Session 2: Introduction to Logic in Data Entry

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

- Use the command errmsg to display messages to the interviewer
- Use *if then else* statements to implement consistency checks across multiple variables
- Implement both "hard" and "soft" edit checks
- Use the command *skip* to skip fields
- End rosters from logic using *endgroup*
- Automatically fill in fields using logic

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. But 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 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 BEDROOMS:

```
PROC BEDROOMS

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

The *if* statement only executes the code between the *then* and then *endif* if the condition (BEDROOMS > 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. Everything on a line after // is considered a comment and is ignored by CSPro. It is good practice to use comments to document your logic.

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 less informative error message.

Let's take another example. We will display an error message if the user enters a person with relationship set to spouse and marital status set to single. Which proc do we put this check in? It needs to go in the marital status proc since it is only after entering marital status that we will have both the

values for marital status and relationship. With consistency checks between variables the logic always goes in the proc of the second variable to be captured.

```
PROC MARITAL_STATUS

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

Here we see that it is possible to combine multiple conditions using *and*. This condition will only be true if both subconditions are true. It is also possible to combine conditions using *or* which will be true if either the first subcondition or the second subcondition is true. For example if we expand our check to include spouses who are single, divorced or widowed we could write:

```
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 "<>" which means "not equal to":

```
// 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.

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 BEDROOMS and all lowercase for CSPro keywords like *if, then, errmsg and reenter*. Finally you should use comments as much as possible to help others and your future self better understand your code.

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 BEDROOMS

// Ensure that number of bedrooms does not exceed number of rooms.
if BEDROOMS > ROOMS then
    errmsg("Number of bedrooms, %d, exceeds number of rooms, %d.",
           BEDROOMS, 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. We can do this with errmsg but instead of %d we use %s since the name is an alpha variable.

```
// 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, 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 which means that they get padded with blank spaces. We can remove the trailing blank spaces by using the *strip()* function.

```
// 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), 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()* which returns the value set as an alpha value. It takes the name of the variable (or value set) to use 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.

```
// 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), 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. Going back to the bedrooms example:

```
PROC BEDROOMS

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

With the *select* we no longer need the *reenter* since CSEntry will automatically reenter the field they select.

Note that the select is part of the *errmsg* statement so there is no semicolon in between the two.

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 BEDROOMS

// Ensure that number of bedrooms does not exceed number of rooms.
if BEDROOMS > ROOMS then
    errmsg("The number of bedrooms, %d, is greater than the number of rooms,
%d.",
           BEDROOMS, ROOMS)
           select("Fix number of rooms", ROOMS,
                  "Fix number of bedrooms", 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 “rent or own”.

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.

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), getlabel(MARITAL_STATUS, MARITAL_STATUS))
    select("Fix marital status", MARITAL_STATUS,
           "Fix relationship", RELATIONSHIP);
endif;
```

Let’s take another example. We will display an error message if the head of household is less than 12 years old. Which proc do we need to put this check in? It has to go in the AGE proc since only then will we have entered both AGE and RELATIONSHIP.

Again we can use the operator *and* to combine multiple conditions in a single if expression. In this case we want to show when the relationship is head of household AND the age is less than 12.

```
PROC AGE

// Do not allow head of household to be under 12 years of age.
if RELATIONSHIP = 1 and AGE < 12 then
    errmsg("%s cannot be head of household and only %d years old",
        strip(NAME), AGE)
        select("Correct Relationship", RELATIONSHIP,
```

```
                "Correct Age", AGE);
endif;
```

Preproc and Postproc

So far all of our examples have used the postproc: logic that is executed after the data is entered. We can instead add logic to the preproc which is executed before the data is entered.

Let's fill in the date of interview automatically so that the interviewer doesn't have to enter it. We can do this in the preproc of INTERVIEW_DAY:

```
PROC INTERVIEW_DAY
preproc

// Fill in interview date automatically based on current date
INTERVIEW_DATE = sysdate("YYYYMMDD");
```

To avoid having to enter data for this field we can use the statement *noinput*.

```
PROC INTERVIEW_DAY
preproc

// Fill in interview date automatically based on current date
INTERVIEW_DATE = sysdate("DDMMYYYY");
noinput;

PROC INTERVIEW_MONTH
preproc
noinput;

PROC INTERVIEW_YEAR
preproc
noinput;
```

Alternatively we can make the fields 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. Be aware that if you do not set the value of a protected field before the interviewer gets to the field CSEntry will give you an error.

There is one small problem with filling in the date automatically. If we go back to an old case in modify mode, we will change the interview date to the current date. To avoid this we can check to see if we are in add or modify mode using the *demode()* function.

```
PROC INTERVIEW_DAY
preproc

// Fill in interview date automatically based on current date
// if in add mode.
if demode() = 1 then
```

```
    INTERVIEW_DATE = sysdate("DDMMYYYY");
endif;
```

Skips

If the house is not rented we should not ask the monthly rent. We do this with the *skip* command in the postproc of E03 rent or own:

```
PROC RENT_OR_OWN

// Skip monthly rent if house is not rented
if RENT_OR_OWN = 1 then
    skip to ROOFING_MATERIAL;
endif;
```

Note that once we have skipped the field MONTHLY_RENT 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 RENT_OR_OWN. This is an important difference between system controlled and operator controlled data entry mode.

Now let's skip questions B6-B10 for household members under 12. In which proc will we put the skip? If we put it in AGE we would skip PLACE_OF_BIRTH so we put it in the postproc of PLACE_OF_BIRTH. What do we skip to? We want to skip to LINE_NUMBER but on the next row of the roster so we use *skip to next*.

```
PROC PLACE_OF_BIRTH

// Skip rest of row for household members under 12
if AGE < 12 then
    skip to next;
endif;
```

endgroup and endlevel

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 to the roster. Let's call it MORE_PEOPLE and give it the value set Yes - 1, No – 0. We will put it at the end of the 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 = 0 then
    endgroup;
endif;
```

With this we no longer need the occurrence control field of the roster however we do need to change our usage of *skip to next* in the adult roster to *skip to MORE_PEOPLE* otherwise we will not be able to end the roster after entering a household member under 12.

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

Exercises

1. Add a consistency check in section E that displays an error message if the interviewer specifies that the household has a flush toilet in question E09 and does not also specify piped water inside house for question E10. Use a select clause to allow them to correct either E09 or E10.
2. Display an error message if the highest level of education is college or graduate but the age is less than 18. Make this a "soft check" so that the interviewer can ignore the error if they wish.
3. If the answer to question E08, have toilet, is no skip question E09, type of toilet.
4. Add an additional question to E11 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.
5. Add a record and a form for section D: Fertility. Do not include the name of the woman but do include the IDNO. Implement the skip patterns described on the questionnaire. Don't worry about the last question on the form that displays the total births. We will cover that in a later lesson.
6. Make the interview start time protected and write logic to fill it in automatically. You can use the function *systime()* to get the current time. Note that *systime*() is a bit different than *sysdate*() so you may want to look in the online help for hints on how to get the hours and minutes from it.