Organisation of an Extensive BLAISE Project

Holger Hagengooth, Federal Statistical Office Germany

Table of contents:

1. Introduction

In the German official statistics, BLAISE is used, amongst other things, in two large, extensive surveys: the Microcensus and the ongoing sample surveys. These two surveys receive considerable attention from the political sphere, and the data gathered are requested and further processed by many social science institutes. BLAISE is therefore of great strategic significance to the German official statistics.
In May 2000, I assumed responsibility for the technical aspects of the German Blaise application for the Microcensus.
Until then, the BLAISE application had been developed by the specialist department, largely without support from the IT Department. As the years passed, the specialist requirements made of the BLAISE survey program became ever more stringent, and new possibilities were identified, whilst at the same time the procedures previously carried out on the mainframe were transferred into the BLAISE application. The German BLAISE application reached a level of complexity that could no longer be managed by the specialist department alone. The whole application had to be completely reorganised in line with modern, abstract principles of software development, including Internet- and object-orientated technologies, in order to retain its ability to be deployed flexibly and maintainably. This article explains how these principles are implemented in BLAISE, and which concepts of BLAISE can be used for this. My many years of experience in managing complex EDV projects in the German official statistics were very helpful here. Since there was very little knowledge of many of these possibilities among German BLAISE programmers, it appears to be very necessary to point this out in particular. Many of the ideas put forward in this article are of value in my view for all the more complex BLAISE applications. Guidelines should be developed for other German BLAISE applications on the basis of this paper: for example, to ensure that all applications in future use modern principles and that the software development is organised effectively, standard modules can be developed allowing the wide variety of features offered by BLAISE to be put to better use in the German official statistics.

In Chapter 2, I will report on the organisation of the German Microcensus and the resulting requirements made of the BLAISE program, and in Chapter 3, I will introduce the desired aims and principles of modern software development, then in the following Chapters, I will describe how these goals can be implemented in BLAISE.

2. The German Microcensus

Roughly one percent of the German population is surveyed once every year in the German Microcensus statistics (by data on identity, family connections, economic and social situation, training, occupation, residence, health, etc.). The selection of the surveyed population is effected by a mathematical random sample procedure.

The list of questions changes each year, several new questions being added and several being deleted, whilst some old questions are re-activated.

The German official statistics are organised in line with the principles of Federalism. The actual survey of the data is effected in the Federal Länder and the local Land Statistical Offices. The Land Statistical Offices provide the data to the Federal Statistical Office, where they are combined. In accordance with federalist principles, the Federal Statistical Office primarily has the role of coordinator for the State Offices. The technical equipment of the individual State Offices varies greatly, and the organisation of work within the individual authorities is based on a variety of structures. In order to ensure that the data in Germany can be combined and surveyed in an uniform manner, the same BLAISE program is used to collate the data in all State Offices. This BLAISE program is developed centrally at the Federal Statistical Office, and then provided to the 16 Land Statistical Offices, where it is installed and used. No programming is carried out in BLAISE in the 16 State Offices, so that only prepared objects are provided, and no sources. The various sets of technical and organisational conditions at the Land Statistical Offices must be taken into account in developing the application at the Federal Office.

The population is surveyed by interviewers who visit the households in person to collect the data either with a laptop or a paper questionnaire. Data are also transferred from the laptops to the Land Statistical Offices in very different ways across the Federal Länder (modem or diskette, daily or at the end of the survey). The data collected using paper questionnaires are integrated into the BLAISE program in the Land Statistical Offices.

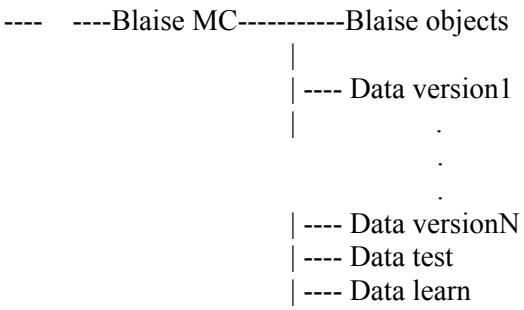
These points lead to the following requirements for the BLAISE program:

1. Flexibly controlled data model (list of questions)
2. The program should be deployable in and adaptable to a wide variety of technical and organisational structures.
3. The program should be adaptable to various environments by environment variables and a profile data model.
4. Installation of the application should be simple and flexible, largely without specific requirements.
5. It should be possible to distribute the application in various network environments.
6. Data backup and transfer largely independent of technical requirements.

3. Goals of the reorganisation of the German Microcensus Blaise application

It should be possible to maintain a strict separation of the Blaise objects and the data objects within the application, so that several versions can be processed, test data can be prepared, and learning data can be set up, etc.

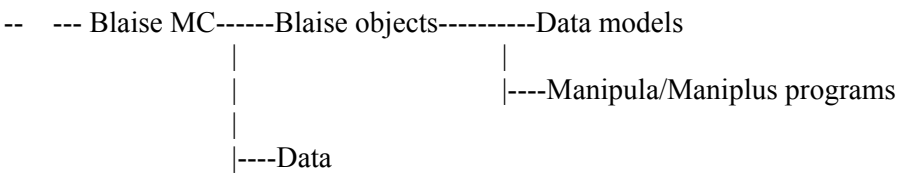
This can be achieved using the following folder structure:



When the application is launched, a selection box is displayed in which the available data folders are offered for selection.

A layered model is followed; this seeks to ensure a separation of the data storage between the application level and the presentation layer.

This can be achieved using the following folder structure:



Data storage : data
Application level : Manipula/Maniplus programs
Presentation level : data models

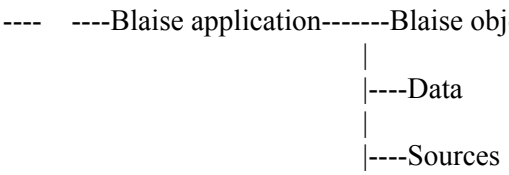
A Blaise object (data model or Maniplus program) should be perceived as a separate object, with its own interface, identified only by a name (address). The object's pathname and the drive on which it is located should be irrelevant to the running of the application.
For each BLAISE object, therefore, an interface file is created with the necessary external objects (USES, EXTERNAL, INPUTFILE, UPDATEFILE, OUTPUTFILE, LIBRARIES); from the name (identifier or address) of this object the complete filename (including path information) is dynamically formed and dynamically entered into the call command (RUN, CALL and EDIT). Thus, the application can be distributed at will and deployed in a wide variety of network architectures; objects can be called up from standard libraries, etc.

4. Folder structures

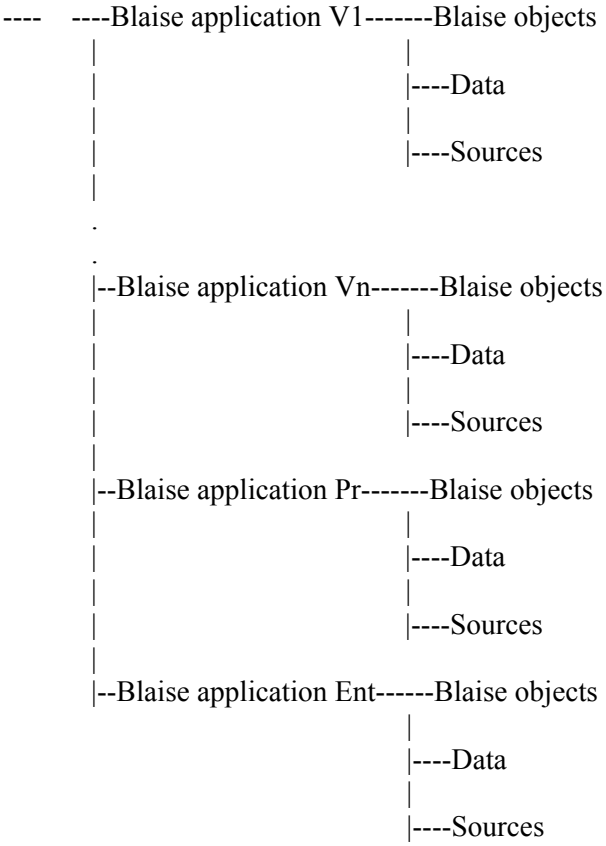
BLAISE has no CASE TOOLs, that is to say, tools that generate the application from an abstract diagram (such as an entity relationship). Indeed, BLAISE has no need of this facility.
It is possible to use a PC tool in order to give a graphic view of the structure of the application, but it is probably sufficient to structure and order the application using a good folder structure. With the aid of a hierarchical folder structure with informative folder designations, the logical structure of the application and its functionality become very clear.
Error search and maintenance are therefore made easier.

Some information has been provided above regarding the folder structure; a proposal can be found in the BLAISE DEVELOPERS GUIDE on p. 678.

On principle, a Blaise application should have the following structure:

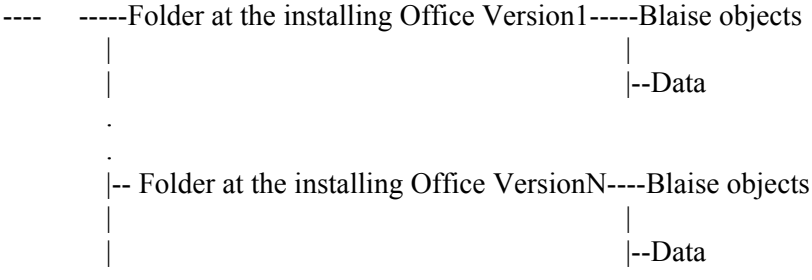


The advantage of such a structure becomes clear if only by virtue of the fact that it is very simple to operate several versions of the application in parallel, such as a production and a development environment; in each version the right sources are available for the Blaise objects:



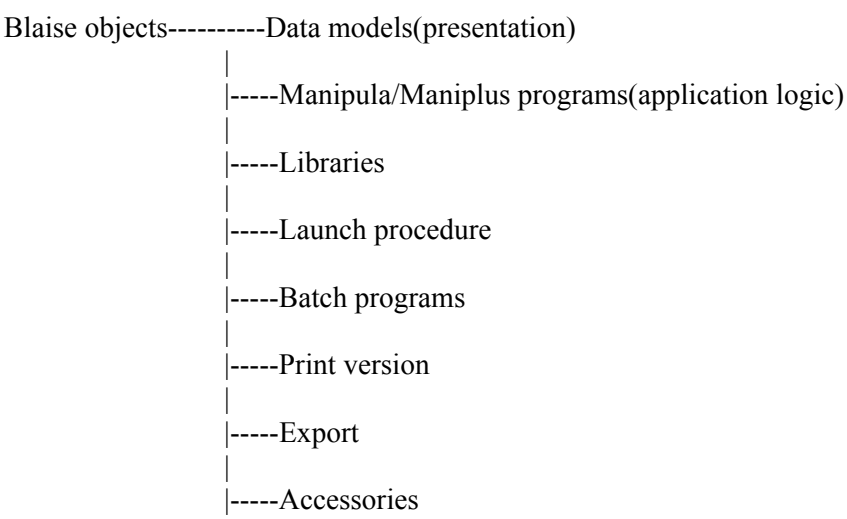
The names "Blaise application V1, Blaise application Vn, Blaise application Pr, Blaise application Ent" are arbitrary and can be selected as desired.

Such a structure is also extremely useful for installing the application in another authority; all you need to do in that authority is to create a folder corresponding to the name conventions of the authority; the folders "Blaise object" and "Data" are copied under this folder and the application can run; several versions can be set up without difficulties:



As stated above, the names "Folder at the installing Office Version1, Folder at the installing Office VersionN" can be selected at will.

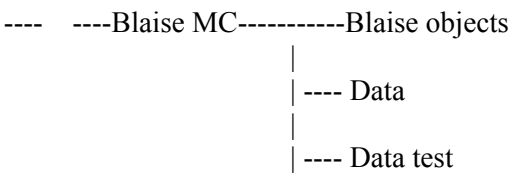
In order to clarify the structure of the application, the Blaise objects folder has been further sub-divided:



In an organisation such as the German official statistics, guidelines and rules should be defined for this subdivision to enable rapid orientation in all Blaise applications within the organisation. Thus, standard modules can be developed and synergy effects can be used.

5. Data organisation

The strict separation of data and Blaise objects was pointed out in item 3 above. In a Blaise data model, much metainformation, such as information on checks carried out, was stored in addition to the data. The effect of any change to the CHECK rules is that the old data become unreadable. This fact has been frequently criticised within official German statistics. This problem can however be solved by sensibly organising the data and the check data:



In the "Check data" folder, there is for each data model of the "Data" folder a data model containing only the fields of the data model, but not the check rules. If a change is made to the model in the "Data" folder, the data from the "Data" folder can be transferred to the "Check data" folder at any time. Once the change has been made, the data can be read back. If a field is changed in the "Data" folder, this must also be entered in the "Check data" folder. The "Check data" folder therefore serves as abstract data storage. Corresponding fields or notes can be used to enter the significance of the check data record and then searched for, functions that are not required in the "Data" folder. It is also possible to create the "Check data" folder as a database independently of Blaise (for example using ACCESS). The principle of saving and reading back works in the same way.

6. Pathnames in BLAISE

The pathnames and filenames in Blaise must be given particular attention so that the abovementioned goals can be achieved. Each object called up in Blaise must be fully identifiable in the network environment; the complete pathnames and filenames must be available.
With the need for flexible installation of the Blaise application, and installation of the application using any chosen folder name, the programmer does not know the pathname, so it must be determined when running the program. The full pathname must be put together implicitly or explicitly in the Blaise program when it is running.

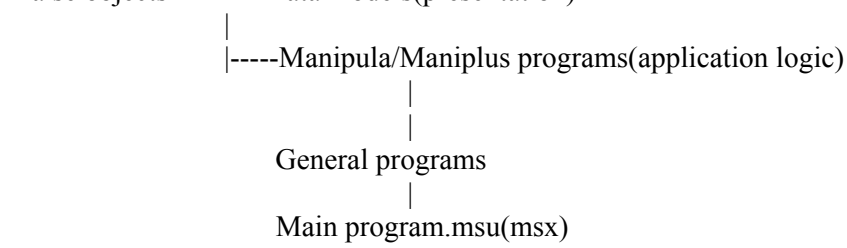
6.1 Dynamic pathnames

All pathnames that the programmer does not know, and all pathnames dynamically determined either implicitly or explicitly when running the application, are referred to below as "dynamic pathnames".

If the application is installed using the following structure:

the pathname of the drive up to "Folder at the installing Office Version1"
is a dynamic pathname.

6.2 Static pathnames



The path "Blaise objects\ Manipula/Maniplus programs (application logic) \ General programs" is known to the programmer who develops the project because he/she decides on the name.

6.3 Sub-pathnames

Consider the following example:

C:\Ord1\Blaise\Blaise Microcensus\Blaise objects\Manipula/Maniplus Programs\General programs\Main program.msu

The complete pathname is: "C:\Ord1\Blaise\Blaise Microcensus\Blaise objects\Manipula/Maniplus programs\General programs", whereas "C:\Ord1\Blaise\Blaise Microcensus" and "Blaise objects\Manipula/Maniplus-programs\General programs" are two sub-pathnames of the complete pathname. By combining (concatenating) the two pathnames, the complete pathname is created. In most cases, the complete pathname consists of a dynamic sub-pathname and a static sub-pathname. How the pathnames are divided into sub-pathnames, which of them are dynamic and which are static, which operations are defined for pathnames, is the basic project decision which determines the flexibility of the installation.

6.4 Pathnames in program development

If the source of a Blaise object is loaded into the text editor of the Blaise Control Center, the intention is to be able to implement the Prepare command directly. For this, all data models needed for conversion (USES section) must be found (i.e. with the full path). In the source code of the object, therefore, only the static sub-path should be stated. The dynamic sub-path should be permanently set in the project options. This means that the dynamic path for running the project must also be entered statically for development. Since the development takes place in a fixed environment, this is not so laborious; it only has to be entered once in the projects, and then applies to all objects within the project. If a new version of the project is also created for development, then in the new version only the pathnames in the project objects must be adapted; this is not a burdensome requirement.
If the B4Cpars.exe interface is used to prepare objects (for instance using a procedure) all dynamic sub-pathnames can be determined while the application is running and transferred as parameters. Therefore, the static entries in the options of the project are not needed. This is however not practical during development. A new version of the project is not needed that often.

6.5 Pathnames when running the program

All static sub-pathnames in the source of a Blaise object (USES, EXTERNAL, INPUTFILE, UPDATEFILE, OUTPUTFILE, LIBRARIES) that are permanently allocated can be overwritten while the application is running, and the commands CALL, EDIT and RUN can be transferred as parameters. The parameters are described in the Developers Guide on pp. 667 et seqq. Thus, the folder structure can be made flatter in the "Blaise objects" folder than in the "Sources" folder. When the application is running, a description of the structure may not be required in the same form as it is during development of the application in the "Sources" folder. Also, the division of the complete pathname into dynamic and static pathnames can be effected differently while the application is running than during project development.

7. Working folder

The working folder is the folder accessed by the application with MANIPULA.EXE. The working folder forms the basis for DOS commands made in the Manipula program, and sub-pathnames stated in the USES INPUTFILE refer to the working folder. The treatment and administration of the working folder is hence of fundamental significance for the organisation of the pathnames.

It is possible to determine the working folder during running by the following means:

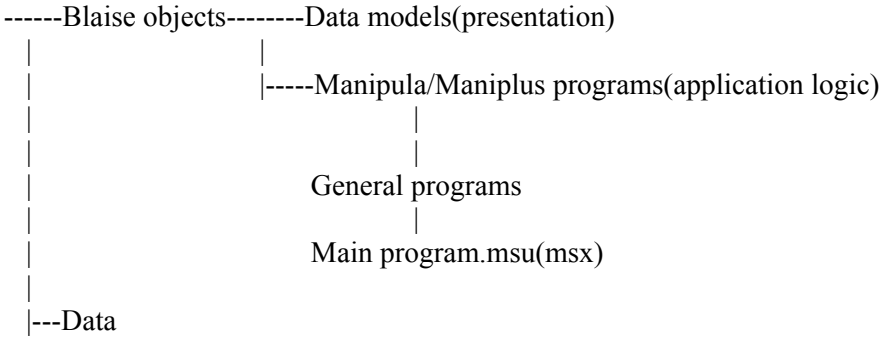
- to use the MS-DOS procedure and environment variables, which cannot entirely be done directly,
- in the Manipula start program to use a dummy file and the Manipula/Rules function pathname. This dummy file must not have any path information in the INPUTFILE section, and must be located in the same folder as the Start program.

It is possible to change the working folder during running by the following means:

- To use the Run and CD commands in a Maniplus program e.g.: Reslt := RUN ('CD STARTPROCEDURE' , WAIT)
- To use the parameter /W when calling up Manipula.exe, Dep.exe

If the working folder was set when launching the application in the Start program, it can be transferred using the parameter /P to all Maniplus programs and it is available there. If the Start program launches a Maniplus program in another folder, such as: Reslt := CALL('Manipula-Maniplus Programs\General_Programs\ AnzPC), the working folder of the start program remains the present working folder in the AnzPC program. If the AnzPC folder is intended to become the present working folder, /WManipula-Maniplus-Programs\General_Programs must be transferred as a parameter.

If the application is developed using the following structure:



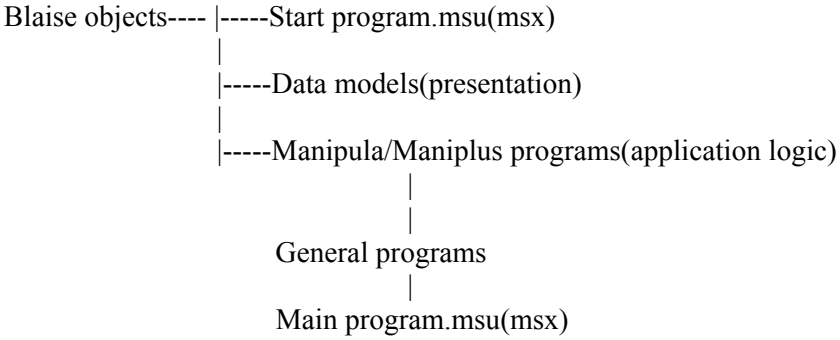
it has therefore proven to be practical to take the "Blaise objects" folder as the working folder for all objects. The Parameter /W is then not needed.
If the application is distributed more (standard modules), the working folder - as described above - must naturally be changed.

8. Management of pathnames in a Blaise project

In addition to the working folder, pathnames and filenames may be dealt with in different ways. Those possibilities I consider most important will be described, and their advantages and disadvantages discussed.

8.1 Information on the static filenames in the commands USES, EXTERNAL, INPUTFILE, UPDATEFILE, OUTPUTFILE, LIBRARIES, RUN, CALL and EDIT

In the structure:



the start program is launched in the Blaise objects folder.
The working folder is therefore "Blaise objects". Using Call, the main program is to be launched. All data models and files needed in the main program are stated with a full sub-path relating to the working folder.

Main program:

```
USES
    look         'Data models\Organisational data\Look\Look'
    doublecd   'Data models\Organisational data\doublecd\Doublecd'
    staff        'Data models\Organisational data\Staff'
    MaxP       'Data models\Microcensus data records\MaxP'
    err_lap     'Data models\Organisational data\err\Err_lap'

INPUTFILE    colleag:  staff('Data\Organisational data\Staff',BLAISE)  SETTINGS OPEN=YES
ACCESS=SHARED


INPUTFILE    looknb:  look ('Data\Organisational data\Look\Look',BLAISE)
                SETTINGS OPEN=NO
```

The effect of the CALL command ('Manipula/Maniplus programs\General programs\Main program') is that the name Manipula/Maniplus programs\General programs\Main program is implicitly supplemented dynamically during running by the working folder, and is correspondingly found and launched. The same applies to the data models and files. All names apart from the working folder are therefore static and stated permanently in the source code.

Advantages: maximum performance, very easy to understand, simple to install

Disadvantages: lack of flexibility, no way of distributing the application on a variety of drives, no differentiation between program development and running, and hard to change: if the structure is expanded or changed, all changes must be entered in the sources.

8.2 Static file names in an INCLUDE file

When a Maniplus setup is called up, the name of the chosen setup does not have to be explicitly stated; a STRING variable is possible.
All static pathnames/filenames can be listed in an INCLUDE file in string variables; the system setup can be highly flexible with a corresponding name convention for the string variables.

Definition of the string variables:

```
AUXFIELDS

{The variables for the filenames and pathnames are defined below.
Only relative pathnames are needed here for data files and MANIPULA/MANIPLUS SETUPS.
 All other relative pathnames are stated in the program.
 The following convention is introduced here:
 String : This is the character chain of the name
 Path    : This is a (relative) pathname
 File     : This is a (relative) filename
 from   : The path (filename) starts with the folder following 'from' (drive)
 to       : The path (filename) ends at the folder following 'to' (filename)
 The variables are set at the start of the program (in the 2nd INCLUDE file) so that in the event of a
change to the path structure only this variable setting needs to be changed.
 System refers to the folder in which the BLAISE application was installed.
 If the variables with + are concatenated, it is not possible to recognise immediately which path is
formed. A pathname starts with a letter and ends with a \  .}
```

```
String_Path_from_Drive_to_Blaise_Objects_to_Run          : STRING [255]
String_Path_from_Drive_to_Data                           : STRING [255]
String_Path_from_Data_to_Data                            : STRING [255]
String_Path_from_Classifications_to_Classifications      : STRING [255]
String_Path_from_Guide tapes_to_Guide tapes              : STRING [255]
String_Path_from_Organisational data_to_Organisational data : STRING [255]
String_Path_from_Organisational data_to_Save             : STRING [255]
String_Path_from_Organisational data_to_Copy             : STRING [255]
String_Path_from_Organisational data_to_Count            : STRING [255]
String_Path_from_Microcensus data records_to_Microcensus data records : STRING [255]
String_Path_from_Microcensus data records_Laptop_to_Microcensus data records_Laptop :
                                                                    STRING [255]
String_file_from_Manipula_Maniplus_programs_to_Start_MC3_Check  : STRING [255]
String_file_from_Manipula_Maniplus_programs_to_Main program     : STRING [255]
String_file_from_Manipula_Maniplus_programs_to_dataretg         : STRING [255]
String_file_from_Organisational data_to_Staff            : STRING [255]
String_file_from_Data models_to_Staff                    : STRING [255]
```

Occupation of the string variables:

```
String_Path_from_Drive_to_Blaise_objects_to_Run        := PARAMETER(1)
String_Path_from_Drive_to_Data                         := PARAMETER(2)
String_Path_from_Data_to_Data                          := PARAMETER(3)
String_file_from_Manipula_Maniplus_programs_to_Main program :=
          'Manipula-Maniplus programs\General_programs\Main program'
String_file_from_Manipula_Maniplus_programs_to_dataretg    :=
          'Manipula-Maniplus programs\General_programs\dataretg'
String_file_from_Organisational data_to_User               :=
              'Organisational data\User'
String_file_from_Organisational data_to_Staff              :=
              'Organisational data\Staff'
String_file_from_data models_to_Staff                  :=
              'Data models\Organisational data\Staff'
String_file_from_Organisational data_to_pcnum              :=
              'Organisational data\pcnum'
```

Accordingly, variables can be defined for calling up a SETUP:

String_Call_up _Main program : STRING[255]

And this call up variable is allocated according to the parameter conventions: String_Call up_Main program :=
String_File_from_Manipula_Maniplus_Programs_to_Main program + '/W' +
String_Path_from_Drive_to_Blaise_objects_to_Run + ...
The call up is then: CALL (String_Call_up_Main program)

Advantages: Separation of development and program running, easier to change since in the event of a change, only the entry in the String variable must be effected, and not in each separate object. No measurable performance disadvantages as against 8.4. Easy to install.

Disadvantages: Since the INCLUDE file is read when preparing the object and is permanently anchored in the prepared Blaise object, a flexible distribution of the application is not possible on installation. Somewhat more work is involved in programming.

8.3 Dynamic pathname search using an MS-DOS procedure or a database

As in 8.2, all call ups from Maniplus setups and links to external files are implemented via variables. These variables are set dynamically when running. There are very many ways of doing this. Basically, the system can be designed to be as variable as possible:

- Each object is given a name (a virtual address). Using this name, the current filename and pathname is found in a database. Hence the application can be installed and distributed with considerable flexibility. If the application is distributed differently, only the entries in the database need to be changed, and the application can run. In the same way, the external objects needed by a program are entered, and the interface is built up dynamically in line with the name (using a separate interface file for each object).
- The object is found in selected directories, for instance by an MS – DOS procedure, and the complete pathname and filename is set in this manner. The same procedure can be followed with the required external objects for calling up the program.
- The system administrator can use a profile data model to enter drives and pathnames.

Advantages: Separation of development and program running, easy to change, highly flexible in distribution and installation.

Disadvantage: More burdensome to administrate, higher performance, harder to install.

Since this is a procedure that does not depend on projects, the procedures that have been developed (modules) can be provided for all projects. The greater effort needs to be made only once.

9. Data model

Various procedures can be used when building up a Blaise data model. It is possible to program following the progression of the questionnaire, such as:

```
IF EF1 = 1 AND EF2 = 1
  THEN
     EF3
     EF4
  ELSE
     EF5
     EF6
ENDIF
```

This leads very quickly to conditional constructions that are hard to follow.

Since the display of the fields and their sequence is set statically in the electronic questionnaire, a field-related view is recommended:

```
IF EF1 = 1 AND EF2 = 1       { enquiry from EF3 }
  THEN
     EF3
ENDIF
IF EF1 = 1 AND EF2 = 1       { enquiry from EF4 }
  THEN
     EF4
ENDIF
IF NOT(EF1= 1 AND EF2 = 1)       { enquiry from EF5 }
  THEN
     EF5
ENDIF
IF NOT(EF1 = 1 AND EF2 = 1)       { enquiry from EF6 }
  THEN
     EF6
ENDIF
```

For each field, the condition is given in its entirety as to when it is queried, when it is displayed, when it is possibly deleted, etc.
The core of this method of observation lies in the field and not the enquiry logic. The application is given a clearer structure. After each field enquiry, the checks are performed. The checking logic is unlinked from the enquiry logic as far as possible. The core of the check logic is a complete test. Each test is effected using its own check routine in which all fields that are important in the check are used as parameters.
Below, therefore, the method is described that is used in developing the highly complex German data model for the Microcensus.

9.1 Building up a complex data model

- Determining the field sequence to follow a possibly existing paper questionnaire
- It is determined for each field when it is queried, when it is displayed, etc.
- For each field, the plausibility checks are described and programmed in a separate routine referring to this field and to all previously enquired fields.

Program example: enquiry logic:

{Enquiry of the Change field}

```
IF Result = Implem AND HHNumber <> 00 OR
        Result = questioned
  THEN
    IF RedV = 1 OR RedV = 2 OR RedV = 4
      THEN
        Change
    ENDIF
ENDIF

        {The following checks refer to the fields:
        Result ResultVJ HHNumber HHNumberVJ Redv Change}

Check_BG09 ( HHNumber, RedV, Result)
```

{N.B.:
The result ´GoneDead´ may not apply to a household newly included in the survey.}

Pruef_BG10b (RotV, HHNumber, HHNumberVJ, ResultVJ, Result)
{N.B.:
In 2000, the household had the questionnaire result ´^ResultVJ´ (gonedead).
Therefore there cannot be a result to the questionnaire for 2001.}

Check_BG10a (RedV, HHNumber, HHNumberVJ, ResultVJ, Result)
{N.B.:
A household number used the previous year for an empty dwelling can only be
used for the dwelling if it is still unoccupied.}

Check_BG12 (RedV, HHNumber, HHNumberVJ, ResultVJ, Result, Change)
{N.B.:
The new inclusion of the household was accepted although the questionnaire result in
2000 was "questioned" or "cancelled".}

Program example: Check routine

```
{Check BG04}

PROCEDURE CHECK_BG04
  PARAMETERS
    PHHNumber : THHNumber
    PResult    : TResult
    PNo.Pers : TNo.Pers
    PKomp2    : TKomp2
  RULES
    IF PResult = Implem AND PHHNumber >= 01 AND PHHNumber <= 98
      THEN
        IF PNo.Pers <> DONTKNOW
          THEN
            PKomp2 = DeuOA OR PKomp2 = DeuHw OR PKomp2 = DeuNw OR PKomp2 =
           Ausl OR
            PKomp2 = StAoA
            "This category cannot be correct since the household size
             (^PNo.Pers person(s)) is stated!@/
             If no further information is available, category '5'
             (nationality of reference person unknown) is to be stated.(BG04)"
        ENDIF
    ENDIF
ENDPROCEDURE
```

9.2 Metadata model

If the data model is structured as in 9.1, it is very simple to control the data model and, for instance, to activate and deactivate checks and blocks, etc. It is much more difficult, if not actually impossible, to control enquiry logic.
For the data model, a metadata model is formed, containing a field for each block and each check, which is used as a switch determining whether the enquiry or check is to be implemented:

```
Check:
IF P_F_Aus.S_P_Check_No.pers  = Yes
          THEN
            Check_No.pers (No.pers)
ENDIF

Enquiry:

IF P_F_Aus.S_B_FNR = Yes
                THEN
                  TFNR
ENDIF
```

10. Tools

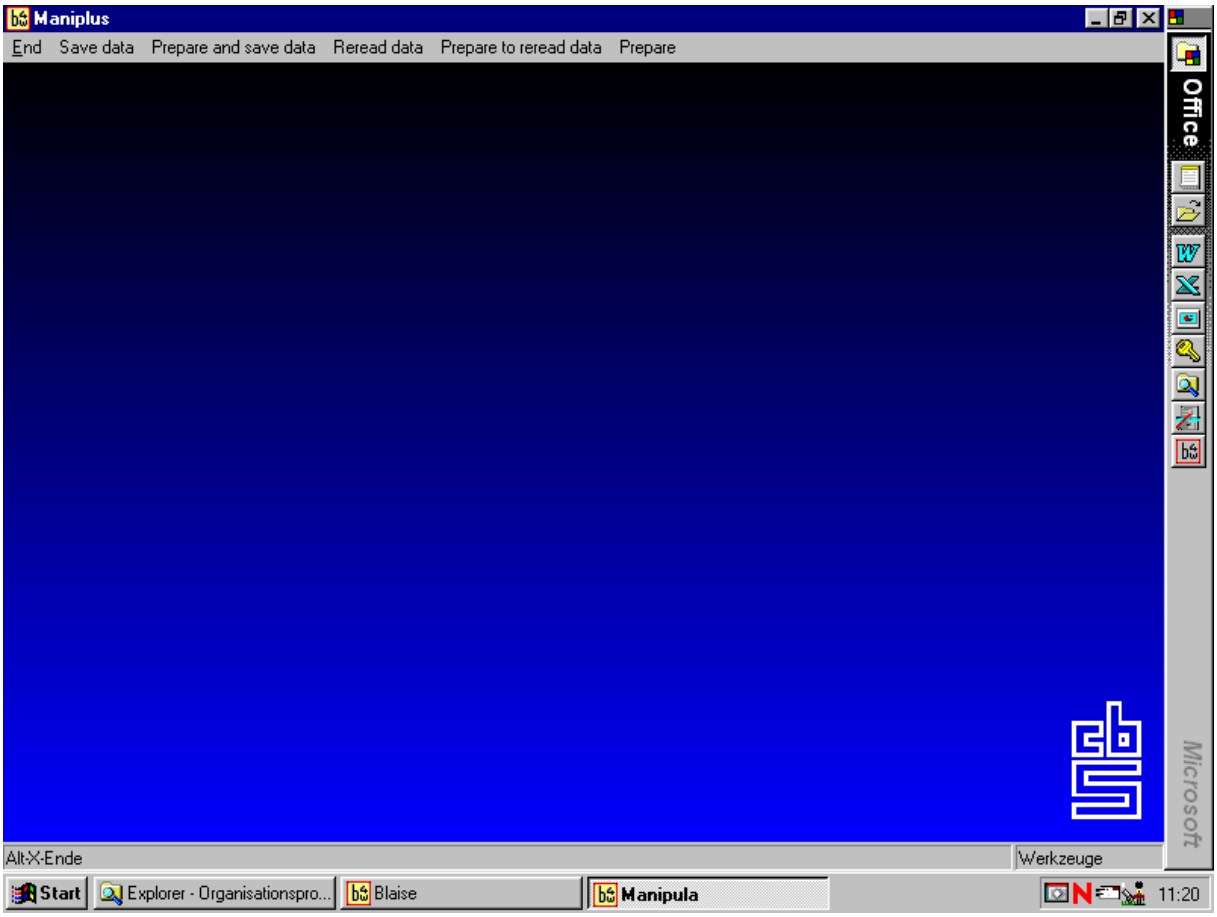
If the Blaise application is developed using the following structure:

```
----    ----Blaise application------------Blaise objects
                                          |
                                          |----Data
                                          |
                                          |----Data test
                                          |
                                          |----Sources
```

it is possible to create extremely effective, powerful tools with which the development can be greatly accelerated.

- A tool to prepare all objects in a batch.
- A tool to prepare selected objects, such as all Blaise source files containing a text string. If a data model is changed, all Maniplus objects in the USES section needing this data model are found and have to be newly prepared.
- If the "Data check" folder is organised as an abstract data store – cf. 5. – tools can be formed to store and re-read the test data. If a check rule is changed in a data model, the data are transferred after a data check. The data model is changed; the data are re-read and are available once more.
- Tools to manage the project automatically, for example to compare the "Sources" and "Blaise-objects-to-run" folders. If for each source an object in exists in "Blaise-objects-to-run", and vice versa, the versions are correct.

All these tools can be launched using a Maniplus application which is available and can be used constantly during development:


Maniplus
End
Save data
Prepare and save data
Reread data
Prepare to reread data
Prepare
Office
Microsoft
Alt-X-Ende
Werkzeuge
Start
Explorer - Organisationspro...
Blaise
Manipula
11:20