PRESENTING ANSWERS IN RANDOM ORDER.

A generic approach for presenting enumeration answers in random order in the Blaise Data Entry Program.

Marien Lina, Division of Business Statistics, Statistics Netherlands

1. Summary

A known methodological issue in survey research is structural response bias in case enumeration answers are presented in a fixed order. Respondents tend to select the first answers that are presented in an answer list. This occurs in self interviews such as internet surveys, but also in face to face surveys and telephone surveys. To miminize the amount of structural bias due to this undesirable response behaviour, answers may be presented in a random order.

There have been several attempts in the past to realize a random presentation order of answer categories in a Blaise questionnaire. Until recently, in Blaise this could only be done in a relatively complex way. The methods used in the old examples are not generic. They require modification for every new question. More generic solutions have been proposed but they require using the Blaise API, which is not available to everybody.

A new approach with Manipula in Blaise 4.8 makes it possible to realize this in a less elaborate way without using the Blaise API. This newly proposed method uses a Manipula setup and a generic include file for the Blaise data entry program. The Manipula procedures can be used in the data entry program relatively easy. For using the procedure a minimal amount of additional fields and instructions is required in the Blaise rules. This paper explains this generic approach to present answer actegories in random order. A practical example will be added in the appendix.

2. Purpose

Structural bias appears for enumeration questions with a fixed order of presented answers. Presenting answers in a random order may help to avoid structural bias in measurement instruments. Respondents may prefer to respond a yes in stead of a no. Also there may be a tendency to take the answer category that has been offered first. When answers are presented in a fixed order, the answer that is presented first may cause a selective context and affect the attitude towards the second and the third answer category. If the answers are presented in a fixed order then respondents may be moved to the same context, which may lead to structural bias.

Different results may depend on the homogeneity and difficulty of the presented questions, the education level of the respondent, social desirability and context bias, the sensitivity and the subjectivity of the topic.

3. Used technical features

Various new and old technical features of Blaise have been combined to create a number of more or less generic procedures to display answer categories in random order.

3.1 Hiding enumeration categories

A characteristic feature used here is the option to hide categories of an enumeration when the descriptions of a category is empty. This can be used for a dummy enumeration question with a large number of answer categories. A limited number of answer categories can be displayed when the other categories have been left empty.

3.2 Calling Manipula procedures in the data entry program

Relatively new in Blaise is that there are methods for running manipula procedures in a Blaise data entry program. Results of Manipula procedures can be passed to the rules of the data entry program.

3.3 Retrieving meta information in Manipula.

Another relatively new option in Manipula is the retrieval of meta information. Meta information - for example the number of categories to be displayed - can be read in a manipula procedure, and used in a data entry program, as described above.

3.4 Compute and fix a random order of enumeration answers

In the rules of a Blaise datamodel, a presentation order can be computed. As soon as it is computed, the sequence can be fixed and it can be saved in the data, if replication of the same order of presenting answer categories is required.

3.5 Separate fields for asking and saving

A known trick for generating questions in a random order is to add a dummy question in a datamodel. Text fills can be used for the answers in the dummy questions. The array cells are always presented in the same order, but the text fills can be arranged to follow the random order as computed as described in 3.4.

3.6 A generic approach.

A generic include file allows you to use procedures for presenting categories in random order for enumeration questions for a variable number of answer categories. Any Blaise datamodel can include the generic include file, listed in appendix 2. The include file requires the availability of the prepared generic Manipula setup listed in appendix 3. The include file arranges that an enumeration question will be asked with a random presentation order of the possible answers. You will find a more detailed description below.

3.7 Specific modelib options.

The generic solution requires specific modelib options for the preparation of the datamodel that uses the include file. The generic procedure in this example uses a layout called *CatText*, using specificic layout options that are listed in appendix 4.

To make it work, it is also necessary that, apart from the fields to be saved in the data, dummy array fields have been defined in the datamodel for asking the questions.

4. A generic solution

The example which is supplied in the appendix shows how you can combine the Manipula setup "shuffleanswers.man" (see appendix 3) with the generic include file "randinclude.bla" (see appendix 2) to use a random order for answer categories. The example works for enumeration questions with answer categories from 1 to 9. Code 0 will be treated as being empty. The values *dontknow* and *refusal* will be used and recognized. For recognizing empty as a value you will have to adapt the procedures in the include file and in the manipula setup. It is also quite simple to adapt the same example to make it work for a larger enumeration range, say from 1 to 97,

4.1 A simple datamodel.

Here is a small example datamodel that contains one enumeration question only.

```
DATAMODEL MyTranspo
TYPE
TTransport= (Walking, Bike, Car, Taxi, Bus, Train), dk, rf, empty
FIELDS   { the question order is saved, the question order of the previous session is used!}
Transport "Can you tell hus how you got to work today?" : Ttransport
RULES
Transport
ENDMODEL
```

The answers in *Transport* are presented in a fixed order, as defined for *TTransport*.

4.2 Another simple datamodel.

To use the generic include file and the related manipula setup, some changes are required in the datamodel. Take notice that you should use specific modelib options (see appendix 4).

```
DATAMODEL MyRandomAnswers
TYPE
```

```
  TTransport= (Walking, Bike, Car, Taxi, Bus, Train), dk, rf, empty
INCLUDE "RandInclude.BLA"
AUXFIELDS  {The questions order is not saved. The order changes from session to session!}
  TransOrder   : INTEGER { !! not saved in the data !! }
FIELDS   { the question order is saved, the question order of the previous session is used!}
  Transport "Can you tell hus how you got to work today?" : Ttransport
RULES
  Transport.KEEP
  TransOrder.KEEP
  ShuffleAnswers('Transport', TransOrder)
ENDMODEL
```

To arrange random answers for the question *Transport*, the actual question is not asked but instead it is kept in the rules:

```
RULES
  Transport.KEEP
```

The auxfield *Transorder* has been introduced to compute the random order of the answer categories. Take notice that this is an *AUXFIELD*. The presentation order will not be saved and computed again and again after re-opening the same form in the data entry program. If you would make it a *FIELD* then the order will be saved in the data and will stay the same after reopening the same data form.

The include file "randinclude.bla", contains the procedure "*ShuffleAnswers*". This procedure arranges the random presentation of the answers.

```
                                                    ShuffleAnswers('Transport', TransOrder)
```

The procedure contains a dummy enumeration question. The question text of the field *Transport* will be used in the dummy question. The answer categories will be copied from Transport to the dummy question, in the order as provided by the parameter *TransOrder*. Afterwards, the procedure returns the proper answer to the field *Transport*.

It is in this part where manipula procedures are called to retrieve meta information such as the question texts.

```
PROCEDURE GetQuestionText
  PARAMETERS
    IMPORT FieldName: STRING
    EXPORT QuestionText : STRING
  ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'GetQuestionText')
ENDBLOCK
```

Other procedures ask for meta information, such as category names and the number of categories. These procedures are available generic include file and they are calling external procedures in the manipula setup. The files are listed in appendix 2 and 3.

4.3 Using the solution in other datamodels.

A more elaborate questionnaire of the previous questionnaire is available in appendix 1. It contains four enumeration questions that will present answers in a random order. For each enumeration question you want to have a random presentation order of the answers, like for the transport question, you will have to define a *AUXFIELD* or *FIELD*. This field is used as parameter for the procedure "shuffleanswers"in the include file. In the example above this was the *AUXFIELD TransOrder*. After that, the include file should be added in the datamodel, and the datamodel must be prepared with a specific modelib options (see appendix 4).

APPENDIX

Appendix 1 - An example Datamodel.

```
DATAMODEL MyRandomAnswers

AUXFIELDS
  Hello "This example shows answer categories of enumeration fields in random order.
       @/Press Enter to start..." : (Enter), EMPTY
  Goodbye "This completes the questionnaire. Press <Enter> to continue." : (Enter), empty
{ the answer descriptions are text fills of the array Answer[i] }

TYPE
  TTransport= (Walking, Bike, Car, Taxi, Bus, Train, Balloon), dk, rf, empty
  TWorldMap = (America, Europe, Asia, Australia, Africa, AntArcitica), dk, rf, empty
  TAnimal =   (Tiger, Camel, Rabbit, Cow, Horse),dk, rf, empty
  TYesNo =    (Yes, No),dk, rf, empty

INCLUDE "RandInclude.BLA"

AUXFIELDS
  TransOrder   : INTEGER { !! not saved in the data !! }
  CoffeeOrder : INTEGER
FIELDS
  AnimalOrder: INTEGER { !! saved in the data !! }
  WorldmapOrder : INTEGER
  Transport "Can you tell hus how you got to work today?" : TTransport
  Animal    "What is your favorit Animal?"              : TAnimal
  WorldMap  "Where do you live?"                        : TWorldMap
  Coffee    "Would you drink coffee?"                   : TYesNo

RULES
  Hello
  Transport.KEEP
  TransOrder.KEEP
  ShuffleAnswers('Transport', TransOrder)
  Animal.KEEP                { these keep instructions are necessary }
  AnimalOrder.KEEP                    { before calling the procedure }
  ShuffleAnswers('Animal', AnimalOrder)
  WorldMap.KEEP         { these keep instructions are necessary }
  WorldMapOrder.KEEP             { before calling the procedure }
  ShuffleAnswers('WorldMap', WorldMapOrder)
  Coffee.KEEP           { these keep instructions are necessary }
  CoffeeOrder.KEEP              { before calling the procedure }
  ShuffleAnswers('Coffee', CoffeeOrder)
  Goodbye
ENDMODEL
```

N.B. using SHOW in stead of KEEP may be helpful to understand what is happening in the data entry program.

Appendix 2 - Generic Include file.

```
TYPE
  TTextArr = ARRAY[1..9] OF STRING

  DummyEnum = (Item1 "^TempAnswer[1]",
               Item2 "^TempAnswer[2]",
               Item3 "^TempAnswer[3]",
               Item4 "^TempAnswer[4]",
               Item5 "^TempAnswer[5]",
```

```
            Item6 "^TempAnswer[6]",
            Item7 "^TempAnswer[7]",
            Item8 "^TempAnswer[8]",
            Item9 "^TempAnswer[9]"),dk,rf,empty

AUXFIELDS
  PassScore    : INTEGER, dk, rf, empty
  QuestionText : STRING
  Answer, TempAnswer   : TTextArr
  EnumSize     : 1..9
  dumdum : STRING

PROCEDURE GetQuestionText
  PARAMETERS
    IMPORT FieldName: STRING
    EXPORT QuestionText : STRING
  ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'GetQuestionText')
ENDBLOCK

PROCEDURE GetSize
  PARAMETERS
    IMPORT FieldName1: STRING
    EXPORT EnumSize : INTEGER
  ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'TellSize')
ENDBLOCK

PROCEDURE GetOrd
  PARAMETERS
    IMPORT FieldName1: STRING
    EXPORT OrdVal : STRING
  ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'GetOrd')
ENDBLOCK

PROCEDURE GetCatNameI
  PARAMETERS
    IMPORT FieldName1: STRING
    IMPORT XVal : STRING
    EXPORT A1   : STRING
  ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'GetCatnameI')
ENDBLOCK

PROCEDURE GetRandomSequence { this procedure sets the random order of categories for one question }
  PARAMETERS
    IMPORT Size : INTEGER   { the number of categories in the enumeration }
    EXPORT Order: INTEGER    { a number that holds the sequence of the questions }
  AUXFIELDS
    Stringeling : ARRAY[1..9] OF INTEGER
    RandomSet: ARRAY[1..9] OF INTEGER { random numbers used to sort stringeling }
    DNum : integer
    ENUMSIZE : 1..9
  LOCALS I,J: INTEGER
  RULES
    EnumSize:=Size
    FOR I:= 1 to EnumSize DO
      Stringeling[i]:= i { initial categories are sorted to ORD value }
      RandomSet[i]:= random(999999999999999999) { high number to minimize double numbers }
    ENDDO
    FOR i:= 1 to 8 DO  { sort the array to random numbers  }
      FOR j:= 2 to 9 DO
        if (i<j) AND (j<=enumsize) THEN
          IF RandomSet[j]>RandomSet[i] THEN
            Stringeling.exchange(j,i)
            RandomSet.exchange(j,i)
          ENDIF
        ENDIF
      ENDDO
    ENDDO
```

```
    Dnum:= 0 { represent the order as a number }
    j:=1
    FOR i:= 1 TO 9 DO
      IF i<= enumsize THEN
        DNum:= DNum + j*stringeling[i]
        j:= j * 10
      ENDIF
    ENDDO
    Order:= DNum
ENDPROCEDURE

PROCEDURE RandomAsk { Use a dummy field of an enumeration type to ask the questions and }
                    { display the proper question and answer texts with the question }
  PARAMETERS
    IMPORT Desc: STRING
    TRANSIT D : integer     { import the previously selected category, export the selected categorie in the procedure }
    IMPORT Order:integer    { import the order of the answers }
  AUXFIELDS
    AskIt "^QuestionText" / "^DESC" : DummyEnum
    MyOrder : String
    MyAnswer: ARRAY[1..9] of string
    Stringeling : ARRAY[1..9] OF 1..9
  LOCALS
    i : INTEGER
    j : INTEGER
  RULES
    MyOrder:= Str(Order)
    FOR i:= 1 to EnumSize DO
      Stringeling[i]:= val(substring(Myorder,i,1))
    ENDDO
    FOR i:= 1 to 9 DO
      IF i<=EnumSize THEN
        TempAnswer[i]:= Answer[Stringeling[i]]
      ELSE
        TempAnswer[i]:= ''
      ENDIF
    ENDDO
    AskIt.KEEP
    IF AskIt=EMPTY THEN
      IF D= 10 THEN
        ASKIT:=DK
      ELSEIF D= 11 THEN
        ASKIT:=RF
      ELSE
        FOR i:= 1 to EnumSize DO
          IF D = val(substring(Myorder,i,1)) THEN
            AskIt:= i
          ENDIF
        ENDDO
      ENDIF
    ENDIF
    AskIt
    IF AskIt = RESPONSE THEN
      D:= Stringeling[ORD(AskIt)]
    ELSEIF AskIt= DONTKNOW THEN
      D:= 10
    ELSEIF AskIt= REFUSAL THEN
      D:= 11
    ELSE
      D:= 0
    ENDIF
LAYOUT
  At ASKIT fieldpane CatText
ENDPROCEDURE

PROCEDURE GetMeta
  PARAMETERS IMPORT MetaName:STRING
```

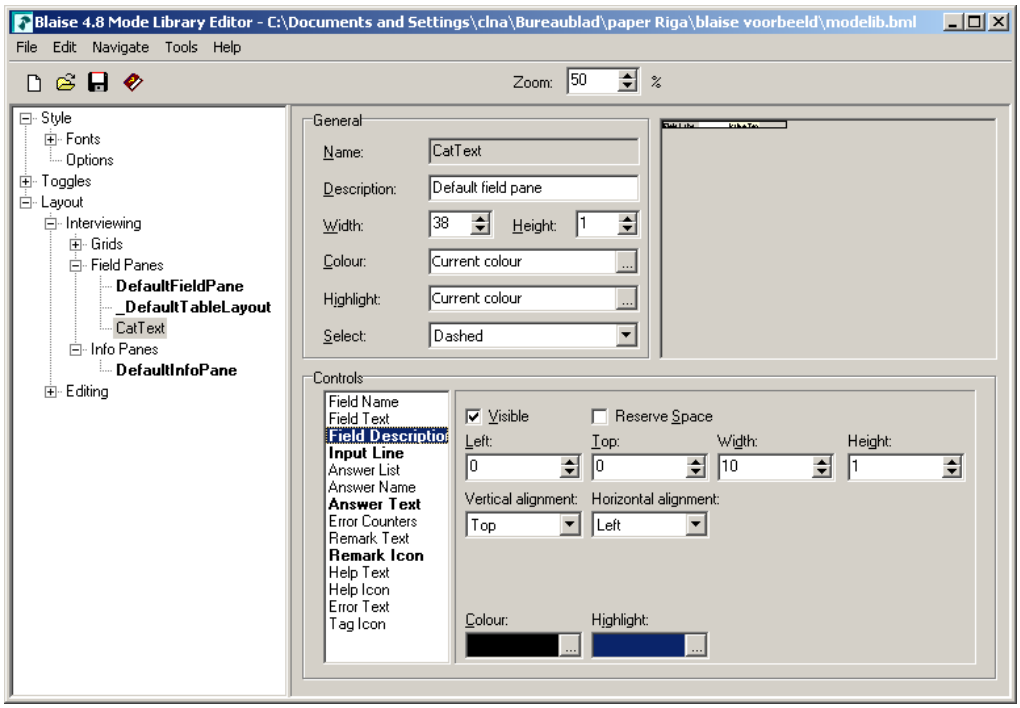
```
  LOCALS counter:integer
  RULES
    GetQuestionText(MetaName, QuestionText)
    GetSize(MetaName, EnumSize)
    FOR Counter:= 1 to 9 DO { assigning the names of the categories }
      GetCatNameI(MetaName, STR(Counter), Answer[Counter])
    ENDDO
ENDPROCEDURE

PROCEDURE UnShuffle
  PARAMETERS
    IMPORT FName : STRING
    IMPORT Score : INTEGER
  ALIEN('ShuffleAnswers.msu /KMyMeta=$dictionaryname', 'UnShuffle')
ENDBLOCK

PROCEDURE ShuffleAnswers
  PARAMETERS
    IMPORT MetaName:STRING
    TRANSIT ItemOrder: INTEGER
  LOCALS
    OrdVal: STRING
  RULES
    GetMeta(MetaName)
    IF ItemOrder=EMPTY THEN
      GetRandomSequence(EnumSize, ItemOrder) { computing the random sequence if still empty }
    ENDIF
    GetOrd(MetaName, dumdum)
    PassScore:= val(dumdum)
    RandomAsk(MetaName, PassScore, ItemOrder)
    UnShuffle(MetaName, PassScore)
ENDPROCEDURE
```

Appendix 3 - Generic Manipula Setup

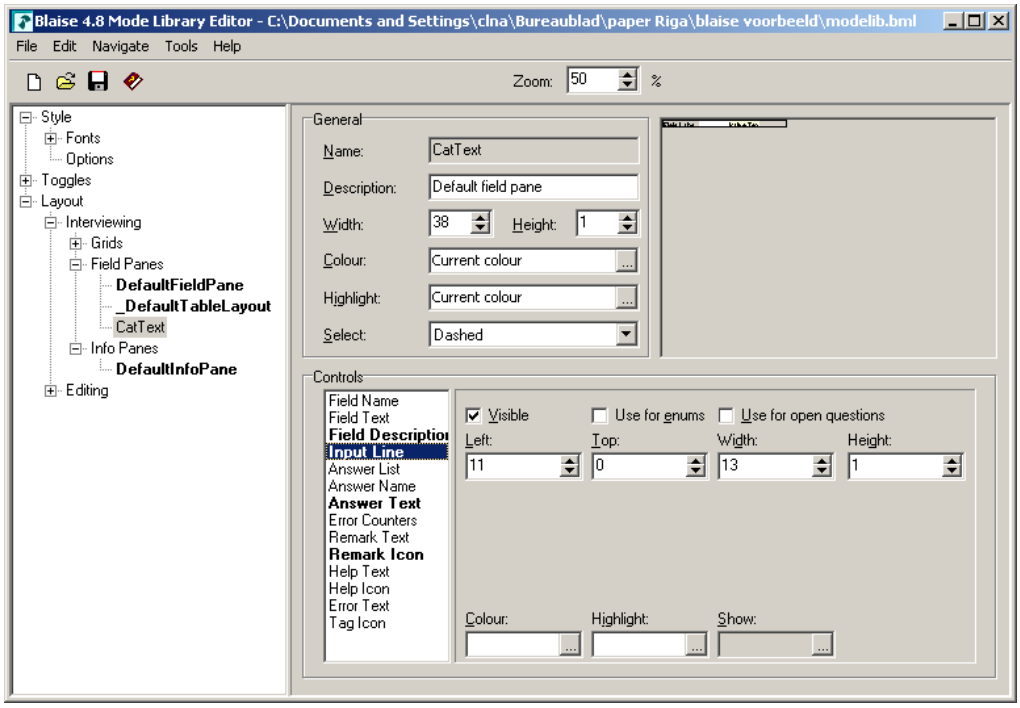
```
PROCESS TestMeta
USES MyMeta (VAR)
TEMPORARYFILE x:MyMeta
SETTINGS INTERCHANGE=SHARED

PROCEDURE UnShuffle { put answered value "Score" back to field "FName" }
  PARAMETERS
    IMPORT FName : STRING
    IMPORT Score : STRING
INSTRUCTIONS
  IF val(Score) = 10 THEN
     x.PUTVALUE(FName, DONTKNOW) { take into account DK values }
  ELSEIF val(Score) = 11 THEN
     x.PUTVALUE(FName, REFUSAL) { take into account RF values }
  ELSEIF val(Score) >0 THEN
     x.PUTVALUE(FName, Score)
  ELSE
     x.PUTVALUE(FName,EMPTY)
  ENDIF
ENDPROCEDURE

PROCEDURE GetOrd { Read the current value in field "FieldName1" }
  PARAMETERS
    IMPORT FieldName1 : STRING
    EXPORT OrdVal : STRING
  AUXFIELDS
    FieldStat : STRING
INSTRUCTIONS
  FieldStat:= x.GETFIELDINFO(FieldName1, 'FIELDSTATUS')
  IF FieldStat = 'DONTKNOW' THEN
    Ordval:= '10'       { take DK and RF values into account }
  ELSEIF FieldStat = 'REFUSAL' THEN
```

```
    Ordval:= '11'
  ELSEIF FieldStat = 'EMPTY' THEN
    Ordval:= '0'
  ELSE
    OrdVal:= x.GETVALUE(FieldName1,UNFORMATTED)
  ENDIF
ENDPROCEDURE

PROCEDURE TellSize { how many categories are in the enumeration }
  PARAMETERS
    IMPORT FieldName1 : STRING
    EXPORT Size : STRING
  AUXFIELDS
    ix : integer
INSTRUCTIONS
  ix:= val(x.GETFIELDINFO(FieldName1,'CATEGORIES.COUNT'))
  Size:= STR(ix)
ENDPROCEDURE

PROCEDURE GetCatNameI { returns the category name of FieldName[XVal]}
  PARAMETERS
    IMPORT FieldName1 : STRING
    IMPORT XVal       : STRING
    EXPORT A1         : STRING
  AUXFIELDS
    ix : INTEGER
INSTRUCTIONS
  ix:= val(x.GETFIELDINFO(FieldName1,'CATEGORIES.COUNT'))
  IF VAL(XVAL)<= IX THEN
    A1:= x.GETFIELDINFO(FieldName1,'CATEGORIES[' + XVAL + '].NAME')
  ELSE
    A1:= ''
  ENDIF
ENDPROCEDURE

PROCEDURE GetQuestionText { returns the questiontext of FieldName }
  PARAMETERS
    IMPORT FieldName: STRING
    EXPORT QuestionText : STRING
INSTRUCTIONS
  QuestionText:= x.GETFIELDINFO(FieldName,'QUESTIONTEXT')
ENDPROCEDURE
```

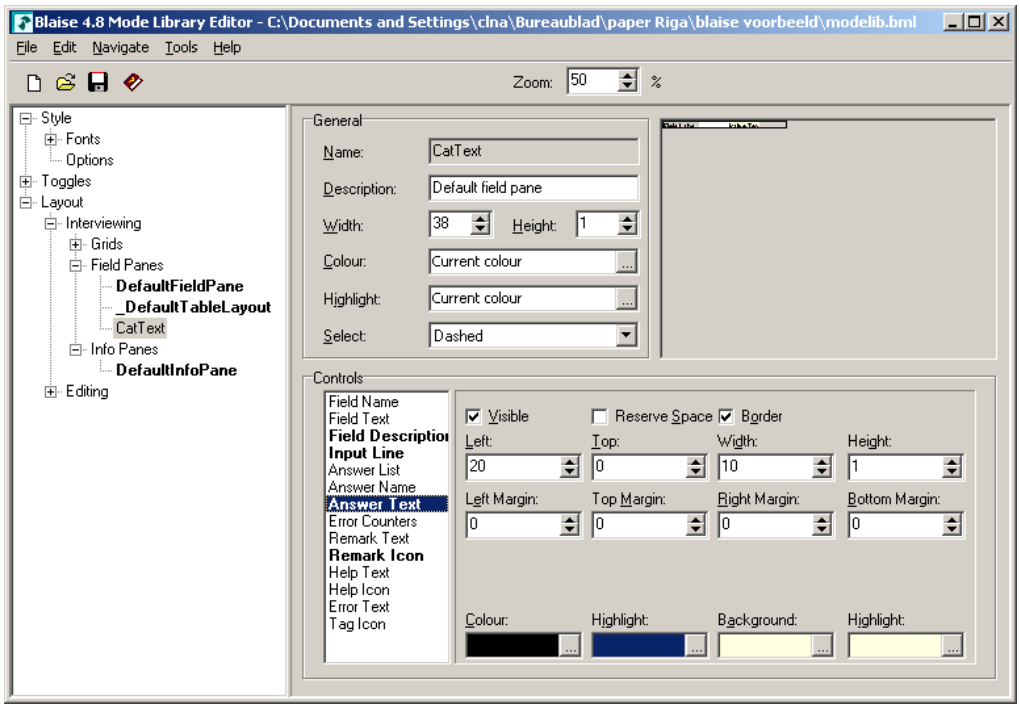
Appendix 4 – Used Modelib Settings



The Field Pane *CatText* is used in the generic include file for presenting and asking the dummy question with the random order of the answer categories of the enumeration.



The figures show the settings for displaying the *Field Description*, *Input Line* and *Answer Text* for the Field Pane *CatText*. The Remark icon is in the default position.


Blaise 4.8 Mode Library Editor - C:\Documents and Settings\clna\Bureaublad\paper Riga\blaise voorbeeld\modelib.bml
File Edit Navigate Tools Help
Zoom: 50 %
Style
Fonts
Options
Toggles
Layout
Interviewing
Grids
Field Panes
DefaultFieldPane
_DefaultTableLayout
CatText
Info Panes
DefaultInfoPane
Editing
General
Name: CatText
Description: Default field pane
Width: 38 Height: 1
Colour: Current colour
Highlight: Current colour
Select: Dashed
Controls
Field Name
Field Text
Field Descriptio
Input Line
Answer List
Answer Name
Answer Text
Error Counters
Remark Text
Remark Icon
Help Text
Help Icon
Error Text
Tag Icon
Visible
Reserve Space
Border
Left: 20
Top: 0
Width: 10
Height: 1
Left Margin: 0
Top Margin: 0
Right Margin: 0
Bottom Margin: 0
Colour:
Highlight:
Background:
Highlight: