Basil, A New Tool for CASI in Blaise

Roger Linssen & Jo Tonglet (Statistics Netherlands)

1. Introduction

In the past decade, computer assisted self interviewing has become more and more popular. CASI is a cheap way to get data from respondents: you don't need expensive call centres or interviewers.

Statistics Netherlands is experimenting with different types of CASI surveys. One of the experiments was the use of a downloadable program that handles the survey and sends the data back to Statistics Netherlands through the internet.

2. Requirements

There were several user requirements for such a downloadable program, listed in Appendix A. A couple of important requirements:

- Size < 2 MB
- No reboot after installation
- Reusable for different surveys
- Data transmission through internet or e-mail
- Lots of specific layout requirements (see also Appendix B)
- Time to complete the project: 5 months

3. Project Decisions

After carefully examining the requirements, it was decided that the program should be programmed in Delphi. Blaise is programmed in Delphi, and therefore, all Blaise functionality could be used in the program. Also, Delphi can produce stand-alone executables, which don't require any run-time libraries, and therefore an easy installation is guaranteed.

The next decision was to handle all layout requirements within the Blaise language. It was not possible to change the Blaise compiler within such a short period of time, but it was possible to use the Blaise language concept to store layout information. Another important decision was to use the Blaise paging mechanism, instead of creating a new algorithm. An 'index' control, which Blaise doesn't have by default, had to be created, and linked to the paging mechanism.

It was soon clear that some form of event handling was necessary: the layout requirements included buttons, hyperlinks, help icons, etc. It was decided to use Manipula (or rather, Maniplus) to facilitate event handling. This decision had major consequences: all event handling had to written by the programmer of the data model. However, this way TEPS (Tool for Electronic Production Statistics) could be a generic program, which could be used for all sorts of data models. Dialogs, error handling, printing and even transmission of data could be handled by Maniplus.

4. Prototyping

In the Netherlands, people can fill out their tax form using a program similar to TEPS (see Appendix C). This Tax Disc can be downloaded and installed on a PC, handles the tax 'questionnaire', and sends the data back to the tax department. All in all, a perfect prototype.

Typically, the tax questionnaire has an index and an image on the left side of the screen, and a content area on the right side of the screen. The bottom part of the screen contains a navigational area and a menu area.

4.1 Application

The first step was to define an 'application definition', in which a specification of the global layout of the instrument could be made. This specification is XML-based. The Blaise datamodel description was the ideal place to put this definition:

```
DATAMODEL Test
"<application width=800 height=600 Title='Test Application'>
   <panel align=left width=100 color=blue>
     <panel align=bottom height=100>
       <img src='LOGO' left=0 top=0 width=100 height=100>
     </panel>
     <panel align=client>
       <index-area>
     </panel>
   </panel>
   <panel align=client>
     <panel align=bottom height=25 color=blue>
       <button left=0 top=0 height=25 src='EXIT'>
     </panel>
     <panel align=client color=green>
       <content-area>
     </panel>
   </panel>
 </application>
"
...
ENDMODEL
```

This code snippet shows how to define an instrument, divide it into areas, and put controls (image, button, index and content) on the areas. It is possible to divide areas (panels) into sub-areas.

4.2 Content area

Next, the content-area needed to be filled with (Blaise) questions. To achieve this, a question layout definition was needed. A question can be seen as a container, which holds one or more elements. The question text part of a question definition is used to describe the layout of a question:

```
FIELDS Q
"<question>
   <label left=0 top=0 width=100 height=25 text='Name?'>
   <input left=150 top=0 width=75 height=25>
 </question>
" : string[40]
```

This code will define a question with two display elements: a label and an input control. The type of input control depends on the type of the Blaise field. In this case, an edit box will appear, because the type of the Blaise field is string. If the Blaise type was an enumeration, the input control would have been a radio button group. It is possible to change the look and feel of the default control by setting its properties, but it is also possible to use a different control. For an enumeration, for

instance, it is possible to use a dropdown control, a checkbox (useful if there are only 2 options in the enumeration), or a button.

4.3 Index area

The Index area was especially tricky. It should be possible to define the index in the Blaise language, but Blaise has no concept of an index. It was decided to use Blaise block field descriptions to represent the index. Nested blocks can then be used to add structure to the index.
The following piece of code shows how to define an entry in the index:

```
BLOCK BBlock
  {Field Definitions}
  ...
ENDBLOCK

FIELDS
  B "<index>
       <img left=0 top=0 width=16 height=16 src='ICON'>
       <label left=20 top=0 width=40 height=25 text='Section 1'>
     </index>
    " : BBlock
```

The index element consists of an image and a label. By clicking on the index element, the first eligible field of BBlock will be focused.

4.4 Event handling

The next challenge was to add event handling to the data model: when certain events occur, a piece of (procedural) code should be executed. For instance, by clicking on a button, data could be saved to disk, or sent to a web server. It should also be possible to create dialogs in an event handler, which present a user interface to perform certain tasks. For instance, when the user tries to leave the program, a custom dialog could pop up which asks the user if he's really sure he wants to exit.
Blaise does not support event handling. However, one of the Blaise tools, Maniplus, has all the characteristics to be used for event handling: it is a procedural language, and it can create user interfaces.
In the application section, one can define the Maniplus setups that are used for event handling

```
DATAMODEL Test
"<application width=800 height=600 setups='Events.msu'>
   ...
 </application>
"
```

The Maniplus setups that are specified in the application section are preloaded during initialization of the program, and are kept in memory after that. This way, the performance is sped up significantly.

The actual event handler, written in Maniplus, could look something like this:

```
PROCESS Events
USES
  dmTest 'Test'

TEMPORARYFILE tfTest : dmTest
SETTINGS
  INTERCHANGE = SHARED

DIALOGBOX DlgSave "Save"
  ESCAPE = NO
  SIZE   = (370, 100)
  COLOR  = '239 255 255'
  TEXT   = ('Press the OK button to save your data', 10, 10)
  BUTTON
    CAPTION = 'OK'

PROCEDURE SaveData
  dlgSave
  tfTest.WRITE
ENDPROCEDURE
```

This event handler will show a dialog first, after which a record is written. To use this event handler, for instance when a button is clicked, assign the event handler in the data model:

```
DATAMODEL Test
"<application width=800 height=600 setups='Events.msu'>
   <panel align=bottom height=50>
     <button left=0 top=0 width=75 height=25 src='SAVE'
             onclick='Events.SaveData'>
   </panel>
   ...
 </application>
"
```

4.5 Variables & Expressions

Blaise has the possibility to put variables (fills) in question texts. This way, a piece of text can vary depending on the value of a field. TEPS extends this concept by allowing two other fill types to be used in texts: $-fills and %-fills.

Both new fill types require the fully qualified field name:

```
BLOCK BBlock
  FIELDS
    Q1 : string[30]
    Q2 "<question>
          <label left=0 top=0 width=100 text='^Q1'>
          <label left=0 top=50 width=100 text='$(B1.Q1)'>
          <label left=0 top=100 width=100 text='%(B1.Q1)'>
          <input left=0 top=150 width=100 height=25>
        </question>
       " : integer[1]
ENDBLOCK

FIELDS
  B1: BBlock
```

This will create an area with three labels and an input control. Each label will have the value of Q1 as text. The normal fill (^Q1) does not require the fully qualified name, whereas the other two fills do.

If the structure of the question (e.g. the components it is composed of) doesn't change, then the %-fill is preferable. If a $-fill is used in a question text, the structure of the question is rebuilt every time fills are calculated. This means that the screen may flicker.

Notice the (optional) parenthesis around the fully qualified fieldname. This allows you to add special characters directly after the fieldname:

```
<label left=0 top=0 text='Year:$(Year).'>
```

This way, the dot will not be interpreted as being part of field Year.

The $ and %-fills also support expressions:

```
FIELDS
  NrOfPeople: integer[2]
  Q "<question>
       <label left=0 top=0 visible='%NrOfPeople>0'>
       <input left=0 top=150 width=100 height=25>
     </question>
    " : integer[1]
```

The label will only be visible when NrOfPeople holds a value greater than zero.

5. Electronic Production Statistics

After prototyping with the Tax Disc, it was time to implement a Statistics Netherlands questionnaire. Five EPS questionnaires were chosen for to be implemented 'by hand'. After that, it should be possible to generate the remaining 188 questionnaires from a meta server and a datamodel server using the first five questionnaires as a template.

Because the design of the tool was such, that it could be reused for multiple questionnaires, the decision was made to ship the tool with Blaise. The name TEPS (Tool for Electronic Production Statistics) changed rapidly into BASIL (Blaise Alternative Self Interviewing Language).

The design of the prototype proved to be very solid. By inventing an XML-like language, it was possible to add new keywords and properties very easily. Implementing the first four questionnaires proved to be fairly easy.

After the completion of the first EPS questionnaire, it was evaluated by 10 respondents. This resulted in several additions and corrections to Basil:

- Improved navigation
- Facilitations to make an audit trail
- Index improvements
- Additions to Maniplus, which made it possible to access all meta data of a questionnaire

During installation of the five questionnaires, Basil was also used to display a username/password dialog.

In March 2006, all five questionnaires went in production. They were sent to (or rather, downloaded by) 7400 respondents. The results thus far are very satisfying. Approximately 95% of the respondents use the internet to send data to Statistics Netherlands. The rest uses e-mail.

6. Conclusions

It proved to be possible to meet a deadline of half a year. This was accomplished by an intensive cooperation between developers, questionnaire designers and layout specialists.

Most of the requirements as mentioned in Appendix A were met. Those that weren't met were deliberately removed from the feature list.

By designing the application as a generic program, it is possible to reuse the tool for different projects, and even ship it as a standard tool with Blaise.

Appendix A: Feature List

Download & installation

- Downloadable through a respondent portal
- Maximum program size: 2 MB
- No reboot obligation after installation
- Large font support
- Default screen resolution: 800 x 600

Login

- Authentication possibility (no obligation) at login using a password
- Automatically reload previously filled out data at startup

Design

- Different panes with different colors:
  - Pane for index
    - Tree view (index)
    - Expandable tree
    - Hyperlink possibility
    - "Teletext" principle (numbers)
    - Only one chapter expanded at any moment
    - Possibility to place information (company info) at any position on the index pane
  - Pane for content
    - Possibility to place question and answer at arbitrary positions within the pane
  - Pane for navigation
    - Possibility to place clickable navigation buttons at arbitrary positions
  - Pane for menu
    - Possibility to place clickable menu buttons at arbitrary positions

Navigation

- Clear navigation
- Overview of the questionnaire + clear indication of the position in the questionnaire
- Navigation with both mouse and keyboard

Data Entry

- Confirm possibility (confirm button) on a page
- Immediately save entered data (even before confirmation)
- Progress indicator that indicates what part has been filled out and what part still has to be filled out/corrected
- Remark field, available for every answer
- Calculator field functionality: available for every answer (when applicable)
- Roll-up menu (lookup file), table from which selections can be made

Explanation

- Available with a single mouse click or keystroke
- Possibility to add a clickable or non-clickable button/image to every question/answer
- Explanation must be able to appear at any position
- Explanation is removable with a single mouse click or keystroke
- Windows Screen must be available
- A pop-up must be able to appear at any moment
- "Google-like" search function at any place in the questionnaire

Auxiliary Functions

- Windows Calculator (in menu area), callable with a mouse click or keystroke

- Printing of the questionnaire at different levels, for instance:
  - Empty form
  - Filled out form
  - Chapter
  - Clarification

Transmission

- Transmission:
  - option through internet (default)
  - option through email

Appendix B: Layout wishes

Figure 1: Layout structure

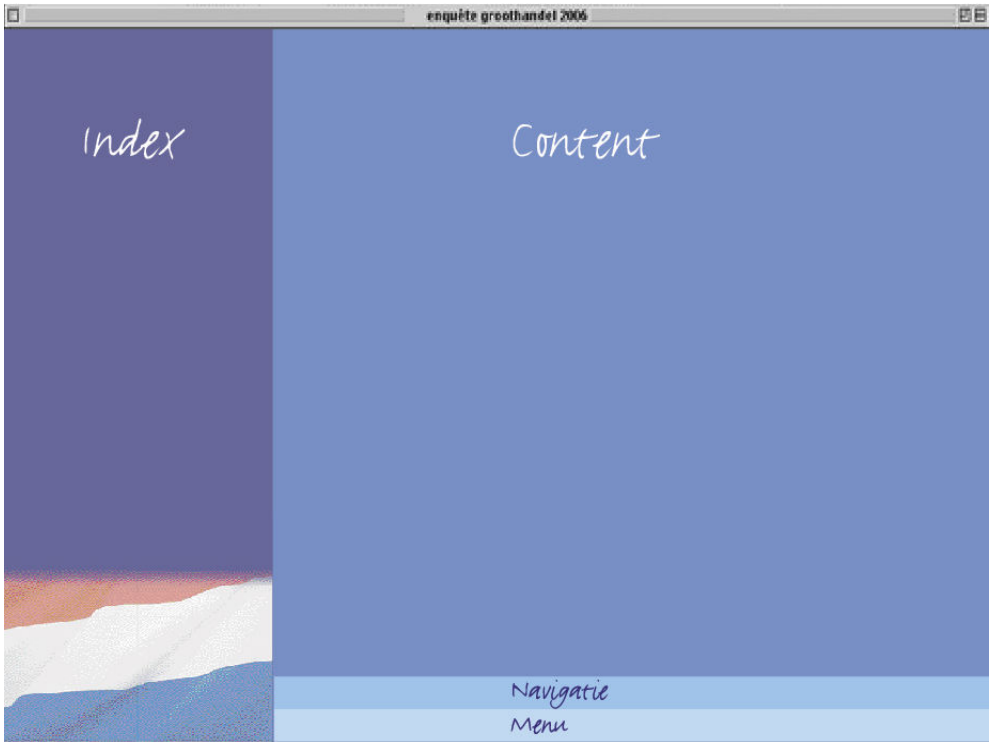


Figure 2: Sample page

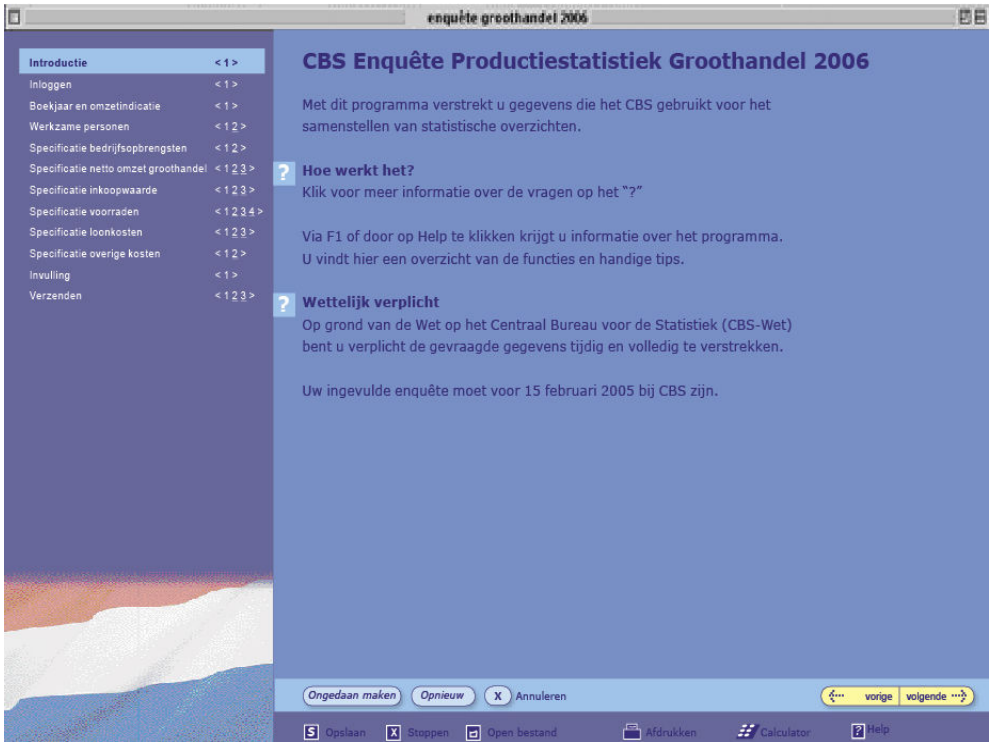


Figure 3: Context Sensitive Help

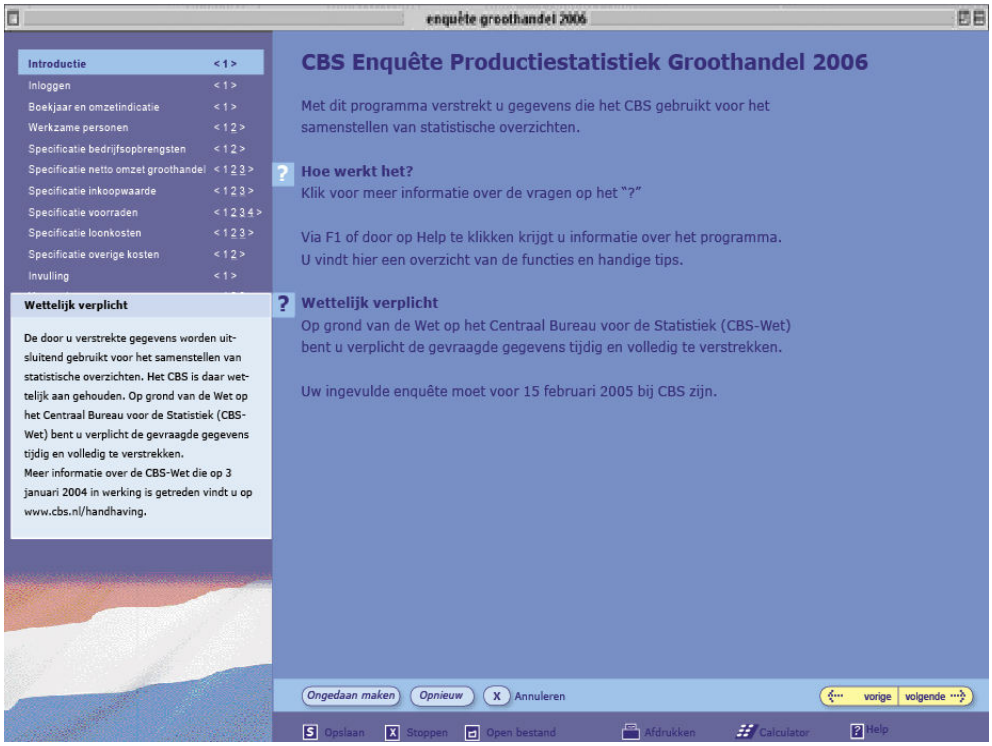


Figure 4: Index & Teletext principle

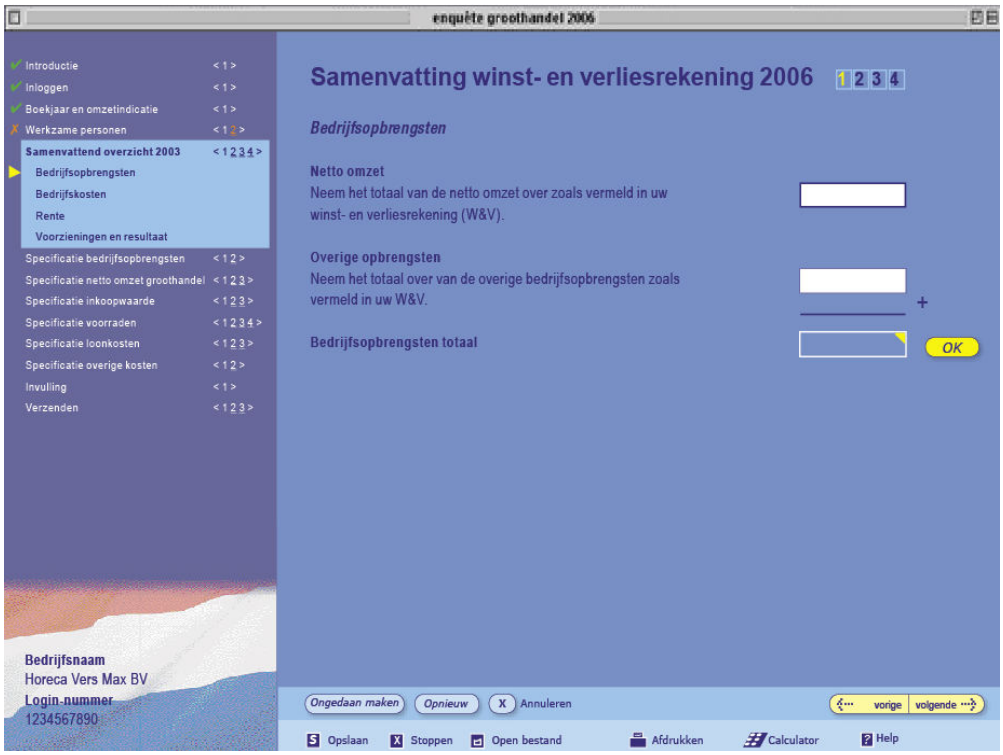


Figure 5: Calculator Fields

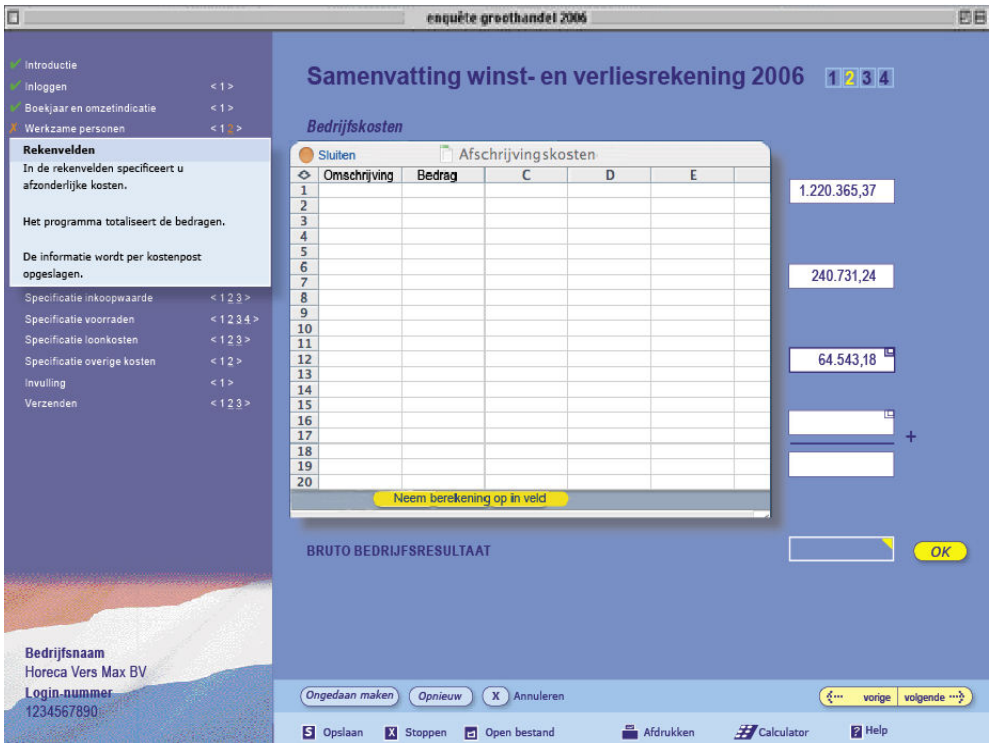


Figure 6: Error handling

Figure 7: Printing

Figure 8: Transmission

Appendix C: Tax Disc Prototype

Figure 9: Tax Disc (Original) Introduction Screen

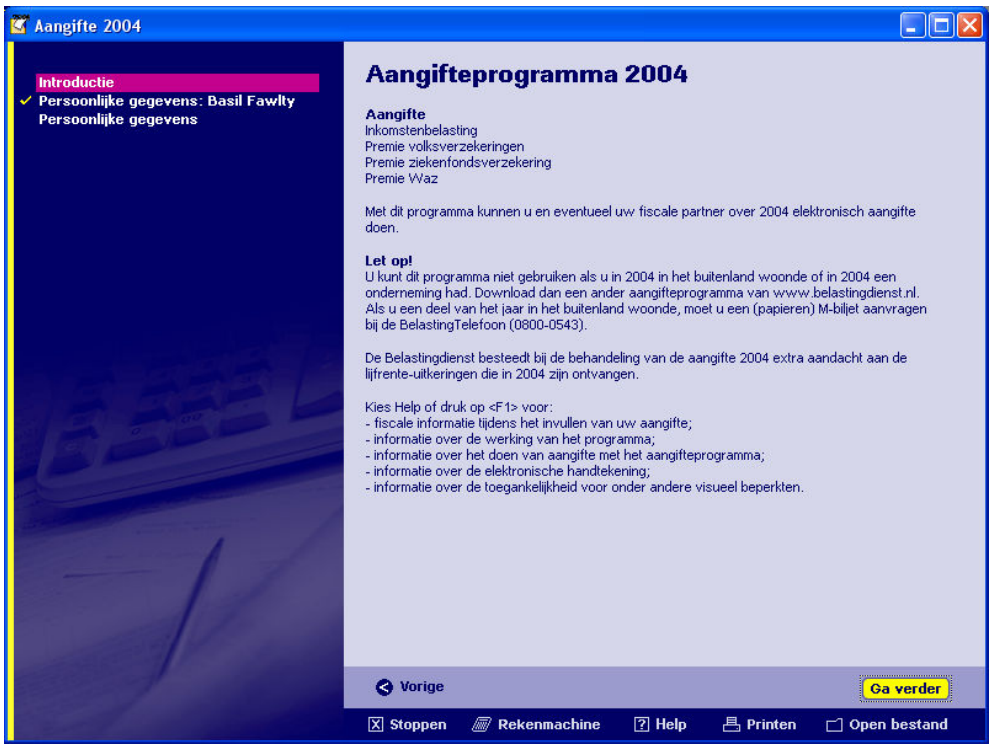


Figure 10: Tax Disc (Prototype) Introduction Screen

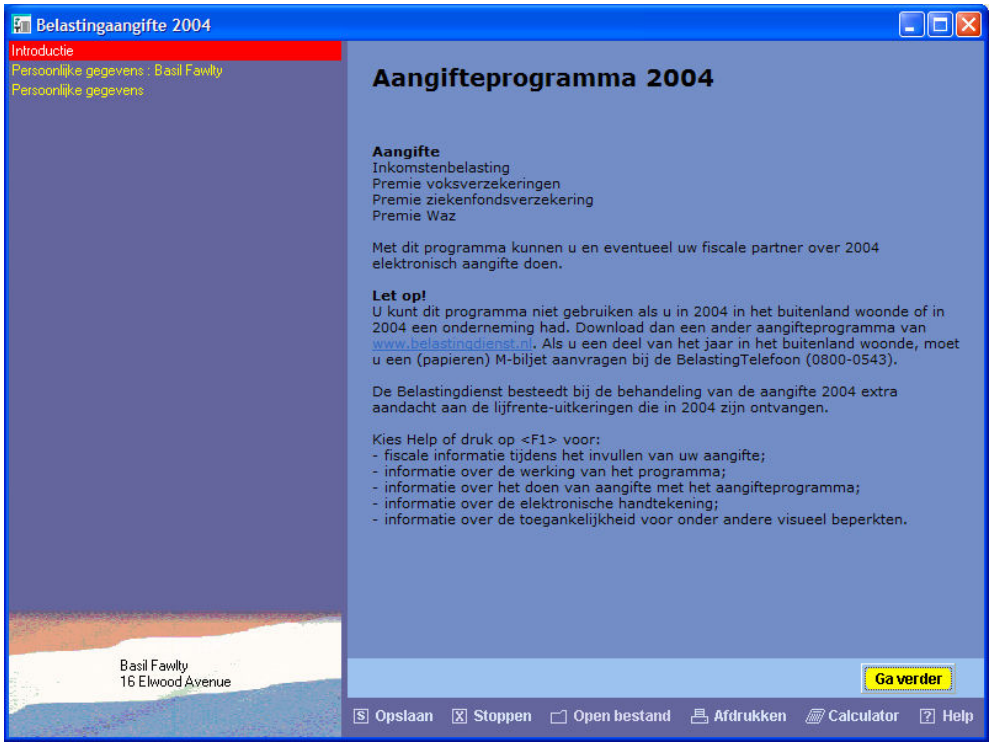


Figure 11: Tax Disc (Original) Sample Screen

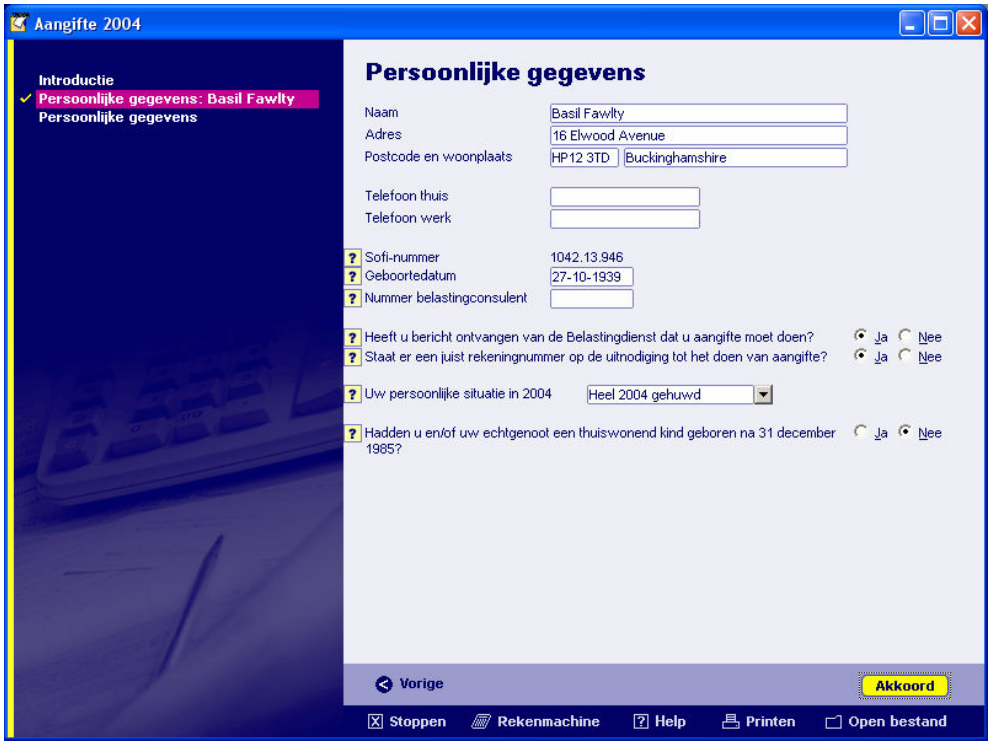
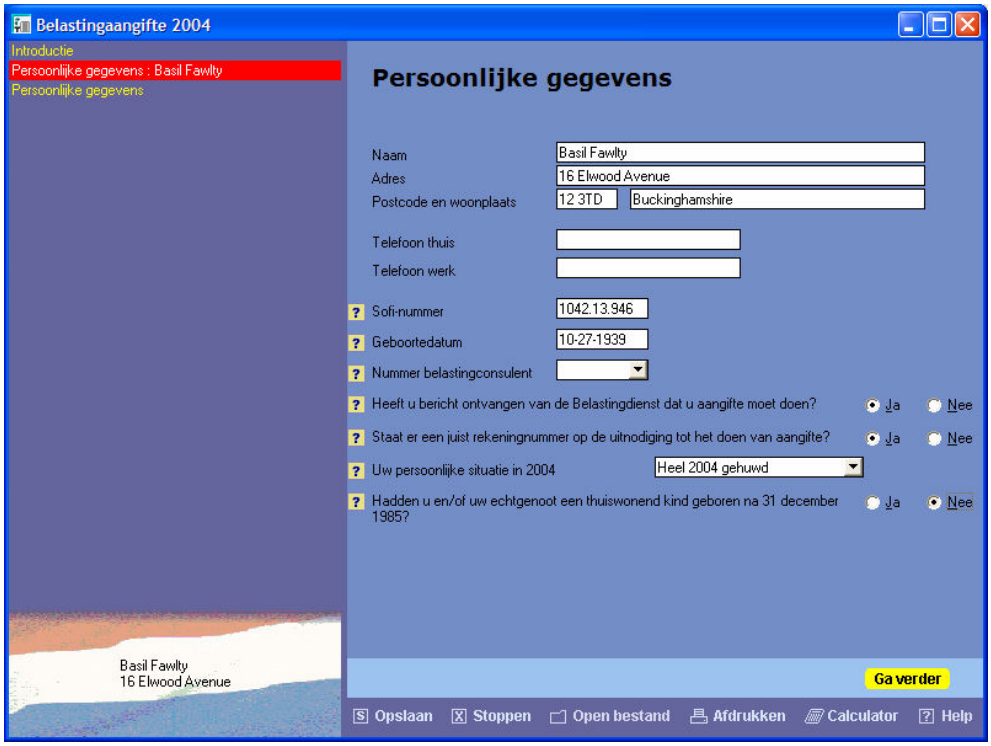


Figure 12: Tax Disc (Prototype) Sample Screen



Session 3.3: Transformations

Computer Assisted Coding by Interviewers[1]

Wim Hacking, John Michiels & Saskia Janssen-Jansen (Statistics Netherlands)

1. Introduction

Coding activities play an important role in a statistical production process. These activities are usually associated with the assignment of responses to predefined codes in a classification, so that these responses become available for further data-editing operations. Coding can be done at various stages of statistical production: during data collection by respondents or interviewers or during the data-editing process by coding experts and/or automated systems. In most cases coding is a rather trivial exercise: when questions are clear and unambiguous and the number of answering categories is limited an answering category can easily be provided by respondent or interviewer. The situation becomes more difficult when the number of answering categories is large or several open text answers are required. Examples of these more difficult coding activities are the assignment of a respondent's educational background or occupation to a corresponding classification. In these examples usually more than one question is involved in gathering the information required for assigning valid codes and part of these questions are of the open text type.

In previous years the approach adopted at the *division of social statistics* for coding (open text) responses from a number of related questions was the following. First, the answers to these questions are collected using CAPI or CATI modes of data collection. In the case of long answers they are interpreted and modified by the interviewer to obtain a concise response. At the statistical office the collected and sometimes edited information is then fed to a 'batch' process for automated coding. The records that are not coded are classified interactively by coding experts in a second pass. Coding economic activities at the *division of business statistics* was partly done manually at Statistics Netherlands and partly externally, at the chambers of commerce.

In this paper we would like to describe alternative coding techniques that are more successful than the traditional approach. The technique chosen depends on a number of factors (and possibly more than given here):

- The desired level of detail for statistical output (publications, international obligations) and for intermediate processes (e.g. weighting).
- The desired quality of coding (error rates).
- The available budget and expertise for development, implementation, operation, and maintenance.

The first point is relevant because for obtaining low detail classifications in most cases less elaborate methods can be developed that produce good results. The desired quality puts similar restraints on the choice of coding method. As higher demands on detail and accuracy require more complex coding techniques, it will in general also mean higher development costs.

In this paper we will discuss an alternative coding strategy that has been developed and selected for implementation at Statistics Netherlands: Computer Assisted

[1] The views expressed here are those of the authors and do not necessarily reflect the position of Statistics Netherlands

Coding by Interviewers (CACI). The new strategy is more cost effective than traditional approaches with comparable level of detail and reliability. The available expertise for implementing this approach can usually be found at a statistical office; if not, the required effort to obtain this knowledge is certainly less than for some more advanced methods in machine learning or for expert systems. In the next section we will describe the methods that have been applied for semi-automatic coding. In section 3 we will discuss the interactive coding approach as applied in the interview process at Statistics Netherlands, based on the methods from section 2. Finally, section 4 shows results using these new techniques. Here the percentage of coded records and coding reliability for each technique will be considered, along with results on interview lengths for CACI (based on field work done by the division of social statistics). In addition, some early results for the coding of economic activity at the division of business statistics are given.

2. Coding Techniques Used

We will start by giving an overview of the coding techniques used. The merits and drawbacks of each technique will also be discussed in brief. Basically, three coding techniques have been chosen depending on the (coded) material at hand: in 2.1 we discuss the situation that (many) previously coded records are available in electronic form. In 2.2 we start with a registry containing an extensive description for each code. In 2.3 no electronic material is available and a search file is constructed specific for the coding process. Depending on what initial material is at hand methods have selected for semi-automatic coding.

2.1. Using previously coded material

A number of techniques for automatically classifying text strings has been developed and implemented by the Institute for Knowledge and Agent Technology (IKAT) at Maastricht University: Nearest Neighbours (NN), Term Frequency Inverse Document Frequency (TFIDF), and Naïve Bayesian (NB) (Smirnov 2003, Kaptein 2005). All of these techniques involve learning algorithms that need a training data set with combinations of text strings and corresponding codes for ‘optimization’ or ‘training’. The nature of this optimization depends on the technique being considered. Training a learning algorithm produces a text-classifier with an approximate mapping of text strings to codes. The text-classifier assigns a weight or probability to each code in the classification and the code with the largest weight or probability is then selected.

The problem is to determine the reliability of this type of classification. There are ways to do this using another kind of learning-machine called meta-classifiers. These classifiers use ‘meta-information’ from text-classifiers: in case the code assigned by the text-classifier equals the true code in the training data set the combination of text string and code vector is labelled as ‘good’, otherwise it is labelled as ‘bad’. The meta-classifier uses this kind of information in order to optimize a set of rules for correctly assessing text-classifier results: with these rules the meta-classifier decides whether the codes assigned by the text classifier for new text strings are to be considered as ‘good’ or as ‘bad’. There are different ways in which these rules can be constructed and hence there are different meta-classifiers. Examples of meta-classifiers are described in the thesis of Kaptein 2005 and the main results are mentioned in section 4.2.

Instead of coding the material afterwards based on classifier techniques we can also apply them during the interview in the field. We will an example to clarify the approach chosen. Suppose that we are interested in coding the open text ‘carpenter’ (as a possible answer to the question: what is your current occupation?). This answer has been given in previous surveys many times before and at the statistical

office it has been assigned a number of different occupation codes (using additional information on job activities). It is now possible to calculate conditional probabilities $P(Code_i|$ *'carpenter')* representing the frequency with which each code has been assigned to 'carpenter'. Therefore to each word in the open text answer a vector of conditional probabilities can be determined. In case the open text contains more than one word the vectors are added to produce a combined vector of weights (not probabilities). For example, in the open text answer 'carpenter at shipyard' there are probability vectors for 'carpenter', 'at', and 'shipyard'. These vectors are added to produce a vector of weights:

Carpenter: $P(code_1|$ *'carpenter')=0,60 ;* $P(code_2|$ *'carpenter')=0,20 ; .....*
At: $P(code_{14}|$ *'at')=0,02 ;* $P(code_2|$ *'at')=0,01 ;* $P(code_{11}|$ *'at')=0,01 ; .....*
Shipyard: $P(code_2|$ *'shipyard')=0,50 ;* $P(code_4|$ *'shipyard')=0,35 ; .....* +

Carpenter + at + Shipyard : Weight($code_2$)=0,71 ; Weight($code_1$)=0,60 ; .....

Or in a formula:

$$Weight(Code_i) = \sum_j P(Code_i \mid Word_j)$$

The probability vectors for the individual words are stored in index files containing the conditional probabilities $P(Code_i|Word_j)$. After calculating the weights for each code based on the search string, the code descriptions of codes with the largest weights are then presented to the respondent if the combined weight of the first (say) 6 codes is above a certain threshold. The respondent then selects a particular code description and a code is uniquely established. In case the combined weight is too small there are too many codes possible for the given answer and these can not all be presented; in that case it is better to ask additional information in order to limit the number of possible code descriptions.

The main advantage of using text- and meta-classifiers is that they represent a cost-effective technique as far as operational and maintenance costs are concerned. Another advantage is that the use of these classifiers does not require a deep understanding of the classification problem at hand. Accurate training data are not necessary although "noisier" data degrades the classifier performance. However, there are some drawbacks: the cost of developing text- and meta-classifiers can be high. And often the expertise needed for development and implementation is not available at a statistical office. Moreover, a lot of data are needed to train text-classifiers and the technique does not increase our knowledge about the coding process itself.

2.2. Using registries

The most cost effective approach to coding is probably by using results that have already been collected elsewhere. For example, if one is interested in coding the economic activity of one-man businesses in the Netherlands one can link the complete files of the population and business administrations using the social security number as the connecting key. Although in principle registries could work well for coding purposes one usually faces the following problems: the registry does not exist (yet), the registry does exist but is not available to the statistical office, the quality of the linking field(s) or the target variables is insufficient, or the information is outdated. In the latter case there is still need of a questionnaire and the respondent should be asked whether the (outdated) information is still valid. Formulating appropriate questions is not always easy. Also one has to link sampling frame and registry in order to identify sampling elements for which the information in the registry is applicable.
On can also use the registry as a search file which contains information about the item to be coded. For example, in the case of businesses, we use a search file that consists of records containing information about business names (legal and trade