Programming Guidelines for Good Data Documentation

Sue Ellen Hansen, University of Michigan

Abstract

Prior International Blaise Users Conference (IBUC) meetings have had sessions and papers on screen design standards (e.g., Couper 2000, Gatward 2003, Hansen and Dinkelmann 2003, Kuusela 2003, and Wensing 2003). Past meetings have also had papers on programming approaches and standards (e.g., Soper and Dyer 2001 and Altvater, Stanford & Ziesing 2001). Generally, however, programming and interface standards have been defined separately from attempts to design data structures and to produce data documentation. Recent efforts at the University of Michigan and elsewhere suggest that it is time to revisit Blaise programming standards, with a view toward providing good (complete and readable) data documentation.

The University of Michigan (Michigan) has created a tool for producing XML-based codebooks and questionnaires (Sparks and Youhong 2004), which may be printed or viewed as web pages. In the course of the development, the system was tested on several complex instruments. The results varied in the "usability" of the documentation produced.

This paper describes the problems encountered, identifies specific programming styles or approaches that hindered adequate documentation, and proposes programming guidelines for future instruments that would ensure well-documented Blaise instruments and datasets.

1.0 Introduction

Couper (2000) suggests that there are three aspects to computer assisted interview (CAI) design that need to be balanced. These may be viewed perhaps as three legs of a development stool: (1) programming, (2) user interface, and (3) data output and documentation. An emphasis on one aspect can lead to neglect of at least one of the other aspects, and thus create an imbalance. The requirements of one aspect also may be in conflict with another. For example, it may take twice as long to program a Blaise instrument for good user interface than for poor user interface. Finally, those who are most knowledgeable about one aspect of design may be ill informed about other aspects, and may be unable to achieve a good balance when addressing programming, interface design, and data output and documentation concerns.

Many organizations have developed programming and screen design guidelines, but few have addressed the impact of programming and design decisions on data output and documentation. Michigan developed a preliminary set of CAI instrument specification guidelines, which to date have been used for programming a handful of Blaise instruments, and have resulted in reduced programming costs. The recent development of an automated Blaise Documentation System (see Sparks and Youhong 2004) has forced us to revisit these specifications, as well as programming and screen design guidelines, and to modify them so that instrument specifications lead to better documented survey datasets. The goal is to have three sets of guidelines for development of CAI instruments, all of which reflect a balance among programming, screen design, and data output and documentation:

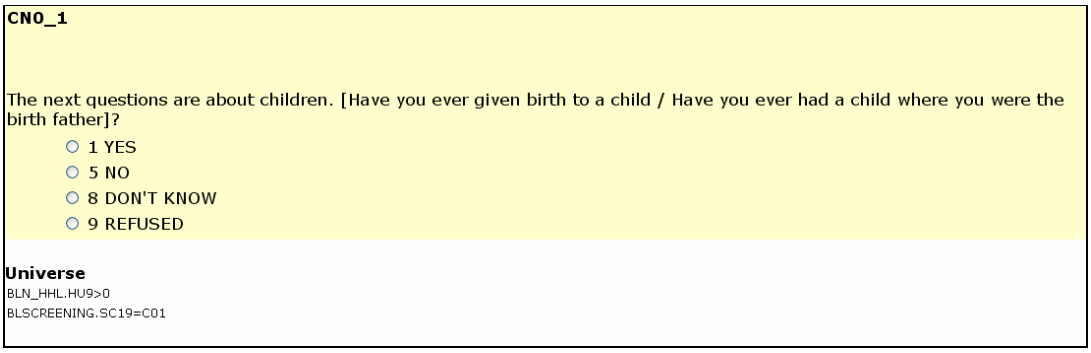
- CAI Instrument Specification Guidelines
- CAI Screen Design Guidelines
- CAI Programming Guidelines

The remainder of this paper (1) describes the components of basic data documentation; (2) discusses aspects of instrument specification, screen design, and programming that need to be considered to generate good data documentation; and (3) and concludes with a summary of preliminary specification and programming guidelines.

2.0 Basic Data Documentation

Figures 1 and 2 show Blaise variables as documented by the Blaise Documentation System, the first as part of questionnaire documentation, the latter for a codebook with frequencies[1].

Figure 1. Basic Questionnaire Documentation



The key elements for questionnaire documentation are:

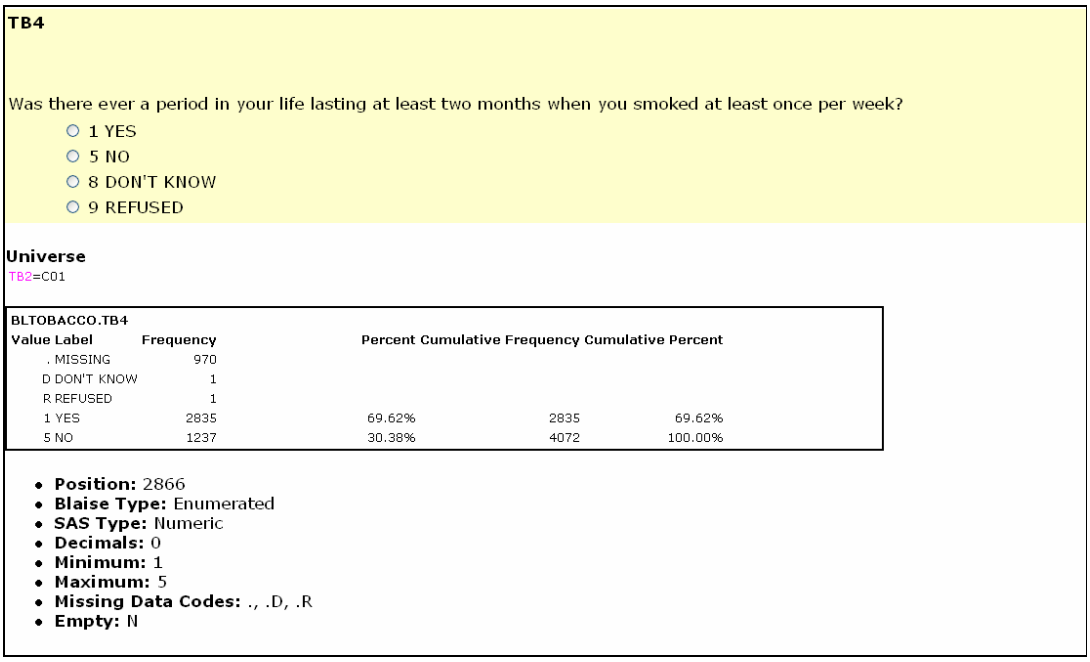
- Question identifier
- Question
- Response options
- Universe

For codebook documentation, the additional elements are:

- Frequencies and Percents
- Variable characteristics (in this case for SAS)
  - Dataset position
  - Blaise variable type
  - SAS variable type
  - Number of decimals
  - Minimum value
  - Maximum value
  - Missing data codes
  - Whether an empty value is allowed (Blaise attribute EMPTY)

[1] These examples represent documentation generated from programming that deviates in some respects from Michigan guidelines.

Figure 2. Basic Codebook Documentation



TB4

Was there ever a period in your life lasting at least two months when you smoked at least once per week?

- 1 YES
- 5 NO
- 8 DON'T KNOW
- 9 REFUSED

Universe

TB2=C01

BLTOBACCO.TB4

Value Label	Frequency	Percent	Cumulative Frequency	Cumulative Percent
. MISSING	970			
D DON'T KNOW	1			
R REFUSED	1			
1 YES	2835	69.62%	2835	69.62%
5 NO	1237	30.38%	4072	100.00%

- **Position:** 2866
- **Blaise Type:** Enumerated
- **SAS Type:** Numeric
- **Decimals:** 0
- **Minimum:** 1
- **Maximum:** 5
- **Missing Data Codes:** ., .D, .R
- **Empty:** N

3.0 Instrument Specification

Pierzchala and Farrant (2000) make a strong case for providing non-programmers with enough of an understanding of Blaise to provide complete specifications for instruments. What is missed in the specifications will be left up to the programmer to decide on how to implement, with a probable impact (not necessarily positive) on the Blaise interface and data output and documentation.

In the CAI instrument specification guidelines under development at Michigan, designers are asked to provide detailed specifications about each question or variable (Blaise field) they expect to have in the final dataset. Each variable described in the specification for an instrument should have following information:

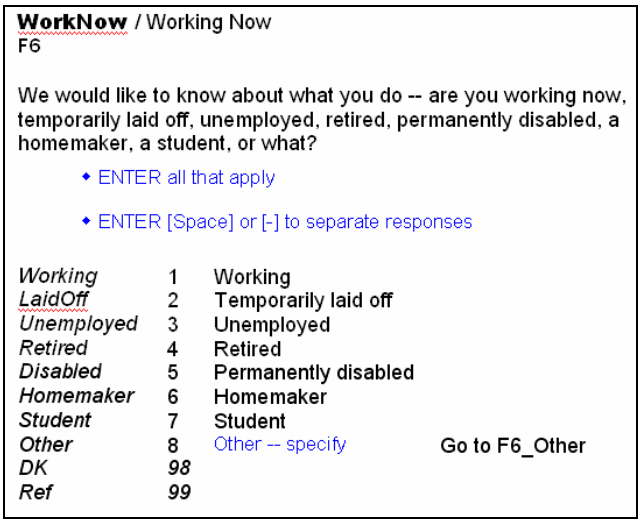
- Field Name,
- Question text / Interviewer checkpoint text,
- Field description,
- Skip [Go to] instructions, and
- Answer Categories/Data Type
  or
- Valid numeric range

Other components of a field would be specified as necessary:

- Field Tag
- Logic for Fills
- Interviewer instruction text, including text of probes
- Entry masks
- Soft Consistency Check (SIGNAL)
- Hard Consistency Check (CHECK)
- Field level DK, RF, EMPTY attribute

Figure 3 gives an example of basic specifications for a variable, "Working Now." In this example, the field name is "WorkNow," the descriptor "Working Now," and the tag "F6."

Figure 3. Basic Specifications for the Variable "Working Now"



WorkNow / Working Now
F6

We would like to know about what you do -- are you working now, temporarily laid off, unemployed, retired, permanently disabled, a homemaker, a student, or what?

- ENTER all that apply
- ENTER [Space] or [-] to separate responses

Working	1	Working	
LaidOff	2	Temporarily laid off	
Unemployed	3	Unemployed	
Retired	4	Retired	
Disabled	5	Permanently disabled	
Homemaker	6	Homemaker	
Student	7	Student	
Other	8	Other -- specify	Go to F6_Other
DK	*98*		
Ref	*99*		

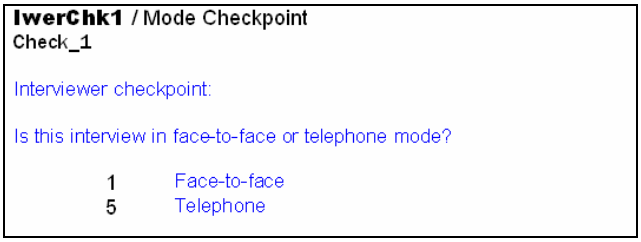
The aspect of this specification relevant to data documentation is variable identification. Michigan recommends mixed case and meaningful field names and descriptors, and that all three identifiers including the tag are specified. This allows the use of any or all identifiers in the documentation, and for more flexibility in screen design (use of the tag in screen design is discussed in the next section)[2].

3.1 Preload variables, checkpoints, and constructed variables

Specifications should contain information on all variables expected to be in the final survey dataset. These include data that are to be preloaded into the Blaise data record at the beginning of the interview, explicit interviewer checkpoints, implicit internal checkpoints, and any constructed variables that analysts want in the final dataset (including randomly generated numbers). If care is taken to put such variable information in the original instrument specifications, programmers will know to declare such variables as fields and not auxiliary fields. This ensures that they are exported with the survey data and that they are appropriately documented.

Figure 4 provides an example of an explicit interviewer checkpoint specification. In this case question text is in blue indicating it is a question to be answered by the interviewer, with nothing read to the respondent. Figure 5 shows the associated Blaise screen.

Figure 4. Specification for Interviewer Checkpoint "Mode Checkpoint"



IwerChk1 / Mode Checkpoint
Check_1

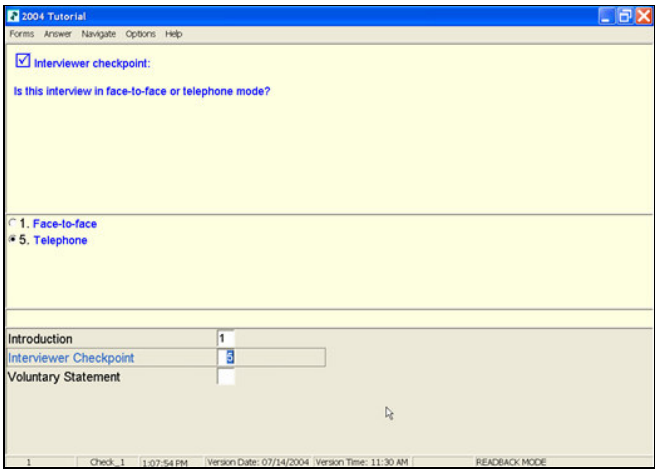
Interviewer checkpoint:

Is this interview in face-to-face or telephone mode?

1 Face-to-face
5 Telephone

[2] As indicated, Michigan is revising all CAI instrument development guidelines. Thus, examples do not necessarily reflect final guidelines. However, basic naming conventions and the use of the three variable identifiers will remain.

Figure 5. Blaise Screen for Interviewer Checkpoint "Mode Checkpoint"



4.0 Screen Design

The primary programming issue related to screen design that has affected data documentation has been whether or not to include a question identifier in the form pane or question text area of the Blaise screen. While Blaise allows display of the tag in the form pane, programmers have no control over its format (e.g., color) and placement in relation to the question text. Thus, programmers are often asked to include a question identifier in the form pane. However, for data documentation systems such as Michigan's that automatically places an identifier above the question text, the result is duplicate identifiers in the codebook, as shown in Figure 6.

Figure 6. Codebook Example:
Variable Name Identifier with Explicit Tag in Question Text

GRPATT1

EA_1b

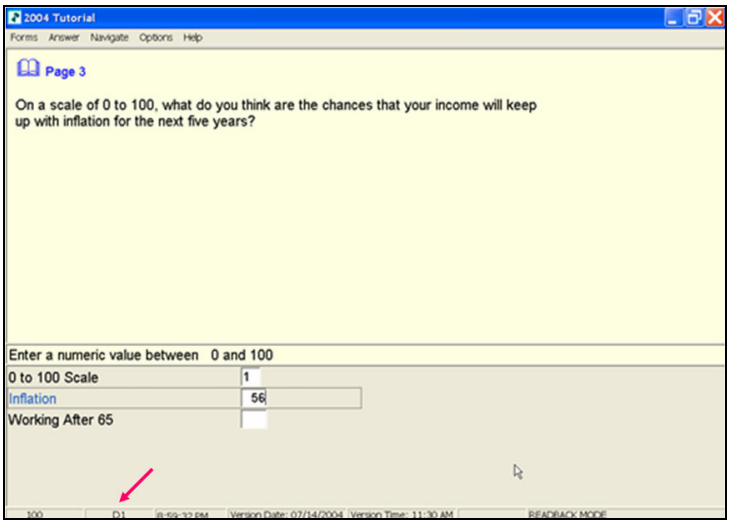
Have you attended a meeting of the organization in the past twelve months?

If R belongs to more than one organization in this group, ASK: Have you attended a meeting of any of them in the past twelve months?

- 1 Yes
- 5 No
- 8 DON'T KNOW
- 9 REFUSED

While a variable descriptor or name is the preferred identifier in the form pane or input area of a Blaise screen (Figure 5), the tag is the preferred screen identifier. This is because it is short and easy to refer to when troubleshooting problems and when training interviewers to use an instrument. In our example it is much easier to refer to screen "EA_1b" than to screen "GRPATT1." Michigan's solution has been to suggest that the tag not be put in the form pane or question area, but instead be placed in the status bar at the bottom of the screen, as shown in Figure 7.

Figure 7. Example of Screen with Tag in Status Bar Used for Identification

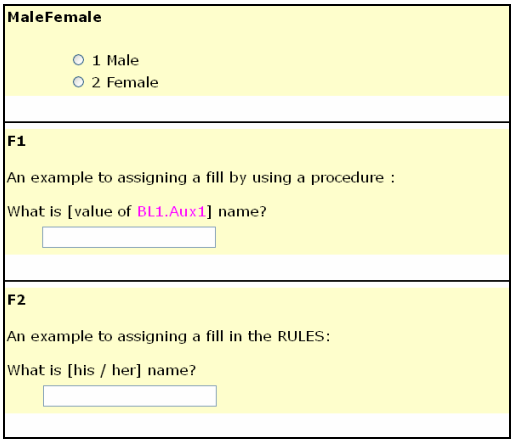


In a more general sense, screen design has a tremendous impact on the readability of any documentation created from Blaise instruments. Screen design standards affect both the instrument interface and the data documentation, since both take question text and response options from the Blaise datamodel. If question text (including all elements displayed in the info pane) and response options are poorly formatted for usability and interface design, then they will be equally poorly formatted for data documentation.

5.0 Programming

Sparks and Youhong (2004) discuss two aspects of Blaise programming that affected data documentation using Michigan's Blaise Documentation System, and they are both related to display of fill text. The first is that if an instrument relies on en external database lookup for creation of fills, they cannot be displayed in the data documentation. Figure 8 shows how a fill would display if created through a call to a procedure capturing the data from an external database (F1), and how it would appear if created directly in the instrument RULES section (F2).

Figure 8. Example of Problematic Fill Display When Use Procedure to Create Fill



The preferred display is as it appears for F2. One option for achieving this would be to avoid the use of fill procedure calls in Blaise instruments. The other would be to add to the Blaise Documentation System the option to capture fills from an external database. This may be done in a future version.

The second problem related to fills arises when a fill variable name is used multiple times in the same block, something which had been done in several Michigan instruments. All possible values associated with the variable will display in the data documentation, as shown in Figure 9 (F2). Figure 10 shows the mock code that generated the documentation.

Figure 9. Example of Problematic Fill Display When Reuse Fill Variable Name in Block

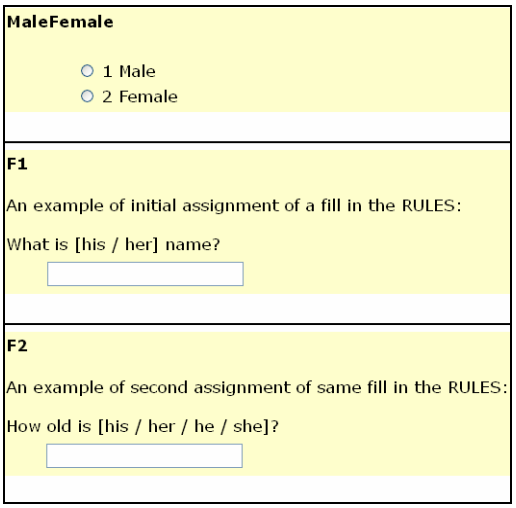


Figure 10. Blaise Code for Generating Reused Fill Example

```
DATAMODEL    TESTFill
  AUXFIELDS
    Aux1 : STRING
  BLOCK B1
    FIELDS
      MaleFemale :(Male, Female)
      F1 "An example of initial assignment of a fill in the RULES:
          @/@/What is ^Aux1 name?" : STRING [1]
      F2 "An example of second assignment of same fill in the RULES:
          @/@/How old is ^Aux1?" : STRING [1]
    RULES
      MaleFemale
     IF MaleFemale = Male THEN
         Aux1 := 'his'
     ELSE
         Aux1 := 'her'
     ENDIF
     F1
     IF MaleFemale = Male THEN
         Aux1 := 'he'
     ELSE
         Aux1 := 'she'
     ENDIF
     F2
   ENDBLOCK
   FIELDS
    BL1 : B1
  RULES
    BL1
ENDMODEL
```

6.0 Conclusions and Recommendations

Good data documentation must be easy to read and must be complete. The issues raised in this paper are primarily focused on those two aspects of data documentation. Good screen design can lead to readable data documentation, as can good specification and programming standards. Based on what Michigan has learned on the Blaise Documentation System development initiative, we would make the following recommendations:

- Ask clients and project managers to fully specify instruments, most importantly by providing meaningful field names and descriptors, as well as tags that can be used as short references to fields
- Do not put anything in an info pane that you would not want to appear in data documentation (such as an identifier above the question)
- Have fully elaborated screen design guidelines that are reflected in detailed instrument specification guidelines
- Do not use procedure calls to create fills from external data (unless you have a documentation system that can work with the external data
- Do not reuse fill variable names within the same block

The experience has also convinced us that all of Michigan's guidelines (specification, screen design, and programming) need to be updated, expanded, and cross-referenced, so that regardless of the backgrounds of the users these documents they will lead to efficient programming, usable instruments, and good documentation.

Contact: sehansen@umich.edu

7.0 References

Altvater, David, Ventrese Stanford, and Curtis Ziesing. 2001. Programming Techniques for Complex Surveys in Blaise. Paper presented at the 7th International Blaise Users Conference, Washington, September.

Couper, Mick P. 2000. Development and evaluation of screen design standards for Blaise for Windows. Paper presented at the 6th International Blaise Users Conference, Kinsale, May.

Gatward, Rebecca. 2003. Developing and updating screen layout and design standards. Paper presented at the 8th International Blaise Users Conference, Copenhagen, May.

Hansen, Sue Ellen, and Karll Dinkelmann. 2003. Screen design guidelines for Blaise instruments. Paper presented at the 8th International Blaise Users Conference, Copenhagen, May.

Pierzchala, Mark, and Graham Farrant. 2000. Helping non-programmers to specify a Blaise instrument. Paper presented at the 6th International Blaise Users Conference, Kinsale, May.

Soper, Ellen, and Ed Dyer. 2001. Challenges of Instrument Development. Paper presented at the 7th International Blaise Users Conference, Washington, September.

Sparks, Peter, and Youhong Liu. 2004. Blaise Documentation System. Paper presented at the 9th International Blaise Users Conference, Ottawa, September.

Wensing, Fred, Jane Barresi, and David Finlay. 2003. Developing an optional screen layout for CAI. Paper presented at the 8th International Blaise Users Conference, Copenhagen, May.