Exercise 3: Derived fields and Check file commands unrelated to a specific field

At the end of this exercise you should be able to:

a. Create automatically calculated fields from one or more existing fields by editing the `*.qes` and `*.chk` files

b. Opening the whole CHK file and make additions to it that cannot be done at the field level

Is the laboratory serial number really unique?

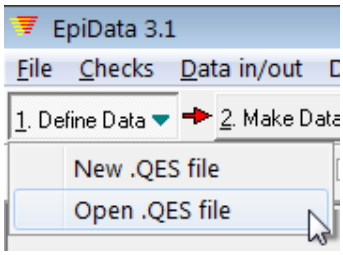
We noted earlier that the laboratory serial number is unique for one given year and for a single laboratory. If the study involves more than one year, then it is not unique; a combination of serial number and year of registration will make it unique. If the study involves more than one laboratory, then even this is not unique; you will then require a combination of laboratory identifier, serial number, year of registration to make it unique. How do we construct a variable which derives information from other variables entered and is auto-generated? That is the topic of our discussion now.

This is what we are going to do in this exercise. But first, we need to have a method (that we are going to use for every exercise from now on) to preserve what we already got and save it with a new name and then make the necessary changes.

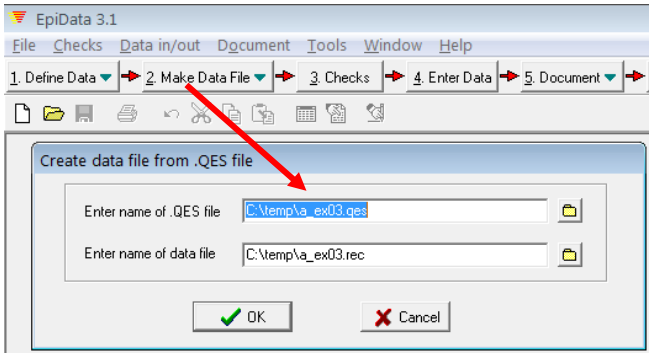
Making new `*.QES`, `*.REC`, and `CHK` files from existing files

First download the solution from Exercise 2 and overwrite your existing files. In this way, we reach a uniform place from where we can begin to build the next exercise. Then, let us use this to create the new set of EpiData triplets and name them `A_EX03.`

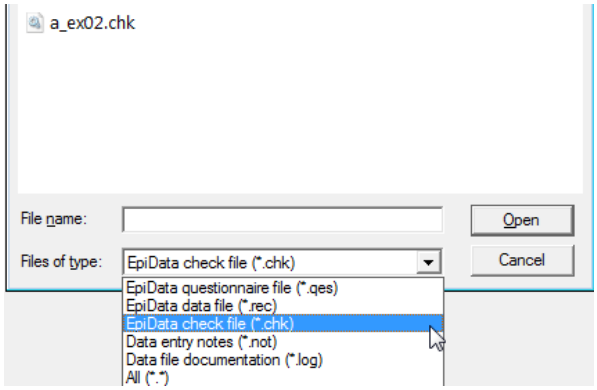
First, let us make `A_EX03.QES` from `A_EX02.QES`: This is easily done by opening:



the already existing `A_EX02.QES` and save it right away as `A_EX03.QES`. It is then edited as needed, saved and then the `A_EX03.REC` file is made from it as we did before:



What is missing is the A_EX03.CHK. Using "3. Checks" is not an option as this will create a new check file with none of the checks we have previously created. Hence, we need to open the A_EX02.CHK file and save it as the new A_EX03.CHK file. Click thus "File" | "Open" (or more simply **CTRL+O**) and select the correct file type:



Open A_EX02.CHK and save it as "A_EX03.CHK". Before you close it, let's remove the KEY UNIQUE from the field SERNO, as the latter will no more stay unique and IDCODE will become the new unique identifier:

```
serno
  KEY UNIQUE 1
  MUSTENTER
END
```

Save and exit.

Note: This is another way of opening the CHK file and here you will click on the CHK file to open it. Remember the other way of opening the CHK file - that is via "3. Checks", you have to click on the REC file for which the checks have to be defined.

Task: We have to create a new variable IDCODE which will be auto-generated by the combination of laboratory identifier (let us say LABCODE), year of registration (which can be extracted from REGDATE) and laboratory serial number. This identifier will thus have a form like LABCODE-YEAR-SERNO.

As you will quickly notice, we will have to create three new fields, one for the unique identifier (IDCODE), one for laboratory identifier (LABCODE), and since it is easier to enter laboratory names rather than to remember codes for each laboratory before entry, we have to

create one more field for the laboratory name (LABNAME). Where do we have to go to create new fields? That's right! It's the QES file!!

Editing the new A_EX03.QES file

We will make one new field that is a MUSTENTER field, with the field name LABNAME. It is a text field (any string) and will have a length of 20 characters:

```
labname Name of the laboratory ____________________
```

Let us have a label block for the laboratory names and codes, where the laboratory name will be the Field value and the laboratory code will be the Value label. We will extract this laboratory code and make it a component of our derived variable.

Additionally we will make two fields that will be generated by the software, i.e. the CHK file will make the calculation and the result will be written into these two fields. Of course, if the value of a field is calculated by EpiData Entry, it is best not to allow a data entry person to get into such a field as this could lead to inadvertent deletion of the content. You will thus learn the use of the command NOENTER for such fields.

The first of these latter two fields will get the laboratory code: We have a label block for the laboratory names and codes, where the laboratory name is the Field value and the laboratory code is the Value label. Instead of writing (as we did in the previous exercise) the Value label to the right of the field definition, we will be writing it into another field. The field:

```
labcode Code of the laboratory ____
```

will also be a text field (any string) with the Field name LABCODE and a length of 4.

The second of the NOENTER fields will receive the new unique identifier, consisting of a combination of the laboratory code (length 4), the year of registration (length 4), and the laboratory serial number (length 4). This would give a length of 12, but for better visual display, we will add a hyphen between the three elements, thus 2 additional characters. The final length of this field with Field name IDCODE will thus be a string of 14 characters:

```
idcode Unique identifier _______________
```

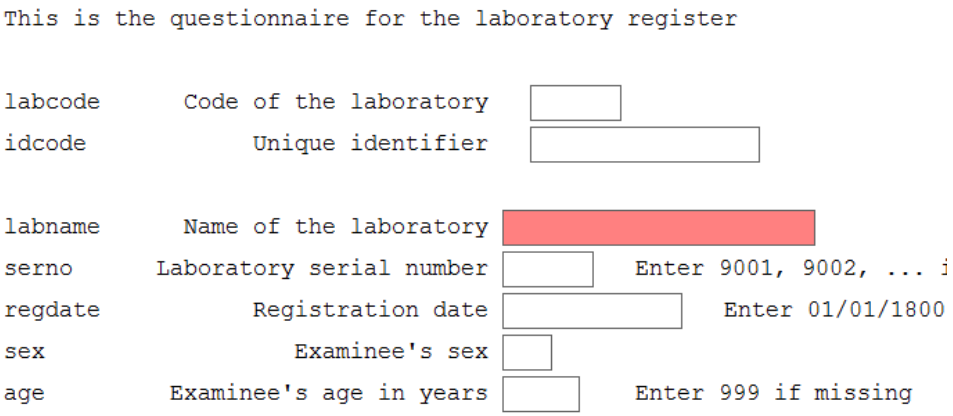
It doesn't really matter where NOENTER fields are placed, but for the sake of tidiness we prefer to place them separately from the MUSTENTER fields, for instance to the top of the questionnaire file:

```
This is the questionnaire for the laboratory register

labcode      Code of the laboratory   ____
idcode            Unique identifier   _______________

labname      Name of the laboratory _____________________
serno      Laboratory serial number ####   Enter 9001, 9002, ...
```

or, shown in the Data form preview:



Once done, save and make a new data (A_EX03.REC) file. *Note this important step of updating the* REC *file once you have made changes in the* QES *file.*

Editing the A_EX03.CHK file

Using "3. Checks" allows making all field-specific checks and it has the advantage that when you edit the checks they are checked for grammar when you "Accept and close". We will thus start with that, conscious that we cannot do all here. We will later need to exit this mode and make additional changes in the Check file as a whole. Finally, we then return again to here.

Thus open the Check file for A_EX03.REC and make the fields LABCODE and IDCODE NOENTER fields but the field LABNAME a MUSTENTER field. The command NOENTER cannot be picked from the list, it must be entered manually using "Edit".

The field IDCODE will newly become the unique identifier, thus edit the field to expand to:

```
idcode
  KEY UNIQUE
  NOENTER
END
```

EpiData will automatically add "1" to become "KEY UNIQUE 1" as we have already removed it from SERNO.

Similarly make the field LABCODE also NOENTER. It will look like this.

```
labcode
  NOENTER
END
```

We will make a label block for the field LABNAME, but with a single entry as follows:

```
LABEL Label_labname
  Awuna "ML_J"
END
```

"Awuna" is thus the Field value and "ML_J" the Value label. This differs from what we did before when the value was always an Integer. In fact, EpiData allows the value to be any of the following, an extraordinarily powerful feature of the software:

→ An integer
→ A float

→ A text

As before, we Edit the field but modify slightly. If doing as before, we would write:

```
labname
  COMMENT LEGAL USE label_labname SHOW
  TYPE COMMENT
  MUSTENTER
END
```

to show the label block as a pop-up box and to type the Value label to the right of the field. This time we do not want to type the Value label to the right of the field but to write the Value label (i.e. the laboratory code) into the field LABCODE that we made a NOENTER field. To this end, we expand the above and very simply add the name of the field after TYPE COMMENT:

```
labname
  COMMENT LEGAL USE label_labname SHOW
  TYPE COMMENT labcode
  MUSTENTER
END
```

After this, you have to write a command to generate (calculate) the IDCODE. For this we will use the AFTER ENTRY command. First, remember that we can generate IDCODE only after all the component variables have been entered. So, since the last of the component fields to be entered is REGDATE, we will write and AFTER ENTRY command for this variable.

```
regdate
  RANGE 01/01/2000 31/12/2005
  LEGAL
    01/01/1800
  END
  MUSTENTER
  AFTER ENTRY
    idcode=labcode+"-"+YEAR(regdate)+"-"+serno
  END
END
```

Note here that we have been able to extract the year out of the variable REGDATE. We can extract any component from a date field:

Extract day:	`DAY(datefield)`
Extract month:	`MONTH(datefield)`
Extract year:	`YEAR(datefield)`

Note also, that the hyphen that is to separate the components should be placed into quotation marks.

Is it possible to bypass the fields though it is **MUSTENTER**?

Unfortunately, the answer is yes!

The fact is that you have to enter the field to which any checks are to be applied. Once you enter the field, it will be difficult to exit without entering a value. But if you by-pass the fields using the mouse, then it is possible to skip the fields and leave them blank. If a field is blank, it is always a problem especially if this happens to be your unique identifier variable.

Is there a way to ensure that certain key fields cannot remain blank? Is there a way to circumvent this problem and ensure that no record can be saved without the unique identifier?

Fortunately, the answer is yes again!

However, to achieve this, we will need to manipulate the CHK file in a totally different way than we have been doing until now. Since this is a CHK command not specific to any field but related to the record as a whole, we cannot use the earlier method of editing the CHK file (Clicking on 3. Checks) which works strictly on the field level. We will have to open the full CHK file.

How do we do it?

With "File" | "Open" (or with **CTRL+O**) and click on the A_EX03.CHK file. Let us spend some time examining this file.

At its beginning, you see that EpiData placed there all Label blocks in one large LABELBLOCK that finishes with END:

```
LABELBLOCK
  LABEL label_labname
    Awuna  ML_J
  END
  LABEL label_sex
    1  Female
    2  Male
    9  "Sex not recorded"
  END
  LABEL label_reason
    0  Diagnosis
    1  "Follow-up at 1 month"
    2  "Follow-up at 2 months"
    3  "Follow-up at 3 months"
    4  "Follow-up at 4 months"
  ....
  END
END
```

Each LABEL command inside also finishes with an END as we already saw when we wrote the Field labels.

Afterwards, the fields with their checks follow:

```
labcode
  NOENTER
END
```

Preventing that a record can be saved without an identifier

We have already used AFTER ENTRY with a field, there is also a BEFORE ENTRY that goes with a specific field. But then there are commands that are not related to a specific field:

```
BEFORE FILE
AFTER FILE
BEFORE RECORD
AFTER RECORD
```

These commands must therefore be written by opening the entire CHK file as demonstrated above. It doesn't really matter whether you but them before the fields or at the end of the record (but it will not work if you place them between fields). EpiData will rearrange things

and place them after the END command following the LABELBLOCK and before the first field. We might thus as well place them there ourselves, because then we make no placement error and see everything that is outside the field level just before the information at the field levels begins. *All the above commands are block commands and must finish with an* END.

What we had in mind was a command that would check after all information has been entered and the record is about to be saved, whether the unique identifier contains a value. If not, the entry person should be given a warning and then experience a forced return to the field that contains the first element contributing to the unique identifier. What we need is an AFTER RECORD statement:

```
AFTER RECORD
  Some commands to execute
END
```

These "Some commands …" must check whether the field IDCODE is empty. The designation for "empty" is a period (.), thus:

```
AFTER RECORD
  IF idcode=. THEN
    Some commands to execute
  ENDIF
END
```

We use the command HELP to convey the relevant message to the person entering the data. At the line end, we can expand to indicate the type of pop-up box that should appear for the data entry person with an additional command:

```
AFTER RECORD
  IF idcode=. THEN
    HELP "There is no identifier" TYPE=WARNING
  ENDIF
END
```

So far, we get the warning displayed but if the user accepts it, nothing is going to happen and the record can still be saved to disk, which is, precisely what we want to prevent. We must therefore direct the cursor to go to the very first MUSTENTER field that contributes to the idcode:

```
AFTER RECORD
  IF idcode=. THEN
    HELP "There is no identifier" TYPE=WARNING
    GOTO labname
  ENDIF
END
```

This way it can be prevented that the data entry person ever gets out of the record without an identifier. That the identifier is unique is checked in the field itself and is of no concern here.

We can use the block commands mentioned above to interact with the data entry operator. For example we can provide a welcome message when the data entry operator opens the REC file by using a command BEFORE FILE or alert the person to take back-up of data when he finishes data entry for the day and closes the REC file using AFTER FILE. Observe the couple of examples below:

```
BEFORE FILE
  HELP "Welcome! Please do not use MOUSE"
END
```

```
AFTER FILE
  HELP "Thank you! Remember to back-up your data"
END
```

Please note that all the block commands must finish with an END. Not mentioning anything with TYPE will provide the message as INFORMATION. The other options with TYPE are CONFIRMATION and ERROR.

Let us now turn to another concept.

Our identifier field IDCODE has a length 14 and is of the type LABCODE-YEAR-SERNO. However, because our serial number is an integer, its actual length might vary from 11 to 14 characters: the length of our IDCODE will not be uniform. Though, this will not affect functionality in any manner, it is cosmetically not very appealing. It is actually more than just visual: in analysis it becomes impossible to verify that each identifier is correctly coded with 14 position by looking at position 14. It will also cause issues with sorting which will no more follow sequential numbering, etc, etc.

Is it possible to write leading zeros to get a consistent length of 4 for SERNO and thus a consistent length of 14 for IDCODE?

It is possible if the field is a text field and thus can take what we call "leading zeros" like in "0023". What we thus need to do is to create a temporary string variable of length 4 into which the serial number is written, with leading zeros as appropriate, and be kept until used to construct the value for the field IDCODE.

What are these temporary variables? How to create them?

Here, we are introducing a new, powerful feature of EpiData, called "temporary variables". What are these? These variables are defined in the CHK file and not in the QES file (as we have been doing until now for all variables). Hence, they will be retained in the memory but will not become part of the actual dataset. This can be very useful, if some intermediate calculations are to be made and these should be staying in memory until they are used. Temporary variables can have a field name length of 16 (rather than the usual 10). This is of very practical utility, because we can just add then the suffix "Temp" to any "hard-wired" variable, recognize it immediately, and never have to worry that it is too long.

The creation of this temporary variable would go into a BEFORE FILE statement and be globally valid. The place for such a statement is also after the LABELBLOCK … END. The command is written as:

```
BEFORE FILE
  DEFINE sernoTemp ___ GLOBAL
END
```

Note: The optional term GLOBAL helps to inform EpiData that this can be used across the records and related databases. In this specific example, we could omit it and we could also use BEFORE RECORD instead of BEFORE FILE.

With this, we are done in the open Check file. While we could finalize everything here, it is better to do things that work at field level with "3. Checks" as the grammar is being checked there.

Editing more Checks at the field level

Open Checks (**ALT+3**) for A_EX03.REC to edit Checks at the field level. The first thing to tackle is to write the SERNO into the SERNOTEMP string field, with leading zeros where indicated. This must be done in an AFTER ENTRY statement in the field SERNO which we expand accordingly:

```
serno
  MUSTENTER
  AFTER ENTRY
    Do something
  END
END
```

EpiData executes commands sequentially line by line. A good approach is thus to assign a default value to a variable and to define subsequently and sequentially, respecting the hierarchy, the conditions for which exceptions must be made. Thus, as a default, we could assign that the temporary variable takes the same value as SERNO:

```
serno
  MUSTENTER
  AFTER ENTRY
     sernoTemp=serno
    Do something
  END
END
```

The flow of the hierarchy suggests that the next change to make is if the SERNO is less than 1,000, then if it is less than 100, and finally if it is less than 10. If it is less than 1,000 (but 100 or more), only one leading zero has to be added:

```
serno
  MUSTENTER
  AFTER ENTRY
     sernoTemp=serno
    IF serno<1000 THEN
      sernoTemp="0"+serno
    ENDIF
    IF ….
      Do the hierarchically next thing
    ENDIF
    …
  END
END
```

Note that "0" is in quotation marks because it is text, not a number. It is connected with a plus to a number but because the field is text and the first character (0) is text, this will result in a string. If the value of SERNO were 511, then the value of SERNOTEMP would be "0511". You will have an opportunity during task solving to complete this block for yourself. In order to prevent an error message, save it for the time being with only the first 'IF' statement. Also, IDCODE has to be created using sernoTemp rather than serno.

Tasks:

- ***Complete the revised*** A_EX03.CHK ***file and verify your*** QES-REC-CHK ***triplet***
- ***Test the system by trying to enter data (***4. Enter data***)***