Making the most out of Manipula-Maniplus

Fred Wensing, Independent IT contractor, Australia

1. Introduction

Most people who are introduced to Blaise soon become aware of the questionnaire language and its capabilities. They start to develop questionnaires and hope to start collecting data. Before long they realise that they will need to draw on the other components of the Blaise software to help them load the input data, extract the output data and in many cases help them to manage and run the survey.

If they have their own survey infrastructure they may look at the Blaise API and its features. But not every organisation has the infrastructure or skilled staff to work with the Blaise API. That is usually the first time when they think that Manipula may help, so they may turn to the Online Assistant or the Manipula Setup Wizard.

For many people using the Blaise software, the Manipula Setup Wizard is their first exposure to the Manipula language and capability. However, the Wizard only handles some simple data conversions. There is much more to Manipula than is found in the Wizard. In particular there is a whole suite of statements and functionality that has been added to Manipula, called Maniplus.

Maniplus has been a feature of Blaise since version III and it has been extended with every release since then. Maniplus now provides comprehensive functionality which can enable full systems to be developed with little reliance on other software. This paper will highlight some of the features of Manipula-Maniplus that make it an excellent tool for developing systems.

This paper is not aimed at replacing or competing with the comprehensive documentation in the Blaise Online Assistant but will just highlight useful features that some people may not be aware of to encourage you to give it a go.

2. Basic description of Manipula-Maniplus

Manipula is a processing language that can be used to carry out manipulations of Blaise and other data files (ASCII, XML, OLEDB). It can combine data, derive new data, sort records, define filters, and perform complex computations.

A basic Manipula program consists of a list of settings, datamodels and file definitions followed by some processing logic which is contained within a MANIPULATE step which contains the main program logic. The program logic can be extended with a PROLOGUE, SORT and/or a second MANIPULATE step for text file processing. Additional functionality can be obtained using PROCEDURE sections which are called from either of the MANIPULATE steps.

The Maniplus language encompasses almost all of the Manipula language with extended statements that provide for interaction with the operator (through dialogs or menus) as well as the ability to call other programs, file management and other special functionality.

A Maniplus program commences with the keyword PROCESS and the main processing logic is contained within the MANIPULATE step. Flexibility is obtained within Maniplus through the use of PROCEDUREs or calls to other programs.

3. Blaise datamodel awareness

The overriding advantage of using Manipula or Maniplus is the ability to interact directly with Blaise datamodels. To access any Blaise file you just need to mention the relevant datamodel in the USES section and associate it with the corresponding file (INPUTFILE, OUTPUTFILE, UPDATEFILE etc). The file can be a native Blaise data file or one that can be accessed using OLEDB Information file (BOI file). The latter capability is part of the Datalink features of Blaise Components (licensed separately).

Text files (ASCII, ASCIIRELATIONAL, FIXED, XML) can also be accessed using a Blaise datamodel but that datamodel can be defined locally (within the program) rather than separately, if so desired.

4. Some auto-execute features of Manipula

Manipula was developed to make some basic processing easy to do. For that reason there are some auto-execute features of Manipula. I will call them AUTOread, AUTOcopy and AUTOwrite. The first two of these are settings which you can change to YES or NO. The last is not a setting but a behaviour outcome if you leave out the MANIPULATE step.

These auto-execute features can make the programming of Manipula quite easy, provided that you know they are active (or not). They can, however, be a source of confusion for inexperienced programmers which is why I suggest that the AUTOREAD and AUTOCOPY settings should be specified with their relevant setting at the top of the program, even if you are using the default settings.

In Maniplus AUTOREAD is always set to No, so file reading is your responsibility. That can readily be handled using READNEXT and the REPEAT UNTIL logic:

```
Infile.OPEN
REPEAT
   Infile.READNEXT
   <other statements>
UNTIL Infile.LASTRECORD
```

AUTOCOPY is a great feature that copies the input fields automatically to the output fields with matching names, immediately after reading the record (by any method). This means that most data transfer can happen “automatically”. Blaise does this by connecting the matched fields. Individual Blaise files can be withheld from the auto-copy feature by turning off by changing the CONNECT setting to NO.

5. Sort and summarise

Before I mention the features of Maniplus, I want to mention a useful feature of Manipula which is the ability to sort and summarise. This feature only applies to the first ASCII output file in a Manipula Setup. Sorting enables you to output your text file in whatever order you like. By adding summary keywords such as SUM, MEAN, MAXIMUM, MINIMUM, STDDEV, VARIANCE and MEDIAN to your sort variables you can readily obtain some summary data that can be used for reporting. While the Blaise software no longer includes a tabulation component this feature of Manipula allows for some basic summary data to be obtained. If you output the data to CSV or Fixed format you can produce some simple reports for a system.

6. Dialogs – the visible interactive part of Maniplus

The most useful aspect of Maniplus is the addition of Dialog boxes which can be used to interact with the operator. Early versions of these were basic but the flexibility has improved greatly over the years.

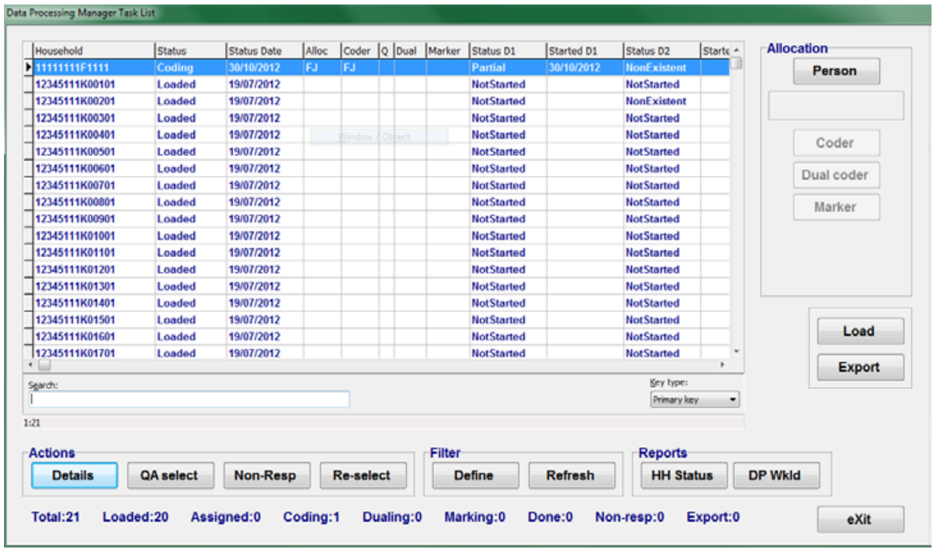
The dialog boxes can show a list of records, content of fields, text elements and buttons to enable actions to be initiated. The elements can be made visible and/or enabled based on Boolean expressions. The placement of elements on the screen is done using x,y coordinates that are measured in pixels.

With Dialog boxes it is possible to create a comprehensive interface that can provide the user with access to lots of functionality.

Dialog boxes can also contain multiple lookups which can be synchronised by using 'on change' instructions to refresh the lookup entries.

Some examples are shown here.

Example 1. Typical screen to manage a survey list with action buttons, filters and record counts.



Household	Status	Status Date	Alloc	Coder	Q	Dual	Marker	Status D1	Started D1	Status D2	Starte
11111111F1111	Coding	30/10/2012	FJ	FJ				Partial	30/10/2012	NonExistent	
12345111K00101	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K00201	Loaded	19/07/2012						NotStarted		NonExistent	
12345111K00301	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K00401	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K00501	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K00601	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K00701	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K00801	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K00901	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K01001	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K01101	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K01201	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K01301	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K01401	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K01501	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K01601	Loaded	19/07/2012						NotStarted		NotStarted	
12345111K01701	Loaded	19/07/2012						NotStarted		NotStarted	

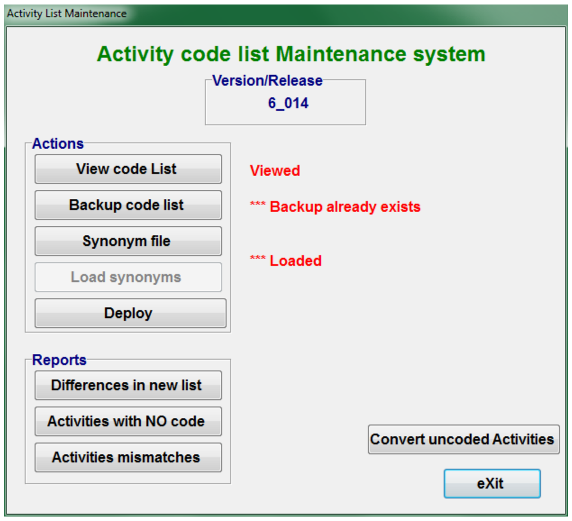
1:21

Total:21 Loaded:20 Assigned:0 Coding:1 Dualing:0 Marking:0 Done:0 Non-resp:0 Export:0

Buttons and objects are made visible or enabled using Boolean conditions attached to the button:

```
BUTTON aButton1 VALUE=bAllocA
    CAPTION='Coder'
    ENABLED=((aRole<>EMPTY) AND (aOneCoder<>EMPTY))
BUTTON aButton1 VALUE=bCHStatus
    VISIBLE=(aTestMode='ON')
```

Example 2. Small utility screen for code list maintenance.



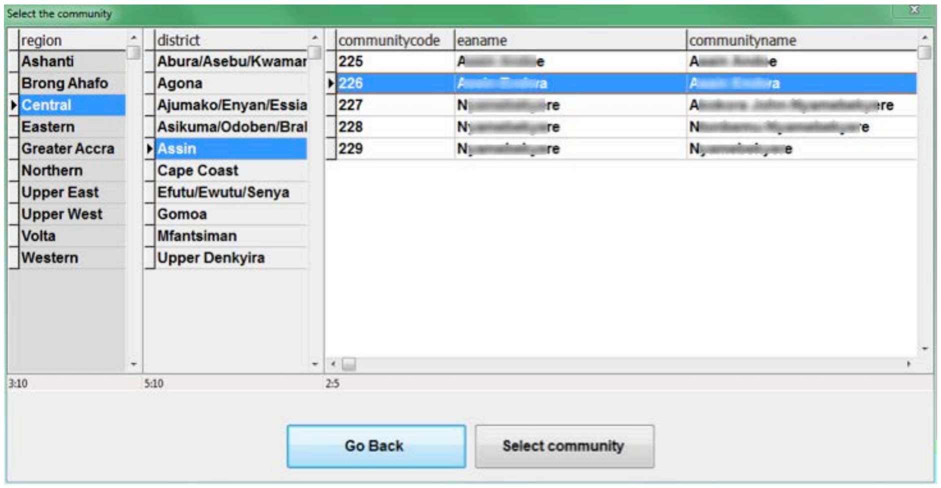
Viewed

*** Backup already exists

*** Loaded

Buttons are intended to be used in sequence and outcomes are reported back to the screen.

Example 3. Selector screen showing three synchronised lookups.



region
Ashanti
Brong Ahafo
Central
Eastern
Greater Accra
Northern
Upper East
Upper West
Volta
Western

3:10

district
Abura/Asebu/Kwamar
Agona
Ajumako/Enyan/Essia
Asikuma/Odoben/Bral
Assin
Cape Coast
Efutu/Ewutu/Senya
Gomoa
Mfantsiman
Upper Denkyira

5:10

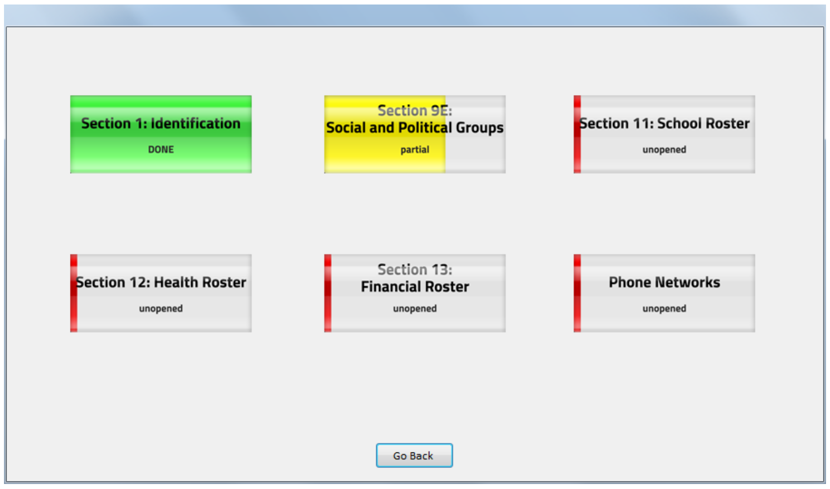
communitycode	eaname	communityname
225	[redacted]	[redacted]
226	[redacted]	[redacted]
227	[redacted]	[redacted]
228	[redacted]	[redacted]
229	[redacted]	[redacted]

2:5

Changing the selection in lookup columns 1 and 2 will cause immediate change in the value of the other lookup columns. This is done using temporary files for each lookup and reloading them using ONCHANGE to activate a procedure to do the reload:

```
lookup tmpR
    size=(150,520)
    onchange=prc_districts
lookup tmpD
    size=(200,520)
    position=(150,0)
    onchange=prc_Communities
lookup tmpC
    size=(674,520)
  position=(350,0)
```

Example 4. Screen showing buttons that are color coded for progress. These buttons are displayed using a set of graphic images.


Section 1: Identification
DONE
Section 9E:
Social and Political Groups
partial
Section 11: School Roster
unopened
Section 12: Health Roster
unopened
Section 13:
Financial Roster
unopened
Phone Networks
unopened
Go Back

The graphic images are stored in a resource folder and activated using APPEARANCE and SRC options. The references to the images are changed by the program so they will differ depending on progress through a section:

```
BUTTON aTryButton SIZE=(175,75) POSITION=(70,50)
   APPEARANCE = IMAGEBUTTON
   SRC = "^trybuttonfile[7]"
   VALUE = bu_s02Land
   CAPTION = ' '
```

7. Menus – less visible but functional

Another useful aspect of Maniplus is the provision of Menu facilities. These are activated by applying the MENU method to a file containing the menu specifications. Selecting from the menu effectively selects one of the records from the file. The selection can be tested and actions or procedures activated.

The attributes of each record in the menu file define the action that will happen.

The Blaise Online Assistant and sample files can show you how to implement this feature.

Use of menus is a little limiting because you cannot have them on the same screen them together with action buttons.

I must admit that I prefer the Dialog box functionality over the Menus when it comes to building systems for inexperienced users because dialogs are more visible and interactive.

8. Parameters

The operations of any Manipula program can be tailored through the use of parameters. This is particularly useful for passing information from other systems or between programs.

The use of parameters makes it easier to break down your system into manageable parts. It enables those parts to be developed and tested separately, and sometimes run stand-alone, before integration into a system.

Parameters are passed into a Manipula or Maniplus program via one of the command line options (/P). Multiple parameters are separated by the ‘;’ character.

Parameters are accessed using the PARAMETER function with a reference to the parameter number. It is useful to set some default values in case the parameters are not invoked. Code like this can be put at the top of the MANIPULATE step:

```
{Check parameters for system settings}
IF PARAMETER(4)<>'' THEN
   aTestMode := PARAMETER(4)
ELSE
   aTestMode := 'OFF'
ENDIF
```

Aside from the user defined parameters, Manipula also has a system PARAMETER(0) which returns the name of the program and path which is executing the Manipula/Maniplus Setup. For example:

```
aManiProgram:=PARAMETER(0)
```

on my computer this returns:

```
C:\Program Files\StatNeth\Blaise 4.8 Enterprise\Bin\Manipula.exe
```

This result can conveniently be used to extract the path to enable Maniplus to call other Blaise executables like the B4CPars.EXE or CAMELEON.EXE.

9. Command Line options and Command Line options File (BCF)

In addition to parameters the command line can also be used to affect changes to the way that the Manipula or Maniplus program operates.

You can supply the names of files to be used (using /I and /O or /N), the names of working folders (/W) or search paths (/E) and other operational behaviour. In this way the same program can be used to operate on different files. For example:

```
CALL (‘myprog.MSU /CC:\Surveys\Config\General.miw’
    + ‘ /EC:\Surveys\External /HC:\Surveys\MySurvey’
    + ‘ /WC:\Surveys\MySurvey’)
```

When the command line options become numerous it can be easier to put the options into a small text file and reference that file using the command line option of @<filename>. The format of such a file follows the style of an INI file with the structure

Option_name=value(s)

The full list of Manipula command line options (and others) can be found in the Blaise Online Assistant under “Miscellaneous / Command line options”.

A typical use of this feature may be to set the working folder, configuration files and meta search path. For example, the above command line settings would look like:

```
[ManipulaCmd]
ConfigFile=C:\Surveys\Config\General.miw
ExternalSearchPath=C:\Surveys\External
MetaSearchPath=C:\Surveys\MySurvey
WorkFolder=C:\Surveys\MySurvey
```

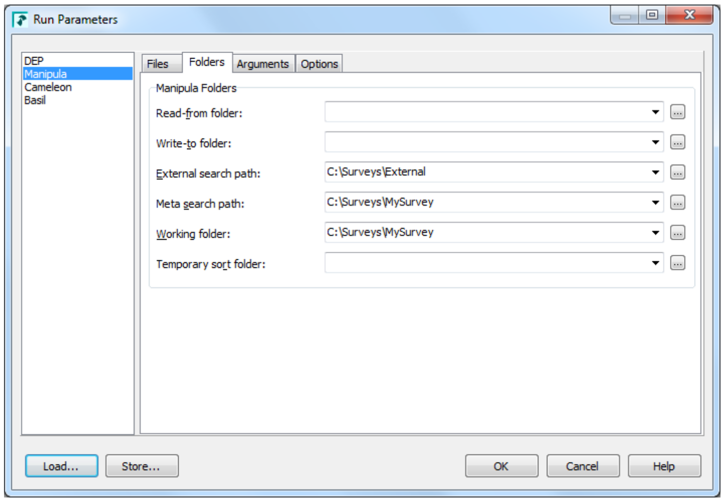
The first line of this file indicates that the settings/options apply to Manipula/Maniplus.

When the Manipula is invoked this file of settings is passed in using @<filename> in the Call to Manipula, namely:

```
CALL (‘myprog.MSU @Myprog.BCF’)
```

Blaise Command line options file (BCF) goes one step further than parameters or Command lines in that they contain the command line and environment settings and/or run parameters for multiple applications (eg. for Manipula and DEP at the same time).

A BCF can be created from within a Blaise project by using the Store button at the end of updating the Run / Parameters. Of course you can also create one manually.



Manipula Folders

Read-from folder:

Write-to folder:

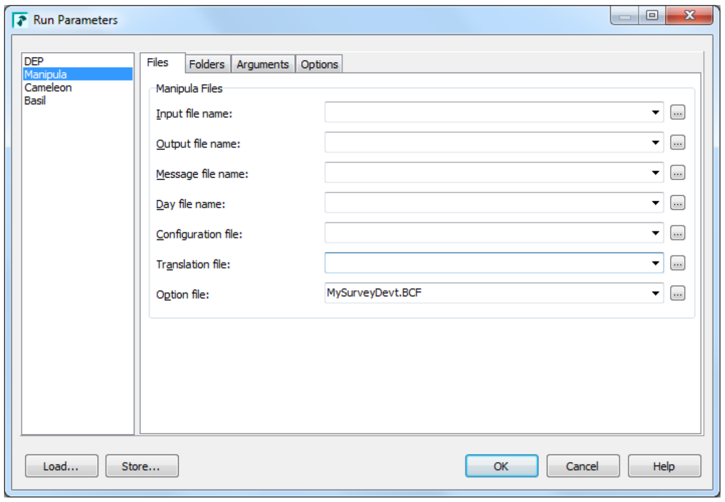
External search path: C:\Surveys\External

Meta search path: C:\Surveys\MySurvey

Working folder: C:\Surveys\MySurvey

Temporary sort folder:

The BCF can also be used in conjunction with a Blaise Project File to ensure that the relevant settings are applied while developing or testing your project. The BCF can be entered into the Option file entry in the Run Parameters screen.



Option file: MySurveyDevt.BCF

Using a BCF means that all the settings are stored in the one file. That BCF can be used in a Windows shortcut or in any call to a Manipula program.

Of particular use in these BCF files are the path (or folder) entries which avoid the need to have full path references in the names of data files or data models. By setting up different BCF files for Development, Test and Production environments you can simply move the application between environments without the need to recompile it each time.

Although the BCF file can be created using the Store button the file that it creates is just a text file and additional entries can be added if needed.

10. Using an INI file

If you have a Maniplus system then parameters can also be passed in at the start but that can make it difficult for an operator who doesn’t know what parameter settings to use. This can be resolved by setting defaults within the program or by providing a small initialisation text file (*.INI) containing various settings that control the system.

An INI file is a text file that contains a list of settings in the form:
Setting_name=value
Of course the Manipula program will need to read this file to obtain the settings. That can be done with a simple file definition and some logic to read the settings.

For example. The datamodel, file definition and procedure to read such a file could look like:

```
USES
    DATAMODEL mSettings
       FIELDS
          ItemName : STRING[20]
          ItemValue : STRING[200]
    ENDMODEL
INPUTFILE
    fSettings : mSettings ('',ASCII)
    SETTINGS
        SEPARATOR='='  OPEN=NO
```

```
PROCEDURE proc_settings
  fSettings.OPEN ('TUSDiary_System.ini')
  fSettings.READNEXT
  REPEAT
      IF UPPERCASE(ItemName)='VERSION' THEN
          aVersion:=ItemValue
      ELSEIF UPPERCASE(ItemName)='INSTALLER' THEN
          aInstaller:=UPPERCASE(ItemValue)
      ELSEIF UPPERCASE(ItemName)='SYSFOLDER' THEN
          aSysfolder:=ItemValue
      ELSEIF UPPERCASE(ItemName)='CODERLOCATION' THEN
          aCoderLocation:=ItemValue
      ENDIF
      fSettings.READNEXT
  UNTIL fSettings.EOF
  fSettings.CLOSE
ENDPROCEDURE
```

11. Controlling the start-up screen

When Manipula or Maniplus start-up the system brings up a splash screen. It is possible to change the splash screen colours, logo icon and default font through the MANIPULA.MIW file. The Manipula.MIW file is based on standard INI file structure.

The following is an example of a typical MIW file that uses full screen, shows a logo and sets the default font to bold 12 and has a splash colour gradient from purple to teal:

```
[Maniplus]
Maximize=1
OwnLogo=1
LogoName=AgencyLogo.bmp
FontSize=12
Bold=1
ColorStart=Purple
ColorEnd=Teal
FontName=Arial
```

This is the first way that you can customise your Manipula/Maniplus processes.
To find out more lookup “How to use… / Maniplus / Customising a Maniplus Setup” in the Blaise Online Assistant.

12. Running other programs

Maniplus can run other programs, either through the CALL instructions (for Manipula/Maniplus programs) or the RUN instruction which can execute any other type of program. And, of course a Maniplus program can be used to call the Data Entry Program (DEP) through the EDIT method.

Manipula/Maniplus components can be developed separately and connected to the main program by having it CALL those parts.

If non-Blaise programs are needed (eg. STATA or SAS) then they can be called using the RUN command.

These features give a Maniplus system lots of flexibility. When the programs you call or run are Manipula/Maniplus programs then the system is more homogeneous. More importantly the Blaise files that you use or create in one part are directly accessible to the other part of your system.

With the file handling extensions to the RUN function even some of the file management can be handled in Maniplus (see next section).

13. File handling

In the early implementation of Maniplus the only way to create folders and copy or delete files was to use the various command line statements to be executed by the CMD.EXE Windows command line interpreter.

From version 4.6 onwards, however, the RUN instruction handles a range of file management commands making it much smoother to handle file management tasks. Some of these commands include:

```
RUN('SETREADONLYFLAG FileName')
RUN('REMOVEREADONLYFLAG FileName')
RUN('DELETEFILE FileName')
RUN('MOVEFILE SourceName TargetName')
RUN('COPYFILE SourceName TargetName')
RUN('CREATEDIRECTORY DirName')
RUN('REMOVEDIRECTORY DirName')
RUN('DELETEBLAISEFILE  FileName')
RUN('MOVEBLAISEFILE SourceName TargetName')
RUN('COPYBLAISEFILE SourceName TargetName')
RUN('RENAMEBLAISEFILE SourceName TargetName')
RUN('DISCONNECTBLAISEFILE FileName [ IGNORESESSIONS ]')
RUN('DISCONNECTDICTIONARY FileName')
```

Of particular use are the instructions to copy or delete Blaise files because these are aware of all the file types which make up a Blaise file. The Blaise datamodel is unaffected when some of these file management commands are applied to the data.

```
{Close the data file before copying}
fMaster.CLOSE
fMaster.RELEASE
{copy the main Blaise data files to another location and name}
fMaster.COPYDATAFILE(C:\Surveys\Data_Local_'+aIDNumber)
```

In this example the COPYDATAFILE copies the Blaise data to the new location C:\Surveys with the name Data_Local_<aIDNumber>.

14. Zip utilities

Zip functionality was added to Maniplus with version 4.8 of Blaise. Zip and Unzip operations can now be performed seamlessly within a Maniplus application.

For example the following statements Zip the main data file to a backup folder with a date-specific name:

```
{set up date for zip file names}
aZipDate:=DATETOSTR(SYSDATE,YYMMDD)
{set up name of Zip file}
aFilenameOut:='Backup_Main_'+aZipDate

{set name of HH file to be zipped}
aFilename:='MasterHH'
{Prepare exclusion list for Zip files}
IF NOT (FILEEXISTS('Exclude_zip1.txt')) THEN
   Proc_CreateExcludes (aFilename,'Exclude_zip1.txt')
ENDIF
{Zip main HH file to backup using an exclusion list}
aResult := ZIPFILES ('"backup\'+aFilenameOut+'.zip"
         "'aFilename+'.*" /X@Exclude_Zip1.txt')
```

This example uses a small procedure called Proc_CreateExckudes to create a list of files to be excluded from the ZIP process and that file is invoked using the /X@<filename> option. The procedure is listed here for information:

```
{Create a file with the list of filetypes which don't need to be put into
the Zip file}
PROCEDURE Proc_CreateExcludes
  PARAMETERS
    Datafile : STRING
    Outfile : STRING
  INSTRUCTIONS
    fSetup1.OPEN (Outfile)
    fSetup1.TextData := Datafile+'.bat' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.bpf' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.bla' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.bdm' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.bmi' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.bxi' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.bdv' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.bdp' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.bww' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.buf' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.aif' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.adt' fSetup1.WRITE
    fSetup1.TextData := Datafile+'.~lk' fSetup1.WRITE
    fSetup1.CLOSE
ENDPROCEDURE
```

15. Late binding (or generic setups)

One issue that can be a problem is that Manipula and Maniplus programs are usually compiled with the datamodels that they need to access the various Blaise data files. This means that if you change one of the datamodels in any way you will need to recompile the program before you can use it.

Since version 4.8 of Blaise, however, Manipula can be prepared with the name of the datamodel specified as a variable provided on the command line at the time of execution. This feature is known as late binding which enables the program to be compiled without knowledge of the datamodel. A program which uses this feature is considered to be a generic setup because it can be used with any datamodel (provided it is compatible with the statements).

You will be restricted in the kind of statements that can be executed in such a program but it does provide the flexibility for creating stand-alone utilities that do not need to be recompiled every time you wish to use it.

To use this technique the USES statement will have the VAR option, for example:

```
USES
    {instrument metadata is passed as a parameter}
    mData (VAR) 'xxx'
```

Then when the program is executed the name of the metadata is passed in using the /K command line option:

```
MakeZipData.MSU /KmData=C:\Surveys\Mysurvey\MasterHH.BMI
```

With late binding the statements in the Manipula program you can only access fields from those files which use the variable meta using GETVALUE / PUTVALUE / GETREMARK / PUTREMARK. These statements do not produce an error if the field being accessed does not exist in the datamodel.

For example the statements here read data from a few fields and write values to other fields:

```
{check the dates and copy status}
aCopyFlag:=Survey.GETVALUE('xCopyFlag',UNFORMATTED)
aSurveyDate:=VAL(Survey.GETVALUE('xDateChanged',UNFORMATTED))

{other statements}

IF aCopyFlag='1' THEN {not copied yet}
    {update the xCopyFlag on the main survey file}
    Survey.PUTVALUE ('xCopyFlag','2') {copied}
    Survey.WRITE
    {save the OfficeData and update xCopyFlag}
    OfficeData.WRITE
    OfficeData.PUTVALUE ('xCopyFlag','2') {copied}
ENDIF
```

16. Metadata access

A very useful extension of Maniplus in version 4.8 is the ability to access the metadata of a datamodel through a series of GET methods. Some of them are:

```
GETFIRSTFIELDNAME
GETNEXTFIELDNAME
GETNEXTROUTEFIELDNAME
GETMETAINFO
GETFIELDINFO
GETTYPEINFO
```

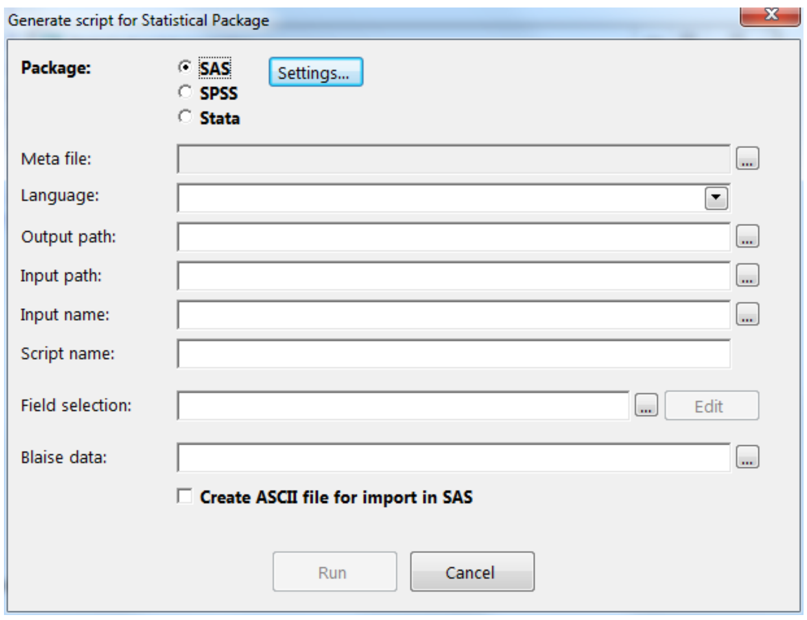
With these it is possible to use Maniplus to generate reports or construct other programs (in Manipula or other software language) to process Blaise files.

This new feature will eliminate the need to use Cameleon which is good because Cameleon will not be available in Blaise 5.

Recently a utility was completed that uses the Metadata extensions of Maniplus to a great extent. It is called the Statistical Script Generator and was produced by a collaborative team from the Blaise developers (Statistics Netherlands) and some Blaise users. I understand that the utility will be available in the next release of Blaise 4.8.4.

This utility can be used to produces a SAS, SPSS or Stata script to read exported Blaise data. It also has the functionality to do this for a selection of blocks/fields of a datamodel.

The main screen of this utility is shown here:


Generate script for Statistical Package

Package: [✓ SAS] SPSS Stata

Meta file:
Language:
Output path:
Input path:
Input name:
Script name:
Field selection:
Blaise data:

☐ Create ASCII file for import in SAS

17. Other features

Some other extensions of Manipula/Maniplus features which I will mention but I don’t have time to describe are:

- It is now possible to define functions using the FUNCTION specification
- The INTERCHANGE file setting allows sharing of files with the calling process (DEP or Maniplus)
- Maniplus can now interact with the active DayBatch files to add, delete or modify cases through various DAYBATCH file methods

18. Concluding remarks

Manipula / Maniplus has many features only some of which are described in this paper. Its main strength is its ability to handle Blaise files and Blaise metadata, but the functionality has been extended considerably over the years.

Manipula/Maniplus is well worth a second look if you are planning to build a system.

19. Acknowledgements

I would like to acknowledge that some of the examples used in this paper came from Lon Hofman (Statistics Netherlands) and Elana Cohen-Khani (ISSER - University of Ghana).