Session 6: Navigation & System Control

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

- Create cascading questions
- 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

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 same country) 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	
This country	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);
```

When removing the item and adding the subitems, we will lose the skip in the place of birth proc and we will need to add it into the proc for the subitem.

In addition, now that the variable PLACE_OF_BIRTH is no longer on the form since we replaced with its subitems, anywhere we use it in the demographics roster, we now need to use curocc() as a subscript. For example, the prefill of the YEAR_MOVED_TO_DISTRICT will change to:

```
PROC YEAR_MOVED_TO_DISTRICT
preproc
// Prefill date moved to district if district
// of household matches district of birth
if PLACE_OF_BIRTH(curocc()) = DISTRICT then
    $ = 9999; // Born in current district
    noinput;
endif
```

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 create a dictionary by going to “Add Files” in the “File” menu and entering the name of a new external dictionary. Let’s call it “districts.dcf”. This dictionary will have the province code and district code as the id-items and a single record containing a single item: the district name. 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. 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. The easiest way to do this is to copy and paste from the main dictionary and then modify the names.

Once we have the dictionary we need to convert the Excel spreadsheet into a CSPro data file. We can use the **Excel2CSPro** tool to do this. We point it at the dictionary and spreadsheet, set the worksheet number and start row, enter the name of the resulting data file (let's call it "districts.dat") and hit "Convert".

Finally in the district proc we 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 variable DI_DISTRICT_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 .dat 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 PROVINCE

// Assign province name to dictionary variable
// so we can display it on form.
PROVINCE_NAME = getlabel(PROVINCE, PROVINCE);
```

```
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 district name from lookup
    // to main dictionary variable so we can
    // display it on form.
    DISTRICT_NAME = DI_DISTRICT_NAME;
endif;
```

The Resource Folder

Instead of copying the lookup file to the Android file each time it is updated we can put it 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 lookup files. Let's create a resource folder, add it to our application and put our lookup file 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 file districts.dat in the resource file and rebuild the pen file.

Group Exercise: EA Lookup File

Use the Excel spreadsheet EnumerationAreas.xlsx to create a lookup file to validate the enumeration area field. The Excel file contains columns for province, district, and enumeration area. First create a dictionary called EnumeraionAreas.dcf for the lookup with these columns as variables. Which will be the id-items? Use Excel2CSPro to generate a data file named EnumerationAreas.dat and put it in the resource folder. Add the new dictionary to the application and use loadcase in the postproc of ENUMERATION_AREA to check that the enumeraton area code is valid for the province and district chosen.

Dynamic Value Set from a Lookup File

As a final example we will combine the lookup file with dynamic value sets to create the dynamic value set for the district. Up to now our lookup files have had a one to one correspondence between keys and values. For this case we need to get a list of multiple districts per province. To do this we make a new dictionary containing the province as the only id-item and add a multiply occurring record to contain the district codes. Let's call this dictionary "districtsperprovince.dcf". Create the lookup file for it from the district codes in annex 2 of the questionnaire. This data file will actually be identical to the one we created for checking the province and district. Unfortunately, CSPro will not let us use the same data file for two different external dictionaries (since both could potentially modify the file simultaneously) so we will need to create a copy of the lookup data file.

Once we have the dictionary and lookup file we use loadcase as before, however now the case that is loaded contains more than one record occurrence. There will be one occurrence for each district in the province. We loop through all the districts and add them to the codes and labels array to make the dynamic value set.

```
PROC DISTRICT

onfocus
// Create dynamic value set of districts in selected province
// using lookup file
if loadcase(DISTRICTSPERPROVINCE_DICT, PROVINCE) = 1 then
    // In the lookup file, DISTRICTSPERPROVINCE_REC is a multiply occurring
    // record since there is more than one district per province. We
    // need to iterate through all the districts in the province
    // and them to the value set.
    numeric i;
    do i = 1 while i <=
count(DISTRICTSPERPROVINCE_DICT.DISTRICTSPERPROVINCE_REC)
        labels(i) = DPP_DISTRICT_NAME(i);
        codes(i) = DPP_DISTRICT_CODE(i);
    enddo;
    codes(i) = notappl;
    setvalueset(DISTRICT, codes, labels);
endif;
```

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 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
// 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 (B13) and the line number of mother of deceased E08.

```
PROC MOTHERS_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(DEMOGRAPHICS_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 DECEASED_MOTHERS_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(DEMOGRAPHICS_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(DEMOGRAPHICS_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 SURVEY_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 member names",
                             "Demographics");
    if section = 1 then
        skip to IDENTIFICATION_FORM;
    elseif section = 2 then
        skip to NAMES_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… 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",
                             "Household member names",
                             "Demographics");
    if section = 1 then
        advance to IDENTIFICATION_FORM;
    elseif section = 2 then
        advance to NAMES_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 member names",
                             "Demographics");
    if section = 1 then
        move to IDENTIFICATION_FORM advance;
    elseif section = 2 then
        move to NAMES_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 demographics roster. If they choose "demographics" 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 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 cancelled.
function pickFromHouseholdRoster()
    numeric i;
    do i = 1 while i <= totocc(DEMOGRAPHICS_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
        move to NAMES_FORM advance;
    elseif section = 3 then
        numeric index = pickFromHouseholdRoster ();
        if index > 0 then
            move to SEX(index) advance;
        endif;
    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(DEMOGRAPHICS_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 do the same with the interview date using the function sysdate():

```
PROC INTERVIEW_DATE
preproc
// Prefill interview date with current time.
if visualvalue($) = notappl then
    $ = sysdate("YYYYMMDD");
endif;
```

Exercises

1. Extend the *goto()* function to include the remaining sections of the questionnaire (D through H).
2. Extend the goto() function so that when the interviewer chooses section E (deaths) the application displays a table with the name of the deceased and the month and year of death. When the interviewer chooses one of the deaths, go the first field in the chosen row of the deaths roster.
3. 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.
4. For B18, occupation, allow the user to choose the occupation based on the hierarchical occupation code. Let them first choose from the 1 digit categories (Managers, Professionals, Technicians and associate professionals…) then from the 2 digit categories that fall under the top-level category chosen, then the three digit categories and finally the 4 digit categories. For example, if they choose 2 for the first level code then the value set for the second level code should be all of the 2 digit codes that start with 2 (21, 22, 23…). If they then pick 23 for the second level code then the value set for the third level should be all codes that start with 23 (231, 232, 233, 234…). If they pick 234 for the third level code then the value set for the fourth level code should be all the codes that start with 234 (2341, 2342).
5. Implement a check on the minimum and maximum per unit possession values in question G01. Use the spreadsheet in annex 5 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. Make sure to put the lookup file in the resource folder.
6. Fill in the interview end time automatically using the *systime*() command and make the interview time fields protected. Note that unlike with the start time, you will need to do this at the end of the interview, i.e. in the postproc of the questionnaire. However, since you cannot leave a protected field blank, you will need to put a value in the end time fields when you pass through them in section A. You can prefill them with 99 in their preprocs, but make sure only to do so if they are blank, otherwise you will overwrite the end time when re-opening the case in modify mode.
7. 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.

1 Interview completed
2 Non-contact
3 Vacant
4 Refused
5 Partially complete