Paper 202-28

Accelerating the Construction of Data Entry Applications in UNIX Systems for Epidemiology and Healthcare Policy Researches

R. Barishev*, A. Ziv*, G. Kotler**
* Information & Computer Unit, Gertner Institute for Epidemiology and Health Policy Research
** Executive Information Systems Ltd.

ABSTRACT

Data entry is a very important phase in the epidemiology research process. At the Computer Division at the Israel Gertner Institute for Health Policy, we must provide rapid, qualitative program development. Although there are differences among the data entry forms used in various projects, there are common requirements for computerized data entry.

The superiority of data entry applications that are based on SAS® software is the result of eliminating unnecessary data transfer, the potential for data analysis during the data entry process, and the provision of a single, cohesive tool. We have developed a user-friendly interface framework for the SAS programmer, which will accelerate the development of data entry applications using several SAS products (Base SAS, SAS/FSP®, and SAS/AF®) in the UNIX operating environment. By implementing our guidelines, the development and use of data entry applications should be simplified in several areas: screen design, sophisticated menus, dynamic format and range updates, and control data entry via arrays, key controlling and navigation between existing and new records.

The paper is aimed for SAS programmers with basic knowledge in SAS/AF® and SAS/FSP®, who is going to implement efficient data entry systems.

INTRODUCTION

The Gertner Institute for Epidemiology and Health Policy Research is a national research agency for the study of epidemiology and healthcare policy, which aims to contribute to the development and formulation of informed health policies. Its primary goals are to conduct studies based on objectives defined by the Ministry of Health, to design, develop and analyze databases in cooperation with other stakeholders in the healthcare system and to incorporate and assess health-related innovative processes of national significance.

The Institute employs approximately 150 members of staff including researchers, statisticians, computer programmers, systems analysts and administrative staff. It operates two divisions, which act as complementary bodies in terms of task allocation and joint collaboration. The Institute also operates three internal service units, aiming to provide comprehensive support to the other Institute units.

The Information and Computerization Unit is a lateral service unit, whose main aim is to provide centralized computer services to the Institute's staff. The Unit is staffed by a team of system analysts and programmers engaged in the following primary areas of activity: technological infrastructure, data storage & data processing and special projects.

The computer infrastructure in our Institute is based on performance and price, taking into consideration the analysis of national files (with millions of records and numerous variables), and various users' requirements, capability and knowledge:

- PC's for every user tailored to accommodate the requirements of his tasks, be it statistical programs, word-processing, internet access, data handling and storage etc;
- NT-server and exchange server to supply Internet, Intranet, e-mail, general security, etc
- SUN server for data processing and multi-user access to common and personal data, using SOLARIS 8 and SAS 8.2 version both for data managing and processing. X-Windows emulator based on EXCEED X-server (Fig 1) provides access to the SUN server from the PC.

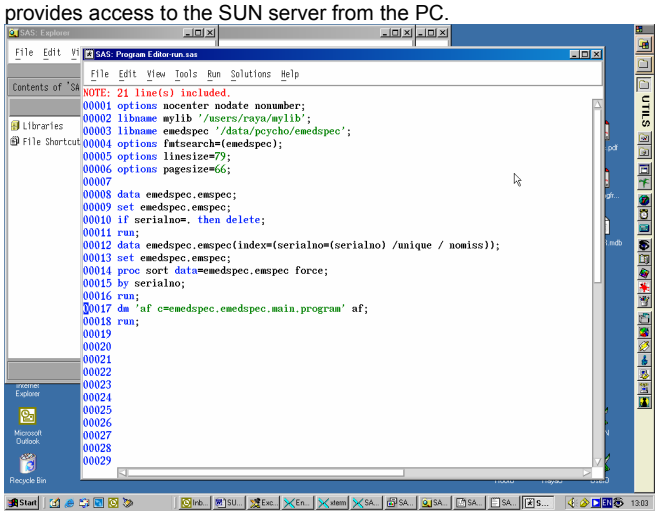

Fig1. SAS windows using X-WINDOW emulation

The proposed recommendations concern the following areas:

- screen design - building a user-friendly screen as simple and as rapidly as possible;
- creation of common templates for the main screen and menus;
- advanced data dictionary managing;
- using arrays for building shorter and more effective code;
- program driven screen navigation considering Hebrew (right-to left) language;
- full function key control;
- navigation between forms and records;

SCREEN DESIGN

As a rule, the data is gathered by questionnaires, so it is preferred that the data entry form will look similar. The questionnaire is usually prepared as a word document. When writing the data building program we copy the questionnaire from the word document to the SAS program and use the questions as labels. For data entry forms building we use the procedure FSEDIT:

```
PROC FSEDIT LABEL DATA = <LIBRARY>.<DATA NAME>
SCREEN = <LIBRARY>.<CATALOG>.<DATA NAME> MOD;
```

So the screen draft is built automatically and only enhancement for a more esthetical view is required.

COMMON TEMPLATES FOR THE MAIN SCREEN AND MENUS

Instead of using SAS/AF® FRAME entries which drastically slows down the data entry application performance and complicates the application development, we decided to limit ourselves to the SAS/AF® program entries. The MAIN screen is built on the basis of a template, which is a SAS/AF® program that contains a simple screen with a menu and a source code for the menu handling. (Fig.2).

The application navigation is fully menu driven. The main parts of the menu are data entry (Fig. 3) and dictionary tables update (Fig. 4).

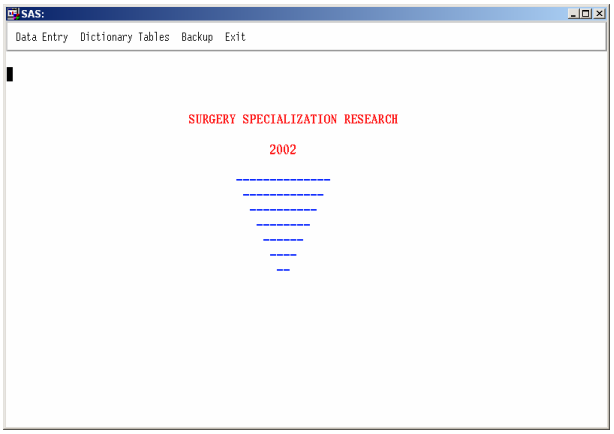

Fig.2 Template of the main screen

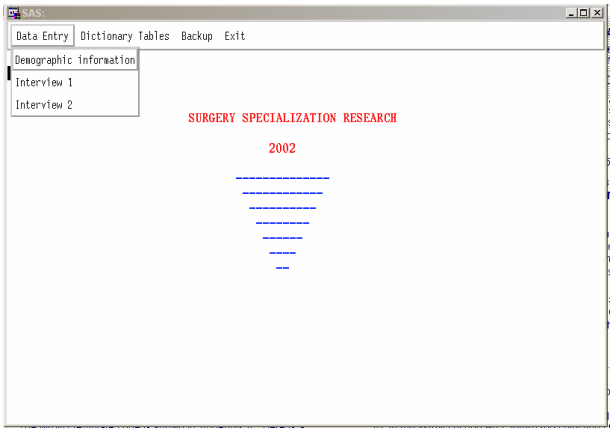

Fig 3 Data entry submenu

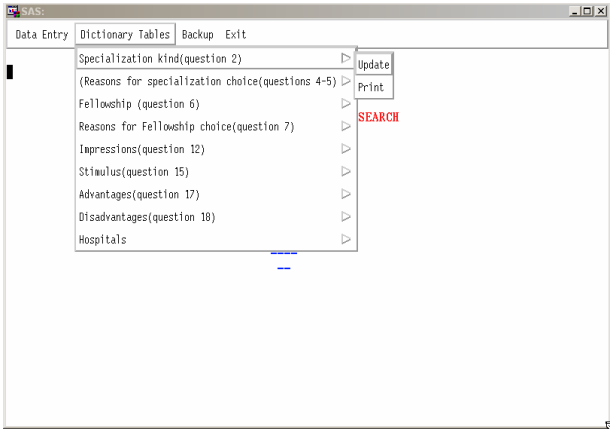

Fig 4 Dictionary tables update submenu

The menu template code is shown in Appendix A. Here is a small fragment of the program:

```
menu m1;
 item 'update' selection=a11;
 item 'print' selection=a12;
 selection a11 '11';
 selection a12 '12';
```

All the menu selections aren't SAS command strings, but flags handled by the main AF program. Actually this is its major job. Using the WORD SCL function, the program identifies the last issued command and selects the further processing accordingly.

MAIN:

```
/* checking the first word of the issued
command which may be produced by the menu
selection */
      cmd=word(1,'u');
       select(cmd);
        when('11')  do;
/* program processing of the menu selection*/
         . . .
        end;
        when('12')  do;
         . . .
        end;
```

This method enables using the menu in a wider and more flexible way and executing a program fragment instead of one command.

ADVANCED DATA DICTIONARY MANAGING

Data dictionary update during the data entry process is a rather complicated procedure, which requires programmer involvement to revise the tables, formats and value ranges and delays the data entry process. In order to increase the efficiency by saving time of the programmer and the typist, reducing the risk of errors and accelerating the data entry process, we suggest advanced data dictionary managing comprising of:

- dynamic update of data dictionary tables during data entry, followed by automatic revision of formats and ranges ;
- wide use of different selection lists types.

DYNAMIC UPDATE OF DATA DICTIONARY TABLES, FOLLOWED BY AUTOMATIC REVISION OF FORMATS AND RANGES

Dictionary tables are a very important part of data entry processing. They are used for information, data selection, value range control and data presentation. Since the formats are built on the basis of the dictionary tables, we can achieve full identity between the data presentation in the dictionary and on the screen. The data dictionary update includes the following steps:

- choosing the required table through the menu;
- preparing the parameters for the chosen table (data set name, formula entry name, format name and format label);
- updating the table using formula entry, which is the easiest way to update a table in a table view mode (Fig 5);
- revising the format using the updated table.

The entire process is performed using one subroutine.

An example of a data dictionary element update is presented below. When the user chooses to update the table SPECIALIZATION (Fig.4), the menu selection is '11'(Appendix A). In the MAIN section we capture the selection string ('11'), prepare the parameters for the subroutine MEDIT: DB (data set name), FL (formula entry name), FMTN (format name) and NAMEC (format label) and submit the subroutine:

```
MAIN:
/* checking the first word of the issued
command */
 cmd=word(1,'u');
 select(cmd);
/*update the reference table specialization*/
  when('11') do;
   /* data set name */
    db='<library>.specializations';
    /* formula */
    fl='<library>.<catalog>.specs.formula;
   /* format name */
```

```
      fmtn='spec';
      /* format label */
      namec='specialization';
      link MEDIT;
     end;
```

The subroutine MEDIT calls FSVIEW program for updating the table, builds a temporary data set FM and rebuilds the format.

```
MEDIT:
 /* updating the reference table */
 call fsview(db,'edit',fl);
 submit continue;
 /* assigning the record number to the
variable code */
data &db;
 set &db;
 code=_n;_
run;
/*creating input control data set for format
building */
data fm;
 set &db;
 start=_n;_
 label=&namec;
 fmtname=&fmtn;
run;
/* building formats from input control data
set fm*/
 proc format
 library=<library>
 cntlin=fm;
```

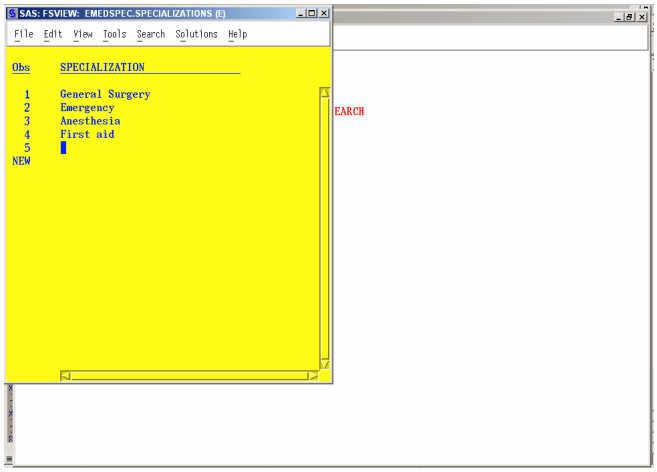

Fig 5 Dictionary table update using FORMULA entry

SELECTION LISTS

Using selection lists reduces the risk of data entry errors and assists the operator to find the required value presented in a natural way (the format value instead of the real value). The kind of selection depends on the dictionary table length. Usually, if the dictionary table is not very long, a single or multiple (for repeated values) selection is used. In long tables we search with the aid of a substring. Single and multiple selections are trivial and don't necessitate special explanations. Below we show an example of data seeking using a substring in the table of medicines. The typist can enter '0' to open the entire table, or the leading substring of the medicine name to choose only the medicines with the required leading substring:

```
MEDICINE:
 if length(medicine)>=1  then do;
  if medicine='0' then do;
   medicine=datalistc(medicinen,'code
   name',' choose the medicine:');
  end;
  else do;
   submit continue;
   /* creating a temporary data set with an
   additional variable POS which shows the
   position of the required substring in the
   medicine name */
    data _a_;
     set emedspec.medicine;
     pos=index(name,'&medicine');
    run;
   /*selecting the records with the required
   substring into a temporary data set */
    data _a_;
     set _a_;
     where pos=1;
    run;
   endsubmit;
   dsid=open('_a_');
   obs=attrn(dsid,'ANY');
   if obs>0 then do;
    call wregion(4,5,30,70);
    medicine=datalistc(dsid,'code name',' choose
    the medicine:');
   end;
   else do;
    medicine='';
   end;
   rc=close(dsid);
   rc=delete(_a_);
  end;
 end;
return;
```

USING ARRAYS FOR BUILDING SHORTER AND MORE EFFECTIVE CODE

Basically we use arrays for three purposes:

- navigation between pages on pressing PgUp, PgDown;
- processing similar fields;
- screen navigation .

NAVIGATION BETWEEN PAGES

The cursor navigation in our applications is screen sensitive. It is affected by current field and typed data. For example, if we have fields 'smoking', 'year smoking', 'cigarettes per day', 'next field' and the cursor is positioned on the field 'smoking' it's behavior after pressing any key depends on the value of the field. If the value is 'NO', the cursor will skip the next two fields, if the value is "YES", the cursor will go to the next field ('year smoking'). The code fragment looks like this:

```
MAIN:
 …
    if curfld()='smoke' then do;
     if smoke='YES' then cursor year_smoking;
     else cursor next_field';
    end;
```

But on pressing PGDown or PgUp the cursor must be positioned on the first field of the next (previous) page regardless of its current position and the field value. To implement this we perform two steps:

- define pages as character arrays, for example:

```
array page1{18} $ 16
      ('serialno','case_control','Q_1','Q-11,…);
array page2{18} $ 16 ('Q_2','Q-21,…);
array page3{18} $ 16 ('Q_3','Q-31,…);
```

– check the last pressed function key using the function LASTKEY() and if it was PgDown or PgUp position the cursor on the first field of the next(previous) page according to the page the current field is located in:

```
if ky=55 or ky=56 then do;
if ky=56 then do;  /* PgDown */
 if curfld() in page1 then cursor Q2;
 if curfld() in page2 then cursor Q3;
 if curfld() in page3 then cursor serialno;
end;
if ky=55 then do;  /* PgUp */
 if curfld() in page2 then cursor serialno;
 if curfld() in page3 then cursor Q2;
 if curfld() in page1 then cursor Q3;
end;
end;
else do;
…
```

PROCESSING SIMILAR FIELDS

When many fields have one range of values, use the same selection list or have similar processing, it is recommended to use arrays to shorten and simplify the code and reduce the risk of programming errors. For example, if the questions Q4_1, Q4_2, Q4_3, Q4_4, Q4_5, Q5_1, Q5_2, and Q5_3 use the same selection list for reply, we process the next steps:

```
/* define an array  */
array reason{8} q4_1 q4_2 q4_3 q4_4 q4_5 q5_1
                q5_2 q5_3;
/*open the file containing the list of reasons*/
reasonn=open('emedspec.reasons');
/* on changing of one of the fields the next
code  is executed */
Q4_1: Q4_2: Q4_3: Q4_4: Q4_5: Q5_1: Q5_2: Q5_3 :
do i=1 to dim(reason);
 if reason(i)=0 then do;
  call wregion(4,5,30,70);
  reason(i)=datalistn(reasonn,'code reason',
            'choose reason');
 end;
 else do:
   …        /*  validity check */
 end;
end;
return;
```

SCREEN NAVIGATION

In many cases a part of a screen looks like a table. On Fig 6-7 you can view such a screen fragment. Here we handle two problems:

– the navigation between fields must be conducted from right to left (Hebrew language);

– the cursor navigation depends on the entered value. The navigation is executing due to the flowchart presented on Fig 8.

To shorten the program we use the same subroutine for processing all the fields of one column.

The column 'Pregnancy year' is processed by PREGNSUB

The column 'Normal Birth' is processed by BIRTHSUB

The column 'Abortion' is processed by ABORTNSUB

Here are the program fragments that execute the flowchart:

```
/* declare arrays of fields) */
array pregn{6} pregny1 pregny2 pregny3
               pregny4 pregny5 pregny6;
array birth{6} birth1 birth2 birth3 birth4
               birth5 birth6;
array abort{6} abortion1 abortion2 abortion3
               abortion4 abortion5 abortion6;
array abortra{6} abortrs1 abortrs2 abortrs3
                 abortrs4 abortrs5 abortrs6;
/* declare  arrays of field names   */
array pregnyn{6} $ 8
      ('pregny1', 'pregny2', 'pregny3',
       'pregny4','pregny5','pregny6');
array birthn{6} $ 8
      ('birth1', 'birth2', 'birth3',
       'birth4','birth5', 'birth6');
array abortn{6} $ 9
      ('abortion1','abortion2','abortion3',
        abortion4','abortion5','abortion6');
array abortran{6} $ 10
      ('abortrs1', 'abortrs2', 'abortrs3' ,
       'abortrs4', 'abortrs5', 'abortrs6');
MAIN:
…
/* searching the field where the cursor is
   currently positioned and calling  the
   necessary subroutine */
if curfld() in pregnyn then link pregnsub;
if curfld() in birthn then link birthsub;
if curfld() in abortn then link abortsub;

PREGNSUB:
 jmp=0;
 do i=1 to dim(pregnyn)
  if pregnyn(i)=curfld() then j=i;
 end;
 do i=1 to dim(pregn);
  if i=j then do;
   if pregn(i)>0 then do;
    rc=field('cursor',birthn{j});
    jmp=1;
   end;
  end;
  end;
  if jmp=0 then cursor children;
return;

BIRTHSUB:
 jmp=0;
 do i=1 to dim(birthn);
  if birthn(i)=curfld() then j=i;
 end;
 do i=1 to dim(birth);
  if (birth(i)=1 & i=j) then do;
   abort(j)=.;
   abortra(j)=.;
   jmp=1;
  end;
 end;
 if (jmp=1 & j=6) then jmp=2;
 select (jmp);
  when(0) rc=field('cursor',abortn{j});
  when(1) rc=field('cursor',pregnyn{j+1});
  when(2) rc=field('cursor','children');
 end;
return;
```

```
ABORTSUB:
 …
return;
```

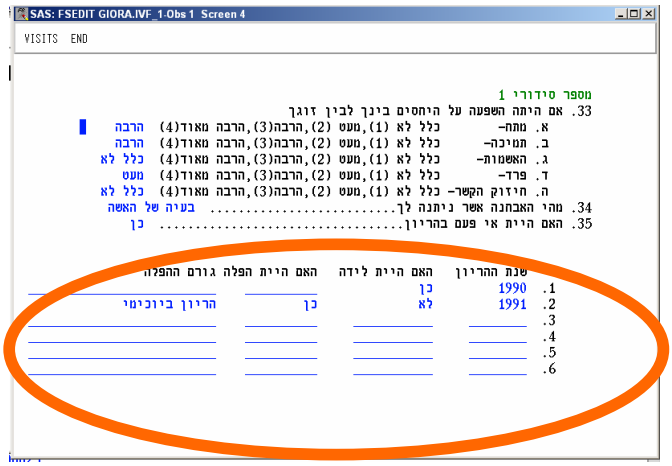

Fig 6 Table in form

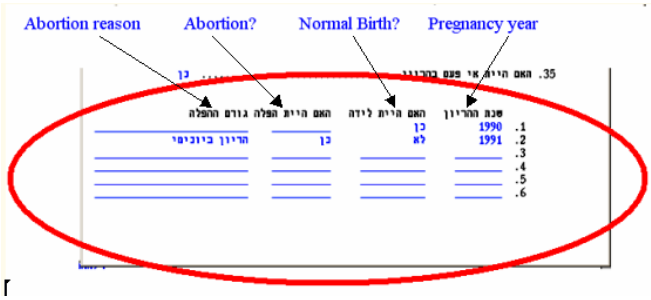

Fig 7 Table form translation

FULL FUNCTION KEYS CONTROL

To prevent additional errors on pressing a wrong function key we provide total key control, which:

- restricts the unwanted function keys using the function SETFKEY, for example , to restrict the function key F3(End) :

```
keyname = fkeyname(3);
CALL SETFKEY(keyname,' ');
```

- checks the pressed function key with the function LASTKEY and process only the function keys that are permitted. Every permitted function key is handled by the appropriate part of the program, all others are ignored.

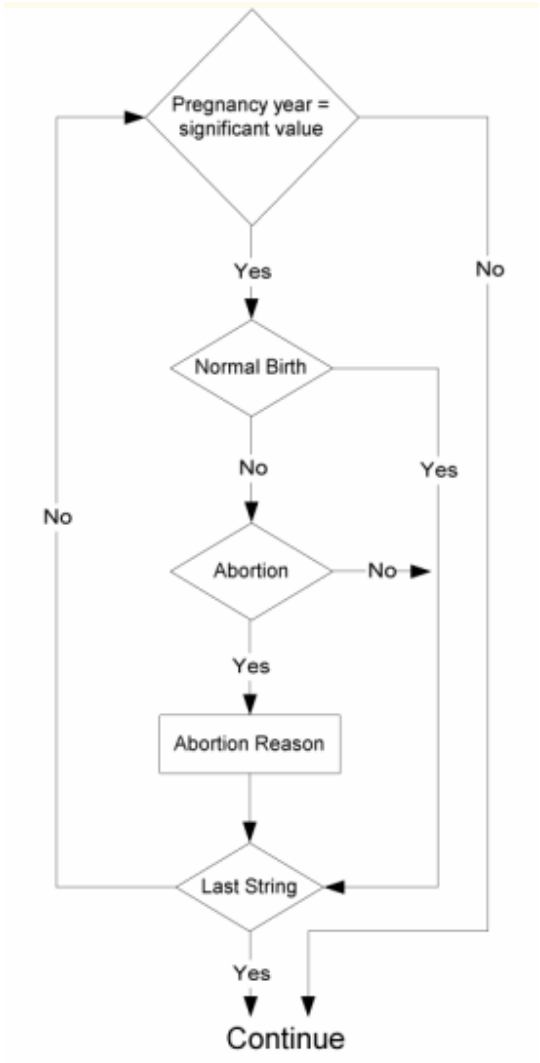

Fig 8. The table navigation flowchart

NAVIGATION BETWEEN FORMS AND RECORDS

The data entry operators usually make some common errors such as pressing the key ADD several times or entering the same serial number (or another unique identifying field) twice or more. Instead of catching and processing SAS errors we forbid the ADD command by converting the 'Allow ADD/DUP command:' property to 'N', and add an additional field for the serial number enter (Fig 9). When the operator enters the serial number, the program looks for this number in the file and if it is found - uploads the record. If not – asks the operator for confirmation using the MESSAGEBOX function:

```
commandlist=insertc(commandlist, 'you want to
            add serial number'||serialno||'are
            you sure?');
cmd=messagebox(commandlist,'!','yn','','n','`);
```

The original serial number field is locked.

The project often contains more than one file with one binding unique field (usually serial number), each with its form. To prevent errors when navigating between screens we keep the current serial number in an SCL list:

```
/* creating an scl list sn for keeping the
   serial number in the MAIN program) */
   l1=envlist('g');
   l2=insertn(l1,-1,1,'sn');
   k=getnITEMn(l1,'sn',1);
```

```
/* saving the serial number */
l2=insertn(l1,serialno,1,'sn');
/*  retrieving the saved  serial number */
l1=envlist('g');
 snn=getnITEMn(l1,'sn',1);
```

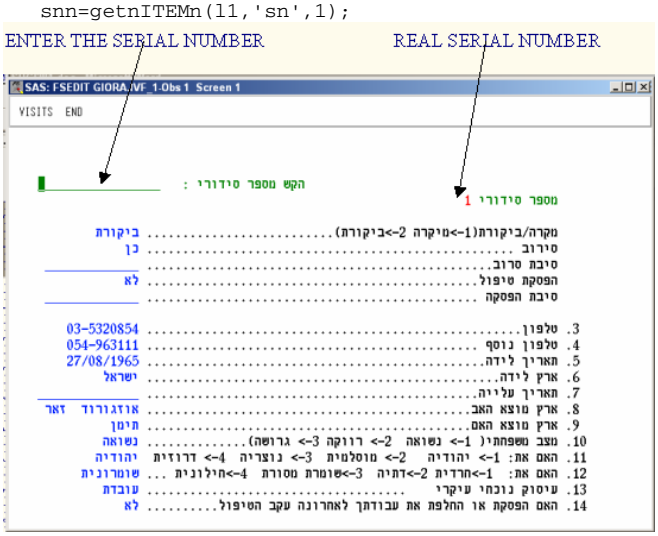

Fig 9 Additional field for serial number entry

CONCLUSION

The suggested method is widely used in the Institute for construction of data entry applications. It proved its efficiency allowing quickly construction of reliable and easy to use applications.

APPENDIX A. FRAGMENT OF A MENU BUILDING PROGRAM

```
/* the main menu */
PROC PMENU cat=emedspec.emedspec;
 MENU MAINMENU;
  ITEM 'Data Entry' MENU=quests;
  ITEM 'Dictionary Tables'  MENU=tablesu;
  ITEM 'backup'  MENU=backup;
  ITEM 'exit' SELECTION=fin;
  selection fin '999;'
/*data entry submenu */
 MENU quests;
  ITEM 'demographic information' selection= d;
  ITEM 'interview 1' SELECTION = v1;
  ITEM 'interview 2' SELECTION = v2;
   SELECTION d  '501;'
   SELECTION  v1 '502;'
   SELECTION  v2 '503;'
/* dictionary tables update submenu   */
MENU tablesu;
 ITEM 'specialization kind(question 2)'
MENU=m1;
 ITEM '(reasons for specialization
        choice(questions 4-5)' MENU=m2;
 ITEM 'fellowship (question 6)' MENU=m3;
 ITEM 'reasons for fellowship choice(question
       7)' MENU=m4;
 ITEM 'impressions(question 12)' MENU=m5;
 ITEM 'stimulus(question 15)' MENU=m6;
 ITEM 'advantages(question 17)' MENU=m7;
 ITEM 'disadvantages(question 18)' MENU=m8;
 ITEM 'hospitals' MENU=m9;

 MENU m1;
  ITEM 'update' SELECTION=a11;
  ITEM 'print' SELECTION=a12;
  SELECTION a11 '11;'
  SELECTION a12 '12;'
          .
          .
          .
 MENU m9;
  ITEM 'update' SELECTION=a91;
  ITEM 'print' SELECTION=a92;
  SELECTION a91 '91;'
  SELECTION a92 '92;'

  MENU backup;
   ITEM 'backup'  SELECTION=b1;
   SELECTION b1 '1000;'
quit;
run;
```

REFERENCES

1. SAS® Language reference: Dictionary, Version 8
2. SAS® .Procedure Guide, Version 8
3. SAS® Component Language: Reference, Version 8

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Raya Barishev
Gertner Institute for Health Policy
Ministry of Health
Sheba Medical Center
Tel Hashomer 52621, ISRAEL
Work Phone: 972 3 530 3282
Fax: 972 3 635 4389
Email: rayab@gertner.health.gov.il

TRADE MARKS