back to the check on the number of bedrooms and display the number of rooms and the number of bedrooms in the message:

```
PROC NUMBER_OF_BEDROOMS

// Ensure that number of bedrooms does not exceed number of rooms.
if NUMBER_OF_BEDROOMS > NUMBER_OF_ROOMS then
    errmsg("Number of bedrooms, %d, exceeds number of rooms, %d.",
           NUMBER_OF_BEDROOMS, NUMBER_OF_ROOMS);
    reenter;
endif;
```

The “%d”s in the message are replaced by the values of the variables that follow in the order that they are listed.

Let’s do the same for the check on marital status and relationship:

```
// Ensure that spouse is not single
if MARITAL_STATUS in 1,3,4 and RELATIONSHIP = 2 then
    errmsg("Relationship is spouse and marital status is %d. Spouse cannot be
single, divorced or widowed.", MARITAL_STATUS);
    reenter;
endif;
```

It would be better if we could include the name of the person in the error message as an additional clue to the interviewer. Since the name is an alpha variable we use %s instead of %d. (%d is for decimal value.)

```
PROC MARITAL_STATUS

// Ensure that spouse is not single
if MARITAL_STATUS in 1,3,4 and RELATIONSHIP = 2 then
    errmsg("%s has relationship spouse and marital status %d. Spouse cannot be
single, divorced or widowed.",
           NAME(curocc()), MARITAL_STATUS);
    reenter;
endif;
```

Why is there a whole bunch of extra space after the name in the error message? Remember that alpha variables in the dictionary are fixed length. This means that they get padded with blank spaces. We can remove the trailing blank spaces by using the `strip()` function.

```
PROC MARITAL_STATUS

// Ensure that spouse is not single
if MARITAL_STATUS in 1,3,4 and RELATIONSHIP = 2 then
    errmsg("%s has relationship spouse and marital status %d. Spouse cannot be
single, divorced or widowed.",
           strip(NAME(curocc())), MARITAL_STATUS);
    reenter;
endif;
```

Finally, rather than show the numeric value of the marital status it would be nicer to show the label from the value set. We can do this using the function `getlabel()`. This function returns the label from value set as an alpha value. It takes the name of the variable (or value set) and the value to use. In this case, both the variable and the value are **MARITAL_STATUS**. Since `getlabel()` returns an alpha we need to change the %d to a %s.

```
PROC MARITAL_STATUS

// Ensure that spouse is not single
if MARITAL_STATUS in 1,3,4 and RELATIONSHIP = 2 then
    errmsg("%s has relationship spouse and marital status %s. Spouse cannot be
single, divorced or widowed.",
           strip(NAME(curocc())), getlabel(MARITAL_STATUS, MARITAL_STATUS));
    reenter;
endif;
```

Errmsg with select

If we add `select` to the `errmsg` we can give the user the option of going back to correct either field. We can use this in our bedrooms example:

```
PROC NUMBER_OF_BEDROOMS

// Ensure that number of bedrooms does not exceed number of rooms.
if NUMBER_OF_BEDROOMS > NUMBER_OF_ROOMS then
    errmsg("Number of bedrooms, %d, exceeds number of rooms, %d.",
           NUMBER_OF_BEDROOMS, NUMBER_OF_ROOMS)
           select("Fix number of rooms", NUMBER_OF_ROOMS,
                  "Fix number of bedrooms", NUMBER_OF_BEDROOMS);
endif;
```

With the select we no longer need the `reenter` since CSEntry will automatically reenter the field that the interviewer selects.

Note that the `select` clause is part of the `errmsg` statement so there is no semicolon in between the `select` and the `errmsg`.

Let's add a select to the relationship-marital status check too:

```
// Ensure that spouse is not single
if MARITAL_STATUS in 1,3,4 and RELATIONSHIP = 2 then
    errmsg("%s has relationship spouse and marital status %s. Spouse cannot be
single, divorced or widowed.",
           strip(NAME(curocc())), getlabel(MARITAL_STATUS, MARITAL_STATUS))
    select("Fix marital status", MARITAL_STATUS,
           "Fix relationship", RELATIONSHIP);
endif;
```

Soft edit checks

If you want to allow the user to ignore the error and enter the next field, you can add an option to the select statement with the keyword continue:

```
PROC NUMBER_OF_BEDROOMS

// Ensure that number of bedrooms does not exceed number of rooms.
if NUMBER_OF_BEDROOMS > NUMBER_OF_ROOMS then
    errmsg("The number of bedrooms, %d, is greater than the number of rooms, %d.",
           NUMBER_OF_BEDROOMS, NUMBER_OF_ROOMS)
           select("Fix number of rooms", ROOMS,
                  "Fix number of bedrooms", NUMBER_OF_BEDROOMS,
                  "Ignore error", continue);
endif;
```

Now the message dialog will have a third button labelled "ignore" that, when clicked, will move to the next field, in this case to type of main dwelling.

This is commonly referred to as a "soft edit check" as opposed to the previous "hard edit check" that does not allow the interviewer to move on until the inconsistency is fixed. The advantage of a soft edit check is that the interviewer won't get stuck, however, data quality may suffer. In a CAPI census soft edit checks are generally preferred in order to prevent interviewers from getting stuck and to keep interview times to a minimum.

For soft edit checks, CSPro provides a warning function, which is identical to errmsg except that if you have already ignored the message once, when you navigate through the field by clicking on the case tree or resuming from partial save, the message is not shown a second time. To make the rooms/bedrooms a soft check, in addition to adding the ignore option, we would use `warning` instead of `errmsg`.

```
PROC NUMBER_OF_BEDROOMS

// Ensure that number of bedrooms does not exceed number of rooms.
if NUMBER_OF_BEDROOMS > NUMBER_OF_ROOMS then
    warning("The number of bedrooms, %d, is greater than the number of rooms, %d.",
           NUMBER_OF_BEDROOMS, NUMBER_OF_ROOMS)
           select("Fix number of rooms", ROOMS,
                  "Fix number of bedrooms", NUMBER_OF_BEDROOMS,
                  "Ignore error", continue);
endif;
```

Testing Consistency Checks

When testing consistency checks involving multiple variables it is important to test all possible combinations of the variables. To do this we can create a test matrix. For example, to test all possible combinations of our consistency check between **MARITAL_STATUS** and **RELATIONSHIP** we would use the following matrix that shows the expected result for each combination of the variables involved:

		Marital Status		
Relationship	Never married (1)	Married (2)	Divorced (3)	Widowed (4)
Spouse (2)	Error	OK	Error	Error
Not-spouse (<> 2)	OK	OK	OK	OK

Using this matrix, you can test each of the 8 possible combinations and make sure that you get the expected result. This ensures that all possible cases are tested.

By combining the test matrices from all the consistency checks in the application, you can create a test plan for the survey/census application to verify the application works correctly when changes are made. Often times changes to one part of the questionnaire can have unintended effects on skips and consistency checks in other parts of the questionnaire so it is important to have a test plan that is used after every set of changes to ensure that no problems are introduced.

Calculations

In question **D02** the respondent gives the number of boys living in the household, girls living in the household and the total number of children living in the household. Let's check that the sum of the girls and the boys is equal to the total.

```
if BOYS_IN_HOUSEHOLD + GIRLS_IN_HOUSEHOLD <> CHILDREN_IN_HOUSEHOLD then
    errmsg("Boys (%d) plus girls (%d) does not match total (%d).",
           BOYS_IN_HOUSEHOLD, GIRLS_IN_HOUSEHOLD, CHILDREN_IN_HOUSEHOLD )
        select("Correct boys", BOYS_IN_HOUSEHOLD,
               "Correct girls", GIRLS_IN_HOUSEHOLD,
               "Correct total", CHILDREN_IN_HOUSEHOLD);
endif;
```

CSPro logic supports the following mathematical operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulo (remainder)	%
Exponentiation	^

Checking Dates

Let's add a consistency check between the date of birth and the age. We can use the function `datediff()` to calculate the age based on the interview date and the date of birth:

```
PROC DAY_OF_BIRTH

// Ensure that age matches date of birth
if datediff(DATE_OF_BIRTH, DATE_OF_INTERVIEW, "y") <> AGE then
    errmsg("Age (%d) does not match date of birth (%d)", AGE, DATE_OF_BIRTH)
        select("Correct Age", AGE, "Correct date of birth", DATE_OF_BIRTH);
endif;
```

This works but it would be better if the error message informed the interviewer what the date of birth that we calculated was.

```
PROC DAY_OF_BIRTH

// Ensure that age matches date of birth
if datediff(DATE_OF_BIRTH(curocc()), DATE_OF_INTERVIEW, "y") <> AGE then
    errmsg("Age (%d) doesn't match date of birth (%d). Based on date of birth age
should be %d",
        AGE, DATE_OF_BIRTH(curocc()), datediff(DATE_OF_BIRTH(curocc()),
        DATE_OF_INTERVIEW, "y"))
        select("Correct Age", AGE, "Correct date of birth", YEAR_OF_BIRTH);
endif;
```

We should also handle the case where the age or date of birth is unknown.

```
PROC DAY_OF_BIRTH

// Ensure that age matches date of birth
if AGE <> 999 and DATE_OF_BIRTH(curocc()) <> 99999999 then
    if datediff(DATE_OF_BIRTH(curocc()), DATE_OF_INTERVIEW, "y") <> AGE then
        errmsg("Age (%d) doesn't match date of birth (%d). Based on date of birth
age should be %d",
            AGE, DATE_OF_BIRTH(curocc()),
            datediff(DATE_OF_BIRTH(curocc()), DATE_OF_INTERVIEW, "y"))
            select("Correct Age", AGE, "Correct date of birth",
                   YEAR_OF_BIRTH);
    endif;
endif;
```

Logic variables

It would better if we didn't repeat the datediff calculation twice. Repeating code makes it harder to maintain. We may fix a bug in one copy of the code but forget to do so another. To avoid repeating ourselves we can declare a logic variable to hold the value of our calculation. Logic variables are like dictionary variables but are only for use in calculations in logic and don't get shown on forms or saved to the data file. To create a variable, we declare it as follows:

```
numeric aNumber;
string anAlphanumeric;
alpha(20) anAlphaNumericWithAFixedLength;
```

Numeric variables hold numbers (including numbers with fractional parts) and string variables hold alphanumeric values. Alpha variables are like strings but with a fixed length. In early versions of CSPro string variables didn't exist and it was common to use alpha variables. In modern CSPro you should always use string variables instead of fixed length alpha variables.

Coding style

We recommend using mixed case (a.k.a. camelCase) for logic variables in order to distinguish them from dictionary variables.

We can declare a variable to hold the result of the datediff as follows:

```
numeric calculatedAge;
calculatedAge = datediff(DATE_OF_BIRTH(curocc()), DATE_OF_INTERVIEW, "y");
```

Now we can use that variable in place of the datediff:

```
if calculatedAge <> AGE then
    errmsg("Age (%d) does not match date of birth (%d). Based on date of birth age
should be %d",
        AGE, DATE_OF_BIRTH(curocc()), calculatedAge)
        select("Correct Age", AGE, "Correct date of birth", YEAR_OF_BIRTH);
endif;
```

It is possible to combine the declaration and the initialization of the variable into a single line:

```
numeric calculatedAge = datediff(DATE_OF_BIRTH(curocc()), DATE_OF_INTERVIEW, "y");
```

When to Use Consistency Checks

It is tempting to place consistency checks on every question in your application in order to maximize data quality but it is important to realize that every check you add comes at a price. Beyond the time it takes to implement and properly test each check, you also need to consider the additional time required to train and support a more complex application. Every check you add increases the risk that there will be bugs in your application when it gets to the field. Unlike a key from paper application, a bug in the field can mean an enumerator being stuck and abandoning an entire questionnaire. For surveys, you generally have a small sample and better trained enumerators so this is less of a concern. However, for a census, enumerators are not as experienced and supporting them in the field is much more of a challenge. In a census, due to the number of respondents, you can use imputation to correct many inconsistencies with minimal impact on the overall results. In any case, just the fact that the CAPI instrument imposes range checks and skip patterns will be a significant improvement in data quality over a paper census. For a CAPI census application, we recommend only adding consistency checks on the key demographic fields in the household roster and leaving potential inconsistencies in other parts of the questionnaire to be corrected post data collection. In addition, we recommend using mainly “soft” edit checks so that the enumerator is never blocked by the application.

Exercises

Implement the following consistency checks from the edit specifications. Test that they are correct. Make sure to test all possible cases (test matrices can help with this).

1. Display an error message if the head of household is less than 12 years old.
2. Ensure that **B14** (age at first marriage) is less than or equal to the current age (**B05**).
3. Display an error message if the highest level of education (**C02**) is university or graduate but the age is less than eighteen. Make this a “soft check” so that the interviewer can ignore the error if they wish.
4. Add an error message if the highest level of education (**C02**) is “standard 5” or higher and the individual does not know how to read or write (**C03**). Make this a soft check.
5. Add a consistency check in section F that displays an error message if the household has a flush toilet in question **F10** and does NOT also specify piped water inside house for question **F12**. Use a select clause to allow them to correct either **F10** or **F12**.
6. Ensure that the total number of children in **D03** is equal to the sum of the boys and the girls. Do the same for question **D04**.
7. Check that the date that the death occurred in question **E05** is within the last five years, since the section should only include deaths occurring in that period. Use the interview date to calculate the number of years since the death occurred. For example, if the interview date is 2016-10-05 and the date of death is 2016-09 then you should show an error message. You can use datediff to calculate the difference between the interview date and the date of death as we did for the age and date of birth check, however you will need to construct a full date from just the month and year of death. Since no day is given, use 1 as the day (assume first day of the month). Note that datediff rounds to the nearest year so to do the check correctly you will need to get the difference in months and compare it to 60 (5 years times 12 months).

Session 4: More Logic in Data Entry, Rosters, Subscripts and Loops

At the end of this lesson participants will be able to:

- Implement consistency checks across individuals in the household roster
- Understand the order of PROCs and where to place edit checks
- Use subscripts, `totocc()`, `count()`, `seek()` and `curocc()` to work with rosters
- Use loops (`do while`) to implement edit checks on repeating records/items (rosters)
- Understand and use "special values" (`notappl`, `missing` and `default`) in logic
- Set occurrence labels on rosters from logic

Groups and Subscripts

Let's add a check to the line number of mother to ensure that the line number entered corresponds to a woman 12 or over in the household. To do this we need to link the line number entered in **B10** back to the corresponding row of the roster in section B. This is done through subscripts to get the age and sex of the mother based on the line number.

For any item that is repeated, adding a number in parentheses after it gives you the value of a particular occurrence of the variable. For example, `AGE(1)` will be the age of the first household member, `AGE(2)` the age of the second member… If we omit the subscript when executing logic in a proc of a roster CSPro will assume that we want the one for the current row of the current roster. However, in this case we don't want the sex and age of the person in the current row, we want the age and sex in the row specified by **MOTHER_LINE_NUMBER**. To get that we need to use `AGE(MOTHER_LINE_NUMBER)` and `SEX(MOTHER_LINE_NUMBER)`.

```
PROC MOTHER_LINE_NUMBER

// Ensure that line number of mother is line number of woman over 12
if not MOTHER_LINE_NUMBER in 87:88 then

    if SEX(MOTHER_LINE_NUMBER) = 1 then
        errmsg("Sex of mother %s is male",
               strip(NAME(MOTHER_LINE_NUMBER)))
               select("Correct mother",
                      MOTHER_LINE_NUMBER,
                      "Correct sex of " +
                      strip(NAME(MOTHER_LINE_NUMBER)),
                      SEX(MOTHER_LINE_NUMBER));
    endif;

    if AGE(MOTHER_LINE_NUMBER) <> 999 and AGE(MOTHER_LINE_NUMBER) < 12 then
        errmsg("Age of mother %s is %d but must be at least 12",
               strip(NAME(MOTHER_LINE_NUMBER)),
               AGE(MOTHER_LINE_NUMBER))
               select("Correct mother", MOTHER_LINE_NUMBER,
                      "Correct age of " + strip(NAME(MOTHER_LINE_NUMBER)),
                      AGE(MOTHER_LINE_NUMBER));
    endif;
endif;
```

Blanks and special values

What happens if we refer to an occurrence of a row in the section B roster that doesn't exist? Try entering a line number for the mother greater than the number of rows in the section B roster. Our tests for age and sex are not triggered. What are the values of **NAME**, **SEX** and **AGE** when they are empty? Let's print them out using `errmsg` and see.

```
errmsg("NAME = %s, AGE=%d, SEX=%d", strip(NAME(MOTHER_LINE_NUMBER)),
        AGE(MOTHER_LINE_NUMBER), SEX(MOTHER_LINE_NUMBER));
```

The alphanumeric variable **NAME** is just empty but the numeric values **AGE** and **SEX** are `notappl`. What does this mean? Generally, numeric fields that are blank (skipped or not yet entered) have a special value called `notappl` that can be used in comparisons in logic. For example, to test if the **SEX** is blank we can use the following comparison:

```
if SEX(MOTHERS_LINE_NUMBER) = notappl then
    // Sex is blank, must be an empty row in section B roster
    errmsg("%d is not a valid line in section B", MOTHERS_LINE_NUMBER);
    reenter;
endif;
```

There are other special values that are used in CSPro:

- `Missing`: can be used as an alias for no response/refused codes (9, 99, 999…). You must create an entry for it in the value set.
- `Default`: results from an error reading from a data file or from a calculation error (like trying to calculate notappl + 2 or dividing by zero) or moving a value into a variable that is too small to hold the value. For example, if AGE is a two digit field, AGE = 118 will result in `default`.

Getting the size of a roster

In our error message, it would be nice to tell the interviewer what the maximum valid line number is. We can do that using the function `totocc()` which gives you the total number of occurrences of a group.

```
if MOTHER_LINE_NUMBER > totocc(HOUSEHOLD_MEMBERS_ROSTER) then
    // This is beyond the end of the roster
    errmsg("%d is not a valid line in section B. Must be between 1 and %d.",
        MOTHER_LINE_NUMBER, totocc(HOUSEHOLD_MEMBERS_ROSTER));
    reenter;
endif;
```

Setting Occurrence Labels in Logic

If you look at the case tree in Android you will see that the roster occurrences are displayed with the name of the roster and the occurrence number "Demographics(1), Demographics(2)…" which is not useful. Using logic, we can set the occurrence labels to the names of the individuals instead. For that we use the command `setocclabel()` which takes the name of the group (roster or repeating form) and the string to set it to. For example, to set the occurrence label of each row of the different rosters once the name is entered we can do the following in the postproc of the name field (**NAME**):

```
PROC NAME
setocclabel(HOUSEHOLD_MEMBERS_ROSTER, strip(NAME));
setocclabel(DEMOGRAPHICS_ROSTER, strip(NAME));
setocclabel(EDUCATION_ROSTER, strip(NAME));
setocclabel(FERTILITY_ROSTER, strip(NAME));
```

This works fine when we are adding a new case, however we open an existing case in modify mode the occurrence labels are not set until we get to the person roster even though the occurrences already exist. In modify mode, while still on the demographics roster you can scroll the case tree to see Demographics (1), Demographics (2)… as we had before. In order to prevent this, we need to set the occurrence labels for the rosters as soon as we open the case.

More About Procs

What proc can we use to set the occurrence labels? We could use the preproc of the first field of the first form but there is a better option. It turns out that every element in the form tree has a PROC. Not only do variables have procs but there are procs for forms, rosters, levels and even the application itself. Everything you see in the forms tree can have both a preproc and a postproc. Understanding the order in which procs are executed is important in understanding CSPro logic.

The general rule is that

1. Parent items have their preproc called first,
2. then the procs of the child items are called
3. and finally, the postproc of the parent is called.

Group Exercise: Proc Order

Form teams of three to five people. The instructor adds errmsg to the postproc and preproc of the following: **POPSTAN2020_FF** (application), **POPSTAN2020_QUEST** (level), **HOUSEHOLD_MEMBERS_FORM** (form), **HOUSEHOLD_MEMBERS_ROSTER** (roster), **NAME** (variable), **NUMBER_OF_ROOMS** (variable). Each team receives slips of paper with the names of each preproc and postproc. BEFORE running the application, the teams have to put the slips in the order that the errmsgs will be shown when the application is run. Teams have three minutes to complete the exercise. Then the application is run and teams see if they have the correct results.

Setting Occurrence Labels (take 2)

Now that we understand procs, which proc should we set the occurrence labels in? The preproc of the questionnaire! We can do something like the following:

```
PROC POPSTANLFS_QUEST
preproc
setocclabel(HOUSEHOLD_MEMBERS_ROSTER(1), strip(NAME(1)));
setocclabel(HOUSEHOLD_MEMBERS_ROSTER(2), strip(NAME(2)));
setocclabel(HOUSEHOLD_MEMBERS_ROSTER(3), strip(NAME(3)));
```

Loops

The problem is that we want to do this for each household member, but the number of household members varies from case to case. The above works only if we know the size of the household in advance. In order to handle any size household, we need a loop. We can use a do loop which lets you repeat something until a certain condition is true. How many times do we loop? We use the function totocc() that gives us the total number of occurrences of the roster.

```
PROC POPSTAN2020_QUEST
preproc

// Fill in occurrence labels in rosters when entering a case that
// has existing data (partial save or modify mode). If we don't do
// this then the case tree will not have correct occurrence labels
// until after we pass through the household members roster.

do numeric i = 1 while i <= totocc(HOUSEHOLD_MEMBERS_ROSTER)
    setocclabel(HOUSEHOLD_MEMBERS_ROSTER(i), strip(NAME(i)));
    setocclabel(DEMOGRAPHICS_ROSTER(i), strip(NAME(i)));
    setocclabel(EDUCATION_ROSTER(i), strip(NAME(i)));
    setocclabel(FERTILITY_ROSTER(i), strip(NAME(i)));
enddo;
```

Counting Heads of Household

It is currently possible to enter multiple heads of household or no head of household at all. How can we add a consistency check to ensure that there is exactly one head of household? In which proc would such a check go? We can put it in the postproc of the roster since that will be run after all the rows have been entered.

How do we determine the number of heads of household? In each row of the roster, we can check the relationship to see if it is head of household. To do this we need to a logic variable to keep track of the number of heads of household.

```
numeric numberOfHeads = 0;
```

Now we can increment the variable for each head of household we find:

```
PROC HOUSEHOLD_MEMBERS_ROSTER

// After person roster is complete, check to make sure that there is
// exactly one head of household.

numeric numberOfHeads = 0;

if RELATIONSHIP(1) = 1 then
    numberOfHeads = numberOfHeads + 1;
endif;

if RELATIONSHIP(2) = 1 then
    numberOfHeads = numberOfHeads + 1;
endif;

if RELATIONSHIP(3) = 1 then
    numberOfHeads = numberOfHeads + 1;
endif;

if numberOfHeads <> 1 then
    errmsg("Number of heads of household must be exactly one");
    reenter RELATIONSHIP(1);
endif;
```

Since there is more than one row in the roster we need to use a subscript to tell CSEntry which row of the roster to look in. `RELATIONSHIP(3)` refers to the relationship of the 3rd row of the roster.

The above works but only if we know the size of the household in advance. In order to handle any size household, we need a loop. We can use a do loop. Just like with the occurrence labels, we can loop from 1 to the number of rows in the roster.

```
PROC HOUSEHOLD_MEMBERS_ROSTER

// After adults roster is complete, check to make sure that there is
// exactly one head of household.

numeric numberOfHeads = 0;
do numeric i = 1 while i <= totocc(HOUSEHOLD_MEMBERS_ROSTER)
    if RELATIONSHIP(i) = 1 then
        numberOfHeads = numberOfHeads + 1;
    endif;
enddo;

if numberOfHeads <> 1 then
    errmsg("Number of heads of household must be exactly one");
    reenter RELATIONSHIP(1);
endif;
```

Activity: Acting out a loop

Show a household with four members on the whiteboard/flipchart and show the code above on the screen. Pick three volunteers. One volunteer plays the role of the variable numberOfHeads, a second plays the role of the variable i and a third is the loop director. Give them each a sign so everyone knows who they are playing. The two people playing variables should show their current values by holding up the correct number of fingers. The director is responsible for walking through the code line by line and updating the variable values until the loop ends explaining what they do step by step.

Now that we all understand loops let's see how we can simplify our code by using the function `count()` instead of a loop.

```
PROC HOUSEHOLD_MEMBERS_ROSTER

// After demographics roster is complete, check to make sure that there is
// exactly one head of household.

numeric numberOfHeads = count(PERSON_REC where RELATIONSHIP = 1);

if numberOfHeads <> 1 then
    errmsg("Number of heads of household must be exactly one");
    reenter RELATIONSHIP(1);
endif;
```

Force Head of Household to First Row

Instead of allowing the interviewer to enter the head of household on any line of the roster, we can greatly simplify our logic if we force them to list the head on the first row. We can implement this in the postproc of **RELATIONSHIP**.

```
PROC RELATIONSHIP
// Don't allow non-head of household on first row
if RELATIONSHIP <> 1 and curocc() = 1 then
    errmsg("Please enter the head of household before the other household
members.");
    reenter;
endif;

// Don't allow head of household on a row other than first row
if RELATIONSHIP = 1 and curocc() <> 1 then
    errmsg("Only one head of household is allowed. Head of household was already
entered.");
    reenter;
endif;
```

This replaces our logic for counting the number of heads of household in the postproc of the roster. This is simpler and will also simplify any other checks involving the head. For example to ensure that the head of household is at least 12 years older than his/her children we can simply compare the age of the individual to the age of head. Since the head is on the first row, the age of the head will always be `AGE(1)`. Since the head is entered first, we can do this type check in the proc for **AGE** instead of in the roster postproc.

```
PROC AGE
// Compare age of child to age of the head
if RELATIONSHIP = 3 and AGE(1) - AGE < 12 then
    errmsg("Child %s is %d years old but head %s is %d. Parent must be at least 12
years older than child.",
                strip(NAME),
                AGE,
                strip(NAME(1)),
                AGE(1))
                select("Correct age of " + strip(NAME), AGE,
                       "Correct age of " + strip(NAME(1)), AGE(1),
                       "Correct relationship of " + strip(NAME),
                       RELATIONSHIP,
                       "Correct relationship of " + strip(NAME(1)),
                       RELATIONSHIP(1));
    endif;
enddo;
```

Exercises

1. Ensure that the age difference between the head of household and the grandchild should not be less than 24 years. If age of head is X years and the age of the child is Y years, then X-Y ≥ 24.
2. Show an error message if the head of household has at least one spouse in the household and the marital of status of the head of household is not married.
3. Show an error message if the head of household and his/her spouse are of the same gender. Make sure that your logic works with polygamous households as well as households with just one spouse.
4. Set the occurrence labels in the section E roster, deaths, to the names of the deceased in that section. Make sure that the labels are set correctly when resuming from partial save.

Session 5: CAPI Features

At the end of this lesson participants will be able to:

- Add question text and help text
- Add fills in question text
- Use multiple languages in question text, error messages and the dictionary
- Use occurrence labels and logic variables as fills in question text
- Use `setcaselabel` to customize the case listing
- Use dynamic value sets

Question Text

We can add text to each question in our survey by clicking on "CAPI Questions" in the toolbar. This allows us to enter literal question text for the questionnaire that the interviewer will read verbatim. Additionally, we can add interviewer instructions. Let's start with the first few questions in section B.

For **RELATIONSHIP**, enter "What is (name's) relationship to the head of the household?" For **SEX** enter "Is (name) a male or a female?". Run the application on Windows and then on mobile to see how the question text is displayed.

In addition to the text, we can add instructions to the interviewer. For example for **AGE** we can have:

> How old is (name) in completed years?

Followed by the instruction:

> *Enter age in completed years ("000" for children less than one year old)*

To make it clear to the interviewer that this is an instruction, we use italics to distinguish it from the question. We can also use different colors and fonts as well. You can use whatever scheme you like as long as it is consistent throughout the application.

If you have your question text in Word or Excel you can copy and paste into CSPro and it will preserve the formatting.

Help Text

In addition to question text, we can provide additional instructions to the interviewer as "help text". Help text is not shown by default but can be displayed by using the F2 key on Windows or tapping the help icon next to the question text on Mobile.

Let's add the following help text to the **NAME** field in section B:

> *Include all persons living in this house who have common arrangements for cooking and dining.*

Run the application on both Android and mobile and see how the help text is displayed.

Fills in Question Text

It is possible to have the question include the values of dictionary variables. For example, for **SEX**, instead of asking "Is (name) a male or a female?" we can include the respondents name by using **%NAME%** inside the question text. At runtime, this will be replaced by the contents of the variable **NAME**.

Let's change the text for **RELATIONSHIP**, **SEX** and **AGE** to use **%NAME%** in place of "(name)".

Conditional Question Text

Sometimes, the question text will be completely different depending on other factors and using fill variables is not sufficient to implement the differences. For example, the question text for the NAME field should really be:

What is the name of the head household? – for the first row of the roster

What is the name of the next member of the household? – for the remaining rows of the roster.

We can implement this using the conditions window underneath the question text window. By default, this window contains a single empty condition. Right-clicking on the conditions lets you add a new condition. Let's modify the first condition so that **Min Occ** and **Max Occ** are both 1 and then add a second condition where **Min Occ** is 2 and **Max Occ** is 20. Now we have two different question texts, one that will be shown for the first occurrence and the other that will be shown for the rest. Edit these two questions texts as above and test it to verify that it works.

	Min Occ	Max Occ	Condition
1	1	1	
2	2	20	

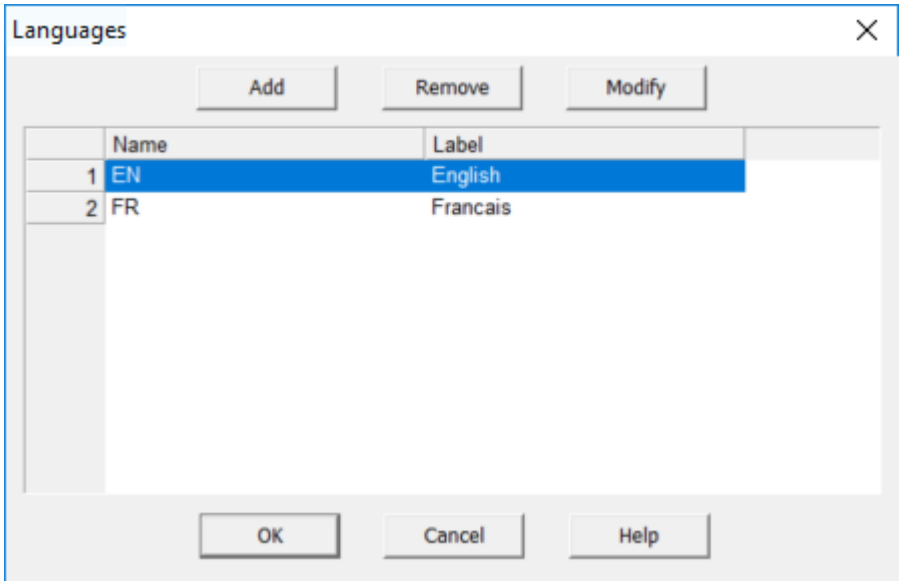
In addition to using the occurrence numbers it possible to specify a logic expression in the condition column to determine whether or not to display the text. For example setting the condition to "SEX=1" will display the question text only for males.

Supporting Multiple Languages

It is possible to add question text, dictionary labels and error messages in multiple languages.

Multiple Languages for Question Text

To have multiple languages for question text you first need to define the languages. This is done from the CAPI Options Menu. Select **"Define CAPI Languages"**. This will bring up the "Language" dialog box. Click on "Add" to add a language, specify the name and label to use for the language and click "OK". The label is what the interviewer will see when choosing a language and the name is what will be used in logic to refer to the language. We generally use abbreviations for the language names such as EN, FR and ES but this is just a convention.



When creating question text, you enter the question text for each language specified. Let's look at the question text for the "Sex" item. There is a Drop Down menu for the languages we specified in the above steps. We select the language and then enter the question text in that language.

Now that you have created the question text in the specified languages, they can be selected during data entry.
In Windows, we can go to the "Options" Menu and select "Change Language" and a menu of the languages we defined will be displayed. Click on the desired language and the question text will be displayed in that language. On mobile devices you will find "Change Language" on the menu during data entry.

Multiple Languages for Dictionary Items

To have multiple language in value sets, you first need to define the languages. This is similar to the process we did to define multiple languages for question text; however, for dictionary items you define languages from the Edit menu of the

dictionary. Make sure you are in the dictionary editor (you may need to click on the little blue book in the toolbar) and choose "Edit" and then "Languages". This dialog is identical to the one used for adding languages to question text. You should make sure to use the same names and labels for languages in this dialog as you entered for question text.

Once you have added additional languages to the dictionary you will see a language dropdown on the toolbar. To add a translation for a dictionary label or value set label, choose the language in the dropdown and edit the label as you normally would. This will set the label for the language you chose.

Multiple Language Error Messages

To add translations used in program logic, such as arguments to `errmsg()`, we can provide translations in the message file. To do this we add the translated messages to the messages tab at the bottom of the logic editor.

```
{Application 'POPSTAN2020' message file generated by CSPro}
FR("Head of household must be at least 15 years old") Le chef de menage doit avoir au mo
FR("Correct age") Corriger l'age
FR("Correct relationship") Corriger le lien de parente
```

On each line of the file, we put the language name followed by the English message text in parenthesis and finally the translation. Then in the logic, we use the function `tr()` to retrieve the translated text. Now when we run the application with the language set to French and encounter the error, it will show the French text.

```
PROC AGE

// Ensure that head of household is at least 15 years old
if AGE < 15 and RELATIONSHIP = 1 then
    errmsg(tr("Head of household must be at least 15 years old"))
        select(tr("Correct age"), AGE, tr("Correct relationship"), RELATIONSHIP);
endif;
```

It is also possible to use numbers to represent messages. In the message file, instead of putting the untranslated message, simply put the number followed by the message text. Mark different languages in the file using "Language=" followed by the language name.

```
{Application 'POPSTAN2020' message file generated by CSPro}
Language=EN
100 Head of household must be at least 15 years old
101 Correct age
102 Correct relationship

Language=FR
100 Le chef de menage doit avoir au moins 15 ans.
101 Corriger l'age
102 Corriger le lien de parente
```

In the logic you can pass the message number directly to `errmsg()`. Inside the select clause, however, you will need to use `tr()` or `maketext()` around the message number.

```
PROC AGE

// Ensure that head of household is at least 15 years old
if AGE < 15 and RELATIONSHIP = 1 then
    errmsg(100)
        select(tr(101), AGE, tr(102), RELATIONSHIP);
endif;
```

Using message numbers makes the code a bit harder to read since you need to refer to the message file to see the text of the messages. Using the literal question text makes the code easier to read, however, if you edit the English question text in the logic and forget to modify it in the message file your translation will not be displayed.

You can also use a separate message file for each language. To add an additional message file, use "Add files" from the files menu and enter the name of the message file next to "External Message File".

Group exercise
Add the French translations for the question text, labels, value sets and error messages for marital status (**B13**).

Using Occurrence Labels in Question Text

We have the following occurrence labels for the housing unit types in question **F05**.

- Traditional round hut
- Detached house
- Semi-detached house
- Flat/apartment
- Improvised (kiosk/container)

We can use these occurrence labels in the question text for **F05**:

How many %getocclabel% units are in this household?

Anytime you use **%getocclabel%** in question text it is replaced by the occurrence label of the current occurrence. With the above, the question text for the first occurrence will be “How many traditional round hut units are in this household?” and the text for the second occurrence will be “How many detached house units are in this household?” …

Using Logic Variables in Question Text

In addition to dictionary variables, and occurrence labels it is also possible to use logic variables in question text.

For question **D05** we want to double check that the total of the children living with the women, living elsewhere and deceased equals the number of total births. We do this by asking the question:

Just to make sure that I have this right, (name) has had in total (total number) births during her life. Is this correct?

We can use the question text for this but we don’t have a dictionary variable for total births. We only have the yes/no variable **IS_TOTAL_BIRTHS_CORRECT**. We could create an additional dictionary variable but instead we can simply create a logic variable in the program for **totalBirths** and use that as the fill value.

We will declare it in the PROC global section so that it is available everywhere. If you declare a logic variable inside the PROC of a dictionary variable or group, it is only available inside that PROC. Anything in the PROC global is available in all the PROCs and in the question text.

To view the proc GLOBAL, in the logic view, click on the first item in the form tree. This shows all of the program logic at once: the proc GLOBAL plus all the other procs. Clicking on any other item in the form tree shows just the procs for that item.

We need to assign a value to **totalBirths**. Which proc do we do that in? We do that in the onfocus of **NUMBER_BIRTHS_CORRECT** since we need to use it when we are in that field. The onfocus is called every time the field is entered. We cannot do this in the preproc since the preproc is not triggered when moving backwards through the questions.

```
PROC IS_TOTAL_BIRTHS_CORRECT
onfocus
// Compute total births to be used as fill in question text
totalBirths = CHILDREN_IN_HOUSEHOLD + CHILDREN_ELSEWHERE + CHILDREN_DECEASED;
```

What happens when one of the fields in this calculation is skipped? The value becomes `notappl` which messes up the entire calculation. We need to be a little smarter in calculating our total to exclude the skipped values.

```
PROC IS_TOTAL_BIRTHS_CORRECT
onfocus

// Compute total births to be used as fill in question text
totalBirths = 0;
if CHILDREN_IN_HOUSEHOLD <> notappl then
    totalBirths = totalBirths + CHILDREN_IN_HOUSEHOLD;
endif;
if CHILDREN_ELSEWHERE <> notappl then
    totalBirths = totalBirths + CHILDREN_ELSEWHERE;
endif;
if CHILDREN_DECEASED <> notappl then
    totalBirths = totalBirths + CHILDREN_DECEASED;
endif;
```

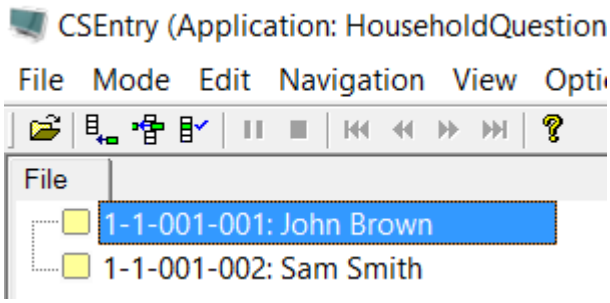
Case Labels

By default, the case listing screen shows the id-items concatenated together. This is not very easy for an interviewer to read. You can customize the case listing for a case using the `setcaselabel` command. As an example let's set the case label to the string "province-district-ea-household number: name of head of household". Since we need to have the name of the head of household to do this, we can add it in the postproc of **NAME**. In order to format the case label from the variables we can use the function `maketext()` which works like `errmsg()` but returns a string instead of displaying it on the screen.

```
PROC NAME

if curocc() = 1 then
    // Set label for case in case listing to an easier to read format.
    // We do this when we first get the name of the head.
    string caseLabel = maketext("%d-%02d-%03d-%03d: %s",
        PROVINCE, DISTRICT, ENUMERATION_AREA,
        HOUSEHOLD_NUMBER, strip(NAME));
    setcaselabel(POPSTAN2020_DICT, caseLabel);
endif;
```

Now after entering a case we have a much friendlier case listing:



Note that `setcaselabel` only works with data files of type CSPro DB.

Dynamic Value Sets

It is often useful to change the value set for a question from logic. This can be done using the command `setvalueset`. Let's start with a simple example. Currently our value set for relationship in section B has labels like "Son/Daughter" and "Brother/Sister" to allow for both genders. However, when we show the relationship value set we already know the gender of the household member so we could show "Son" for males and "Daughter" for females. To do this we create two new value sets for relationship in the dictionary: **RELATIONSHIP_MALE** and **RELATIONSHIP_FEMALE**. Then in the onfocus of relationship we choose between the two value sets:

```
PROC RELATIONSHIP
onfocus
// Show male or female version of value set depending on sex of the person.
if SEX = 1 then
    setvalueset(RELATIONSHIP, RELATIONSHIP_MALE);
else
    setvalueset(RELATIONSHIP, RELATIONSHIP_FEMALE);
endif;
```

Dynamic Value Sets from a Roster

For question **A11**, line number of respondent, we would like to create a value set from the names and line numbers of the eligible household members. For this to work correctly we will need to ask **A11** after entering the names and ages in the household members rosters. Create a new numeric variable in the dictionary for **A11** and place it on the household members form after the roster.

To create the value set from the household roster we need the second form of setvalueset that takes an array of codes and an array of labels. This will allow us to create the list of names in logic instead of in the dictionary. First, we need to declare the two arrays in the PROC global.

```
PROC GLOBAL
numeric totalBirths;
array string labels(100);
array numeric codes(100);
```

An array logic variable is similar to a dictionary item with occurrences. A numeric array of length seven stores seven numbers, each of which is accessed through subscripts.

We will fill in the arrays of codes and labels with names and line numbers of the eligible members in the household. According the specification, only household members 12 and over can be respondents. For example, if we have the following household:

	Line number	Name	Sex	Relationship	Age
1	1	John Brown	1	1	39
2	2	Mary Brown	2	2	40
3	3	Bobby Brown	1	3	11
4	4	Jane Brown	2	7	22

We would fill in the two arrays as follows:

Subscript	Codes	Labels
1	1	John Brown
2	2	Mary Brown
3	4	Jane Brown
4	notappl	

To do this in logic we need to loop through the household roster and add an entry into our arrays for each eligible household member:

```
PROC RESPONDENT
onfocus
// Create the value set for respondent from all household members 12 and over
numeric indexRoster;
numeric nextEntryValueSet = 1;
do indexRoster = 1 while indexRoster <= totocc(HOUSEHOLD_MEMBERS_ROSTER)

    if AGE(indexRoster) >= 12 then
        labels(nextEntryValueSet) = NAME(indexRoster);
        codes(nextEntryValueSet) = indexRoster;
        nextEntryValueSet = nextEntryValueSet + 1;
    endif;
enddo;
```

Finally, we need to terminate the array of codes with a notappl to tell CSPro not to use the whole array and then pass the array of codes and the array labels to the setvalueset command. Make sure to set the capture type of the field to "Radio Button" in the field properties for **RESPONDENT_LINE_NUMBER**.

```
codes(nextEntryValueSet) = notappl;
setvalueset(RESPONDENT, codes, labels);
```

Group Exercise

Implement the dynamic value set for mother line number (**B10**). Only show females in the household over 12. Don't forget to include the codes for non-resident and deceased in the value set.

Cascading Questions

Let's revisit question **B07**, place of birth. Rather than present a giant list of all countries in the world, it would be nice to allow the interviewer to narrow down the list first by continent and the choose from the countries in that continent. This is simple given the way that codes for place of birth are structured. The first digit is the continent (or blank for Popstan) and the next two digits are the country (or district). We can create subitems for the continent and country and drag those onto the form instead of the parent items. For the first subitem we use the following value set to allow the interviewer to choose either this country or a continent:

Continent of birth	Code
Popstan	0
Africa	1
Asia	2
Europe	3
North America	4
Oceania	5
South America	6

Then in the onfocus for the second subitem, once we know the continent, we can loop through the value set for **PLACE_OF_BIRTH** that contains all the countries and create a value set containing only those countries that are in the selected continent.

```
PROC COUNTRY_OR_DISTRICT_OF_BIRTH
onfocus
// Create value set based on continent selected in previous question
numeric i;
do i = 1 while i < 100
    string label = getlabel(PLACE_OF_BIRTH, CONTINENT_OF_BIRTH * 100 + i);
    // A blank label means that there are no more countries in the chosen
    // continent so we can break out of the loop early.
    if label = "" then
        break;
    endif;
    codes(i) = i;
    labels(i) = label;
enddo;
codes(i) = notappl;
setvalueset(COUNTRY_OR_DISTRICT_OF_BIRTH, codes, labels);
```

Dynamic Value Sets from Checkboxes

Question **B16** (primary language) should only show a subset of the languages chosen in **B15** (languages spoken). We can do this using a dynamic value set as well. The trick is that since **B15** uses checkboxes it will have alpha codes (A, B, C…) while **B16** will have numeric codes (1,2,3…) so we need to convert from numeric to alpha to determine if a given language was selected. We could do this with a series of `if then else` statements but an easier approach is to use the string **"ABCDEFGH"** to convert from numeric to alpha by looking up the character at the position of the numeric code. For example, numeric code 1 would give us the character at the first position: A. Numeric code 2 would give us the character at position 2, B etc…

```
PROC MAIN_LANGUAGE
onfocus

// Create value set from items selected in languages spoken
numeric nextEntry = 1;

// Used to translate from checkbox (alpha codes) to numeric codes
string languageCheckboxCodes = "ABCDEFGH";

// Loop through the numeric codes 1-8 and each selected
// to value set
do numeric languageNumericCode = 1 while languageNumericCode <= 8

    // Convert the numeric code to the checkbox alpha code
    // by looking it up in the array.
    string languageCheckboxCode = languageCheckboxCodes[languageNumericCode:1];

    // Check if the language is selected in the checkbox field
    if pos(languageCheckboxCode, LANGUAGES_SPOKEN) > 0 then
        // Language is selected. Add it to the value set.
        codes(nextEntry) = languageNumericCode;
        labels(nextEntry) = getlabel(MAIN_LANGUAGE_VS1, languageNumericCode);
        nextEntry = nextEntry + 1;
    endif;

enddo;

// Mark end of value set
codes(nextEntry) = notappl;

// Modify value set
setvalueset(MAIN_LANGUAGE, codes, labels);
```

What if the interviewer doesn't pick any language in **B15**? Then our dynamic value set is empty. We should add a check to

B15 to ensure that at least one language is chosen.

```
PROC LANGUAGES_SPOKEN

// Ensure that at least on language is chosen
if length(strip(LANGUAGES_SPOKEN)) = 0 then
    errmsg("You must choose at least one language");
    reenter;
endif;
```

Dynamic Checkboxes

Let's implement a dynamic value set for question **G2**, "were assets purchased with a loan". Rather than a series of yes/no questions, we implement this using a single variable with checkboxes. We could have one checkbox for each of the 10 items in the assets roster but it would be better if we only displayed the checkboxes for the assets that the household actually possesses. How do we know if the household possesses an item? The household possesses the item if its quantity is greater than zero. We need to loop through the rows of the roster and add a checkbox to the value set for each item with quantity greater than zero. The only tricky part is that these are checkboxes so we need to use alpha values.

In order to create a value set with alpha values we need a code array of type string. Where do we declare it? PROC GLOBAL.

```
PROC GLOBAL
numeric totalBirths;
array string labels(100);
array codes (100);
array string codesString(100);
```

We build the value set in the onfocus of the checkboxes field. We use the alphabet string trick again to get the alpha codes from the occurrence number. We also use the function `getocclabel()` to get the occurrence label from the assets roster to use in the value set.

```
PROC POSSESSIONS_PURCHASED_WITH_LOAN
onfocus

// Create dynamic value set from assets that have quantity > 0
numeric nextEntry = 1;

string alphabet = "ABCDEFGHIJ";

do numeric assetNumber = 1 while assetNumber <= totocc(POSSESSIONS_ROSTER)
    // Check if household possesses this asset
    if QUANTITY(assetNumber) > 0 then
        // Add to value set
        labels(nextEntry) =
                        getocclabel(POSSESSIONS_ROSTER(assetNumber));
        codesString(nextEntry) = alphabet[assetNumber:1];
        nextEntry = nextEntry + 1;
    endif;
enddo;

// Mark end of array (use "" instead of notappl since field is alphanumeric)
codesString(nextEntry) = "";
setvalueset($, codesString, labels);
```

Exercises

1. Add question text to the rest of section B. Use fills to include the name as we did in the examples.
2. Add a new language, the language of your choice, to the CAPI text and to the dictionary. Translate the question text, labels and value sets for section B into the new language.
3. Add question text for section G and use the occurrence labels to fill in the name of the possessions in the quantity and value fields.
4. In question B06 (date of birth) use a dynamic value set for the day based on the month (January: 1-31, February: 1-28, March: 1-31…) so that the interviewer cannot enter an invalid date like February 30 or April 31. Bonus if you can correctly handle leap years.
5. For E08 (line number of mother of deceased) use a dynamic value set to list the names of all eligible women from the household roster.

Session 6: Lookup files, Navigation & System Control

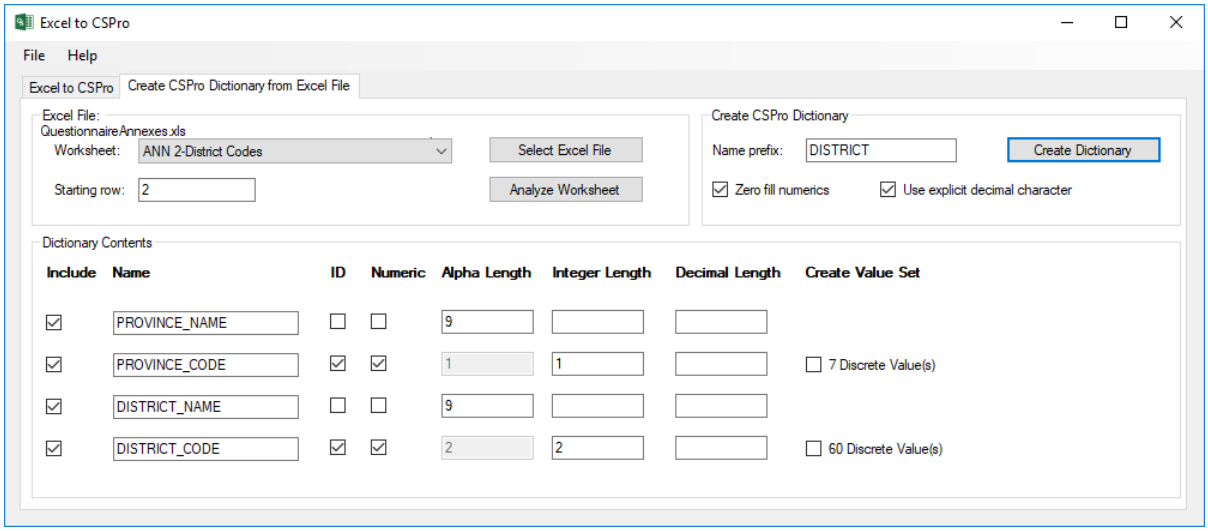
At the end of this session participants will be able to:

- Use lookup files in data entry
- Create user defined functions in CSPro
- Extend the interface of CSEntry with userbar buttons
- Use the commands `advance` and `move` to navigate through the questionnaire
- Use the function `visualvalue` to get the value of variables that are “off path”
- Use the command `showarray` to display tables to the interviewer

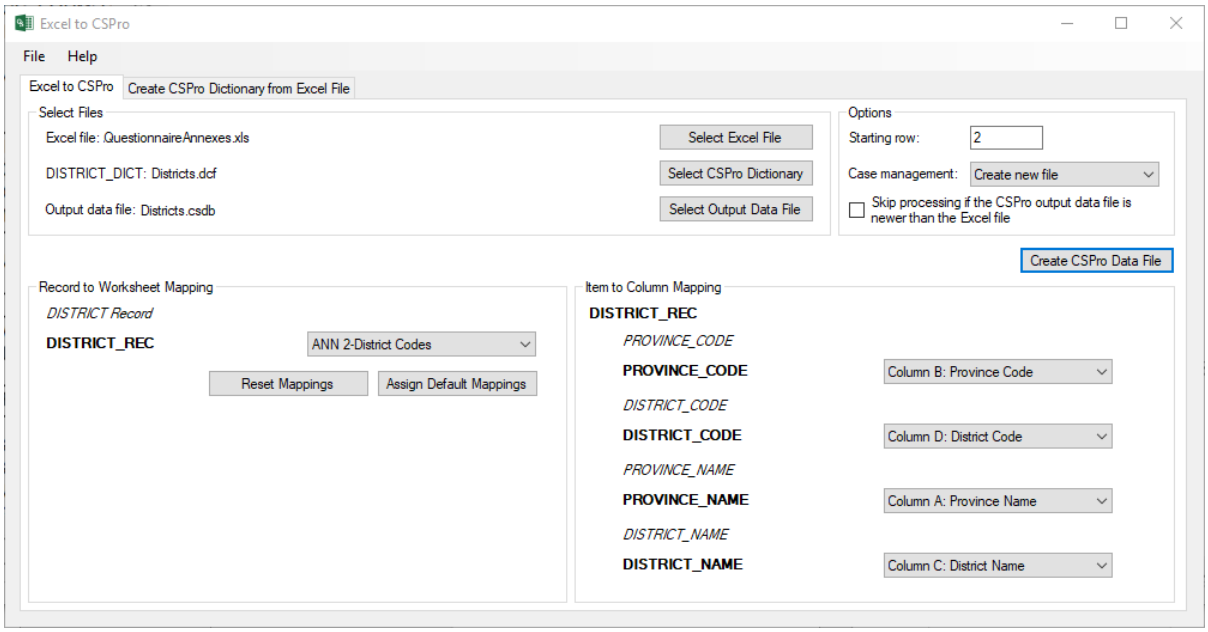
Lookup Files

It is currently possible for the interviewer to enter a district code that is not valid for the province that was selected. In order to verify that the district code is valid for the province we need to use the list of province and district codes in the annex of the questionnaire. We can do this by creating a lookup file from the Excel spreadsheet. We can then use this lookup file to check for valid districts.

First, we need to create a dictionary for our lookup file. For this task, we want to be able to query if the combination of the province and district codes is a valid district. In annex 2 of the questionnaire we have a table with the province code, district code and district name. We can use this as a lookup file where the keys are the province and district code and the value is the district name. We can create a dictionary for this using the Excel2CSPro tool from the tools menu. Select the “Create CSPro dictionary from Excel File” tab, click “Select Excel File” and browse to the file QuestionnaireAnnexes.xlsx. Choose “ANN2 – District Codes” and click “Analyze Worksheet”. This detects the dictionary variables to create from the columns of the spreadsheet. The tool will show four variables, one for each column and will set the default names to column headers of the spreadsheet. Our lookup file dictionary will have the province code and district code as the id-items and province and district names as regular variables. Next to the province and district codes check the “ID” box to indicate that these should be ID items. In order to avoid name conflicts with the main dictionary we can use DI_PROVINCE, DI_DISTRICT and DI_DISTRICT_NAME as the variable names. Modify them in the Excel to CSPro tool.



We must make sure that the lengths and zero-fill settings of the id-items exactly match those in the main dictionary otherwise we won’t be able to use the variables from the main dictionary as keys for the lookup. Verify that the length of the province and district codes match those in the main dictionary and check the box “Zero fill numerics” to make sure that the id-items are zero filled. Once all settings are correct, click create dictionary to generate the CSPro dictionary file. Save it as “Districts.dcf” in the household folder.



Once we have the dictionary we need to convert the Excel spreadsheet into a CSPro data file. We can use the first tab of the Excel2CSPro tool to do this. Select the QuestionnaireAnnexes.xlsx file again along with the Districts dictionary that we just created. Specify a new output data file to write the lookup file to. Under "Record to Worksheet Mapping" choose the second worksheet "ANN 2-District Codes". Under the "Item to Column Mapping" link the dictionary variables to the appropriate columns in the Excel spreadsheet: DI_PROVINCE_CODE to "Column B: Province Code", DI_DISTRICT_CODE to "Column D: District Code", etc… Finally, click "Create CSPro Data File" to generate the file. Verify the file in DataViewer to make it sure it was converted correctly.

Finally, in the district proc use the `loadcase()` command to lookup the province and district codes in the file. Loadcase takes the name of the dictionary (**DISTRICTS_DICT**) and the values to use as keys (id-items) for the lookup. For example, to lookup province 3, district 6 we would do:

```
loadcase(DISTRICTS_DICT, 3, 6)
```

If loadcase finds a record in the lookup file with province code 3 and district code 6 it will return 1 and set the variables in **DISTRICTS_DICT** to the values from the case it found. In this case that means setting the two id-items **DI_PROVINCE**, **DI_DISTRICT** and the variables **DI_DISTRICT_NAME** and **DI_PROVINCE_NAME**.

We can use this to test if the province and district codes are valid in the **DISTRICT** proc:

```
PROC DISTRICT

// Verify that the district code is valid for the province selected.
if loadcase(DISTRICTS_DICT, PROVINCE, DISTRICT)  = 0 then
    errmsg("District code %d is not valid for province %s",
           DISTRICT, getlabel(PROVINCE, PROVINCE));
    reenter;
else
    errmsg("You have selected district: %s", DI_DISTRICT_NAME);
endif;
```

Note that we are using the **PROVINCE** and **DISTRICT** from the main dictionary as arguments to `loadcase`, not the id-items from the districts dictionary. Before calling loadcase the id-items for the external dictionary are all blank. They are only set if `loadcase` is successful.

Note that when you run this application, in addition to copying the pen and pff files to the Android device, you must now also copy the lookup file (the .csdb file).

We can now add alpha variables to the main dictionary, assign the province and district names to them and display them on the form as protected fields:

```
PROC DISTRICT

// Verify that the district code is valid for the province
// selected.
if loadcase(DISTRICTS_DICT, PROVINCE, DISTRICT)  = 0 then
    errmsg("District code %d is not valid for province %s",
           DISTRICT, getlabel(PROVINCE, PROVINCE));
    reenter;
else
    // Assign province and district names from lookup to main
    // dictionary variables so we can display them on form.
    PROVINCE_NAME = DI_PROVINCE_NAME;
    DISTRICT_NAME = DI_DISTRICT_NAME;
endif;
```

Updating a Lookup File

If the data for your lookup file changes once your survey is in the field, rather than completely regenerating the lookup file you can update the existing lookup data file based on a new Excel spreadsheet. Choose the existing csdb file as the output data file and under "Case Management" choose "Modify, add cases" or "Modify, add, delete cases". Instead of erasing and regenerating the data file this will update only the cases that are new or have been modified in the Excel file. Choosing "Modify, add, delete cases" will also delete cases that are in the data file but are not in the spreadsheet. Using this approach is important if you are planning to use CSPro data synchronization to update the lookup file during data collection. If you are using data synchronization to update the file and you generate a completely new data file you will get duplicate cases when the new lookup file is downloaded to the devices. If instead, you update the existing file then data synchronization will simply update the data files on devices that download it.

Dynamic Value Set from a Lookup File

An alternative to using the lookup file to check the district and province codes is to combine the lookup file with `setvalueset` to create a dynamic value set for the district that includes only districts in the selected province. To do this we need to extract all the districts in the selected province from the lookup file. We can do this using the `forcase` loop which iterates over all cases in an external data file. By itself, `forcase` will loop through each case in the data file. You can also add an optional “where” clause to only go through the districts in the selected province.

```
PROC DISTRICT
onfocus
// Create dynamic value set of districts for selected province using lookup file
numeric nextEntry = 1;
forcase DI_DICT where DI_PROVINCE_CODE = PROVINCE do
    codes(nextEntry) = DI_DISTRICT_CODE;
    labels(nextEntry) = DI_DISTRICT_NAME;
    nextEntry = nextEntry + 1;
endfor;
codes(nextEntry) = notappl;
setvalueset(DISTRICT, codes, labels);
```

User defined functions

Often you find you have identical blocks of logic in multiple procs in your application. This can cause problems if you later change the code in one place to fix a bug and forget to change it in the other. In such situations it is better to put the logic in a user defined function which you can then call from all the procs where it is used. User defined functions are defined in the PROC GLOBAL. You call them the same way you call built in CSPro functions.

User defined functions can take arguments and return values just like built in functions. Let’s define a function that we can use in all the places where we check if a household member is a woman of childbearing age. This function will be passed the index (row number) of the household member in the household roster and it will return one if the person is a woman over 12 years old and zero otherwise. This way if we later decide that we should be using 13 or 14 as a minimum age we only have to make the change in one place.

```
// Determine if member of household from household roster is a woman of childbearing
// age. Pass in the index (occurrence number) of the household member.
function isChildbearingWoman(index)
    if SEX(index) = 2 and AGE(index) >= 12 then
        isChildbearingWoman = 1;
    else
        isChildbearingWoman = 0;
    endif;
end;
```

Now we can use this function when building the value sets for line number of mother of child (**B10**) and the line number of mother of deceased (**E08**).

```
PROC MOTHER_LINE_NUMBER
onfocus
// Create the value set for child mother from all eligible women
// in household roster.
numeric indexRoster;
numeric nextEntryValueSet = 1;
do indexRoster = 1 while indexRoster <= totocc(HOUSEHOLD_MEMBERS_ROSTER)

    if isChildbearingWoman(indexRoster) = 1 then
        labels(nextEntryValueSet) = NAME(indexRoster);
        codes(nextEntryValueSet) = indexRoster;
        nextEntryValueSet = nextEntryValueSet + 1;
    endif;
enddo;
labels(nextEntryValueSet) = "non-resident";
codes(nextEntryValueSet) = 87;
nextEntryValueSet = nextEntryValueSet + 1;
labels(nextEntryValueSet) = "deceased";
codes(nextEntryValueSet) = 88;
nextEntryValueSet = nextEntryValueSet + 1;
codes(nextEntryValueSet) = notappl;
setvalueset(MOTHERS_LINE_NUMBER, codes, labels);

PROC MOTHER_OF_DECEASED_LINE_NUMBER
onfocus
// Create the value set for deceased mother from all eligible women
// in household roster
numeric indexRoster;
numeric nextEntryValueSet = 1;
do indexRoster = 1 while indexRoster <= totocc(HOUSEHOLD_MEMBERS_ROSTER)

    if isChildbearingWoman(indexRoster) = 1 then
        labels(nextEntryValueSet) = NAME(indexRoster);
        codes(nextEntryValueSet) = indexRoster;
        nextEntryValueSet = nextEntryValueSet + 1;
    endif;
enddo;
labels(nextEntryValueSet) = "not in household";
codes(nextEntryValueSet) = 99;
nextEntryValueSet = nextEntryValueSet + 1;

codes(nextEntryValueSet) = notappl;
setvalueset(DECEASED_MOTHERS_LINE_NUMBER, codes, labels);
```

Looking at these two procs we could probably move the shared code that creates the value set into a function as well.

```
// Create a value set of all household members that are eligible to be
// mothers by filling in the global codes and labels arrays.
// Labels are names of household members and codes are the corresponding
// line numbers.
// Returns the number of eligible members in the value set.
// The household member in row excludeIndex will not be included in the value
// set. This can be used to exclude someone from being their own mother.
function createMothersValueSet(excludeIndex)

    numeric indexRoster;
    numeric nextEntryValueSet = 1;
    do indexRoster = 1 while indexRoster <= totocc(HOUSEHOLD_MEMBERS_ROSTER)

        if isChildbearingWoman(indexRoster) = 1 and indexRoster <> excludeIndex then
            labels(nextEntryValueSet) = NAME(indexRoster);
            codes(nextEntryValueSet) = indexRoster;
            nextEntryValueSet = nextEntryValueSet + 1;
        endif;
    enddo;

   createMothersValueSet = nextEntryValueSet;

end;

PROC MOTHER_LINE_NUMBER
onfocus
// Create the value set for child mother from all eligible
// women in household roster. Exclude current occurrence so
// that person cannot be their own mother.
numeric nextEntryValueSet = createMothersValueSet(curocc());
// Add additional entries for non-resident and deceased
labels(nextEntryValueSet) = "non-resident";
codes(nextEntryValueSet) = 87;
nextEntryValueSet = nextEntryValueSet + 1;
labels(nextEntryValueSet) = "deceased";
codes(nextEntryValueSet) = 88;
nextEntryValueSet = nextEntryValueSet + 1;
// Mark end of value set with notappl
codes(nextEntryValueSet) = notappl;
// Update the value set with values in codes and labels
setvalueset(MOTHER_LINE_NUMBER, codes, labels);

PROC MOTHER_OF_DECEASED_LINE_NUMBER
onfocus
// Create the value set for child mother from all eligible
// women in household roster
numeric nextEntryValueSet = createMothersValueSet(0);

// Add additional entry for not in household
labels(nextEntryValueSet) = "not in household";
codes(nextEntryValueSet) = 99;
nextEntryValueSet = nextEntryValueSet + 1;

// Mark end of value set with notappl
codes(nextEntryValueSet) = notappl;
// Update the value set with values in codes and labels
setvalueset($, codes, labels);
```

The Userbar

From logic we can add buttons to the CSEntry user interface that call user-defined functions. Here is how to add a userbar button that will create an errmsg dialog that says “hello”. First we define the function hello in the PROC GLOBAL:

```
function hello()
    errmsg("Hello");
end;
```

Then in the preproc of the application we add it to the userbar:

```
PROC POPSTAN2020_FF
preproc
userbar(clear);
userbar(add button, "Hello", hello);
userbar(show);
```

When adding a button in the preproc of the application it is important to call clear first, otherwise we can end up with two or three copies of the same button if we start the application multiple times. We added the button in the preproc of the application but you can add or remove buttons in any proc so you can, for example, only a show button within a certain field or a certain roster.

Let’s try a more interesting example. Let’s add a “Go To…” button that will let the user navigate directly to a particular section of the questionnaire.

```
// Userbar function for navigating directly to different parts of questionnaire.
function goto()
    numeric section = accept("Go to?",
                             "Identification",
                             "Household members",
                             "Demographics");
    if section = 1 then
        skip to IDENTIFICATION_FORM;
    elseif section = 2 then
        skip to HOUSEHOLD_MEMBERS_FORM;
    elseif section = 3 then
        skip to DEMOGRAPHICS_FORM;
    endif;
end;

PROC HOUSEHOLDQUESTIONNAIRE_FF
preproc
userbar(clear);
userbar(add button, "Go To...", goto);
userbar(show);
```

Advance and Move

The above will let the interviewer jump from one section to another but using `skip` means that if we use our *Go To…* button to jump over a section, that section will end up skipped and won’t be saved in the data file. Instead of using `skip` we can use `advance` which moves forward in the questionnaire without marking fields as skipped. Using `advance` also runs all the preprocs and postprocs of the fields that are passed through to ensure that no consistency or out of range checks are missed.

```
// Userbar function for navigating directly to different parts of questionnaire.
function goto()
    numeric section = accept("Go to?",
                             "Identification",
                             "Households",
                             "Demographics");
    if section = 1 then
        advance to IDENTIFICATION_FORM;
    elseif section = 2 then
        advance to HOUSEHOLD_MEMBERS_FORM;
    elseif section = 3 then
        advance to DEMOGRAPHICS_FORM;
    endif;
end;
```

This stops the function from skipping over data when navigating, however it still has a limitation. We can only navigate forward in the questionnaire. To go backwards we need to use `reenter`, but in our goto() function we don't know if the user wants to move forward or backward. Fortunately, CSPro provides the command `move` which will use either `skip` or `reenter` as appropriate. By default, when going forward, move does a skip but you can add `advance` after the field name to make it do an `advance` instead of a `skip`.

```
// Userbar function for navigating directly to different parts of questionnaire.
function goto()
    numeric section = accept("Go to?",
                             "Identification",
                             "Household members",
                             "Demographics");
    if section = 1 then
        move to IDENTIFICATION_FORM advance;
    elseif section = 2 then
        move to HOUSEHOLD_MEMBERS_FORM advance;
    elseif section = 3 then
        move to DEMOGRAPHICS_FORM advance;
    endif;
end;
```

Let's extend our goto function to let the interviewer navigate directly to an individual in the household roster. If they choose "household members" then instead of going to the first person in the roster, we will show them a list of all household members in the roster and the let them choose which one to goto. For this we can use the function `showarray()`. This function takes an array of values and displays them in a grid in a dialog box and returns the row number that the user picks.

Unlike `setvalueset()` that takes two arrays, `showarray()` takes a two-dimensional array. You can think of a two-dimensional array as a grid of variables or as a matrix. You declare a two-dimensional array in the PROC GLOBAL the same way you declare a one-dimensional array except that you specify the size in both dimensions: number of rows and number of columns.

```
array string householdMembersArray(30, 3);
```

In our case we want up to 30 rows, one for each person, and we will use 3 columns so that we can display the name, sex and relationship for each person.

When assigning a value to a two-dimensional array you specify both the row and column you want to put the value in:

```
// Set value in row 4, column 2
householdMembersArray(4, 2) = "This is the 4th row, 2nd column";
```

In our examples we will loop through the members in the household roster and add the name, sex and relationship for each one to our array. Note that for sex and relationship we want the value set labels so we use getlabel(). Since this will be more than a few lines of code let's put this into a function by itself and then call it from our goto() function.

```
// Show list of entries in household roster in a dialog and let interviewer pick
// one. Returns the row number of the person that was picked or zero if the
// dialog was canceled.
function pickFromHouseholdRoster()
    numeric i;
    do i = 1 while i <= totocc(HOUSEHOLD_MEMBERS_ROSTER)
        householdMembersArray(i, 1) = strip(NAME(i));
        householdMembersArray(i, 2) = getlabel(SEX, SEX(i));
        householdMembersArray(i, 3) = getlabel(RELATIONSHIP, RELATIONSHIP(i));
    enddo;
    householdMembersArray(i, 1) = ""; // Mark end
    numeric picked = showarray(householdMembersArray, title("Name", "Sex",
                                                            "Relationship"));
    pickFromHouseholdRoster = picked;
end;

// Userbar function for navigating
// directly to different parts of questionnaire.
function goto()
    numeric section = accept("Go to?",
                             "Identification",
                             "Household members",
                             "Demographics");
    if section = 1 then
        move to IDENTIFICATION_FORM advance;
    elseif section = 2 then
        numeric index = pickFromHouseholdRoster();
        if index > 0 then
            move to NAME(index) advance;
        endif;
    elseif section = 3 then
        move to DEMOGRAPHICS_FORM advance;
    endif;
end;
```

Path and Visualvalue

Our goto() function works when we are in section B or C but when we are in section A the sex and relationship are blank. What is going on? In system controlled mode, CSEntry keeps track of which variables are on and off the “path”. Variables that you have entered are considered “on path” but those that have been skipped, even if there was a value in them before they were skipped, are considered “off path”. As we have seen before, variables that are “off path” are considered blank (`notappl`) in logic. It turns out that variables that are ahead of the current field are also considered “off path” until you pass through them. The idea is that these fields have not yet been validated by running their preproc and postproc with the current values of all preceding fields and therefore cannot be considered final. The effect of this is that the values of all fields ahead of the current field in the questionnaire are `notappl` in logic.

Interestingly, as we can see from our current code, this only applies to numeric items. Our code works just fine for the **NAME** field.

You can see which fields are “on path” by looking at the background color of the field:

- Green: on path
- Dark Grey: skipped
- White: not yet been filled in
- Light grey: protected

Note that the field coloring scheme is different in operator controlled mode.

Fortunately, we can get the value of fields that are “off path” using the function `visualvalue()`. It returns whatever

value is currently visible in the field whether or not it has been skipped or is ahead of the current field. Using this for relationship and sex in our function we get:

```
function pickFromHouseholdRoster()
    numeric i;
    do i = 1 while i <= totocc(HOUSEHOLD_MEMBERS_ROSTER);
        householdMembersArray(i, 1) = strip(NAME(i));
        householdMembersArray(i, 2) = getlabel(SEX, visualvalue(SEX(i)));
        householdMembersArray(i, 3) = getlabel(RELATIONSHIP,
                                              visualvalue(RELATIONSHIP(i)));
    enddo;
    householdMembersArray(i, 1) = ""; // Mark end
    numeric picked = showarray(householdMembersArray,
                               title("Name", "Sex", "Relationship"));
    pickFromHouseholdRoster = picked;
end;
```

All that is left now is to use the appropriate relationship for the sex of the household member:

```
if visualvalue(SEX(i)) = 1 then
    householdMembersArray (i, 3) = getlabel(RELATIONSHIP_MALE,
                                  visualvalue(RELATIONSHIP(i)));
else
    householdMembersArray (i, 3) = getlabel(RELATIONSHIP_FEMALE,
                                  visualvalue(RELATIONSHIP(i)));
endif;
```

Setting field values only once

Let's prefill the interview start time. We can make the field protected and set the value in the preproc just like we did with the **PERSON_NUMBER** field. We can use the function `systime()` which returns the current time as a number formatted according to the format specification passed in.

```
PROC INTERVIEW_START_HOURS
preproc
// Prefill interview start time with current time.
INTERVIEW_START_TIME = systime("HHMM");
```

This works the first time we visit the field but what happens we come back to the question a minute or two later? We only want to record this value the first time the interviewer enters the field and once it is set we don't want it to change. We can do this by comparing the value of **INTERVIEW_START_TIME** to blank (`notappl`) in the preproc and only setting the value to systime if it is blank.

```
PROC INTERVIEW_START_HOURS
preproc
// Prefill interview start time with current time.
if INTERVIEW_START_HOURS = notappl then
    INTERVIEW_START_TIME = systime("HHMM");
endif;
```

But this doesn't seem to work. Why? Is **INTERVIEW_START_HOURS** on path when we are in the preproc of **INTERVIEW_START_HOURS**? It is not. We have to get to the postproc for the variable to be on path. However, we can use `visualvalue()` to get the value of the field in the preproc:

```
PROC INTERVIEW_START_HOURS
preproc
// Prefill interview start time with current time.
if visualvalue(INTERVIEW_START_HOURS) = notappl then
    INTERVIEW_START_TIME = systime("HHMM");
endif;
```

We can also fill in the interview end time automatically using `systime()`. Unlike with the start time, we need to do this at the end of the interview, i.e. in the postproc of the questionnaire.

```
PROC POPSTAN2020_QUEST
postproc

// Set interview end time first time end of questionnaire is reached
if INTERVIEW_END_TIME = notappl then
    INTERVIEW_END_TIME = systime("HHMM");
endif;
```

We should make this field protected, however if we do that we get an error when we don't fill it in while in section A of the questionnaire. While we don't normally want to allow a field to be left blank, in this case we need to make an exception. We can do that using the `setproperty()` command. `setproperty` allows you to modify properties of fields and of the application from within CSPro logic. To allow entry of blanks in a field we need to set the "CanEnterNotAppl" property to "NoConfirm". We can do this in the preproc of the application so that it is done once when the application first starts.

```
PROC POPSTAN2020_FF
preproc
setproperty(INTERVIEW_END_TIME_HOURS, "CanEnterNotAppl", "NoConfirm");
setproperty(INTERVIEW_END_TIME_MINUTES, "CanEnterNotAppl", "NoConfirm");
```

Having captured the interview start and end times as hours and minutes we can calculate the total interview time by subtracting the start time from the end time, however this calculation is tricky because we have to handle the case where the end minutes are less than the start minutes. It would be easier if instead of using `systime()` we used the function `timestamp()` which gives the time in total seconds since January 1, 1970. If we subtract the start timestamp from the end timestamp we get thte total interview time in seconds. Let's add new protected variables for start timestamp, end timestamp and interview duration and fill them in.

```
PROC POPSTAN2020_FF
preproc
setproperty(INTERVIEW_END_TIME_HOURS, "CanEnterNotAppl", "NoConfirm");
setproperty(INTERVIEW_END_TIME_MINUTES, "CanEnterNotAppl", "NoConfirm");
setproperty(INTERVIEW_END_TIMESTAMP, "CanEnterNotAppl", "NoConfirm");
setproperty(INTERVIEW_DURATION_MINUTES, "CanEnterNotAppl", "NoConfirm");

PROC INTERVIEW_START_TIMESTAMP
preproc
// Prefill interview start time with current time.
if visualvalue(INTERVIEW_START_TIMESTAMP) = notappl then
    INTERVIEW_START_TIMESTAMP = timestamp();
endif;

PROC POPSTAN2020_QUEST
postproc

// Set interview end time and total time first time end of questionnaire is reached
if INTERVIEW_END_TIMESTAMP = notappl then
    INTERVIEW_END_TIMESTAMP = timestamp();
    INTERVIEW_DURATION_MINUTES = (INTERVIEW_END_TIMESTAMP -
INTERVIEW_START_TIMESTAMP)/60;
endif;
```

Exercises

1. Extend the goto() function to include the remaining sections of the questionnaire (D through G).
2. Add a button to the userbar called “household summary” that when clicked uses the errmsg function to display a message that shows the name of the head of household, and the number of household members by sex. For example: “Head of Household: John Brown, Total Members: 5, Women: 2, Men: 3”. Bonus if you can get this to work when you click the button from section A. Hint: count and seek will not work with visualvalue so you will need to use a loop to find the number of males of and females.
3. Implement a check on the minimum and maximum per unit possession values in question G01. Use the spreadsheet in annex 4 to create a lookup file containing the asset code, minimum value and maximum value. Use the loadcase command with this lookup file to find the minimum and maximum values for the selected asset and show an error message if the per unit asset value entered in the roster is below the minimum value or above the maximum value. This should be a soft check.
4. Fill in the interview start and end date automatically using the sysdate() command and make the interview start and end date fields protected.
5. For question A10, interview status, we want to fill in the question at the start of the interview if the response is code 2 (non-contact), 3 (vacant) or 4 (refused) and then immediately end the questionnaire without going into any of the subsequent questions. However, if there is a respondent willing to give the interview, then the interview status should be set 5 (partially complete) until the entire questionnaire is complete at which point it should be set to 1 (complete). To implement this, keep the question in its current position in the questionnaire but use a dynamic value set to limit the options so that the interview cannot be set as completed until the entire question has been completed. The first time the field is entered (before any value has been entered) the value set should be: 2 Non-contact, 3 Vacant, 4 Refused, 5 Continue interview. If the interviewer chooses 2, 3 or 4, end the interview, otherwise, if they choose 5, continue to the next field. At the end of the interview (postproc of the level), if the value is currently 5, set it to 1 (complete). If the interviewer returns to A9 after the field has been set to 1 (complete) then display the value set as it is on the questionnaire.

Session 7: Multimedia & GPS

At the end of this session participants will be able to:

- Use images in value sets
- Launch external applications from a data entry application
- Launch photo and document viewers from a data entry application
- Capture GPS coordinates in a data entry application
- Display captured GPS points on a map
- Generate and view reports from a data entry application

Value Set Images

In addition to showing text in a value set we can also display images. To add an image to a value set, select the variable in the dictionary editor and click on the "…" button next to the value. Browse to pick an image file. CSPro supports standard image file formats: png, jpeg, bmp… We recommend using png or jpeg as they tend have smaller file sizes. The images are not added to the pen file by default so you would need to copy them to the device along with the pen file.

Let's add images for the roof types in **F08**. Copy the RoofImages folder into the application directory. Now open the value set for **F08** and add each image to its respective value. We do not have a picture for other (specify) so we will leave that one without an image. Run the application on Windows and on mobile and see the images. Note that on mobile we have to copy the images to the device since they are not part of the pen file by default.

The Resource Folder

Instead of copying the image files to the mobile device each time they are updated we can put them in the resource directory of the application. All files in the resource folder are built into the pen file and extracted on the device when the application is run. This makes it more convenient to distribute an application with additional files. Let's create a resource folder, add it to our application and put our image files in it. To create the resource file just create a new folder in Windows explorer. We will create the folder "resources" in the household folder. Then in CSPro go to Add Files… from the File menu, choose "resource folder" and browse to the new folder. Now put the whole RoofImages folder in the resource folder and rebuild the pen file.

Launching External Applications on Windows

The command `execsystem()` lets you launch another application from logic. It takes the name of the application to launch. For example, using a userbar button you could bring up the Window's calculator:

```
PROC GLOBAL

// Launch the calculator
function showCalculator()
    execsystem("calc.exe");
end;

PROC POPSTAN2020_FF
preproc
userbar(clear);
userbar(add button, "Go To...", goto);
userbar(add button, "Household summary", showHouseholdSummary);
userbar(add button, "Calculator", showCalculator);
userbar(show);
```

For Windows utilities like Calculator and Notepad which are in the system directory you can simply give the name. For other applications that are installed in the Program Files directory you may need to specify the full path. For example, to launch Microsoft Word (Office 2016 version) you would use:

```
execsystem("C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE");
```

Launching External Applications on Android

The `execsystem()` command is slightly different on Android. On Android to launch an application with `execsystem()` use "app:" followed by the package name of the application. For example, to launch Gmail:

```
execsystem("app:com.google.android.gm");
```

Finding the package name can be a bit tricky. The easiest way is to use your web browser to search for the application on the Google Play website (play.google.com). On the page for the application the package name will be the last part of the address in the address bar of your browser, after the "?id=". For example, if you go the page for CSEntry on the Google Play store the address is:

https://play.google.com/store/apps/details?id=gov.census.cspro.csentry

The package name is therefore "gov.census.cspro.csentry".

To launch the calculator from Android we would use: [1]

```
execsystem("app:com.google.android.calculator");
```

To make our code work on both Android and Windows we can use the function `getos()` which returns a number between representing the operating system:

Code	Operating System
10	Windows
20	Android
30	Universal Windows

```
// Launch the calculator
function showCalculator()
    if getos() = 10 then
        // Windows
        execsystem("calc.exe");
    elseif getos() = 20 then
        execsystem("app:com.google.android.calculator");
    else
        errmsg("Calculator not supported on this system");
    endif;
end;
```

Viewing Files on Android

On Android `execsystem()` can also be used to launch files with their default viewer.

```
execsystem("view:/mnt/sdcard/csentry/picture.jpg");
execsystem("view:/mnt/sdcard/csentry/audio.mp3");
execsystem("view:/mnt/sdcard/csentry/movie.mp4");
execsystem("view:/mnt/sdcard/csentry/document.pdf");
```

For this to work, the device must have an application installed capable of viewing the type of file specified. Android determines which application to launch to view the file based on the file extension. All Android devices have viewers for most photo, music, and video file formats, however not all have a built-in PDF viewer. If you do not have one on your device, you can always download one from Google Play.

[1] On some Android devices the default calculator is replaced with a different package and the package name may need be changed.

Let's add a userbar button to show a map of the enumeration area as a visual aid for the interviewer.

```
// Display an image containing a map of the enumeration area
function showEAMap()
    execsystem(
      "view:/mnt/sdcard/csentry/Popstan2020/Housing/resources/maps/EAMap.png");
end;
```

We can make our code a little more flexible by using the function pathname() to get the directory where the application is stored. This way if someone copies it into a different folder on the phone/tablet it will still work:

```
// Display an image containing a map of the enumeration area
function showEAMap()
    execsystem(maketext("view:%s/resources/maps/EAMap.png",
                        pathname(Application)));
end;
```

Our code is currently limited to a single enumeration area. We could improve it to the show a different image based on the current value of the enumeration area variable.

```
// Display an image containing a map of the enumeration area
function showEAMap()
    execsystem(maketext("view:%sresources/maps/EAMap%03d.png",
                        pathname(Application), ENUMERATION_AREA));
end;
```

Taking Photos on Android

To take a picture on Android you can use execsystem with "camera:" followed by the full path in which to save the photo:

```
execsystem("camera:/mnt/sdcard/csentry/photo.jpg");
```

Note that photos do not get saved into the application data file. You need to save them to a folder/file on the device. The filename you use should make it easy to link the photo back to the interview that it was taken for. The easiest way to do this is to make the name of the photo based on the id-items of the questionnaire:

```
string photoFilename = pathname(application) +
                maketext("../Data/HouseholdPhotos/photo%d%02d%03d%d%03d.jpg",
                                PROVINCE, DISTRICT, ENUMERATION_AREA,
                                AREA_TYPE, HOUSEHOLD_NUMBER);
execsystem(maketext("camera:/mnt/sdcard/csentry/%s", photoFilename));
```

Note that we are putting the photo in a subdirectory of the data directory named HouseholdPhotos. We will need to create this folder on the tablet for this to work.

Let's have the interviewer take a photo of the household being interviewed in section A. Rather than adding a userbar button which would be easy for the interviewer to forget, we will make it part of the questionnaire by adding a dummy variable to the dictionary. In the postproc of the variable we will call execsystem() to take the photo. We can add a refused code in case the household doesn't want a picture taken.

```
PROC PHOTO

if PHOTO = 1 then
    // Take photo
    string photoFilename = pathname(application) +
              maketext("../Data/HouseholdPhotos/photo%d%02d%03d%d%03d.jpg",
                                  PROVINCE, DISTRICT, ENUMERATION_AREA,
                                  AREA_TYPE, HOUSEHOLD_NUMBER);
    execsystem(maketext("camera:%s", photoFilename));
elseif PHOTO = 9 then
    // refused, move to next field with no photo
endif;
```

This works the first time through the questionnaire but if we go back through the photo question a second time then we are forced to take another photo. We can avoid this by adding another option to the value set “Keep photo” that the interviewer will select once they are happy with the photo that they took. When “keep photo” is selected we won’t launch the camera. We can add another option “View photo” to let them look at the photo that was taken.

```
PROC PHOTO

string photoFilename = pathname(application) +
              maketext("../Data/HouseholdPhotos /photo%d%02d%03d%d%03d.jpg",
                                  PROVINCE, DISTRICT, ENUMERATION_AREA,
                                  AREA_TYPE, HOUSEHOLD_NUMBER);

if PHOTO = 1 then
    // Take/retake photo
    execsystem(maketext("camera:%s", photoFilename));

    // reenter so that interview can retake if it is not good.
    reenter;
elseif PHOTO = 2 then
    // view existing photo
    execsystem(maketext("view:%s", photoFilename));

    // reenter so that interview can retake if it is not good.
    reenter;
elseif PHOTO = 3 then
    // Keep photo - move to next field
elseif PHOTO = 9 then
    // No photo (refused)
    // Delete photo if it exists
    filedelete(photoFilename);
endif;
```

This works except that it is possible for the interviewer to select “Keep photo” even when there is no existing photo and to attempt to view a photo that doesn’t exist. We can solve this by only showing the “Keep photo” and “View photo” options in the value set if the photo exists already.

```
PROC PHOTO
onfocus
string photoFilename = pathname(application) +
               maketext("../Data/HouseholdPhotos/photo%d%02d%03d%d%03d.jpg",
                                PROVINCE, DISTRICT, ENUMERATION_AREA,
                                AREA_TYPE, HOUSEHOLD_NUMBER);

// Only show the view and keep options in value set
// if the photo exists.
if fileexist(photoFilename) then
    setvalueset(PHOTO, PHOTO_VS_PHOTO_TAKEN);
else
    setvalueset(PHOTO, PHOTO_VS_NO_PHOTO);
endif;
```

Viewing GPS Points

On Android you can use `execsystem` to display a GPS point on a map using "gps:" followed by the latitude and longitude. We can add a userbar button to display the household location on a map:

```
// Display household GPS point on map
function showHouseholdOnMap()
    execsystem(maketext("gps:%f,%f", LATITUDE,LONGITUDE));
end;
```

Capturing GPS Points

To capture a GPS point first you need to start up the GPS hardware:

```
gps(open);
```

Then you request a GPS reading giving it a timeout in seconds and optionally a desired accuracy in meters:

```
if gps(read, 60, 10)  then // Read up to 60 seconds, try for 10m accuracy
    errmsg("Latitude is %f, longitude is %f",gps(latitude),gps(longitude));
else
    errmsg("GPS signal could not be acquired");
endif;
```

Finally, you close the GPS:

```
gps(close);
```

In addition to querying the GPS latitude and longitude you can also query the accuracy, number of satellites, and altitude (although on Android altitude is not very accurate). See the help on GPS for details.
The longitude and latitude reported are in degrees in the WGS84 datum and are returned as decimal numbers.

Let's add a user button to update the GPS coordinates for the household:

```
// Capture current household location using GPS
function getGPS()
    gps(open);
    if gps(read, 60) then // Read for up to 60 seconds
        LATITUDE = gps(latitude);
        LONGITUDE = gps(longitude);
    else
        errmsg("GPS signal could not be acquired");
    endif;
    gps(close);
end;
```

If we want to be fancy, we can display the point on the map first so that the interviewer can view it and then decide whether or not to use it.

```
// Capture current household location using GPS
function getGPS()
    gps(open);
    if gps(read, 60) then // Read for up to 60 seconds
        // Show map so that interview can see result.
        execsystem(maketext("gps:%f,%f",
                   gps(latitude),gps(longitude)));
        if accept("Save this result", "Yes", "No") = 1 then
            LATITUDE = gps(latitude);
            LONGITUDE = gps(longitude);
        endif;
    else
        errmsg("GPS signal could not be acquired");
    endif;
    gps(close);
end;
```

Group Exercise

Build a CSPro application for the household listing questionnaire. Place your application in the folder Popstan2020/Listing. Treat each household as a separate case. In other words, do not use a repeating record and roster to try to match the paper questionnaire. When creating the listing dictionary use "LI_" as a prefix for the all the variable names so that they will not conflict with the names in the household dictionary. There is no need to validate the id-items using a lookup file like we did for the household questionnaire. We will prefill the id-items from the menu program tomorrow. For each household, automatically record the GPS coordinates in the latitude field. Give the interviewer the option of taking a photo of the household but allow for refusal. Name the photo based on the case id-items and put the photos in the directory Popstan2020/Data/ListingPhotos. Go outside and test your listing application on a tablet.

Writing and Viewing Reports

With `execsystem()` on Android you can display text files on your device using "view:" followed by the filename. We can use this to show text documents that we write out from CSPro. To write out files from a CSEntry application we have to first declare a file variable in the PROC GLOBAL:

```
PROC GLOBAL
file tempFile;
```

Then we can use the commands `setfile()`, `filewrite()` and `close()` to open, write to and close the file. Let's create a userbar button to write a simple text report that lists the household members:

```
// Write out and display household summary report
function showHouseholdReport()
    string reportFilename = maketext("%sreport.txt", pathname(Application));
    setfile(tempFile, reportFilename);
    filewrite(tempFile, "Household Summary Report");
    filewrite(tempFile, "------------------------");
    filewrite(tempFile, "");
    filewrite(tempFile, "Province %d District %d EA %d Area type %d Household Number
%d",
                        visualvalue(PROVINCE),
                        visualvalue(DISTRICT),
                        visualvalue(ENUMERATION_AREA),
                        visualvalue(AREA_TYPE),
                        visualvalue(HOUSEHOLD_NUMBER));
    filewrite(tempFile, "");
    filewrite(tempFile, "Household Members:");
    filewrite(tempFile, "");
    filewrite(tempFile, "Name                            Sex    Age  Relationship");
    filewrite(tempFile, "----                            ---    ---  ------------");
    do numeric i = 1 while i <= totocc(HOUSEHOLD_MEMBERS_ROSTER)
        filewrite(tempFile, "%s %-6s %3d  %s",
                            NAME(i),
                            getlabel(SEX, visualvalue(SEX(i))),
                            visualvalue(AGE(i)),
                            getlabel(RELATIONSHIP_VS1,
                            visualvalue(RELATIONSHIP(i))));
    enddo;
    close(tempFile);
    if getos() = 20 then
        // Android - use "view:"
        execsystem(maketext("view:%s", reportFilename));
    elseif getos() = 10 then
        // Windows - use "explorer.exe <filename>"
        execsystem(maketext("explorer.exe %s", reportFilename));
    endif;
end;
```

On Windows we can't use execsystem with "view:" so instead we launch Windows Explorer (explorer.exe) with the name of the file and that will open the file in notepad.

If you are familiar with HTML you can write out HTML and have nicer formatting and pictures. You can use `execsystem()` with "html:file://" followed by the filename to view html files on your device. CSPro also has a built in templating engine to generate HTML reports using templates. See "Templated Reports" in the CSPro help for details.

Exercises

1. Add a button to the userbar that shows the interviewer manual. You can use the PDF interviewer manual that is on the course website (it only has a cover page but is enough for this exercise).
2. Add the following additional information to the summary report. a. Total number of household members, total males, total females. b. List of household assets from section G with quantity. Do not list assets where the quantity is zero.

Session 8: Menu Programs

At the end of this session participants will be able to:

- Understand the how to design the screens of a menu program
- Use the command execpff() to launch one CSPro application from another
- Create a simple menu program to launch the main data entry program
- Create dynamic menus with dynamic value sets

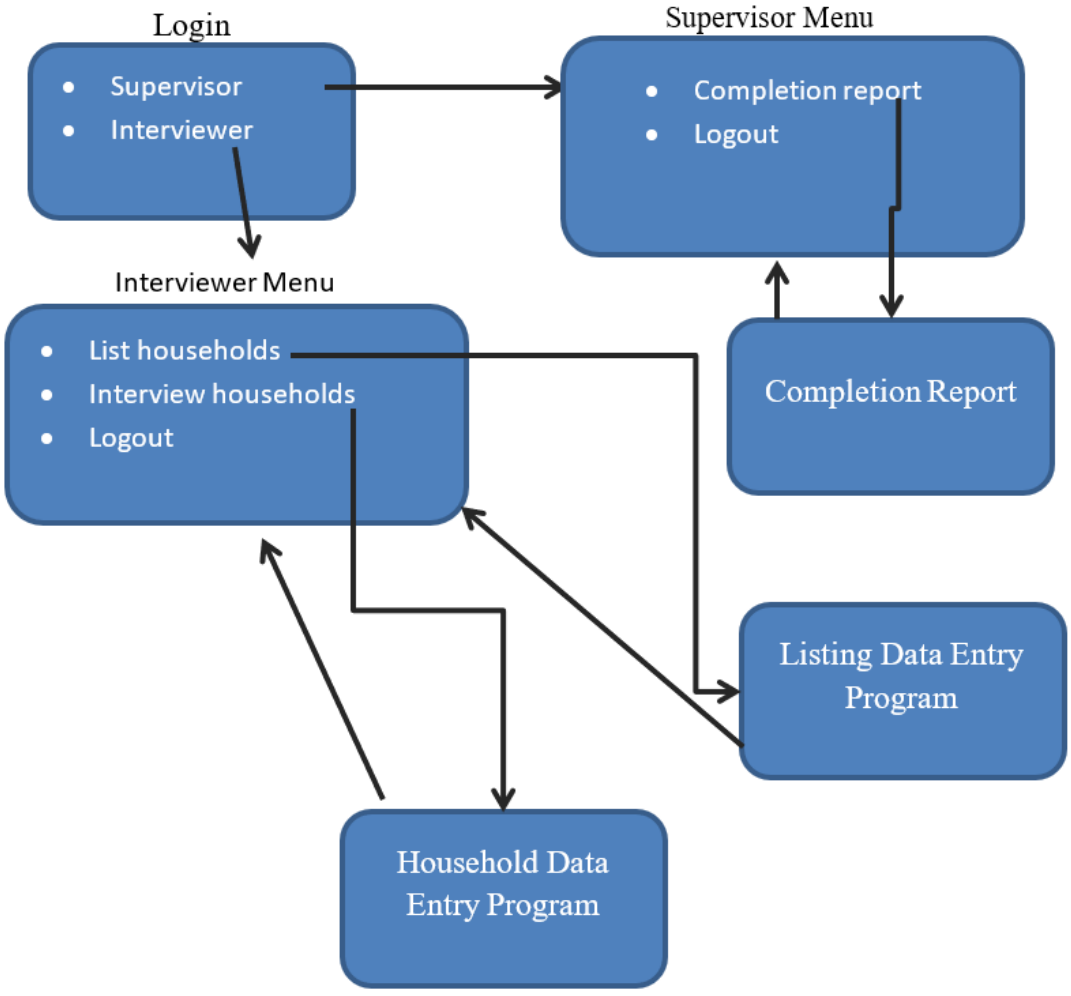
What is a Menu Program?

A menu program is a CSPro data entry application that is used to manage the data entry workflow. A menu program is not used for capturing any interview data itself. Instead, it launches other data entry programs for interviews. Menu programs generally have some or all of following functions:

- Launches other CSPro applications to do data collection (often pre-filling id items)
- Show reports on progress, summary statistics
- Manage user access through userid/passwords
- Manage household listing and interview assignments
- Launch synchronization

Designing the Screens and Flow

Once you decide on which functions you want your menu program to perform, the next step is to design the screens and how they link together. This is most easily done as a diagram like the one below for a fairly simple menu program.



Creating the Dictionary and Forms

Once you have the screens, the next step is to create a new data entry application for your menu program. Let's name ours Menu and put it in the folder *Popstan2020/Menu*.

Each menu screen will be a different variable in the dictionary. The value set for the variable will show the available menu choices to the enumerator on that screen. In the menu example above we will have three variables:

- LOGIN (value set: 1 – Interviewer, 2 – Supervisor)
- INTERVIEWER_MAIN_MENU (value set: List Households - 1, Interview Household – 2, Logout -9)
- SUPERVISOR_MAIN_MENU (value set: Completion report – 1, Logout – 9).

We will keep the default id-item that CSPro creates for us. The menu program dictionary doesn't really need an id-item since we do not need to save our menu choices to the data file. However, CSPro requires that we have at least one id-item so we will keep it.

Create the items in the dictionary and then create a form and drop the items onto the form. Do not drop the id-item onto the form. We can leave it off the form since we will never actually write out a completed case from the menu program. Unlike typical data entry applications, menu programs tend not to have a linear flow. As a result, the order of the variables on the form is less important. We will use skips and reenters to move from one menu to another.

Menu Program Logic

The logic for processing the menu choice for each screen goes in the postproc of the variable for the menu. For example, to process the login menu field in our example we would have the following logic:

```
PROC LOGIN

// Go to the appropriate menu for the role chosen
if $ = 1 then
    skip to INTERVIEWER_MAIN_MENU;
else
    skip to SUPERVISOR_MAIN_MENU;
endif;
```

If the user selects to login as an interviewer we skip to the interviewer main field to show the interviewer menu, otherwise we skip to the supervisor main menu field to show the supervisor menu.

Handling the interviewer menu is similar. Here we will create user defined functions to launch the listing and household programs.

```
PROC INTERVIEWER_MAIN_MENU
postproc

// Handle the menu choice
if $ = 1 then
    // List households
    launchHouseholdListing();
elseif $ = 2 then
    // Household questionnaire
    launchHouseholdDataEntry();
elseif $ = 9 then
    // Logout
    stop(1);
endif;

// Show interviewer menu again
reenter;
```

It is important to make sure that after the postproc of the menu field we do not let CSEntry continue to the next field,

otherwise after the interviewer launches the household listing they would end up in the next field, which, in this case is the supervisor menu. To prevent this, we put a reenter at the end of the postproc so that we go back into the same menu field again.

It looks kind of strange that when we go back into a menu, the previous choice is still selected. We can prevent this by clearing the choice in the postproc of the field before the reenter.

```
// Clear previous entry
$ = notappl;

// Show interviewer menu again
reenter;
```

The supervisor menu is similar to the interviewer menu:

```
PROC SUPERVISOR_MAIN_MENU

// Handle the menu choice
if $ = 1 then
    // Show report
    showCompletionReport();
elseif $ = 9 then
    // Logout
    stop(1);
endif;

// Clear previous entry
$ = notappl;

// Show supervisor menu again
reenter;
```

Launching one CSPro Application from Another

Let's fill in the function to launch the household data entry program. We can launch other CSPro programs from within our data entry program using `execpff()`. You give `execpff` the path to a pff file for a CSPro application and it will start the application using the parameters in the pff file. In addition to the filename, you can also pass execpff either the keyword `wait` or the keyword `stop`. Using `wait` will pause the menu application until the application that was launched exits, while using `stop` will exit the menu application immediately after launching the other application. On mobile devices, we will always use the stop parameter since it is not possible to run two CSEntry applications at the same time on mobile.

The following logic will launch a data entry application using the pff file Popstan2020.pff in the sibling directory to the menu program directory.

```
function launchHouseholdDataEntry()
    execpff("../Household/Popstan2020.pff", stop);
end;
```

This will start the data entry application and immediately exit the menu. The ".." in the path to Popstan2020.pff tells CSEntry to go one directory up, i.e. to go to the parent of the Menu directory and from there to Household/Popstan2020.pff.

Making the Entry Program Return to the Menu Program

When we exit the main questionnaire, we want to return to the menu but the menu program stopped when we launched the main questionnaire. To get it to restart we can add a parameter to the pff file to tell it start the menu program when it exits. Right click on the Popstan2020.pff and choose "Edit". This will bring up the pff file editor. You can then use the options menu to Add a new **On Exit Pff** entry. Use the browse button in that entry to locate the Menu.pff and fill in the path to it.

Deploying multiple applications on Android

When you copy your application to Android it is important that you preserve the same folder structure as you have on the PC. In our case we must create a separate menu folder to contain the pen and pff files for the menu and this menu folder

must be in the same parent directory as the Household folder so that when we use "../Household/" from the menu application it points to the folder containing the entry application we are launching.

One way to simplify deployment is to use a Windows batch file to generate the pen files for both the entry application and the menu application. Copy both applications into a deployment folder that can be copied onto the device. This can greatly speed up testing of your application on Android.

```
setlocal

REM Find CSEntry.exe (path differs on 32 and 64 bit Windows)

SET CSEntry="%ProgramFiles(x86)%\CSPro 7.1\CSEntry.exe"
if exist %CSEntry% goto :gotcspro
SET CSEntry="%ProgramFiles%\CSPro 7.1\CSEntry.exe"
if exist %CSEntry% goto :gotcspro
echo "Can't find CSEntry version 7.1. Is it installed?"
goto :eof
:gotcspro


REM Create deployment directory
rmdir /q /s Deployment
mkdir Deployment
cd Deployment
mkdir Popstan2020
cd Popstan2020
mkdir Menu
mkdir Household
mkdir Listing
cd ..
cd ..

REM Create .pen files
cd Menu
%CSEntry% /pen %CD%\Menu.ent
cd ..\Household
%CSEntry% /pen %CD%\Popstan2020.ent
cd ..\Listing
%CSEntry% /pen %CD%\Listing.ent
cd ..

REM Copy applications to deployment
move /y .\Menu\Menu.pen .\Deployment\Popstan2020\Menu
move /y .\Household\Popstan2020.pen .\Deployment\Popstan2020\Household
move /y .\Listing\Listing.pen .\Deployment\Popstan2020\Listing

REM Copy .pff files
copy /y .\Menu\Menu.pff .\Deployment\Popstan2020\Menu
copy /y .\Household\Popstan2020.pff .\Deployment\Popstan2020\Household
copy /y .\Listing\Listing.pff .\Deployment\Popstan2020\Listing
```

Tidying up the Menu Program

Currently the menu program shows up in the applications list on Android as “Menu” instead of something like “Popstan 2020 Census”. In addition, when we tap on Menu we have to then tap “Start New Case” which doesn’t make sense for a menu. We can fix both of these problems by modifying the pff file for the menu. Right click on the Menu.pff and choose “Edit with pff editor”. Change “Start mode” to “Add” so that we won’t have to tap “Start New Case”. Enter “Popstan 2020 Census” for the description to change what shows in the case listing. While we are at it, open the Menu program in the CSPro designer and in data entry options turn off the display of the case tree since the case tree is not useful for menu programs. Finally, in the data entry options, check "Automatically advance on selection". This way the interviewer will not have to tap next to move from one field to another in the menu program.

Now that we have the menu program to launch the household data entry program, we don't want the household application to appear in the list of applications on the mobile device. By default, CSPro creates the list of applications from all of the pff files in the CSEntry directory on the device. To prevent an application from being listed you can edit the pff file in the pff editor and choose "Hidden by Default" or "Never" for "Show in Application Listing".

Creating a Dynamic Menu

When the interviewer launches the data entry application, they can currently enter any id-items but we would like to restrict them to only interviewing households that have already been listed. We can do this by displaying the households from the listing file in a dynamic value set and having the interviewer choose which one to interview.

In order to read the listing file, we need to add the listing dictionary as an external dictionary in the menu application. We will also need to add a new menu item to list the households. This will be a new variable in the dictionary named **CHOOSE_HOUSEHOLD**. We will make it 10 digits long to hold the full set of id-items (province, district, EA, area type and household number). It will be an alpha field as this will be easier to work with later on. The interviewer menu will need to be modified to skip to this new field instead of calling launchHouseholdDataEntry().

```
PROC INTERVIEWER_MAIN_MENU
postproc

// Handle the menu choice
if $ = 1 then
    // List households
    launchHouseholdListing();
elseif $ = 2 then
    // Household questionnaire
    skip to CHOOSE_HOUSEHOLD;
elseif $ = 9 then
    // Logout
    stop(1);
endif;

// Clear previous entry
$ = notappl;

// Show interviewer menu again
reenter;
```

In the onfocus proc for **CHOOSE_HOUSEHOLD** we need to loop through every case in the listing file. We can use the `forcase` statement to loop through the cases in the listing file and build the value set:

```
PROC CHOOSE_HOUSEHOLD
onfocus

numeric nextEntry = 1;

// Loop through all cases in listing file
// to build dynamic value set.
forcase LISTING_DICT do
    // Values are household ids concatenated together
    codesString(nextEntry)=maketext("%d%02d%03d%d%03d",
        LI_PROVINCE, LI_DISTRICT, LI_ENUMERATION_AREA,
        LI_AREA_TYPE, LI_HOUSEHOLD_NUMBER);

    // Labels have household number and name of head
    labels(nextEntry) = maketext("%03d: (%s)", LI_HOUSEHOLD_NUMBER,
                            strip(LI_NAME_OF_HEAD_OF_HOUSEHOLD));

    nextEntry = nextEntry + 1;
endfor;

// Mark end of array
codesString(nextEntry) = "";

setvalueset($, codesString, labels);
```

When we run this we should see a list of the households in the listing file as the value set for **CHOOSE_HOUSEHOLD**.

While the dynamic value set works correctly when there are households that have been listed, if the listing file is empty we get and empty value set. Instead we should show an error message and return to the main menu.

```
PROC CHOOSE_HOUSEHOLD
onfocus

numeric nextEntry = 1;

// Loop through all cases in listing file
// to build dynamic value set.
forcase LISTING_DICT do
    // Values are household ids concatenated together
    codesString(nextEntry)=maketext("%d%02d%03d%d%03d",
        LI_PROVINCE, LI_DISTRICT, LI_ENUMERATION_AREA,
        LI_AREA_TYPE, LI_HOUSEHOLD_NUMBER);

    // Labels have household number and name of head
    labels(nextEntry) = maketext("%03d: (%s)", LI_HOUSEHOLD_NUMBER,
                            strip(LI_NAME_OF_HEAD_OF_HOUSEHOLD));

    nextEntry = nextEntry + 1;
endfor;

if nextEntry = 1 then
    errmsg("No households have been listed. Please list households first before
interviewing.");
    reenter INTERVIEWER_MAIN_MENU;
endif;

// Mark end of array
codesString(nextEntry) = "";

setvalueset($, codesString, labels);
```

Generating the PFF File

The next step is to handle the menu choice in the postproc to launch the household data entry program using the id-items for the household that was chosen. In order to do this, we will need to specify the id-items as parameters in the pff file. The first step will be to change the launchHouseholdDataEntry() function to first write out the pff file before launching it so that we can customize it. We can use `filewrite` to write out the file as we did with the reports in the last lesson. An easy way to write this logic is to use open the pff file in the pff editor and enable "expert mode". This displays an extra tab containing CSPro logic to write out the pff file. We can copy and paste this into our logic and make a few modifications.

First we change paths to the data entry application and the data file to be based on the path to the application using the `pathname` command. `pathname(Application)` returns the path to the current application, which in this case is the menu program. By adding ".." we go up one level to the Popstan2020 directory and from there we can get to the application and data files.

```
// Launch household questionnaire data entry application
function launchHouseholdDataEntry()
    string pffFilename = pathname(Application) +
                         "../Household/Popstan2020.pff";

    if setfile(pffFile,pffFilename,create) = 0 then
        errmsg("Failed to open file %s", pffFilename);
    endif;

    filewrite(pffFile,"[Run Information]");
    filewrite(pffFile,"Version=CSPro 7.1");
    filewrite(pffFile,"AppType=Entry");
    filewrite(pffFile,"Description=Popstan 2020 Census");

    filewrite(pffFile,"[DataEntryInit]");

    filewrite(pffFile,"[Files]");
    filewrite(pffFile,"Application=" + pathname(Application) +
                      "../Household/Popstan2020.ent");
    filewrite(pffFile,"InputData=" + pathname(Application) +
                      "../Data/Popstan2020.csdb");

    filewrite(pffFile,"[ExternalFiles]");
    filewrite(pffFile,"DISTRICTS_DICT=" + pathname(Application) +
                      "../Household/Resources/Districts.csdb");
    filewrite(pffFile,"HOUSEHOLDPOSSESSIONVALUE_DICT=" +
                      pathname(Application) +
                     "../Household/Resources/HouseholdPossessionValues.dat");

    filewrite(pffFile,"[UserFiles]");
    filewrite(pffFile,"TEMPFILE=%s","");

    filewrite(pffFile,"[Parameters]");
    filewrite(pffFile,"OnExit=%s",pathname(Application) +
                        "Menu/Menu.pff");
    filewrite(pffFile,"PROVINCE=%s", CHOOSE_HOUSEHOLD[1:1]);
    filewrite(pffFile,"DISTRICT=%s", CHOOSE_HOUSEHOLD[2:2]);
    filewrite(pffFile,"ENUMERATION_AREA=%s", CHOOSE_HOUSEHOLD[4:3]);
    filewrite(pffFile,"AREA_TYPE=%s", CHOOSE_HOUSEHOLD[7:1]);
    filewrite(pffFile,"HOUSEHOLD_NUMBER=%s", CHOOSE_HOUSEHOLD[8:3]);

    close(pffFile);

    execpff(pffFilename, stop);
end;
```

We also extract the id-items from the **CHOOSE_HOUSEHOLD** field and pass them as parameters in the pff file so that they

can be retrieved by the household data entry application.

Pre-filling the Household Id-items

The last step is to modify the household data entry application to prefill the id-items using the parameters in the pff file. This can be using the `sysparm()` command which retrieves a parameter by name from the pff file. We can do this in the preproc of each of the id-items in the household application. `sysparm` always returns the result as a string so we need to convert it to a number.

```
PROC PROVINCE
preproc

// Retreive parameters from menu program via pff file
if sysparm("PROVINCE") <> "" then
    PROVINCE = tonumber(sysparm("PROVINCE"));
endif;
```

We use an if statement to only fill in the field if the pff file actually has a value for the parameter. This way we can still easily test our application without using the menu program. We should also make the field protected if we are filling it in. We can do this using `setproperty`.

```
PROC PROVINCE
preproc

// Retreive parameters from menu program via pff file
if sysparm("PROVINCE") <> "" then
    PROVINCE = tonumber(sysparm("PROVINCE"));

    // protect field so the interviewer cannot modify it
    setproperty(PROVINCE, "Protected","Yes");

endif;
```

The logic for **DISTRICT**, **ENUMERATION_AREA**, **AREA_TYPE** and **HOUSEHOLD_NUMBER** is similar.

Since we have not set the start mode in the pff file that we write out, after adding a first case, the next time we launch a case from the menu program, it goes to the case listing screen instead of starting the case. We can fix this by setting the **StartMode** parameter to “add” in the pff for the household application the way we did for the menu program. In order for this to work when we are modifying an existing case we need to add the case id to the **StartMode** line. In the menu program, the case id to use is just **CHOOSE_HOUSEHOLD**. Finally, so that after finishing a case we return to the menu program rather than returning to the case listing we add "Lock=CaseListing" to the pff file which prevents the case listing from ever being shown.

```
filewrite(pffFile,"[DataEntryInit]");
filewrite(pffFile,"StartMode=add;%s", CHOOSE_HOUSEHOLD);
filewrite(pffFile,"Lock=CaseListing");
```

Using a Lookup File for Login

Rather than have the user choose their role, lets assign each interviewer and supervisor a staff code and then create a lookup file containing the staff codes along with the role and the district/enumeration area assigned to the interviewer/supervisor. Here is an example staff file:

Staff code	Name	Role (1=interviewer, 2=supervisor)	Province	District	EA
001	Shemika Rothenberger	2	1	1	1
002	Andrew Benninger	1	1	1	1
003	Angelica Swenson	1	1	1	2
004	Zelma Hawke	1	1	1	3
005	Willis Catron	1	1	1	4

Note that for the supervisor we leave the enumeration area blank since they are assigned to an entire district and not to an enumeration area. The supervisor will supervise all of the enumerators in their district.

Let's create a new external dictionary in the menu program for this file and use Excel2CSPro to convert the spreadsheet to a data file named staff.dat.

Let's modify the login field so that the user enters the staff code instead of picking the role. We will need to increase the size of the field and delete value set. In the postproc we now need to look up the staff code in the staff file, validate it, and then skip to the appropriate menu based on the role.

```
PROC LOGIN

// Verify staff code using lookup file
if loadcase(STAFF_DICT, LOGIN) = 0 then
    errmsg("Invalid staff code. Try again.");
    reenter;
endif;

// Go to the appropriate menu for the role chosen
if STAFF_ROLE = 1 then
    skip to INTERVIEWER_MAIN_MENU;
else
    skip to SUPERVISOR_MAIN_MENU;
endif;
```

Preserving Login when Returning to Menu

Currently when you launch the listing program and return to the menu, you have to enter the user code again. Let's save the login code so that you only have to enter it once. We can do this using the commands `savesetting()` and `loadsetting()`. These commands store and retrieve values in persistent storage. Values saved in the settings are available even after CSEntry is closed and restarted, and they are available in all CSPro applications on the same device. We will save the login code in the login postproc after validating it:

```
PROC LOGIN
postproc
// Verify staff code using lookup file
if loadcase(STAFF_DICT, LOGIN) = 0 then
    errmsg("Invalid staff code. Try again.");
    reenter;
endif;

// Save login so we do not have to enter it again
savesetting("login", maketext("%d", LOGIN));

// Go to the appropriate menu for the role chosen
if STAFF_ROLE = 1 then
    skip to INTERVIEWER_MENU;
else
    skip to SUPERVISOR_MENU;
endif;
```

Settings are always stored as alphanumeric so we need to use `maketext` to convert the numeric **LOGIN** code to a string.

In the preproc we will try to retrieve the login code from the settings and if it is not empty we will use it instead of asking the user to enter the code.

```
PROC LOGIN
preproc
// Check to see if there is an existing login code
// use that
if loadsetting("login") <> "" then
    LOGIN = tonumber(loadsetting("login"));
    noinput;
endif;
```

Finally, we need to clear the setting on logout:

```
PROC INTERVIEWER_MAIN_MENU
postproc

// Handle the menu choice
if $ = 1 then
    // List households
    launchHouseholdListing();
elseif $ = 2 then
    // Household questionnaire
    launchHouseholdDataEntry();
elseif $ = 9 then
    // Logout
    // Clear login from settings
    savesetting("login", "");
    stop(1);
endif;

// Show interviewer menu again
reenter;
```

The Completion Report

Let's add the report to the supervisor menu. We will create a report that shows the total number of households by interview status. Here is an example:

```
Completion Report
-----------------
Province: 01 District: 01

Interview Status
----------------
Completed: 10
Non-contact: 1
Vacant: 2
Refused: 1
Partially complete: 5
Total: 19
```

To generate this report, we need to loop through all the cases in the household dictionary and count the number of cases with each interview status. We can use `forcase` to do this. To compute the totals for each category we create a local variable for each status and increment the appropriate variable every time we encounter a household with that status.

```
// Display the completion report that shows total interview status
// for all cases in the supervisor's district.
function showCompletionReport()

    string reportFilename = maketext("%sreport.txt", pathname(Application));
    setfile(tempFile, reportFilename);

    filewrite(tempFile, "Completion Report");
    filewrite(tempFile, "-----------------");
    filewrite(tempFile, "");
    filewrite(tempFile, "Province %d District %02d",
                        visualvalue(STAFF_PROVINCE),
                        visualvalue(STAFF_DISTRICT));
    filewrite(tempFile, "");
    filewrite(tempFile, "Interview Status");
    filewrite(tempFile, "----------------");

    numeric complete = 0;
    numeric nonContact = 0;
    numeric vacant = 0;
    numeric refused = 0;
    numeric partial = 0;

    forcase POPSTAN2020_DICT do
        if INTERVIEW_STATUS = 1 then
            complete = complete + 1;
        elseif INTERVIEW_STATUS = 2 then
            nonContact = nonContact + 1;
        elseif INTERVIEW_STATUS = 3 then
            vacant = vacant + 1;
        elseif INTERVIEW_STATUS = 4 then
            refused = refused + 1;
        else
            partial = partial + 1;
        endif;
    endfor;

    filewrite(tempFile, "Completed: %d", complete);
    filewrite(tempFile, "Non-contact: %d", nonContact);
    filewrite(tempFile, "Vacant: %d", vacant);
    filewrite(tempFile, "Refused: %d", refused);
    filewrite(tempFile, "Partially complete: %d", partial);
    filewrite(tempFile, "Total: %d", complete + nonContact +
                                     vacant + refused + partial);

    close(tempFile);
    if getos() = 20 then
        // Android - use "view:"
        execsystem(maketext("view:%s", reportFilename));
    elseif getos() = 10 then
        // Windows - use "explorer.exe <filename>"
        execsystem(maketext("explorer.exe %s", reportFilename));
    endif;
end;
```

Exercises

1. Modify the launchHouseholdDataEntry() function to write the staff code as a parameter in the pff file used to launch the household application. Modify the household application to prefill the interviewer code (**A9**) with the staff code from the pff file.
2. Implement the function launchHouseholdListing() in the menu program. It should write out and launch a pff file to run the listing program. Write the province, district, enumeration area and staff code from the staff file as parameters in the pff file and prefill those items in the listing program. Make sure that the listing program returns to the menu program on exit.
3. Modify the logic in the onfocus and postproc of **CHOOSE_HOUSEHOLD** to add an extra option to the end of the list labelled "Back" that will move back to the interviewer main menu.
4. Modify the dynamic value set we create in the onfocus proc of **CHOOSE_HOUSEHOLD** to show the interview status (**A10** in the household questionnaire) in addition to the household number and the head's name. To do this you will need to use loadcase on the household questionnaire dictionary to find the case from the household data file.
5. Create a new menu called "Summary Reports" that has options for the completion report (the one we implemented already) and a new "Total Population Report" that you will implement. This new menu should also have an option to go back to the main menu. The "Summary Reports" menu should be accessed from the Supervisor Main Menu. The new "Total Population Report" should look like:

```
Total Population Report
-----------------------
Province: 01 District: 02
Male: 1020
Female: 1025
Total: 2045
```

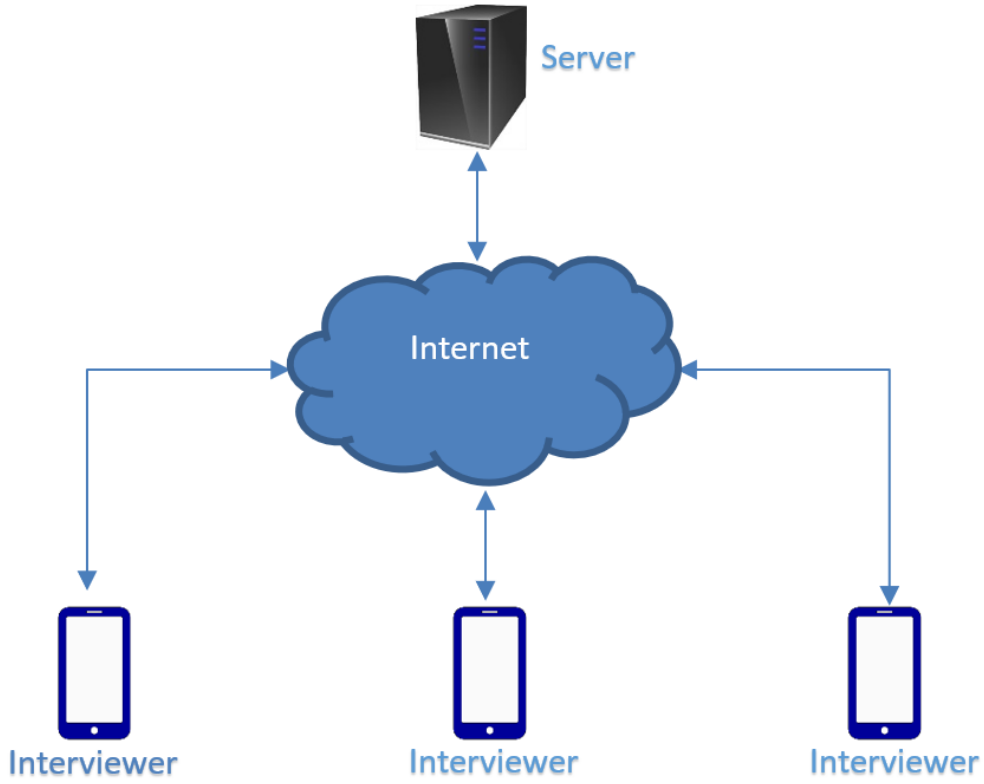
Session 9: Synchronization

At the end of this session participants will be able to:

- Use the synchronization options dialog to add synchronization to an application
- Create advanced synchronizations using CSPro logic to synchronize data files and update applications in the field
- Create peer to peer synchronizations between devices for situations where there is no internet access

Synchronization in CSPro

After collecting data in the field, you need to get the data from the interviewer's devices back to headquarters to create a combined data file. While this can be done manually by connecting each device to a laptop, copying the individual data files and combining them using the Concatenate Data tool, this is rather inconvenient for large surveys. Instead, it is possible to send the data from each tablet over the internet to a central server that will combine the data into a single data file to be used for further processing. In CSPro this is called synchronization.



Direct synchronization between interviewers and central server over the internet

The Sync Server

In order to use internet synchronization in CSPro you need a server accessible over a network. CSPro supports three types of Synchronization servers:

- CSWeb: A free web application that can run in a local data center or hosted on a server in the cloud. Best for large surveys. Requires skills in setting up and administering a website and SQL database.
- Dropbox: A free service for synchronization with servers in the cloud using an account created at www.dropbox.com. Data is stored on the servers managed by Dropbox in the United States. Best for small surveys when the skills and infrastructure for setting up CSWeb are not available.
- FTP: CSPro can synchronize with any FTP (File Transfer Protocol) server accessible over the network. Best for small surveys when the skills and infrastructure for setting up CSWeb are not available and you don't want your data on the Dropbox servers.

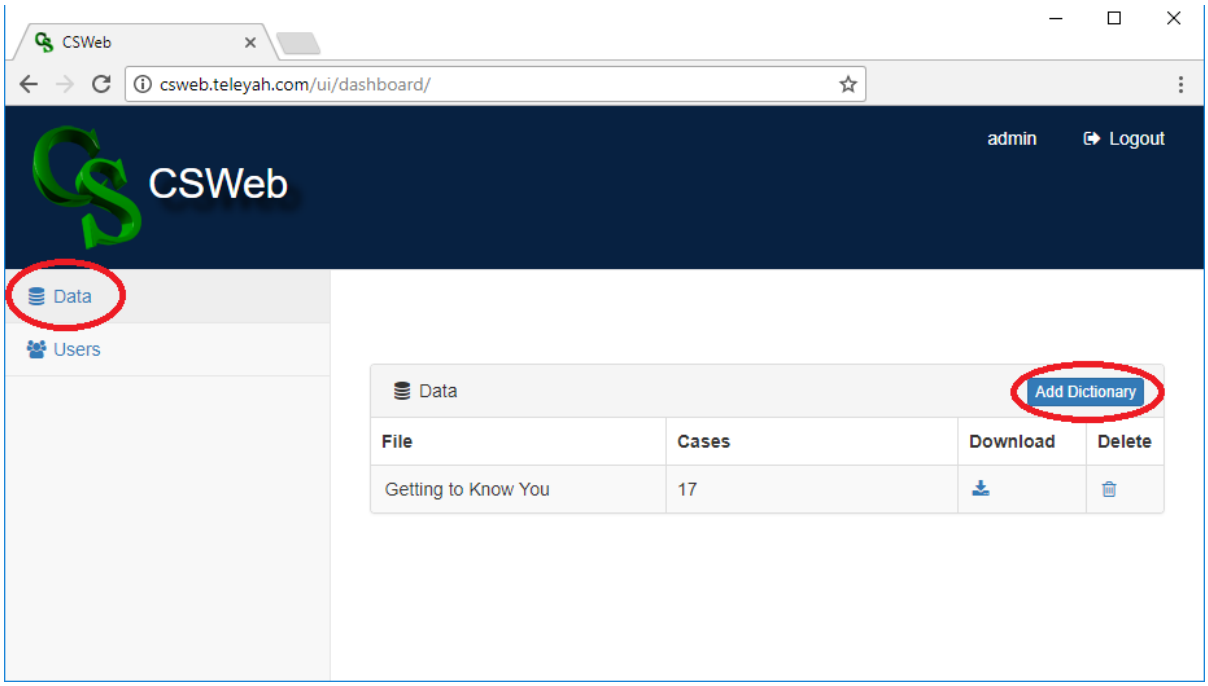
For our examples in this workshop we will use a CSWeb server, however using Dropbox or FTP is very similar. Installing the

CSWeb server is beyond the scope of this training so we will work with a server that is already installed and running. The process for setting up a CSWeb server is well documented in the CSWeb users guide which is included in the CSPro documentation available from the "Help Menu".

Using the CSPro Web Server interface

Before using the CSWeb server for synchronization we need to upload the data dictionary used by the application so CSWeb can create a database table to store the data in. This step is not needed if you are using FTP or Dropbox as the server. With those servers, the dictionary is uploaded on the first sync. To upload the dictionary, first log into the CSWeb server in a web browser.

Next, choose “Data” from the left menu, click the “Add Data” button and browse to the dictionary to upload.

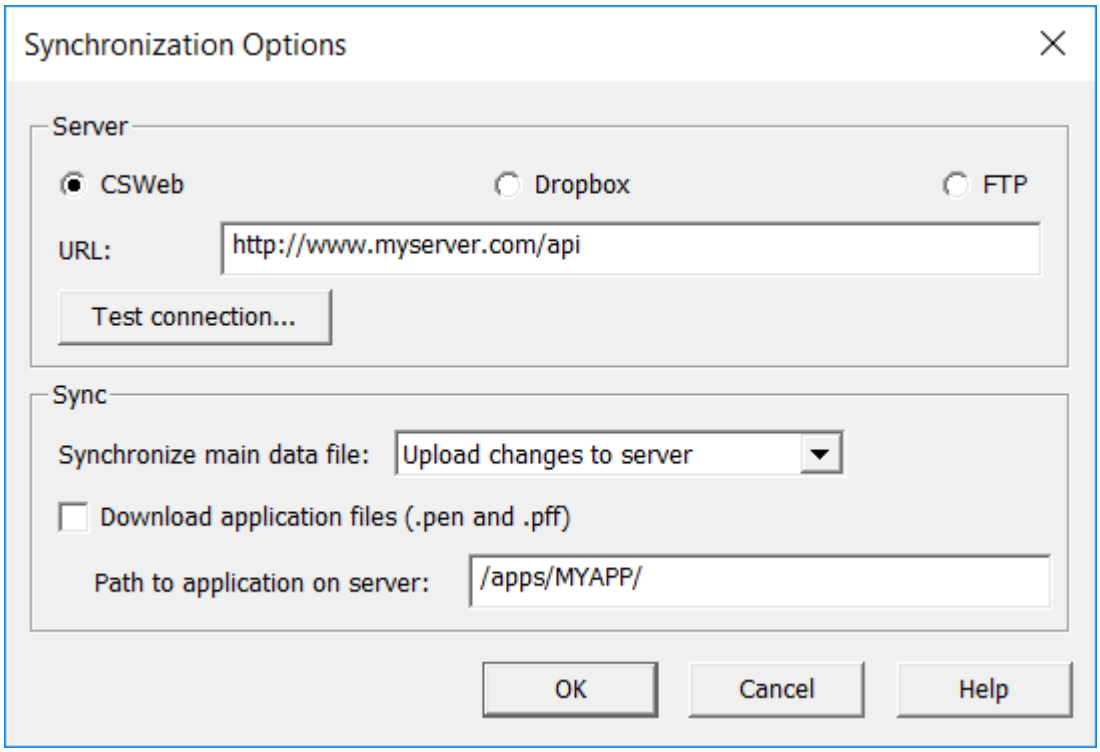


You should now see the dictionary listed in the table under Data. The server is now ready for synchronization.

You can also use this web interface to add users for synchronization.

Adding Synchronization to the Data Entry Application

In order to use synchronization, you must first add the server information to the application to enable synchronization. In CSPro choose “Synchronization…” from the options menu. Choose the server type (CSWeb, Dropbox or FTP). For CSWeb or FTP, enter the server URL (for Dropbox no URL is required). You can use the “Test Connection…” button to verify that the URL is correct. For CSWeb, the URL to use for synchronization in CSPro will usually end with "/api" while the URL to use to access the web interface will usually end in "/ui". The former is for CSPro itself to connect to the server while the latter is for use only in the web browser.



In the dropdown, you have three choices for how to synchronize the main data file:

- Upload changes to server: all new cases and any cases modified locally will be uploaded to the server.
- Download changes from server: all new cases and cases modified on server will be downloaded from server.
- Sync local and remote changes: combines the first two options by sending updates to the server and downloading updates from the server.

In most cases you will want to choose either the first or third option. Use the third option if you want to share cases between multiple devices in the field as this will download every case on the server onto the local device. For a survey operation with a large number of interviewers you may want to avoid this as it will cause you to download a lot of data. For our test, we will use the first option.

In addition to synchronizing the data file you can also download the pen and pff files from the server. This provides a way to do remote updates to the application while interviewers are in the field. In order for this to work you must place a copy of the pen and pff files on the server. For CSWeb these must be placed in the *csweb/files/* on the server. For Dropbox, they may be placed anywhere in the Dropbox folder. For FTP they must be placed in the home directory of the user that will be used to log into the server. Enter the path on the server where the files were placed under “Path to application on server”. We will place these files in the directory */Popstan2020Census* on the server.

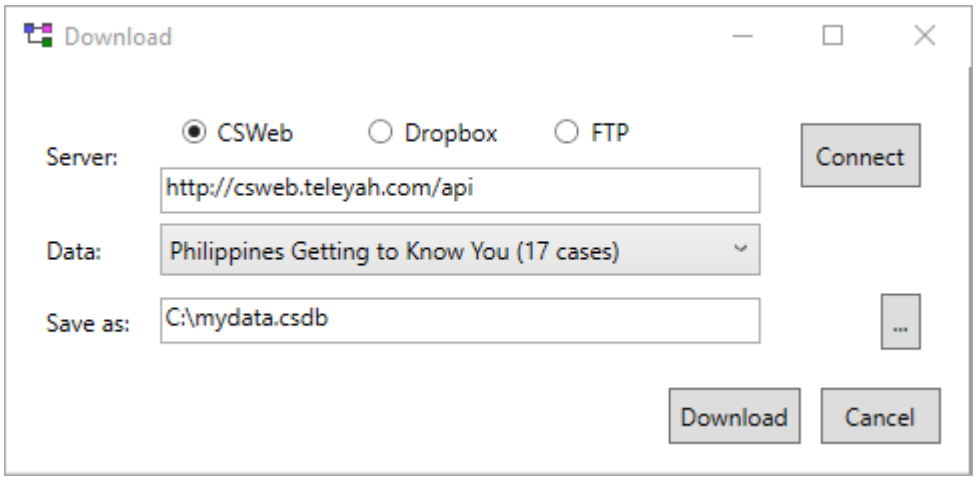
Once the sync settings are complete, regenerate the pen file. The sync settings are stored in the pen file so if it is not updated you won't be able to sync. Copy the pen and pff file into the Popstan2020Census directory under the files directory on the server so that it will be downloaded to the local device during sync.

Running Sync

To synchronize the application with the server, run the application in CSPro and from the case listing screen choose “Synchronize…” from the file menu on Windows or tap the synchronize icon ( ) on Android.

Downloading the Combined Data File

Once you have used synchronize to upload data to the server you can download the data file from the server using DataViewer. This will download a single data file that combines all the data uploaded from all the devices that have synchronized with the server. Open DataViewer, choose the server type, enter the same URL you used in the synchronization options dialog, click "Connect" to download the list of data files available on the server, choose the data file, choose a place to save the downloaded file and click "Download".



If you are using CSWeb, you can avoid filling out the download dialog by clicking on the "Download" icon in the CSWeb interface in the browser. This will download a pff file. When you open the downloaded pff file, it will automatically open DataViewer and begin downloading the combined data. Once you have downloaded the data file the first time, you can use the "synchronize" function in DataViewer to only download the newly updated data rather downloading the entire file a second time. If you are using Dropbox or FTP, CSPro will automatically create a pff file named *DownloadData-<your dictionary name>.pff* in your Dropbox or on your FTP server that will launch DataViewer to download the data.

Group exercise – Sync Race

Split into groups of 3-4 people and create a small CSPro application for the following questionnaire:

Synchronization Exercise Questionnaire

A) Team name: _____

B) Names of team members:

1	
2	
3	
4	

Upload your data dictionary to the following server:

URL: http://csweb.teleyah.com/ui
Admin username: admin
Admin password: adminadmin

Add synchronization to your application:

URL: http://csweb.teleyah.com/api
Username: test
Password: password
Choose to upload data to the server.
Do NOT sync the application files (pen and pff).

Run your application, complete a case and synchronize to send your data to the server.

The first team to get their data uploaded to the server is the winner.

Synchronization from Logic

Sometimes we need more control over the synchronization than the sync options dialog provides. For example, if there are external dictionaries in an application, if you want to sync with a universe or if there are multiple applications to sync at once. CSPro provides logic functions to run synchronizations from within an application that allow more complex synchronizations.

Let's add synchronization to our menu application that will upload the main data file and download the latest version of the pen and pff files for the menu application and for the listing and household questionnaires. We will add a new entry in the supervisor main menu called "Synchronize with Headquarters".

The first step in running sync from logic is to call the command syncconnect() and pass it the server information. If this function succeeds then we are connected to the server and we can make calls to the commands syncdata() and syncfile() to synchronize data files and non-data files respectively. Finally we call syncdisconnect() to end the session.

The commands syncdata() and syncfile() operate differently. syncdata() may only be used with data files in CSPro DB format. With syncdata(), CSPro keeps track of which cases in the file have already been uploaded to the server so that it avoids transferring cases that have already been synced. This reduces the amount of data transferred and keeps data costs and transfer times to a minimum. It also avoids one interviewer overriding changes made by another interviewer when they are working on different cases in the same file. syncfile() may be used with any type of file including text files, images and application files (pen and pff). It does not look at the file contents so it simply uploads or downloads the entire file, overwriting any existing version. When syncing data files, you should always use syncdata().

syncdata() takes two arguments: the direction and the dictionary name. The direction can be PUT (upload data to server), GET (download data from server) or BOTH (upload and download). These options correspond to the drop down in the sync options dialog. The second argument is the name of a dictionary in the application that corresponds to the data file to synchronize. Note that this dictionary must be added to the application as an external dictionary.

In our application, we want to synchronize the household questionnaire dictionary so we need to first add it to the menu program as an external dictionary. Now we can use it in the call to syncdata().

We will also add calls to syncfile() to download the latest versions of the application programs from the server. This function takes the direction (PUT or GET), the “from” directory and the “to” directory. In the case of GET the “from” directory is the local directory on the device and the “to” directory is the path on the server.

```
// Upload cases to web server at headquarters
// and download latest version of application files
function syncWithHeadquarters()

    // Connect to the webserver
    if syncconnect(CSWeb, "http://csweb.teleyah.com/api") then

        // Sync cases from household data file with server
        syncdata(BOTH, POPSTAN2020_DICT);

        // Download latest version of menu application
        syncfile(GET, "/Popstan2020/Menu/Menu.pen", "Menu.pen");
        syncfile(GET, "/Popstan2020/Menu/Menu.pff", "Menu.pen");

        // Get latest household data entry program
        syncfile(GET, "/Popstan2020/Household/Popstan2020.pen",
                      "../Household/Popstan2020.pen");

        // Disconnect from the server
        syncdisconnect();

    endif;
end;
```

Now that we have added the synchronization logic to the menu we rebuild the pen files and copy them to the files directory on the server.

Now we can run the menu program and test our synchronization.

Group exercise – Sync Race II

In your teams modify the sync exercise questionnaire to take a photo of the team and save it to a file.

Synchronization Exercise Questionnaire

A) Team name: _____

B) Names of team members:

1	
2	
3	
4	

C) Team Photo: 1 Yes 2 No

D) Synchronize with server: 1 Yes 2 No

Add two questions, C and D, to your application. For question C, take a photo if the interviewer answers "yes". For question D, add logic to the postproc of the question so that it synchronizes with the server if the user selects "yes".
Use `syncfile()` with PUT to upload the photo to the directory /photos/ on the server.
Run your application, enter a case and synchronize to send your photo to the server.
The first team to get their photo uploaded to the server is the winner.

Security Considerations

It is important to ensure the security of your server and data. Computer and network security is a challenging problem and well beyond the scope of this training. Here a few security considerations to think about. For any large survey or census operation you should consult with a security expert to ensure that your data is safe.

Use HTTPS on the web server for data transfer

This encrypts the data transmission that goes over the internet. It requires purchasing and installing an SSL certificate on the server. Without SSL, server passwords are sent unencrypted and are vulnerable to hackers.

Use a private network

For censuses and large surveys you may be able to work with the telcom provider to create a private network that only your devices can access. This avoids putting your server on the public internet where it could be vulnerable to attacks. This also makes it possible to limit your devices to only be able to access your server and not other websites that you don't want your interviewers to use.

Use device encryption on Android devices

By setting a PIN code on your Android tablet or phone you enable encryption of files on the device using strong hardware encryption. This makes it very difficult for anyone without the PIN code to retrieve data on the device.

Consider how to manage passwords

It is more convenient to have a single password that is shared by all devices and is hardcoded in the pen file however it is more secure to use a different username/password for each interviewer and have them enter it for every sync. For a large number of interviewers managing passwords and resetting passwords for those who have forgotten their passwords could be a significant management burden. You will need to find the right balance of security and convenience. If you want to hardcode the password in your pen file you can pass the username and password in the call to `syncconnect()`. If you do not provide a username and password in this call, the interviewer is prompted to enter the password the first time they sync with the device. After that the username and password are saved on the device so they do not need to be entered for synchronizations. If you don't want the password saved on the device, you can use the `prompt()` function to have the interviewer enter the password before each sync and then pass the password received from the call to `prompt()` to `syncconnect()`.

Synchronization with a Universe

The `syncdata()` command has an optional third parameter, the universe. This is a string that CSPro tries to match to the concatenated case-id items to limit the data transferred. If a universe is provided, then only cases who case-ids match the universe are synchronized. For example, to limit synchronization to only province 1, district 2 we would call:

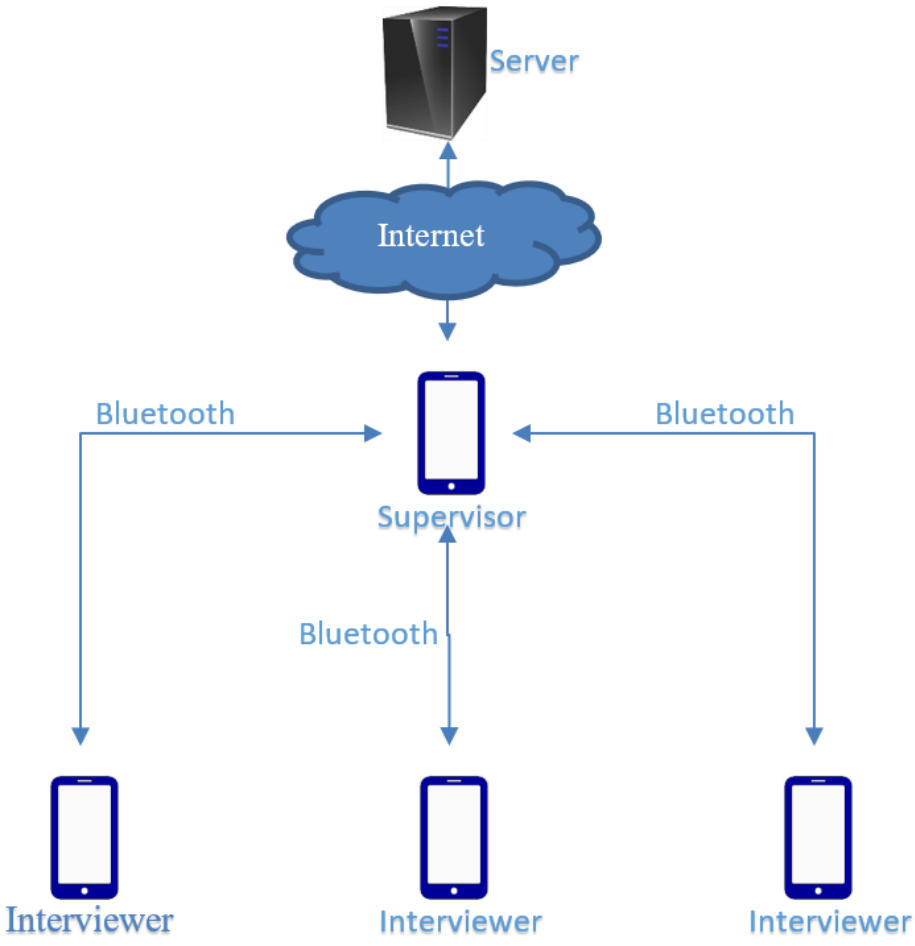
```
syncdata(BOTH, HOUSEHOLDQUESTIONNAIRE_DICT, "12");
```

This will only synchronize cases whose case-ids start with “12”. Since province and district are both single digit fields this limits the sync to cases in province 1 and district 2. Similarly, if we used “1” as the universe all cases in province 1 would be synced.

This is useful when there are a large number of interviewers and you don’t want every interviewer to download the data from every other interviewer. You can specify the geographic area assigned to the interviewer as the universe so that only cases in the interviewer’s area are synced.

Peer to Peer Synchronization Using Bluetooth

In some places, interviewers do not have reliable internet access and cannot synchronize directly with the server. In this case, you can use peer to peer to synchronization. Interviewers synchronize with a supervisor’s tablet or laptop over Bluetooth. Later, the supervisor goes to a location where they can connect to the internet and they synchronize with the central server.



Synchronization with supervisor over Bluetooth when interviewers do not have internet

In order to implement Bluetooth synchronization, we need to add logic on both the supervisor device and on the interviewer device. One device acts like the web server, processing synchronization requests from the client device, and the other device, the client, sends `syncfile` and `syncdata` commands the same way that it would to a web server. For our example let’s make the supervisor device be the server and the interviewer device be the client. This is arbitrary, we could easily reverse the roles.

To extend the menu program to support peer to peer synchronization we will add a new menu option to the supervisor main menu "sync with interviewer" and a new menu option to the interviewer main menu "sync with supervisor".

The logic on the interviewer's tablet is nearly identical to the logic used for web synchronization. The only differences are the arguments to syncconnect(). When using Bluetooth, syncconnect() does not need a URL, username or password.

```
syncconnect(bluetooth)
```

syncconnect() will scan all nearby Bluetooth devices and present a list to the interviewer who can choose which device to connect to. Alternatively, if you know ahead of time the name of the device to connect to you may specify it as a second parameter. In this case syncconnect() will try to connect directly to the named device.

```
syncconnect(Bluetooth, "Supervisor03")
```

The rest of the logic is the same as for web synchronization except that the "from" paths change to match the paths on the supervisor device.

```
function syncWithSupervisor()
    // Connect to the supervisor device
    // We do not specify the device name to connect to which allows the interviewer
    // to pick the device from a list of nearby devices.
    if syncconnect(bluetooth) then

        // Sync main data file.
        syncdata(BOTH, POPSTAN2020_DICT);

        // Download latest application files from the supervisor.
        // The root file on the supervisor tablet is in the "Popstan2020" folder
        // so the from folder starts from there and we need to add "Menu" to get to
        // the Menu programs.
        syncfile(GET, "Menu/Menu.pen", "Menu.pen");
        syncfile(GET, "Menu/Menu.pff", "Menu.pff");

        // Since the current application is in the Menu folder we need to use
        // "../Household" to go up one level and back down into the Household
        // folder for the household application files.
        syncfile(GET, "Household/Popstan2020.pen",
                 "../Household/Popstan2020.pen");

        syncdisconnect();
    endif;
end;
```

The logic for the supervisor is even simpler. We simply call the command syncserver() which runs the Bluetooth server and waits for connections.

```
function syncWithInterviewer()

    // Run the Bluetooth server to receive data from interviewer.
    // Specify ".." as the root directory to make the the "Popstan2020" folder be
    // the starting point for all file get/put commands instead of the menu folder.
    syncserver(Bluetooth, "..");
end;
```

syncserver() takes an optional second parameter which is the root directory for all syncfile() commands. The root directory is prepended to the file path on the server. So if the root directory is "C:/myroot" and the client calls syncfile(PUT, "myfile", "files/myfile") then the destination path will be "C:/myroot/files/myfile". By default, the root directory is the application directory of the server so adding ".." to it makes the root folder the parent directory of the application i.e. the Popstan2020 directory.

Group exercise – Sync Race III

In your teams modify the sync exercise application to support Bluetooth sync.

Synchronization Exercise Questionnaire

A) Team name: _____

B) Names of team members:

1	
2	
3	
4	

C) Team Photo: 1 Yes 2 No

D) Synchronize: 1 Web server 2 Supervisor 3 Interviewer 4 None

Modify question D so that you have the option of synchronizing via Bluetooth.
Run your application on two devices, complete a case on one device and synchronize to send your photo to the other device via Bluetooth. Use the second device to synchronize to send your photo the web server.
The first team to get their photo uploaded to the server is the winner.

Exercises

1. Modify the syncWithHeadquarters() and syncWithSupervisor() functions in the menu program to also sync the listing data file and the staff lookup file. The listing file should be synced using BOTH and the staff file should be downloaded from the server using syncdata with GET.
2. Modify the syncWithHeadquarters() and syncWithSupervisor() functions so that only data in the geographic area assigned to the interviewer/supervisor are synced. For the interviewer this means that only households in the assigned EA are synced and for the supervisor only households in the assigned district are synced.

Session 10: Batch Edit and Export

At the end of this session participants will be able to:

- Implement consistency checks in a batch edit application
- Use batch edit to add calculated variables and recoded variables
- Use batch edit to convert checkbox values to yes/no variables for analysis
- Use the export data tool to take data from CSPro to other software packages

Creating a Batch Edit Application

A batch edit application is like a data entry application but without the forms. It is meant to be run after data entry to detect and fix problems in the data file. A batch edit application takes an input data file and runs logic on it. It generates both a report, called a listing file, and optionally an output data file which is a modified version of the input file. A batch edit application never changes the input file.

To create a batch edit application you choose File->New from CSPro and choose Batch Edit Application. You then choose a dictionary. This is usually the same dictionary that you used for data entry.

The user interface for working with batch applications is similar to the one for working with data entry applications except that there are no forms. Instead there is a tab in the tree one left side of the window for edits. Just like in data entry, you add logic to PROCS. Instead of running interactively, all the error messages are written out to a log file for review after the whole program has run.

Checking for Errors

To add consistency checks we proceed just as we did in our data entry application by adding logic to the appropriate PROC. Let's start with a simple check that the age of first marriage is not greater than the age.

```
PROC AGE_AT_FIRST_MARRIAGE

// Check for age at first marriage less than current age
if AGE_AT_FIRST_MARRIAGE > AGE then
    errmsg("Age at first marriage greater than age");
endif;
```

Next we run the application but first we need a test data file. You can use the file Popstan2020Raw.csdb. Run the application against this test data. After the application runs, we see the log file. The log file reports that we have a case where this error exists.

```
Process Messages

*** Case [3691141112] has 1 messages (0 E / 0 W / 1U)
    U   -20 Age at first marriage greater than age

User unnumbered messages:

   Line    Freq  Pct.  Message text                                        Denom
   ----    ----  ----  ------------                                       -----
     20       1     -  Age at first marriage greater than age                 -

CSPRO Executor Normal End
```

To figure out what the problem is we can open up the problem case in data entry. The printout in the listing file contains the case identifiers which we can use to find the case. You can copy the case id from the listing file and use it with Find Case on the Edit menu in CSEntry.

Getting Summary Information

If you add `summary` after the `errmsg` statement, the individual error cases will not be written to the listing file. With `summary`, only the total number of times each error is encountered is written to the file. This can be helpful when working with large data files that have a lot of errors making the listing file very large.

```
PROC AGE_AT_FIRST_MARRIAGE

// Check for age at first marriage less than current age
if AGE_AT_FIRST_MARRIAGE > AGE then
    errmsg("Age at first marriage greater than age") summary;
endif;
```

Now when we run the application, the listing file no longer contains the individual cases where the errors occurred.

```
User unnumbered messages:

   Line    Freq  Pct.  Message text                                        Denom
   ----    ----  ----  ------------                                       -----
     20       1     -  Age at first marriage greater than age                 -

CSPRO Executor Normal End
```

If you add a denominator using the `denom` keyword, CSPro will calculate the percentage of cases where the error occurred. The denominator to use depends on what is being counted. In our case it is the total number of household members with age 10 or above and marital status married, divorced or widowed. We can calculate that total as part of our batch edit program.

```
PROC GLOBAL
numeric numHHMembers10AndOverEverMarried = 0;

PROC AGE_AT_FIRST_MARRIAGE

if AGE >= 10 and MARITAL_STATUS in 2:4 then
    numHHMembers10AndOverEverMarried = numHHMembers10AndOverEverMarried + 1;

    // Check for age at first marriage less than current age
    if AGE_AT_FIRST_MARRIAGE > AGE then
        errmsg("Age at first marriage greater than age") summary
                denom = numHHMembers10AndOverEverMarried;
    endif;
endif;
```

Now when we run the program we see that 0.2% of the individuals 10 and above who where ever married have an age at first marriage greater than their age.

```
User unnumbered messages:

   Line    Freq  Pct.  Message text                                        Denom
   ----    ----  ----  ------------                                       -----
     20       1   0.2  Age at first marriage greater than age               463

CSPRO Executor Normal End
```

Correcting Errors

In addition to using batch edit to find errors you can also use it to correct problems by modifying variables in your logic. Let's simply cap the age at first marriage to never be greater than the age.

```
// Check for age at first marriage less than current age
if AGE_AT_FIRST_MARRIAGE > AGE then
    errmsg("Age at first marriage greater than age. Capping age at first marriage at
age.");
    AGE_AT_FIRST_MARRIAGE = AGE;
endif;
```

When we run this time we will specify an output file: Popstan2020Edited.csdat. The changes we make will only be made to the output file. We can then rerun the batch application on the output file and make sure that we don't have any error messages.

Instead of just assigning the value of AGE_FIRST_MARRIAGE we can use the impute command which does the assignment just like "=" but also generates a nice report showing the values that were imputed.

```
// Check for age at first marriage less than current age
if AGE_AT_FIRST_MARRIAGE > AGE then
    errmsg("Age at first marriage greater than age. Capping age at first marriage at
age.");
    impute(AGE_AT_FIRST_MARRIAGE, AGE);
endif;
```

The imputation report will be opened in TextViewer after you run the batch application but to see it you will need to go to the Window menu and choose the file that ends in ".frq.lst".

```
                          IMPUTE FREQUENCIES                          Page    1


_____________________________________________________________________________
       Imputed Item AGE_AT_FISRT_MARRIAGE: Age at first marriage - all occurrences

                                  _____________________________ _____________
 Categories                             Frequency       CumFreq       %  Cum %
_____________________________ _____________________________ _____________
 40                                             1             1   100.0  100.0
_____________________________ _____________________________ _____________
 TOTAL                                          1             1   100.0  100.0
```

Adding Calculated Variables

It is often useful to add additional variables to your data file after data collection that are computed from the collected variables. For example, let's add a yes/no/don't know variable to the household record that determines if the household is headed by a child. First we add the new variable **CHILD_HEADED_HOUSEHOLD** to the dictionary (at the end of the housing record so that we do not mess up our existing data). Then we add logic to the PROC of our new variable to impute the value. A household is headed by a child if the age of the head is less than 18.

```
PROC CHILD_HEADED_HOUSEHOLD

// Set calculated variable child headed based on age of head
if AGE(1) < 18 then
    // Head under 18, child headed
    impute(CHILD_HEADED_HOUSEHOLD, 1);
elseif AGE(1)  = 999 then
    // Head age unknown
    impute(CHILD_HEADED_HOUSEHOLD, 9);
else
    // Head over 18, not child headed
    impute(CHILD_HEADED_HOUSEHOLD, 2);
endif;
```

Run the program and look at the imputation report to see how many child headed households are in our data set.

Converting Checkboxes to Yes/No

The checkbox makes a nice interface but the resulting data is a string that is hard to interpret and deal with in other software. To make it easier to use for export we can convert from the checkbox to a repeating item with yes/no options. Let's convert the **LANGUAGES_SPOKEN** field to a series of yes/no variables. Create new yes/no items for each language at the end of the individual record. Add value sets with Yes – 1 and No – 2.

For each of these new variables we want to set the value to yes if the corresponding language is checked in **LANGUAGES_SPOKEN**. What function do we use to determine if an item in a checkbox field is checked? As before we use **pos()**.

```
PROC LANGUAGE_ENGLISH_SPOKEN
if pos("A", LANGUAGES_SPOKEN) > 0 then
    LANGUAGE_ENGLISH_SPOKEN = 1;
else
    LANGUAGE_ENGLISH_SPOKEN = 2;
endif;

PROC LANGUAGE_FRENCH_SPOKEN
if pos("B", LANGUAGES_SPOKEN) > 0 then
    LANGUAGE_FRENCH_SPOKEN = 1;
else
    LANGUAGE_FRENCH_SPOKEN = 2;
endif;

PROC LANGUAGE_SPANISH_SPOKEN
if pos("C", LANGUAGES_SPOKEN) > 0 then
    LANGUAGE_SPANISH_SPOKEN = 1;
else
    LANGUAGE_SPANISH_SPOKEN = 2;
endif;
```

The remaining languages follow the same pattern.

The Export Data Tool

The CSPro export data tool is available from the Tools menu. When you first start Export Data you are prompted to provide a data dictionary. Choose the Popstan2020 dictionary. From the main export screen, you can choose which records/variables to export using the checkboxes next to each one. To start with let's just pick the id items and the first few fields from section F. At the bottom of the export window you can choose the file format to export to. For this exercise we will choose CSV which can be easily opened in Excel. To run the export, click on the traffic light. You are prompted to choose the data file. We will use the Popstan2020Edited.csdb data file. Finally, you are prompted for the name(s) of the exported files. Once the export completes the exported files are displayed in Text Viewer. Let's open them in Excel and see what we have. Note that each household is saved in a separate line in the Excel file.

PROVINCE	DISTRICT	ENUMERATION_AREA	AREA TYPE	HOUSEHOLD_NUMBER	F01	F02	F03
2	14	214	1	1	3	1	3
3	27	301	1	1	9	9	1
1	1	345	1	5	1	1	1
3	26	222	2	2	1	4	1
4	37	422	2	1	1	5	1
2	2	214	3	1	1	2	4
1	1	101	1	1	2	5	1

Exporting Repeating Records

Let's try exporting a few fields from section B: line number, name and sex. Notice for each variable we exported we get 50 columns. Why is that? We are getting one column for each record occurrence. CSPro is still exporting each household on a single row and since there are up to 50 people in the household it generates 50 columns for each variable. Even empty occurrences are still generating columns in the spreadsheet.

B1_01	B1_02	B1_03	B1_04	B1_05	B1_06	B1_07	B1_08	B1_09	B1_10	B1_11	B1_12	B1_13	B1_14	B1_15	
1	2														
1	2	3													
2															
1	2	3	4												
1	2	3	4												
1	2	3													
1	2	3	4	5	6	7									

1 row = 1 household

Having a column for each occurrence can complicate working with the data. For example, in this file it is rather tough to do something as simple as count the total number of people. If instead of choosing the default setting of putting multiple record occurrences in a single row (the *All in One Record setting*) try selecting *As Separate Records*. Now we get each person in the household in a separate row. It is important to include the household id items when doing this so that it is clear which people are in which household.

1 row = 1 person

PROVINCE	DISTRICT	ENUMERATION_AREA	AREA_TYPE	HOUSEHOLD_NUMBER	B1	B2	B3	B5
2	14	101	1	1	1	John	1	27
2	14	101	1	1	2	Mary	2	26
2	27	200	1	1	1	Jane	2	40
2	27	200	1	1	2	Billy	1	13
2	27	200	1	1	3	Tina	2	13

Exporting Multiple Record Types

Now let's try exporting the first few items from both the person record (B) and the deaths record (E). With our current settings, Export Data warns us that only items from the first record will be exported. Why? The problem is that if you put each record in its own row then some rows would have household members on them and others would have deaths on them but then the columns would not match.

The solution is to export each record in a separate file by selecting *Multiple Files (one for each record type)* under Number of Files Created. Doing this generates two files: PERSON.csv which contains the household members and DEATHS.csv which contains the deaths.

PERSON.csv

PROVINCE	DISTRICT	ENUMERATION_AREA	AREA_TYPE	HOUSEHOLD_NUMBER	B1	B2	B3	B5
2	14	101	1	1	1	John	1	27
2	14	101	1	1	2	Mary	2	26
2	27	200	1	1	1	Jane	2	40
2	27	200	1	1	2	Billy	1	13
2	27	200	1	1	3	Tina	2	13

DEATHS.csv

PROVINCE	DISTRICT	ENUMERATION_AREA	AREA_TYPE	HOUSEHOLD_NUMBER	E3	E4	E5
2	14	101	1	1	1	Erwin	20140211
2	14	101	1	1	2	Carmine	20161201
2	27	200	1	1	1	Arnold	20150113

The households in these two files can be linked together based on the id items.

It also possible to have Export Data join together single and multiple records. If you select this option, then one file will be generated for each record with multiple occurrences and the selected columns for all the selected singly occurring records will be added to each row in each of the exported files. For example, if we choose the tenure status (**F06**) and type of main dwelling (**F03**) from the single record housing and also choose the first few items from the person record then **F06** and **F03** will be added to each of the two exported files: PERSON.csv and DEATHS.csv.

PERSON.CSV

PROVINCE	DISTRICT	ENUMERATION_AREA	AREA_TYPE	HOUSEHOLD_NUMBER	F03	F06	B1	B2
2	14	101	1	1	1	1	1	John
2	14	101	1	1	1	1	2	Mary
2	27	200	1	1	2	3	1	Jane
2	27	200	1	1	2	3	2	Billy
2	27	200	1	1	2	3	3	Tina

Id items

Single record

Multiple record

DEATHS.CSV

PROVINCE	DISTRICT	ENUMERATION_AREA	AREA_TYPE	HOUSEHOLD_NUMBER	F03	F06	E3	E4
2	14	101	1	1	1	1	1	Erwin
2	14	101	1	1	2	1	2	Carmine
2	27	200	1	1	1	3	1	Arnold

It is not possible to join multiple records to other multiple records from export.

Note that in some cases it may be easier to do the export of the single and multiple records without joining and then do the join in the software that you have imported the data into.

Importing Data into SAS, SPSS, Stata and R

When exporting data to statistical packages, CSPro generates both a data file and script to run inside the statistical software itself to run the import. For details in how to run this script for each package see the online help for Export Data and look under “How to…”.

Saving your Export Specification

You can save your export settings as CSPro export specification file (.exf). You can later double click on this file or open it from the Export Data Tool to retrieve all the selections that were made.

Exercises

1. Modify the batch edit application to add a check for someone with relationship of spouse but marital status that is not married. Print a message for each case found. This should be done in the batch edit application NOT in the data entry application.
2. Modify the batch edit application to add a check that the total number of rooms (**F01**) is greater than the number of bedrooms (**F02**)
3. Modify the batch edit application to add a check that the type of main dwelling (**F03**) is consistent with the total numbers of each dwelling type in **F05**. For example, if the main dwelling in **F03** is detached house then the total number of detached houses in **F05** must not be zero.
4. Modify the batch edit application to convert the disabilities (**B11**) from alpha (used for checkboxes) to a set of numeric yes/no variables. Create the new variables in the dictionary and write logic to set the value of each of the new variables based on the selections in **B11**.
5. Using the test data, export the household members along with the ID items to the package of your choice (Excel, SPSS, Stata, SAS or R). Use that package to determine the total number of people by sex (total, males, females) and also the number of people by sex for district 01.
6. Export the deaths record and the person record along with the id items into the package of your choice. You should export all the records at once, not one by one. How many households have more than one death? How many households have children under 5 years of age?