Data Checking Methods in a Data Entry System Based on SAS® Software: Exploring the Possibilities

Sandi Tennyson - Immunex Research and Development Corporation

ABSTRACT

The SAS® system offers a flexible applications development environment with a wealth of choices for the developer. With reference to our recent experience in developing a data entry system using SAS/FSP®, various data checking methods, including the SAS/FSP Special Attributes section, the use of SAS Informat, and special elements of the Screen Control Language (SCL) will be discussed in terms of how they can be used to address typical data entry systems development requirements. The data checking methods are analyzed in terms of how they handle 1) single field data checks, 2) cross-field data checks, 3) missing data, 4) the decoding or expanding of data entries, and 5) error messages, alarms, and the overriding of errors. Designing flexible systems will also be addressed, and discussion of the costs in terms of systems developers time is included.

INTRODUCTION

Immunex Corporation is a biopharmaceutical company focused on the discovery, development, manufacture and marketing of products to treat immune system disorders. In the Biometrics department, which is responsible for computing and biostatistical support for all clinical trials, were faced with the task of developing a proprietary clinical data management system, and we chose to use the SAS System. In developing this system we found that it can be challenging for a developer to sort through the myriad of choices available to perform the data checking tasks, alone. It was through trial and error that we determined what worked best for various situations. This paper will share some of what we learned, describing the methods available for data checking in a SAS/FSP-based data entry system in terms of how they meet specific data entry systems requirements.

DATA ENTRY SYSTEMS DESIGN REQUIREMENTS

The data entry portion of our system had to address the requirements typical of any data entry system, some of which are:

- Performing single value, single range, multiple value, or multiple range checks on one field.
- Performing 'relational' value checks on two or more fields; i.e., checking that the values of two or more variables are valid in combination with each other.
- Handling blank, missing or 'unavailable' data.
- 'Decoding' entries to save keystrokes.
- Notifying the operator of errors and invalid entries, and allowing or prohibiting entries which are in error.

Other systems design requirements that can affect which type of data checking method is used are:

- Flexibility of the system, i.e., how well do the entry screens adapt to changing requirements or other applications?
- Availability of systems development time.

The remainder of this paper will be a discussion of some of the data checking methods available under SAS/FSP with respect to how they meet the data entry system design requirements listed above.

DATA CHECKING METHODS

Three methods will be discussed: 1) the Special Attributes method, 2) the use of user-defined informat, and 3) Screen Control Language (SCL), including data set functions and the use of macro variables within SCL.

Data Checking Methods in a Data Entry System Based on SAS® Software: Exploring the Possibilities

Sandi Tennyson - Immunex Research and Development Corporation

ABSTRACT

The SAS® system offers a flexible applications development environment with a wealth of choices for the developer. With reference to our recent experience in developing a data entry system using SAS/FSP®, various data checking methods, including the SAS/FSP Special Attributes section, the use of SAS Informat, and special elements of the Screen Control Language (SCL) will be discussed in terms of how they can be used to address typical data entry systems development requirements. The data checking methods are analyzed in terms of how they handle 1) single field data checks, 2) cross-field data checks, 3) missing data, 4) the decoding or expanding of data entries, and 5) error messages, alarms, and the overriding of errors. Designing flexible systems will also be addressed, and discussion of the costs in terms of systems developers time is included.

INTRODUCTION

Immunex Corporation is a biopharmaceutical company focused on the discovery, development, manufacture and marketing of products to treat immune system disorders. In the Biometrics department, which is responsible for computing and biostatistical support for all clinical trials, we were faced with the task of developing a proprietary clinical data management system, and we chose to use the SAS System. In developing this system we found that it can be challenging for a developer to sort through the myriad of choices available to perform the data checking tasks, alone. It was through trial and error that we determined what worked best for various situations. This paper will share some of what we learned, describing the methods available for data checking in a SAS/FSP-based data entry system in terms of how they meet specific data entry systems requirements.

DATA ENTRY SYSTEMS DESIGN REQUIREMENTS

The data entry portion of our system had to address the requirements typical of any data entry system, some of which are:

- Performing single value, single range, multiple value, or multiple range checks on one field.
- Performing 'relational' value checks on two or more fields; i.e., checking that the values of two or more variables are valid in combination with each other.
- Handling blank, missing or 'unavailable' data.
- 'Decoding' entries to save keystrokes.
- Notifying the operator of errors and invalid entries, and allowing or prohibiting entries which are in error.

Other systems design requirements that can affect which type of data checking method is used are:

- Flexibility of the system, i.e., how well do the entry screens adapt to changing requirements or other applications?
- Availability of systems development time.

The remainder of this paper will be a discussion of some of the data checking methods available under SAS/FSP with respect to how they meet the data entry system design requirements listed above.

DATA CHECKING METHODS

Three methods will be discussed: 1) the Special Attributes method, 2) the use of user-defined informat, and 3) Screen Control Language (SCL), including data set functions and the use of macro variables within SCL.

Combinations of these methods will also be addressed. Generally speaking, the Special Attributes method and the informats method offer a lot of functionality at a low cost in development time, but these first two methods have their limitations. The SCL methods provide much more flexibility, but involve a larger investment on the part of the developer.

SPECIAL ATTRIBUTES

When you select 'Modify' under FSEDIT, menu item 4 allows you to change the 'special attributes' of each field on the screen. Along with changing the color, protection and padding, and other attributes of each field, you can set 'MAXIMUM' and 'MINIMUM' allowable values, as well. Using this method of data checking has the distinct advantage of being simple, fast and easy to develop. Error messages and alarms are automatically generated by the SAS system when invalid values are entered. The developer can choose, on a screen-wide basis, whether or not the data entry operator will be allowed to use the 'override' command to enter a value which is not within the specified range.

With reference to handling blank or missing data, no special action is required on the part of the screen developer. Fields left blank will pass a basic Min-Max range check in the Special Attributes section. When the 'special missing' option of the SAS system is used (which allows a developer to equate the existence of a special alpha character to mean that a value for a numeric variable is missing or blank), special missing characters will also pass the range check. This automatic handling of blank data is very convenient for the developer, especially when compared to how blank data is handled with the other two methods.

The Special Attributes method works well for any continuous numeric range, and SAS dates are a particularly good application for this method because the Special Attributes section automatically takes any informat or format associated with the value into account. Handling SAS dates with the other methods can get tricky, as will be shown later.

There are some limitations to this method. With regard to designing flexibility into a system, since the actual values used in the check are hard-coded and stored with the screen, this method can only be used if the range will not change between applications of the screen, or if there is no concern for having a screen he flexible enough for multiple purposes. However, considering that the range of valid values for a variable may change during data entry within an application, the change is easy to make if the Special Attributes method is used.

Another constraint is that the Special Attributes method works for numeric or character continuous range situations, but not situations where multiple ranges or discrete values are valid. For example, we can check whether the Patient entered has a value between 1 and 50, but if patients 101-150 and 200-250 are valid, we cannot use this method to screen out, for example, patient 105, while still accepting the valid patients. Finally, the Special Attributes method cannot be used to perform relational data checks since only one field at a time is checked, nor can this method handle the decoding or expanding of values entered.

USER-DEFINED INFORMATS

In many cases in our system, we created an informat and associated it with a variable in the data set using the 'Informat' statement in the data step. When using the informat method, the data check is done automatically in FSEDIT by the SAS system. For example, the following informat was used for one study's patient number range:
If informal 'PATN' is used on the Patient variable when the data set is created, then the developer need not take any other action to handle the data check. When data is entered in the Patient variable using FSEDIT, entry of an invalid patient number will automatically generate an error message and an alarm. The "other = ERROR" statement specifies that all values other than the valid values specified should produce an error.

An interesting and valuable characteristic of informats is that they are completely independent of both the FSEDIT screen and the data set. The same FSEDIT screen and data set structure can be used for multiple applications as long as the informat is changed each time. Depending on systems design, this may provide all the flexibility needed. If a range changes, the FSEDIT screen itself is unaffected. (NOTE: if you re-run a FORMAT procedure to change the data check while a SAS/AF session is still active, the new format will NOT take affect. You must re-run the format procedure outside of your SAS/AF application.)

One limitation of the informal method is that it is not a good choice for checking SAS dates. Almost always, you will want to attach a system-supplied informat to your dates (such as 'MMDDYY6:') which will make the dates easy to enter. Therefore, you will not be able to write your own informal to check whether a valid date was entered. It is much better to use the Special Attributes section or screen control language to check SAS date values. Another limitation is that, as with the Special Attributes section, the informal method cannot be used to perform relational data checks, because only one field at a time is checked.

Note that even though you might be creating a numeric informat to be used with a numeric variable, the comparisons are done in character. The values on the left hand side of the equal sign are compared in character, whether or not you put quotes around them. For this reason, you must be careful when specifying numeric ranges. For example, the following check will NOT work:

```sql
*** INFORMAT WHICH WILL NOT WORK ***
proc format;
  invalue patn (just)
    '1' = same_
    '10' = same_
    other = error;
run;
*** INFORMAT WHICH WILL NOT WORK ***
```

When this informal procedure is submitted it will not produce an error; however, the resulting informat will not accept numbers between 1 and 10. For example, the number 2 or 9 will not be accepted because both numbers specified in the range begin with '1', and the range check is done in character. Instead, you must specify each number individually, as shown:

**INFORMAT NEEDED INSTEAD**
```sql
proc format;
  invalue patn (just)
    '11' = same_
    '12' = same_
    '13' = same_
    '14' = same_
    '15' = same_
    '16' = same_
    '17' = same_
    '18' = same_
    '19' = same_
    '20' = same_
    other = error;
run;
```

Similarly, note that the following format procedure would produce an error message indicating that the two ranges overlap:

```sql
***INFORMAT WHICH WILL NOT WORK***
proc format;
  invalue patn (just)
    '11' - '3' = same_
    '101' - '103' = same_
    other = error;
run;
***INFORMAT WHICH WILL NOT WORK***
```

Unlike with the Special Attributes method, fields left blank and special missing values will not be valid unless they are explicitly specified in the informal. If you want to make sure blank values and special missing values are valid, you must specify ' = same_ and 'Z' = same_ (substituting whatever your special missing character is for the 'Z') when building your informal.

The use of informats is especially valuable for 'decoding' or extending values entered. For example, to decode entries for a Disease variable, the following informal could be created and attached to a 'disease' variable in a data set:

```sql
proc format;
  invalue $dis (just)
    '1' = same_
    '11' = 'ACCELERATED'
    '12' = 'CHRONIC'
    '2' = 'BIRD'
    '3' = 'CAT';
run;
```

In this case, the informat is performing two different and valuable functions: 1) it checks to see if the value entered is valid, and 2) it converts the valid values entered and stores a more descriptive character string.

**Combining Methods: special considerations**

Note the following about using informats and formats and the Special Attributes method together. Let's assume you use an informal for a character variable to decode values, such as the following:

```sql
proc format;
  invalue $myfmt
    '1' = 'BIRD'
    '2' = 'DOG'
    '3' = 'CAT';
run;
```

If you also use the Special Attributes section to try to catch a range of entries between 1 and 3, you will notice in the 'MAXIMUM' and 'MINIMUM' modification screens that the number you enter is decoded immediately to the character string. In this case, if you specify 'MAX' is 3 and 'MIN' is 1, then what the SAS system interprets is 'MAX' is CAT, and 'MIN' is BIRD. Therefore, 1 and 3 will be valid entries, but 2 will not be valid because the first character, 'D' is less than the maximum specified, which is 'C'. Thus, the entry '2' for 'DOG' would be invalid.

However, if you use a format (not an informal) to extend the numbers when they are printed, such as:
and you also use the special attribute section to catch a range of entries between 1 and 3. In this case the entered value will be used for the comparison, and 1, 2, and 3 all will be valid.

As a further note, if you change the informat after you have set the 'MAX' and 'MIN' values in your screen, the special attribute settings do not automatically adjust to your change. However, when a format changes after the fact, the special attribute settings do automatically adjust to the change.

Since most SAS programmers are already familiar with using user-defined informats (or, at least user-defined formats), this method of data checking is relatively easy to develop and so provides a lot of functionality for a small cost. However, there are limits to what can be done with informats. For full-blown data checking functionality with the maximum developer control, using SCL is a must.

SCREEN CONTROL LANGUAGE

The use of SCL provides the most flexibility and functionality. With the first two methods, the good news is that the SAS system is doing much of the work, but the bad news is that certain assumptions have to be made which are not always desirable. With SCL control of the environment is completely in the hands of the developer. Of course, this added control does come without its price. Recalling the informat example given earlier for the disease variable, the same check can be accomplished in SCL but it takes much more work on the part of the developer. In this case, the developer must explicitly code the error message and the alarm, and make sure control goes back to the user after the error is identified (using the return statement):

```sas
proc format;
  value Symfmt
    '1' = 'BIRD';
    '2' = 'DOG';
    '3' = 'CAT';
run;
```

In addition to having to explicitly code the error messages and alarms, the developer also must carefully consider the 'override' capability. The difference between using SCL and the Special Attributes method or the informats method is that, with the latter two, a data check is performed only after data has been entered into the field. However, with SCL, all of the code in the MAIN section (including your data check) will be performed every time the user presses Enter, even if no new data was typed into the field. So, while the 'override' command may work once for a field, the data check for the same field may fail again if the user enters data into a different field and presses enter.

One way to prevent this situation from happening with the override command is to use the 'Modified' function in your SCL code, and only perform the data check if the field has been modified. Keep in mind that if you do make use of the modified flag, and if the user types values into multiple fields without pressing Enter after each entry, you may lose your modified flag, and your data check may be skipped. More specifically, if the user enters invalid data into multiple fields without pressing Enter after each entry, and if you return control to the user immediately after you discover the first error, you will have lost your 'modified' flag for the subsequent fields which also were in error.

As shown in the 'disease' examples above, SCL and the other methods discussed can all be used to check the values of single variables, but only with SCL can a developer perform relational edits, or checks between two or more variables. For example, in our system, sometimes the Sites will be assigned specific Patient numbers. To check whether the Site and Patient are compatible, the following SCL code could be used:

```sas
MAIN:
  other SCL code;
  errorflag = 'off';
  select(site);
    when(1)
      if (patient < 100 or Patient > 199) then
        errorflag = 'on';
    when(2)
      if (patient < 200 or Patient > 299) then
        errorflag = 'on';
    when(3)
      if (patient < 300 or Patient > 399) then
        errorflag = 'on';
    otherwise;
      if errorflag = 'on' then
        return;
        _msg_ = 'Patient and site are incompatible.';
        cursor patient;
        _error_;  
  end;
  other SCL code;
```

The developer's time investment is larger when using SCL, but sometimes the system requirements make it necessary. With reference to designing flexible screens that can be used for multiple applications and data checks, note that with the above examples, the FSEDIT screens are not independent of the data check. There are a number of ways to pull the specific data check out of the source code, making the screen more flexible, though sometimes (but not always) making the code more complex.

Of the many ways there are to perform the patient and site cross-check above, one method which is cleaner and more concise than the previous example and also makes the data check independent of the FSEDIT screen is to use a combination of a user-defined informat and SCL. For example, you might first write a user-defined informat, as shown below:

```sas
User-defined 'ptosite' informat:
  proc format;
    value ptosite (just)
      '1100' - '1199' = 1
      '2200' - '2299' = 2
      '3300' - '3399' = 3
    other = _error_;
run;
```

This informat can be used within the SCL code, as follows:
SCL using the 'ptosite' informat:
MAIN:
other SCL code;
charpat = putn(patient,'4.',);
fnbsite = inputn(charpat,'ptosite.);
if (site eq .) then
    site = fnbsite;
else;
do;
    if (site ne fnbsite) then
        erroron site patient;
cursor site;
    _msg_ = 'Site and Patient do not jive.';
    alarm;
    return;
end;
end;
else
    site = symgetn('site');
other SCL code;

The example above shows one way of making the data check independent of the FSEdit screen, by pulling the specific check out into an informat. Another way to accomplish this is by pulling the specific check into a SAS data set through the use of the data set functions in SCL. For example, let's assume the following data set was created, which contains various values and codes for different races:

RACE DATA SET CONTENTS
---Alphabetic List of Variables and Attributes---

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CODE</td>
<td>Char</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>DECODE</td>
<td>Char</td>
<td>20</td>
<td>5</td>
</tr>
</tbody>
</table>

PROC PRINT OF RACE DATA SET
The SAS System

<table>
<thead>
<tr>
<th>OBS</th>
<th>CODE</th>
<th>DECODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>Asian</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>Black</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>Caucasian</td>
</tr>
<tr>
<td>4</td>
<td>H</td>
<td>Hispanic</td>
</tr>
<tr>
<td>5</td>
<td>N</td>
<td>Native American</td>
</tr>
<tr>
<td>6</td>
<td>O</td>
<td>Other</td>
</tr>
</tbody>
</table>

Once this data set is defined, it can be accessed from within an SCL program to see whether a valid race code was entered. While the program becomes more complex, it does provide complete data independence from the FSEdit screen. If a new race code is added to the application, the only change required is to add an observation to the RACE data set. In addition, this technique serves the dual purpose of not only performing the data check, but also 'decoding' or 'extending' the code entered into a more descriptive value (the race itself, not just the one-character code.) A sample piece of SCL code that will perform this check is shown below:

SCL using some data set functions:
FSEINIT:
    dsidopen('glbdat.race','i.');
    cvnum=varnum(dsid,'code');
    dvnum=varnum(dsid,'decode');
other SCL code;
return;
INIT:
other SCL code;
return;
MAIN:
other SCL code;
/************************************************/
/* See if RACE entered is valid, translate if so */
/************************************************/
if modified(racecode) then
    do;
        if (locatec(dsid,cvnum,racecode,'u','a') = 0) then
            do;
                erroron racecode;
            _msg_ = 'The race entered is invalid.';
            alarm;
            return;
        end;
    end;
else;
    do;
        race = getvarc(dsid,dvnum);
    end;
end;

Another feature of SCL that is very powerful and also can help to provide data independence is the capability of using macros and macro variables within SCL. You can use macro variables within your SCL, and set the value of these variables outside of FSEdit altogether. It is important to keep in mind, however, that the macros themselves and references to macro variables are evaluated at compile time not at execution time. You may be able to make use of the following SCL code even though the macro variable is evaluated at compile time:

MAIN:
other SCL code;
if screenvalue < &minvalue then
    do whatever;
other SCL code;
*NOTE: Macro value is evaluated at compile time

This does not provide data independence, though, since the specific value is evaluated when the SCL is compiled, and the value is stored with the compiled source. To make your screen independent of the value of the macro variable, you need to use the symget and symput SCL functions, as shown in the following example:

MAIN:
other SCL code;
minvalue = call symget('minvalue');
if screenvalue < minvalue then
    do whatever;
other SCL code;
* NOTE: Macro value is evaluated at execution time
In this example, the minimum value will be fetched from macro variable 'minvalue' and stored into the SCL variable 'minvalue' at the time the screen is being used (at execution time).

It is sometimes desirable to have the value of the macro variable 10 be evaluated at SCL compile time. For example, in our system we have range-checking macros similar to the one shown below which can be included anywhere in the SCL code of our FSEDIT screens. In this case, the macro variables that are referenced are the FSEDIT screen field name, and the actual minimum and maximum allowable values. We want these to be evaluated at the time the screen is compiled:

```
SCL macro residing outside of FSEDIT screen
%macro numrang(field,min,max);
if modified(&field,'l.') then do;
  xchar = putn(&field,'1.');
  if (xchar ne '1') and (&field ne .) then do;
    if (&field lt &min) or (&field gt &max) then do;
      erroron &field;
      alarm;
      _msg_ "field &field is out of range.
    Range is &min to &max.");
  end;
end;
end;
%Mend numrang;
SCL code using the macro
MAIN:
  other SCL code;
%numrang(patient,101,199);
other SCL code;
```

However, there are other times when we need to use the symget and symput functions and delay the evaluation of the macro variable to the time that the FSEDIT screen is being used. We have macros in our system that check to see whether date values are in range, but the valid date ranges change from study to study. In this case, we use the symget and symput SCL functions, as follows:

```
SCL macro residing outside of FSEDIT screen
%macro chkdate(field);
numin = symgetn('minmindate l');
nummax = symgetn('maxmindate l');
if modified(&field) and (&field ne .) then do;
  if (&field < nummin or &field > nummax) then do;
    erroron &field;
    alarm;
    _msg_ "field &field is invalid.
  "Range is &minmindate to &maxmindate.
  end;
end;
%Mend chkdate;
SCL code using the macro
MAIN:
  other SCL code;
%chkdate(idate);
other SCL code;
```

Note that, as in the above examples, care must be taken to make sure the date field was modified, and also check for blank values. Since the developer has complete control over the environment, SAS can no longer assume that blank or missing values are valid. The developer must explicitly specify this in the SCL code.

**SUMMARY**

There are many different ways to perform data checking within an FSEDIT application. Each method has its strong and weak points, and it is up to the applications developer to choose the optimal method, or combination of methods, for each situation. In an attempt to provide a quick reference to aid in making these decisions, the following chart summarizes the points made in this paper.

<table>
<thead>
<tr>
<th>Special Attributes</th>
<th>User-Defined Informs</th>
<th>Screen Control</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Field Checks</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Cross-Field Checks</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Blank or Missing Data</td>
<td>Automatic</td>
<td>Must be explicitly checked</td>
<td>Must be explicitly coded</td>
</tr>
<tr>
<td>Decoding Entries</td>
<td>Not Possible</td>
<td>YES Good choice</td>
<td>Can do, but more work</td>
</tr>
<tr>
<td>Error Messages and Alarms</td>
<td>Automatic</td>
<td>Automatic</td>
<td>Must be explicitly coded</td>
</tr>
<tr>
<td>Designing Flexible Screens</td>
<td>No</td>
<td>Data Indep. from screen</td>
<td>Data Indep. can be achieved in many ways</td>
</tr>
<tr>
<td>System Developers Time</td>
<td>Minimal</td>
<td>Medium</td>
<td>Maximum</td>
</tr>
</tbody>
</table>

Please feel free to contact the author with questions or comments:

Sandi Tennyson
Immunex Corporation
51 University Street
Seattle, WA 98101
206-587-0430

SAS, SAS/FSF, and SAS/AF are registered trademarks of SAS Institute, Inc.