# Session 2 Lookup Files

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

- Create a lookup file from an Excel file
- Use the `loadcase` function to read and validate using a lookup file
- Use `forcase` to loop through a lookup to create a dynamic value set
- Update a lookup file from CSPro logic

## Using a Lookup File for Login

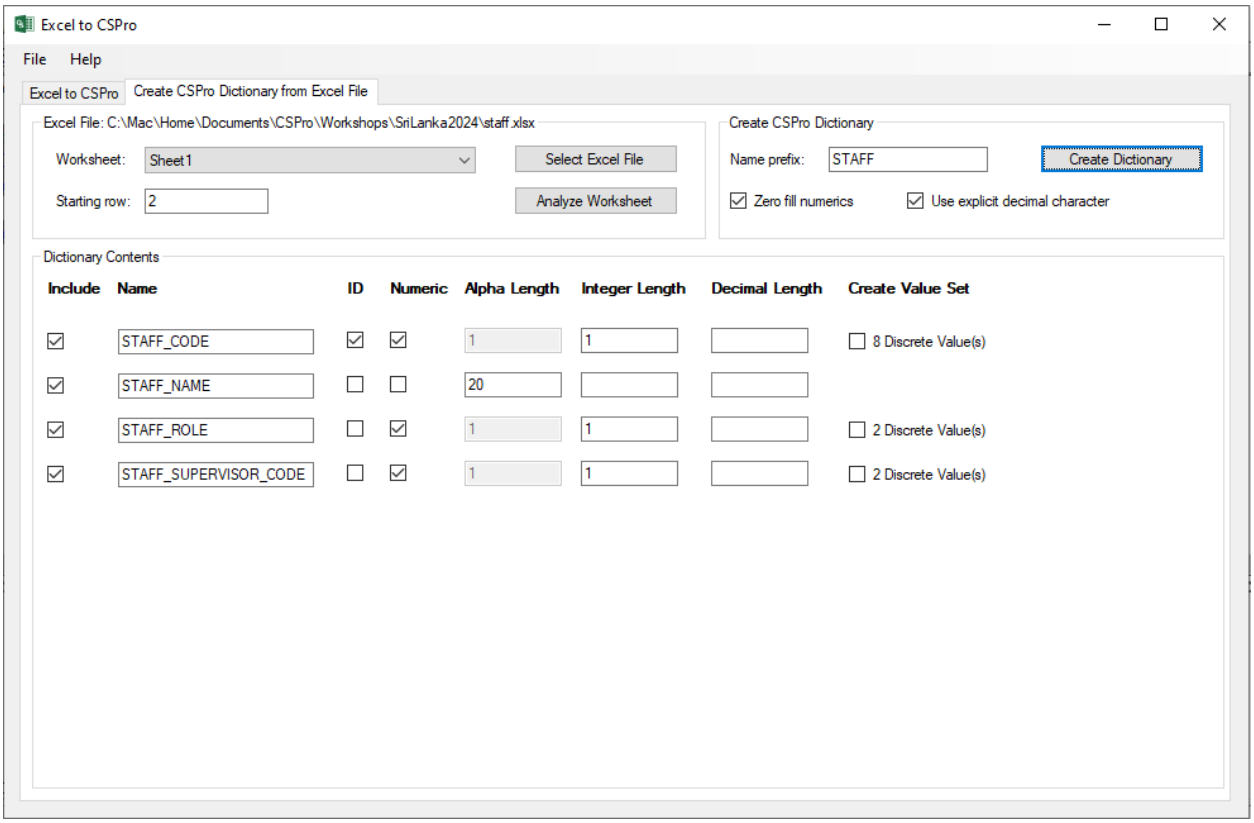
In our menu program, rather than have the user choose their role, enter their name and enter their user code, lets use a spreadsheet that lists all the users along with their role and user code so we can look up the name and role from just the user code. This way the user only needs to enter their code, and we can fill in the rest from the spreadsheet. We can also prevent the user from entering an invalid code. Here is an example staff file:

| Staff Code | Name | Role (1-interviewer, 2-staff) | Supervisor Code |
|---|---|---|---|
| 0001 | Nuwan Perera | 2 | |
| 0002 | Anura Wijesinghe | 1 | 0001 |
| 0003 | Anushka Peris | 1 | 0001 |
| 0004 | Kanchana Jayawardena | 1 | 0001 |
| 0005 | Kasun Fernando | 2 | |
| 0006 | Nadeesha Senanayake | 1 | 0005 |
| 0007 | Asela Jayasinghe | 1 | 0005 |
| 0008 | Chathurika de Silva | 1 | 0005 |

To use this spreadsheet in CSPro as a lookup file we need to convert it into a CSPro data file and create a CSPro dictionary for it. We can use the Excel to CSPro tool to do both.

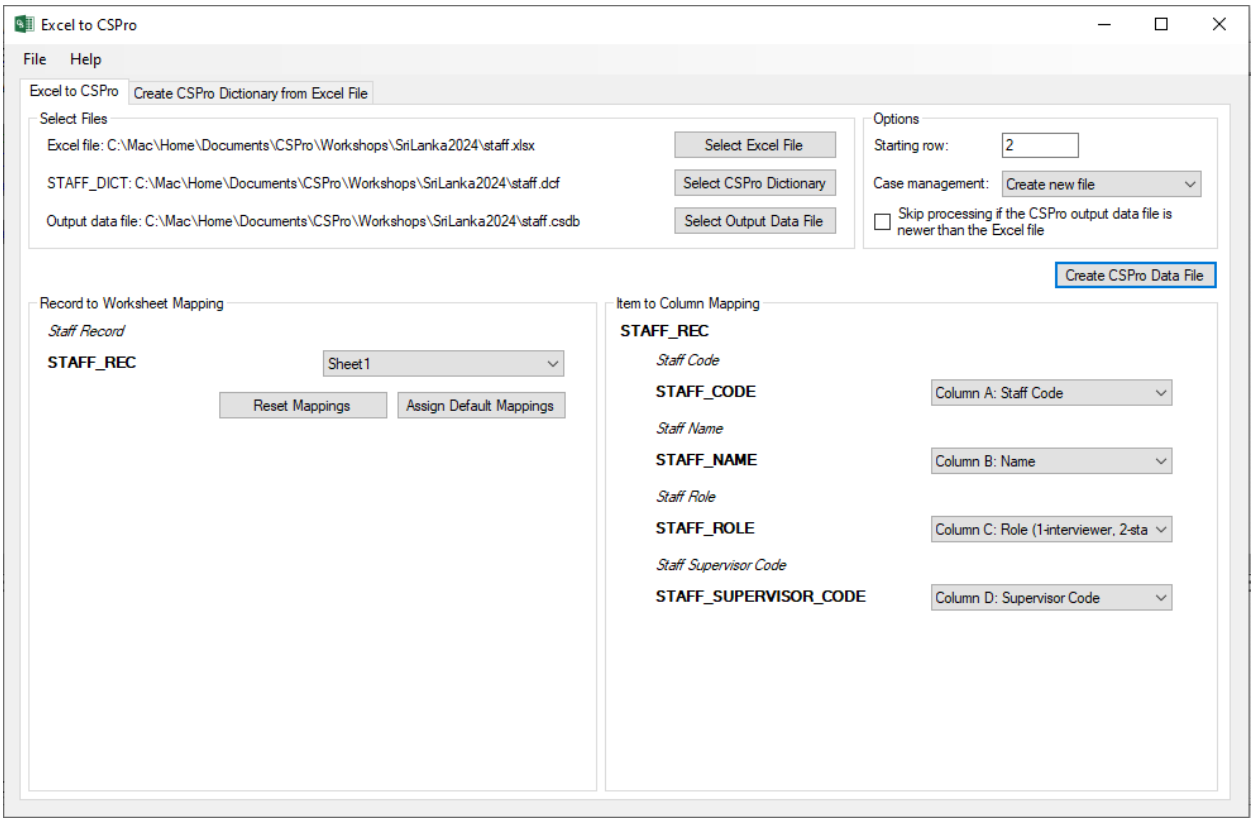
1) Select *Excel to CSPro* from the Tools menu
2) Select the Create CSPro dictionary from Excel File tab
3) Click *Select Excel File* and browse to the file *staff.xlsx*.
4) Click "Analyze Worksheet". This detects the dictionary variables to create from the columns of the spreadsheet. The tool will show four variables, one for each column and will set the default names to column headers of the spreadsheet.
5) Select the id-items. Our lookup file dictionary will have the staff code as the only id-item. The name and role will be regular variables. Next to the STAFF_CODE column check the "ID" box to indicate that it should be an id item.
6) Update the variable names. To avoid name conflicts with the main dictionary I usually add a prefix to all the variable names when creating an external dictionary like “STAFF_”.

7) Click *Create Dictionary* to generate the CSPro dictionary. Save it in the folder Popstan*HouseholdSurvey/Dictionaries*.




Once we have the dictionary, we need to convert the Excel spreadsheet into a CSPro data file. We can use the first tab of the Excel2CSPro tool to do this.

1) Click on the first tab – *Excel to CSPro*
2) Click *Select Excel File* and browse to the staff.xlsx file again
3) Click *Select CSPro Dictionary* and browse to the Staff dictionary that we just created.
4) Click *Select Output Data File* and specify a new output data file to write the lookup file to. Call it *staff.csdb* and save it to the folder Popstan*HouseholdSurvey/Data*.
5) In the *Record to Worksheet Mapping* section choose the worksheet *Sheet1*.
6) In the *Item to Column Mapping* section, link the dictionary variables to the appropriate columns in the Excel spreadsheet: STAFF_CODE to "Column A: Staff Code", STAFF_NAME to "Column B: Name", etc...
7) Click *Create CSPro Data File*" to generate the file
8) Verify the file in DataViewer to make it sure it was converted correctly.




To use the new dictionary in our menu application, we must first add it to the application by going to File -> Add Files and clicking on the … next to External Dictionary. This adds the new dictionary as an external dictionary in the application which allows us to use it as a lookup.

Let's modify the login so that the when the user enters the staff code, we look up the name and role from the lookup file. To do this use the `loadcase` function. `loadcase` takes the name of the dictionary (STAFF_DICT) and the values to use as keys (id-items) for the lookup. For example, to lookup the user with code 003 we would do:

```
loadcase(STAFF_DICT, 3)
```

If `loadcase` finds a record in the lookup file with staff code 3 it will return 1 and set the variables in STAFF_DICT to the values from the case it found. In this case that means setting the id-item STAFF_CODE and the variables STAFF_NAME and STAFF_ROW.

We can use this to test if the staff code is valid and to load the username and role.

```
PROC USER_CODE

// Verify staff code using lookup file
if loadcase(STAFF_DICT, USER_CODE) = 0 then
    errmsg("Invalid staff code. Try again.");
    reenter;
```

```
endif;

// Get the name and role from the lookup file
USER_NAME = STAFF_NAME;
ROLE = STAFF_ROLE;

// Go to the appropriate menu for the role chosen
if ROLE = 1 then
    skip to INTERVIEWER_MAIN_MENU;
else
    skip to SUPERVISOR_MAIN_MENU;
endif;
```

Now that we are loading the name and role from the lookup file, we no longer need them on the form so we can remove those fields from the form and delete the logic in their procs.

**Group Exercise**

In the household data entry application, we do not currently validate that the id-items entered by the interviewer correspond to a valid household in the sample. In your groups, add that validation using a lookup file. Use the *sample.xlslx* spreadsheet which lists the province, district, cluster, household number and name of head of household for each household in the sample. Use *Excel to CSPro* to create a dictionary and data file from the spreadsheet. The new dictionary will need multiple id-items to uniquely identify a household so mark province, district, cluster and household number all as id-items. Add the dictionary as an external dictionary in the household application. In the postproc for HOUSEHOLD_NUMBER in the household application, add logic to use loadcase to verify that the household entered is found in the sample. Note that in this case, you will need to pass multiple variables to loadcase for the lookup since there are multiple id-items. If the loadcase fails, show an error message and have the interviewer reenter the values. If the loadcase succeeds, use the head of household name from the lookup to prefill the name of the first person in the household roster.

## Creating a Value Set from a Lookup File

Another way to prevent the user from entering invalid id-items for the household would be to have them pick from the list of households that are assigned to them. This makes it impossible for them to enter invalid household codes. We can do this by creating a dynamic value set from the households in the household assignments lookup file. In the menu program, before launching the household program, we will loop through the household assignments lookup file, find all the households assigned to the interviewer and add them to a value set for the interviewer to choose from.

First, let's create the household assignments lookup file and dictionary. It will have the household identifiers as id-items and the interviewer code as a regular variable. A row in the spreadsheet means that the household with the codes from the first four columns is assigned to

the interviewer in the last column. This will allow us to look up the interviewer who is assigned to any given household if we have the ids of the household.

| Province | District | Cluster | Household Number | Interviewer Code |
|---|---|---|---|---|
| 1 | 01 | 001 | 001 | 0002 |
| 1 | 01 | 001 | 002 | 0002 |
| 1 | 01 | 001 | 003 | 0003 |
| 1 | 01 | 001 | 004 | 0003 |

For now, we have filled in a few assignments we can test with, but eventually we will allow the supervisor to create the household assignments in the menu program.

To allow the interviewer to pick a household from the assignments, we need to add a variable in the dictionary. Let's call it INTERVIEW_CHOOSE_HOUSEHOLD. We will create a dynamic value set for the variable that contains all the households assigned to the interviewer. What should the length of this variable be? It needs to contain all the identifiers of the household i.e. province, district, cluster and household number. We can concatenate them together to form a variable of length 9. We will use an alpha variable instead of numeric. In this case, a numeric variable would work, but in real-world surveys, the length of the combined id-items of a household often exceeds the maximum length of a number in CSPro which is 15 so it is best practice to use an alpha when combining id-items. After creating the new variable, don't forget to add it to the form and to set the capture type to radio button.

In the interviewer menu, when the interviewer chooses to interview households, instead of immediately launching the household application, we will skip to our new INTERVIEW_CHOOSE_HOUSEHOLD field.

```
PROC INTERVIEWER_MAIN_MENU
onfocus
// Clear previous choice
$ = notappl;

postproc
// Handle the menu choice
if $ = 1 then
    // Household questionnaire
    skip to INTERVIEW_CHOOSE_HOUSEHOLD;
elseif $ = 2 then
    // Sync data
    synchronizeData();
elseif $ = 9 then
     // Exit
     savedLogin = 0; // Clear saved login
     stop(1);
endif;
```

```
// Show interviewer menu again
reenter;
```

In the onfocus proc of the new INTERVIEW_CHOOSE_HOUSEHOLD field we create the value set. To loop through all the households in the household assignments file, we can use a special type of loop called forcase which loops through each case in an external data file and at each iteration loads all the variables from the data file. To limit the cases to only those that are assigned to the current interviewer we can add a filter by adding a where clause to the forcase. For each case we find, we add a value to the value set.

```
PROC INTERVIEW_CHOOSE_HOUSEHOLD
onfocus
// Create a dynamic value set from the assignments file.
valueset string valueSetAssignedHouseholds;

forcase HH_ASSIGN_DICT where ASSIGN_INTERVIEWER_CODE = USER_CODE do
    // The label is made up of the household codes separated by "-" to
    // make it more readable
    string label = maketext("%v-%v-%v-%v",
          HH_ASSIGN_PROVINCE, HH_ASSIGN_DISTRICT,
          HH_ASSIGN_CLUSTER, HH_ASSIGN_HOUSEHOLD_NUMBER);
    // The code is all the id-items from the assignments (province,
    // district, cluster, household number) concatenated together
    // as a string. The key() function returns exactly that.
    string code = key(HH_ASSIGN_DICT);

    valueSetAssignedHouseholds.add(label, code);
endfor;

if length(valueSetAssignedHouseholds.codes) = 0 then
    errmsg("No households have been assigned to you. Please ask your
supervisor to update the assignments.");
    reenter INTERVIEWER_MENU;
endif;

setvalueset($, valueSetAssignedHouseholds);
```

Remember that in order to use a dynamic value set, we need to set the Capture Type to Radio Button in the field properties of INTERVIEW_CHOOSE_HOUSEHOLD on the form.

Now when the interviewer enters the INTERVIEW_CHOOSE_HOUSEHOLD field, the value set will contain the list of households assigned to them. Once the user picks a household from the list, we need to open the household questionnaire and prefill the id-items based on the chosen household. We can update the launchHouseholdDataEntry function to take the concatenated id-items (or the key) of the household as an argument and set it as the *key* property in the pff.

```
function launchHouseholdDataEntry(string householdIds)
    pff householdPff;
    householdPff.load("../Household/Household.pff");
    householdPff.setProperty("OnExit", "../Menu/Menu.pff");
    householdPff.setProperty("INTERVIEWER_NAME", USER_NAME);
    householdPff.setProperty("INTERVIEWER_CODE", USER_CODE);
    householdPff.setProperty("key", householdIds);
    householdPff.exec();
end;
```

When we set the *key* property in the pff, CSPro will automatically pre-fill the id-items in the household questionnaire. In addition, if a questionnaire with the same ids already exists in the data file, it will open the existing questionnaire in modify mode instead of creating a new one.

The next step is to call launchHouseholdDataEntry in the `postproc` of INTERVIEW_CHOOSE_HOUSEHOLD. We pass it the value of INTERVIEW_CHOOSE_HOUSEHOLD as the key. Since the value set was built in such a way that each code corresponds to the key of a household, the value CHOOSE_HOUSEHOLD_INTERVIEW is set to the key of the household that the interviewer chose.

```
PROC INTERVIEW_CHOOSE_HOUSEHOLD
onfocus
...

postproc
// Launch household program with the key of the household chosen
launchHouseholdDataEntry($);
```

Now when we launch the household data entry application, the province, district, cluster number and household number are filled in automatically. However, it is still possible for the interviewer to edit them. To prevent that, we can set them as protected in the case that they are set from the menu program. We do this by checking to see if the value is already set in the preproc of the item. Note that because we are in the preproc of the item, we must us `visualvalue` since the item has not yet been validated. We will learn more about `visualvalue` in a later session.

```
PROC PROVINCE
preproc
// Set as protected if the value was passed in from the menu program
if visualvalue($) <> notappl then
    protect($, true);
else
    protect($, false);
endif;
```

We can do the same thing for the other id-items – district, cluster and household number.

**Group Exercise**

In order for the supervisor to be able to assign households to interviewers, the supervisor needs to first choose an interviewer and then choose a household. In your teams, implement the first part of that – choosing an interviewer.

1) Add a new dictionary variable ASSIGNMENT_CHOOSE_INTERVIEWER and put it on the form.
2) In the supervisor menu, when the supervisor chooses Assign Households, skip to this new field.
3) In the onfocus of ASSIGNMENT_CHOOSE_INTERVIEWER create a dynamic value set made up of the interviewers that are supervised by the supervisor. You can filter using the STAFF_SUPERVISOR_CODE. The value set labels should be the names of the interviewers and the codes should be their staff codes.
4) Test your logic to make sure the correct interviewer list is displayed for each supervisor.

## Updating a Lookup File from CSPro Logic

Now the supervisor can choose an interviewer to assign households to. The next step is to allow them to pick a household to assign to the chosen interviewer. Once we have the both the household and the interviewer, we can update the assignments file.

Again, we need to create a dynamic value set of households for the supervisor to choose from. We can use the sample lookup file which contains all the households in the sample. First, we add a new field ASSIGNMENT_CHOOSE_HOUSEHOLD. Just like INTERVIEW_CHOOSE_HOUSEHOLD it will be an alpha of length 9 that will contain the concatenated household id-items. If we put this field on the form immediately after ASSIGNMENT_CHOOSE_INTERVIEWER, we do not need to skip to it since CSPro will automatically go to it once the supervisor chooses the interviewer.

In the `onfocus` we create the value set from the sample data file.

```
PROC ASSIGNMENT_CHOOSE_HOUSEHOLD
onfocus
// Create a dynamic value set from the sample file.
valueset string valueSetHouseholds;

forcase SAMPLE_DICT do
    string label = maketext("%v-%v-%v-%v",
          SAMPLE_PROVINCE, SAMPLE_DISTRICT,
          SAMPLE_CLUSTER, SAMPLE_HOUSEHOLD_NUMBER);
    string code = key(SAMPLE_DICT);

    valueSetHouseholds.add(label, code);
```

```
endfor;

if length(valueSetHouseholds.codes) = 0 then
    errmsg("The sample file is empty. Please ask for help from IT.");
    reenter SUPERVISOR_MENU;
endif;

setvalueset($, valueSetHouseholds);
```

Now in the `postproc` we can use the chosen interviewer and the chosen household to update the household assignments. We want to add a new case in the file (conceptually adding a new line in the spreadsheet) containing the id-items of the household and the interviewer code to indicate that the household is assigned to the interviewer.

We can use the `writecase` function to do this. It takes the name of the dictionary associated with the lookup file as an argument. It writes the current values of the dictionary variables to the lookup file. For example, if we run the following code:

```
HH_ASSIGN_PROVINCE = 1;
HH_ASSIGN_DISTRICT = 2;
HH_ASSIGN_CLUSTER = 3;
HH_ASSIGN_HOUSEHOLD_NUMBER = 4;
HH_ASSIGN_INTERVIEWER_CODE = 1;
writecase(HH_ASSIGN_DICT);
```

It will add the following case to the lookup file:

| Province | District | Cluster | Household Number | Interviewer Code |
|---|---|---|---|---|
| 1 | 02 | 003 | 004 | 0005 |

Note that if the id-items match an existing case in the lookup file, it will overwrite the existing case instead of adding a new case. A lookup file can never have two cases with the same id-items.

Going back to the `postproc` of ASSIGNMENT_CHOOSE_HOUSEHOLD, we need to use `writecase` just like above, however the individual values of province, district, cluster and household are concatenated together so we need to separate them out. One way to do this is to use [] to extract substrings and then convert those substrings to numbers.

```
PROC ASSIGNMENT_CHOOSE_HOUSEHOLD
postproc

// Update the assignments file based on
// interviewer and household chosen.
HH_ASSIGN_PROVINCE = tonumber($[1:1]);
```

```
HH_ASSIGN_DISTRICT = tonumber($[2:2]);
HH_ASSIGN_CLUSTER = tonumber($[4:3]);
HH_ASSIGN_HOUSEHOLD_NUMBER = tonumber($[7:3]);
HH_ASSIGN_INTERVIEWER_CODE = ASSIGNMENT_CHOOSE_INTERVIEWER;
writecase(HH_ASSIGN_DICT);
```

A second approach would be to use loadcase on the sample data file to lookup the chosen household and then use the id variables from the sample dictionary.

```
PROC ASSIGNMENT_CHOOSE_HOUSEHOLD
postproc

// Update the assignments file based on
// interviewer and household chosen.
loadcase(SAMPLE_DICT, $);
HH_ASSIGN_PROVINCE = SAMPLE_PROVINCE;
HH_ASSIGN_DISTRICT = SAMPLE_DISTRICT;
HH_ASSIGN_CLUSTER = SAMPLE_CLUSTER;
HH_ASSIGN_HOUSEHOLD_NUMBER = SAMPLE_HOUSEHOLD_NUMBER;
HH_ASSIGN_INTERVIEWER_CODE = ASSIGNMENT_CHOOSE_INTERVIEWER;
writecase(HH_ASSIGN_DICT);
```

The advantage of the second approach is that the code won't break if you change the length of one of the id-items.

## Exercises

1. When the supervisor is choosing a household to assign, it would be helpful to see which households are already assigned and to whom. Update the logic that creates the dynamic value set in ASSIGNMENT_CHOOSE_HOUSEHOLD to show the name of the interviewer to which it is assigned in the value set label. If the household is not found in the household assignments file, add "unassigned" to the label. Since the forcase loop in that proc is looping over the sample lookup file and the assignee is in the household assignments lookup file, you will need to use loadcase to retrieve the assignee for each household in the assignments file. This will get the assignee interviewer code. You will then need to use loadcase again on the staff file to get the interview name from the interviewer code.

2. Currently any supervisor can assign households in any cluster. We should limit that so that a supervisor can only work with a set of assigned clusters. Create a new lookup file and dictionary from the cluster assignments.xlsx spreadsheet and add it to the menu program. In the supervisor menu, before the supervisor picks the household to assign, add a new field where they first pick the cluster. Call this new field ASSIGNMENT_CHOOSE_CLUSTER and place it in between ASSIGNMENT_CHOOSE_INTERVIEWER and ASSIGNMENT_CHOOSE_HOUSEHOLD. In the onfocus on the new field show the list

of clusters assigned to the supervisor in a dynamic value set. Then in the ASSIGNMENT_CHOOSE_HOUSEHOLD proc, filter the households shown to include only those in the cluster chosen in the previous field.