Session 9: Advanced Menu Programs

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

- Create dynamic menus with dynamic value sets
- Pass parameters from the menu application to a data entry program
- Launch external applications from a data entry application
- Generate and view reports from a data entry application

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 from the sample. We can do this by displaying the households from the sample file in a dynamic value set and having the interviewer choose which one to interview.

First create the sample file in Excel:

SAM PROV	SAM DIST	SAMD WNO	SAM HHN O	SAM AREA	SAM RPL	SAMG PSLAT	SAMG PSLON	SAMHA	SAMHN A	SAM HCT
1	1	101	101	1	2	40.051 12	-94.877 8	742 Evergreen Terrace	Homer Simpson	55573 34
1	1	102	102	1	2	39.857 65	-94.833 8	741 Evergreen Terrace	Ned Flanders	92342 3423
1	10	1	1	1	2	38.897 96	-77.036 6	1600 Pennsylvani a Ave	Donald Trump	98765 4321
1	10	2	2	1	2	38.848 39	-76.931 1	4600 Silver Hill Rd	Enrique Lamas	73242 2383

Use Excel to CSPro to convert this to a CSPro dictionary and data file and add the dictionary to the menu program.

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 9 digits long to hold the full set of id-items (province, district, dwelling, 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 immediately calling execpff.

```
PROC INTERVIEWER_MENU

if INTERVIEWER_MENU = 1 then
    // Launch questionnaire
    skip to CHOOSE_HOUSEHOLD;
elseif INTERVIEWER_MENU = 2 then
```

In the onfocus proc for CHOOSE_HOUSEHOLD we need to loop through every case in the sample file to build the value set. We can use *forcase* to do this with a where clause to only use the households that are assigned to the interviewer.

```
PROC CHOOSE_HOUSEHOLD
onfocus

// Make value set from households in sample assigned to the
// interviewer who is logged in.
numeric nextEntry = 1;
forcase SAMPLE_DICT where SAMINT = LOGIN do
    codesString(nextEntry) = key(SAMPLE_DICT);
    labels(nextEntry) = maketext("P:%d-DI:%d-DW:%d-HH:%d %s",
                        SAMPROV, SAMDIST, SAMDWNO, SAMHHNO,
                        strip(SAMHNA));
    nextEntry = nextEntry + 1;
enddo;
codesString(nextEntry) = "";
setvalueset(CHOOSE_HOUSEHOLD, codesString, labels);
```

When we run this, we should see a list of the assigned households in the sample 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 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 logic that launches the questionnaire to first write out the pff file before launching it so that we can customize it. To write out files from a CSEntry application we have to first declare a file variable in the PROC GLOBAL:

```
PROC GLOBAL
file pffFile;
```

Once we have a file variable we can use the commands ***setfile(), filewrite()*** and ***close()*** to open, write to and close the file. For example, the following code will write “Hello!” to a file named “MyFile.txt”:

```
setfile(pffFile, "MyFile.txt");
filewrite(pffFile, "Hello!");
close(pffFile);
```

There is a simple to trick to get the logic to write out a pff file. Open the pff file in the pff editor (right-click on the pff file and choose "Edit"). In the pff editor enable "Expert Mode" from the options menu. You will then see two additional tabs that show the text contents of the file and the CSPro logic to write it out using *filewrite*. We can cut and paste from here to get the basic code to write out the pff and then 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 PopstanLFS 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) +
                         "../Questionnaire/PopstanLFS.pff";
    setfile(pffFile,pffFilename,create);

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

    filewrite(pffFile,"[DataEntryInit]");
    filewrite(pffFile,"ShowInApplicationListing=Hidden");

    filewrite(pffFile,"[Files]");
    filewrite(pffFile,"Application=%s",pathname(Application) +
                         "../Questionnaire/PopstanLFS.ent");
    filewrite(pffFile,"InputData=%s",pathname(Application) +
                         "../Questionnaire/PopstanLFS.csdb|CSPRODB");

    filewrite(pffFile,"[ExternalFiles]");
    filewrite(pffFile,"DISTRICT_DICT=%s",pathname(Application) +
                         "../Questionnaire/resources/Districts.csdb|CSPRODB")
;
    filewrite(pffFile,"ISCO_INDEX_DICT=%s",pathname(Application) +
                         "../Questionnaire/resources/ISCO-
Index.csdb|CSPRODB");
    filewrite(pffFile,"OnExit=%s",pathname(Application) +
                         "Menu/Menu.pff");

    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,"DWELLING=%s", CHOOSE_HOUSEHOLD[4:3]);
    filewrite(pffFile,"HOUSEHOLD_NUMBER=%s", CHOOSE_HOUSEHOLD[8:3]);

    close(pffFile);

    execpff(filename(pffFile), 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 questionnaire data entry application to prefill the id-items using the parameters in the pff file. This is done 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 *setproperty* command. Set property allows you to set most of the field properties and data entry options available in CSPro designer from logic.

```
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 the other id-items is similar.

Group Exercise
Modify the pff generation logic and the questionnaire application to prefill the variables GHHRPL and GHAREA from the sample file so that they interviewer does not have to enter them.

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” and Lock to “CaseListing” in the pff for data entry the way we did for the menu program. 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) +
                         "../Questionnaire/PopstanLFS.pff";
    setfile(pffFile,pffFilename,create);

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

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

    filewrite(pffFile,"[Files]");
    filewrite(pffFile,"Application=%s",pathname(Application) +
                         "../Questionnaire/PopstanLFS.ent");
    filewrite(pffFile,"InputData=%s",pathname(Application) +
                         "../Questionnaire/PopstanLFS.csdb|CSPRODB");

    filewrite(pffFile,"[ExternalFiles]");
    filewrite(pffFile,"DISTRICT_DICT=%s",pathname(Application) +
                         "../Questionnaire/resources/Districts.csdb|CSPRODB")
;
    filewrite(pffFile,"ISCO_INDEX_DICT=%s",pathname(Application) +
                         "../Questionnaire/resources/ISCO-
Index.csdb|CSPRODB");
    filewrite(pffFile,"OnExit=%s",pathname(Application) +
                         "Menu/Menu.pff");

    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,"DWELLING=%s", CHOOSE_HOUSEHOLD[4:3]);
    filewrite(pffFile,"HOUSEHOLD_NUMBER=%s", CHOOSE_HOUSEHOLD[8:3]);

    close(pffFile);

    execpff(filename(pffFile), stop);

end;
```

Preserving Login when Returning to Menu

Currently when you launch the questionnaire 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;
```

Interview Result Code

Let's add the final result code to the sample file[1] and allow the interviewer to set it. Add a new variable SAMFIR to the sample dictionary with the following value set:

0. NOT STARTED
1. COMPLETED (FULLY RESPONDING HOUSEHOLD)
2. PARTLY COMPLETED
3. REFUSED
4. NON-CONTACT
5. TEMPORARILY ABSENT
6. INADEQUATE INFORMANT
7. SEASONAL, TEMPORARY RESIDENCE
8. UNDER CONSTRUCTION, VACANT, DEMOLISHED
9. NOT A HOUSING UNIT
10. OTHER (specify):_____

In the sample spreadsheet add a column for this variable, set all the values to 0 (not started) and regenerate the sample data file using CSPro to Excel.

When the interviewer chooses a household, we will now give them a choice between updating the result and opening the questionnaire. Add two new variables to the menu dictionary and form:

1. **HOUSEHOLD_ACTION** with the value set: 1. Open questionnaire 2. Set final result
2. **HOUSEHOLD_FINAL_RESULT** with the same value set as **SAMFIR**

Modify the postproc of **CHOOSE_HOUSEHOLD** to skip to this field instead of launching the questionnaire. Have the postproc of **HOUSEHOLD_ACTION** launch the questionnaire if the interviewer chooses "Open questionnaire" and skip to **HOUSEHOLD_FINAL_RESULT** if they pick "Set final result".

```
PROC HOUSEHOLD_ACTIONS

if HOUSEHOLD_ACTIONS = 1 then
    launchHouseholdDataEntry();
    reenter;
else
    skip to HOUSEHOLD_FINAL_RESULT;
endif;
```

The postproc of **HOUSEHOLD_FINAL_RESULT** should set the final result code in the sample file (**SAMFIR**) to the value chosen. To make sure that we set **SAMFIR** for the correct case, we use *loadcase* to load the

[1] We should consider separating the status and assignment information into a separate data file/files from the sample itself. How we structure these files would depend on who manages this information. We should try to separate the data that will be managed by headquarters from that managed by the supervisor and from that managed by the interviewer. This will make it easier to synchronize the data between the different parties and avoid conflicting changes to the same case.

case chosen earlier into the sample dictionary before modifying the variable. Once we have modified it, we use the *writecase* command to update the sample file on disk with our change.

```
PROC HOUSEHOLD_FINAL_RESULT

loadcase(SAMPLE_DICT, CHOOSE_HOUSEHOLD);
SAMFIR = HOUSEHOLD_FINAL_RESULT;
writecase(SAMPLE_DICT);
reenter HOUSEHOLD_ACTIONS;
```

We should prevent the interviewer from setting the result code to complete if they haven't actually finished the interview. To see if the interview is complete, we can use loadcase on the questionnaire dictionary to determine if the questionnaire exists. If it exists then we can use ispartial to determine if it is incomplete.

```
PROC HOUSEHOLD_FINAL_RESULT

if HOUSEHOLD_FINAL_RESULT = 1 and
    (loadcase(POPSTANLFS_DICT, CHOOSE_HOUSEHOLD) = 0 or
     ispartial(POPSTANLFS_DICT)) then

    errmsg("Questionnaire is missing or partially complete. Please finish
questionnaire before marking as complete.");
    reenter;
endif;

loadcase(SAMPLE_DICT, CHOOSE_HOUSEHOLD);
SAMFIR = HOUSEHOLD_FINAL_RESULT;
writecase(SAMPLE_DICT);
reenter HOUSEHOLD_ACTIONS;
```

In HOUSEHOLD_ACTIONS, we should show some information about the household to the interviewer such as the address, contact info, and name of head of household. We can use variables from the sample dictionary in the question text. For example:

> Household: %SAMPROV%-%SAMDIST%-%SAMDWNO%-%SAMHHNO%
> Head: %SAMHNA%
> Address: %SAMHA%
> Contact Phone: %SAMHCT%
>
> Chose an option:

However, before displaying the question text we need to first load the case in the sample file that was chosen in the previous field. We do this in the *onfocus* so that it is called every time we enter the field and display the question text.

```
PROC HOUSEHOLD_ACTIONS

onfocus
loadcase(SAMPLE_DICT, CHOOSE_HOUSEHOLD);
```

Group Exercise

Add an option to the HOUSEHOLD_ACTIONS menu named “Detailed Status”. It should only be in the menu for partially completed cases. When selected it should go to a new field that shows the following information in the question text:

-Total household members
-Total eligible household members
-Completed labor section interviews

The new field should have only one option in the value set, “Back”, which returns to the previous menu.

You can use global logic variables as fills in the question text to display this information.
Since this information comes from the main questionnaire you will need to make sure to load the appropriate case from that data file using loadcase.

If you use count or totocc on an external dictionary, like POPSTANLFS_DICT, you will need qualify the name of the record with the name of dictionary. For example:

```
count(POPSTANLFS_DICT.DEM_REC)
```

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 launc the calculator from Android we would use[2]:

```
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 10 and 19 for Windows and between 20 and 29 for Android.

```
// Launch the calculator
function showCalculator()
    if getos() in 10:19 then
        // Windows
        execsystem("calc.exe");
    else
        execsystem("app:com.google.android.calculator");
    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.

[2] 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 the interviewer manual:

```
// Display interviewer manual
function showManual()
    execsystem("view:/mnt/sdcard/csentry/PopstanLFS/Questionnaire/resources/I
nterviewerManual.pdf");
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 interviewer manual
function showManual()
    execsystem(maketext("view:%s/resources/InterviewerManual.pdf",
                        pathname(Application)));
end;
```

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 an option in the HOUSEHOLD_ACTIONS menu to display the household on a map using the GPS coordinates from the sample file:

```
// Display household GPS point on map
execsystem(maketext("gps:%f,%f", SAMGPSLAT, SAMGPSLON));
```

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 final result. Here is an example:

```
Completion Status
----------------
Total Assigned Households: 50
Completed Households: 20
Partly completed: 5
Not interviewed: 4
Not yet visited: 21
```

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

```
// Display the completion report that shows number of households
// by status.
function showCompletionReport()

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

```
    filewrite(tempFile, "");
    filewrite(tempFile, "Completion Status");
    filewrite(tempFile, "-----------------");

    numeric complete = 0;
    numeric partial = 0;
    numeric notInterviewed = 0;
    numeric notVisited = 0;
    numeric total = 0;

    forcase SAMPLE_DICT where SAMSUP = LOGIN do
        inc(total);

        if SAMFIR = 0 then
            inc(notVisited);
        elseif SAMFIR = 1 then
            inc(complete);
        elseif SAMFIR = 2 then
            inc(partial);
        else
            inc(notVisited);
        endif;
    endfor;

    filewrite(tempFile, "Total Assigned Households: %d", total);
    filewrite(tempFile, "Completed Households: %d", complete);
    filewrite(tempFile, "Partly completed: %d", partial);
    filewrite(tempFile, "Not interviewed: %d", notInterviewed);
    filewrite(tempFile, "Not yet visited: %d", notVisited);

    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;
```

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 for a text file and in the web browser for an HTML file.

Exercises

1. Modify the dynamic value set we create in the onfocus proc of CHOOSE_HOUSEHOLD to show the final result code (SAMFIR) in addition to the id-items and the head's name.

2. 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 show sum the number of males and females from all households assigned to the supervisor. It should look like:

```
Total Population Report
-----------------------

Male: 1020
Female: 1025
Total: 2045
```