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
- Use the locate/loadcase pattern to loop through cases in an external dictionary

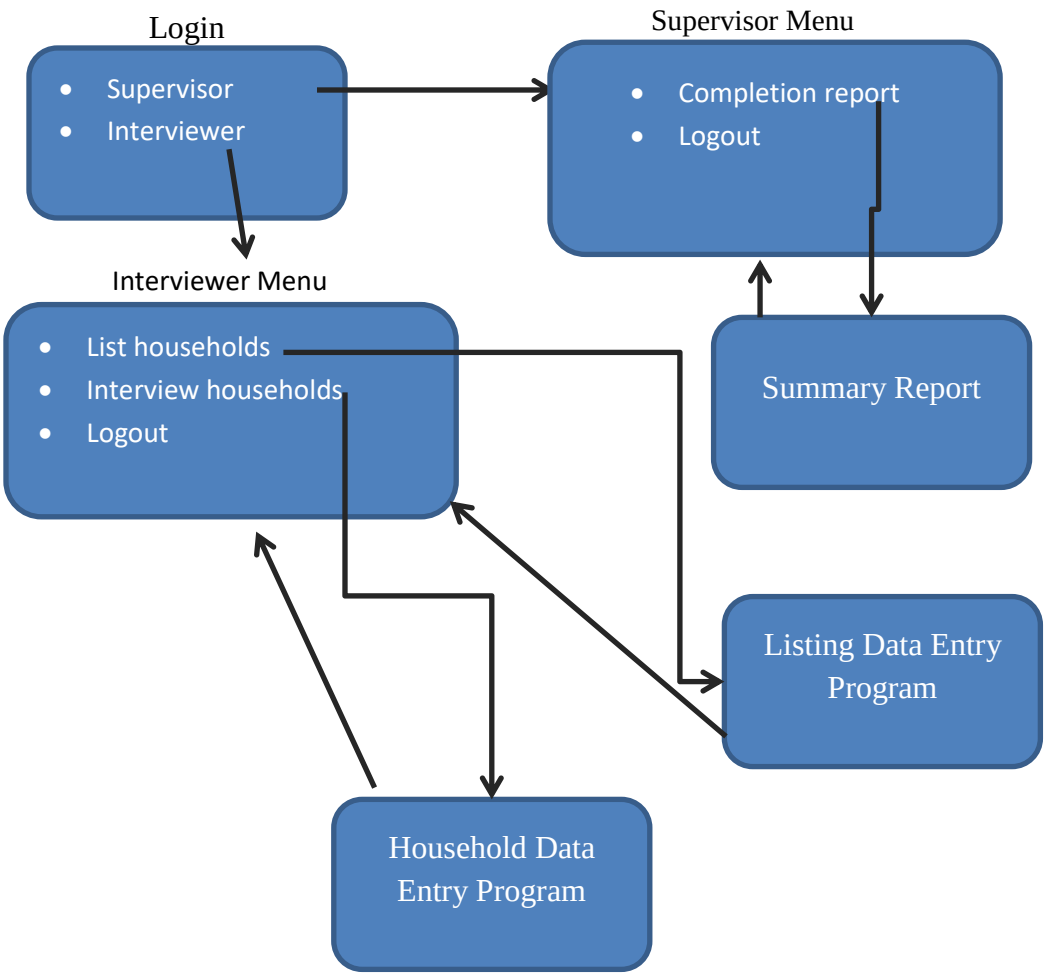
What is a Menu Program?

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 different menu screen will be a 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: Summary 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. 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 data entry 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 Android, we will always use the stop parameter since it is not possible to run two CSEntry applications at the same time on Android.
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

Normally if we start a CSEntry application in add mode and complete a questionnaire, CSEntry will automatically begin entry of the next questionnaire. When starting from the menu program we do not want this behavior. Instead, we want to return to the menu program as soon as we finish the questionnaire. We can do this by adding a call to *execpff()* to relaunch the menu program. Which proc can we put this in so that it will be called when we are done with a questionnaire? The postproc of the level is the last proc to be called for each questionnaire so we can put it there.

```
PROC POPSTAN2020_QUEST

// If launched by menu we should stop and return to menu
// here after questionnaire is completed. If not, in add mode
// we will start to add another case instead of going
// back to the menu.
execpff("../Menu/Menu.pff", stop);
```

We also need to handle the case where the interviewer uses the stop button on Windows or the back button on Android to exit the data entry application. To do this we provide an *OnStop* function in the entry application logic that relaunches the menu program. *OnStop* is a special function. If CSPro finds a function named *OnStop* in your logic it will call it when the interviewer pushes the stop button (or the back button on Android).

```
function OnStop()
    execpff("../Menu/Menu.pff", stop);
end;
```

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 7.0\CSEntry.exe"
if exist %CSEntry% goto :gotcspro
SET CSEntry="%ProgramFiles%\CSPro 7.0\CSEntry.exe"
if exist %CSEntry% goto :gotcspro
echo "Can't find CSEntry version 7.0. 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. Finally, 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.

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 and will be able to support longer id-items if we add new ids 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 a different form of the *loadcase* statement to do this. If you call *loadcase* with no id-items, it will load the *next* case in the file. If there are no more cases in the file, it returns 0. Before we do this we need to move to the start of the file which can be done using the *locate* statement. Locate takes an operator (=, <, >, <=, >=) and a case-id string and moves to the

first position in the file for which the operator and case string are satisfied. We will use >= as the operator and the empty string which will always move to the start of file.
Using locate followed by loadcase is a common pattern in menu programs for looping through all cases in an external file:

```
locate(MY_DICT,>=,"");
while loadcase(MY_DICT) do
      // Do something with case
enddo;
```

We will use this pattern in the onfocus of CHOOSE_HOUSEHOLD to build the value set from the listing file:

```
PROC CHOOSE_HOUSEHOLD
onfocus

numeric nextEntry = 1;

// Loop through all cases in listing file
// to build dynamic value set.
open(LISTING_DICT);
locate(LISTING_DICT,>=,"");
while loadcase(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;
enddo;
close(LISTING_DICT);

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

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

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 7.0");
    filewrite(pffFile,"AppType=Entry");

    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,"DISTRICTSPERPROVINCE_DICT=" + pathname(Application) +
                      "../Household/Resources/DistrictsPerProvince.dat");
    filewrite(pffFile,"DISTRICTS_DICT=" + pathname(Application) +
                      "../Household/Resources/Districts.dat");
    filewrite(pffFile,"ENUMERATIONAREAS_DICT=" + pathname(Application) +
                      "../Household/Resources/EnumerationAreas.dat");
    filewrite(pffFile,"HOUSEHOLDPOSSESSIONVALUE_DICT=" +
                     pathname(Application) +
                    "../Household/Resources/HouseholdPossessionValues.dat");

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

    filewrite(pffFile,"[Parameters]");
    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 the *set attributes protect* command.

```
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
    set attributes (PROVINCE) protect;

endif;
```

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

Handling Add and Modify Mode

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 data entry the way we did for the menu program. However, when we do that it stops us from being able to modify an existing household from the menu. We can fix this by setting the start mode to modify if the household already exists in the data file. In order to do this, we need to add the household questionnaire dictionary (POPSTAN2020_DICT) as an external dictionary to the menu program and use loadcase to see if the household is already in the data file.

When specifying the StartMode, we can also add the case-ids so that if the case exists in the data file already, it will be opened.

```
// 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 7.0");
    filewrite(pffFile,"AppType=Entry");

    filewrite(pffFile,"[DataEntryInit]");
    // Use modify mode if the case already exists in the household
    // data file, otherwise use add mode.
    string mode;
    open(POPSTAN2020_DICT);
    if loadcase(POPSTAN2020_DICT, CHOOSE_HOUSEHOLD) = 1 then
        mode = "modify";
    else
        mode = "add";
    endif;
    close(POPSTAN2020_DICT);
    filewrite(pffFile,"StartMode=%s;%s",mode, CHOOSE_HOUSEHOLD);

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

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

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

    filewrite(pffFile,"[Parameters]");
    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;
```

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	
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. Create a resource directory for the menu program and put staff.dat into the resource directory.
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 user code so that we 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 the locate/loadcase pattern 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;

    locate(POPSTAN2020_DICT, >=, "");
    while loadcase(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;
    enddo;

    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() in 20:29 then
        // Android - use "view:"
        execsystem(maketext("view:%s", reportFilename));
    else
        // 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.
3. Modify the listing program to return to the menu program when the user exits by implementing the OnStop function.
4. 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.
5. 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.
6. 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
```