Session 6: More Logic in Data Entry, Subscripts, Loops and Dynamic Value Sets

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

- Implement consistency checks across individuals in the household roster
- Work with occurrence labels in rosters and question text
- Understand the order of PROCs and where to place edit checks
- Use loops (*do while*)
- Understand and use "special values" (notappl, missing and default) in logic
- Use dynamic value sets to modify responses at run-time

Using Subscripts in Consistency Checks

Let's add a consistency check to show an error message if the son/daughter of the head of household is older than the head of household. Since the head of the household is the parent of anyone with relationship set to son/daughter, the head of household must be older than the child. We can put this check in the postproc of the age (**DEMAG1**). In order to compare the age of the head with the age of the child we will need to use subscripts since the two individuals are on different rows of the roster.

```
// Son/daughter cannot be older than head of household
if DEMREL = 3 and DEMAG1 > DEMAG1(1) then
    errmsg(tr("The head of household (%s) is %d years old and the
son/daughter (%s) is (%d) years old. The child cannot be older than the head
of household."), strip(NAME(1)), DEMAG1(1), strip(NAME), DEMAG1)
    select(tr("Correct age of ") + strip(NAME(1)), DEMBTH_DAY(1), tr("Correct
age of ") + strip(NAME), DEMBTH_DAY, tr("Correct relationship of ") +
strip(NAME), DEMREL);
endif;
```

In the case of the example questionnaire, these consistency checks between different members of the household can easily be accomplished as long as you are careful about using the correct subscripts. In other questionnaires where the head of household is not always entered first, these checks are considerably more complex since you have to identify which row the head is in and handle the case where the head is entered after the child.

Blanks and special values

What happens if we refer to an occurrence of a row in the demographics roster that doesn't exist? Try entering a line number for LBPPNO that is greater than the number of rows in the demographics roster. What are the values of NAME, DEMSEX and DEMAG1 when they are empty? Let's print them out using *errmsg* and see.

```
errmsg("NAME = %s, AGE=%d, SEX=%d", strip(NAME(LPBPPNO)),
        DEMAG1(LPBNNO), SEX(LPBPPNO));
```

The alphanumeric variable NAME is just empty but the numeric values DEMAG1 and DEMSEX 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 DEMAG1 is blank we can use the following comparison:

```
if DEMAG1(LPBPPNO) = notappl then
    // Age is blank, must be an empty row in demographics roster
    errmsg("%d is not a valid line in household roster", LBPPNO);
    reenter;
endif;
```

Of course, the above is not the best way to handle this. Instead we should use totocc() to check the number of occurrences in demographics roster.

```
if not LBPPNO in 1:totocc(DEM_ROSTER) then
    errmsg("%d is not a valid line in household roster", LBPPNO);
    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 n/0 (divide by zero)) or moving a field into a field that is too small to hold the value. For example, if AGE is a two-digit field, AGE = 118 will result in ***Default***.

Occurrence Labels

In "hours worked in main job" (**ACHRSM)** we have a repeating variable with one occurrence for each day of the week. Instead of simply showing the row numbers for this roster, we can show the days of the week by setting occurrence labels in the dictionary. Select the variable in the dictionary editor and choose "Occurrence Labels" from the edit menu. In the window that comes up enter the days of the week starting with Monday. Now remove and drop the variable onto the form again and notice that the row numbers have been replaced with the days of the week. We can also change the language in the dictionary editor and enter the labels in other languages.

Using Occurrence Labels in Question Text

We can also use these occurrence labels in the question text:

And how many hours did (you/NAME) work in (your/his/her) main job on %getocclabel%?

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 "And how many hours did (you/NAME) work in (your/his/her) main job on Monday?" and the text for the second occurrence will be "And how many hours did (you/NAME) work in (your/his/her) main job on Tuesday?" …

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 demographics roster once the name is entered we can do the following in the postproc of the name field (**NAME**):

```
PROC NAME
setocclabel(DEM_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 two to three people. The instructor adds errmsg to the postproc and preproc of the following: POPSTANLFS_FF (application), POPSTANLFS_QUEST (level), DEM_FORM (form), DEM_ROSTER (roster), and NAME (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(DEM_ROSTER(1), strip(NAME(1)));
setocclabel(DEM_ROSTER(2), strip(NAME(2)));
setocclabel(DEM_ROSTER(3), strip(NAME(3)));
```

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 POPSTANLFS _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 this
// then the case tree will not have correct occurrence labels until
// after we pass through demographics roster.

do numeric i = 1 while i <= totocc(DEM_ROSTER)
    setocclabel(DEM_ROSTER(i), strip(NAME(i)));
enddo;
```

Now that we have occurrence labels set in the individual rosters we could also use %getocclabel% in the question text to fill in the names although using the field %NAME% directly will work just as well.

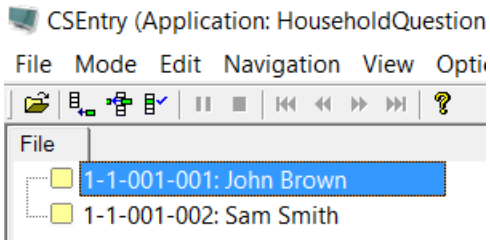
Case Labels

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-dwelling number-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 **DEMREL**.

```
PROC DEMREL

if DEMREL = 1 then
    // Set label for case in case listing
    // to an easier to read format.
    // We do this in the relationship proc since
    // that is where we first get the name of the head
    // of household.
    string caseLabel = maketext("%d-%02d-%03d-%03d: %s",
        PROVINCE, DISTRICT, GHDWNO, GHHHNO,
        strip(NAME));
    setcaselabel(POPSTANLFS_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 the demographics roster has labels like "Son/Daughter" and "Mother/Father" 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: DEMREL_MALE_VS and DEMREL_FEMALE_VS. Then in the preproc of relationship we choose between the two value sets:

```
PROC RELATIONSHIP
preproc
// Show male or female version of value set depending on sex of the person.
if DEMSEX = 1 then
    setvalueset(DEMREL, DEMREL_MALE_VS);
else
    setvalueset(DEMREL, DEMREL_FEMALE_VS);
endif;
```

OnFocus

There is one small bug with our dynamic value set implementation. If we go backwards through the household roster, the value set does not change when we go back from a female member to a male member or vice versa. This is because the preproc is only called when moving forwards through the questionnaire. Instead of using the preproc we can use another proc called **onfocus**. Onfocus is called every time you enter the field, independent of the direction. This is useful for modifying the user interface elements such as the value set or the question text. We generally use the preproc for anything that modifies the data (e.g. prefills) and the onfocus for anything that modifies the interface.

Dynamic Value Sets from Other Variables

Sometimes the text to place in a label depends on values from other variables in the dictionary. For example, in MJBDEE the second response is:

(NAME) IS RESPONSIBLE

Where (NAME) should be replaced with the value of NAME from the demographic roster.

To do this 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
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.

The arrays of codes and labels correspond to the labels and the "from" values in the value set in the dictionary. We set the first elements of the codes and labels array to match the first value set entry.

```
PROC MJBDEE
onfocus
// Set valueset to replace NAME in second response in value set
codes(1) = 1;
labels(1) = "MY EMPLOYER IS RESPONSIBLE FOR THAT";
```

For the second entry, we create the label using the function **maketext**, which formats a string like the **errmsg** statement but instead of displaying it on the screen, it returns the formatted text.

```
codes(2) = 2;
labels(2) = maketext("%s IS RESPONSIBLE FOR THAT", strip(NAME(LBPPNO)));
```

Finally, we need to a final entry of notappl to the codes array to signal to CSPro where the value set ends. Then we can pass the codes and values arrays to **setvalueset**.

```
codes(3) = notappl;
setvalueset(MJBDEE, codes, labels);
```

This works, but it would be better to avoid hardcoding the response text in the logic. This should always come from the dictionary. To do that we can use the function **getlabel** which returns the label from the dictionary for a value in a value set. For the second entry, we edit the value set in the dictionary and replace "(NAME)" with "%s" so that we can pass the dictionary label to maketext directly.

```
PROC MJBDEE
onfocus
// Set valueset to replace NAME in second response in value set
codes(1) = 1;
labels(1) = getlabel(MJBDEE_VS1, 1);
codes(2) = 2;
labels(2) = maketext(getlabel(MJBDEE_VS1, 2), strip(NAME(LBPPNO)));
codes(3) = notappl;
setvalueset(MJBDEE, codes, labels);
```

Dynamic Value Sets from a Roster

For **LBPPN,** we would like to create a value set from the names and line numbers of the eligible individuals in the household roster.

We will fill in the two arrays of codes and labels with names and line numbers of the eligible individuals in the household. 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 demographics roster and add an entry into our arrays for each household member 15 or over:

```
PROC LBPPN
onfocus
// Create the value set for from all eligible
// individuals in household roster
numeric indexRoster;
numeric nextEntryValueSet = 1;
do indexRoster = 1 while indexRoster <= totocc(DEM_ROSTER)

    if DEMAG1(indexRoster) >= 15 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.

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

Exercises

1. Add a consistency check to PXYREL to ensure that in the case where the head of household is the subject of the interview, the relationship in PXYREL matches the relationship of the respondent in DEMREL.
2. Add a dynamic value set for PXYPPN that includes all household members EXCEPT the respondent.
3. Add a dynamic value set for MJBPLC so that the label for the first response includes either “you” or the name of the subject of the interview from the household roster based on the value of PXYCHK.