ManiTabs

Making tabulations in Ms-Excel with Manipula

T.R. Jellema
NAMES B.V.
Korte Raam 1A
2801WE Gouda
The Netherlands

TJELLEMA@NAMESBV.NL
HTTP:\\WWW.NAMESBV.NL

Abstract

Blaise datamodels and Manipula scripts have long had the capability to call DLL's through the alien procedure interface. The article explores the capabilities of Manipula alien procedures with regards to controlling OLE automation servers. As an example we use a Manipula script that automates Microsoft Excel's PivotTable in order to create cross tabulations.

Introduction

This article will illustrate for you the power and flexibility that you can obtain when you combine Blaise for Windows support for ALIEN PROCEDURES with DLL's that are OLE-Automation controllers. The subject matter is technical in nature as it involves the combination of two programming techniques. However we feel that the potential benefits of the application of the technique make it relevant to a wider audience.

The benefits of the technique are quite clear. In your Manipula/Maniplus scripts you can write statements that control other programs. The control you obtain can be quite extensive. You can transfer data into and out of the target program and you can issue commands to have the target program manipulate the data, or output it.

The technique is particularly relevant to offices that have invested heavily in the use of generic 'Office' software and require reports, tables, charts etcetera to be prepared in products such as Excel and Word, and e-mail communication to be performed by Exchange or Outlook. We list a few examples:

- Sending data as e-mail using Ms-Outlook
- Retrieving data from the e-mail system and placing it into Blaise datasets
- Tabulation of data from Blaise data sets using MS-Excel or SPSS-Tables
- Preparation of Charts using Ms-Excel or SPSS-Chart
- Preparing printed reports or form letters using MS-Word instead of using PRINT-Section formatting

In this article we would like to show you a non-trivial example to illustrate the power of the technique. A particular useful feature in MS-Excel is the pivot table. This allows you to interactively define tabulations on multidimensional data. The pivottable is a good example to illustrate the technique because:

- Ms-Excel is ubiquitous. Almost all statistical offices have standardised around Ms-Office, and each desktop will have a copy of Ms-Excel. Many users will therefore be familiar with pivottables.
- Pivottable are extremely user friendly.
- Pivottables are extremely flexible and allow a high degree of customization.

Before proceeding with the Tabulation example, we first present the two building stones of the technique, Alien Procedures in Manipula and OLE-Automation. For both techniques we include a complete example to illustrate that the actual amount of effort in making these techniques is limited. Because we need to program DLL's, it is inevitable that we present some Delphi (=Pascal) source code. We will only show the essential parts that illustrate the technique as we cannot assume that you are familiar with this development environment .

Alien Procedures

At the heart of the technique is Blaise/Manipula's ability to call Windows DLL programs. DLL stands for Dynamic Link Library. DLL's are pieces of programs that are ready to be called by another program and only loaded when required. The Blaise user manual describes the process of declaring and calling such programs.

You cannot just use any DLL however. The procedures that are exported from the DLL must adhere to a standard interface. The interface is documented and implemented in a Delphi unit file called MANWDLLO.PAS that is distributed with the Blaise system. This basically means that you will need to write your own DLL's in order to use ALIEN PROCEDURES in Manipula. You could write the DLL using Delphi (recommended by the Blaise team) or C++.

You would use DLL's for problems that are difficult or impossible to solve using Manipula alone. You can use these DLL's to provide procedures that provide information about the computing environment, that perform certain complicated statistical operations, or that interface into other programs.

In order to provide you with a simple example of what is involved in using DLL's, below we present the Manipula definition of an ALIEN PROCEDURE that shows a message dialog with three choices, Yes, No, Cancel and returns an integer value. For those of you familiar with MANIPLUS this would provide an extension to the CONFIRM dialog. The DLL is called 'MANIYNC.DLL', and it contains the procedure 'ConfirmCancel'

```
PROCEDURE ConfirmCancelDialog
PARAMETERS
  IMPORT Msg : STRING
  EXPORT iRes: INTEGER
ALIEN('MANIYNC','ConfirmCancel')
ENDPROCEDURE {ConfirmCancelDialog}
```

You would call this procedure in your MANIPULATE section as follows:

```
MANIPULATE
   sMsg := 'Do you want to impute the results (Yes/No) or cancel the process '
   iRes := 0;
   ConfirmCancelDialog(sMsg,Ires)
   Case ires of
   0: Display('Returned Cancel',wait)
   1: Display('Returned Yes',wait)
   2: Display('Returned No',wait)
   endcase
```

The implementation of the DLL requires a little Delphi programming. In defining the procedure in Delphi, you will need to adhere to the definition in MANWDLLO.PAS. If you keep to the definitions, writing the procedure is simple. The GetStringValue and SetIntegerValue procedures are used to move the data into and out of the procedure. The DLLInterface parameter contains all of the information on the various parameters passed to the DLL procedure. If the parameters passed conform to the required information then the parameter values are obtained, and a messagedialog is shown that has three buttons. ([mbYes,mbNo,mbCancel]). The outcome is captured in the variable res, and transformed such that cancel = 0, yes = 1 and no = 2.

```
Procedure ShowYesNoCancelDialog(DLLInterface:TDLLParamsInfo);stdcall; export;
var
   msg : string;
   res : integer;
   dllparam1,dllparam2 : tdllparameter;
begin
   if dllinterface.getparametercount= 2 then
   begin
      dllparam1 := dllinterface.getparameter(0);
      dllparam2 := dllinterface.getparameter(1);
      msg := getstringvalue(dllparam1);
      res := MessageDlg(msg,mtconfirmation,[mbyes,mbno,mbcancel],0);
      case res of
      2: res := 0;
      6: res := 1;
      7: res := 2;
      end;
      SetIntegerValue(dllparam2,res);
   end;
end;

exports  ShowYesNoCancelDialog index 1 name 'ConfirmCancel' resident;
```

Finally the ‘exports’ statement specifies how the showYesNoCancelDialog procedure is made available to the outside world. You can see that it can be referred to by a number (index=1) or by an identifier (‘ConfirmCancel’).

The main point about this example is that the actual work is in setting up the exchange of information between Manipula and the DLL. Fortunately this part is easily standardised. The actual work of showing a 3 button dialog is a single line in the program, using a standard function MessageDlg.

OLE Automation

Automation is part of a technology that used to be called OLE (Object Linking and Embedding) and at present is known as COM (Common Object Model) that has been developed by Microsoft and that is a key part of the Windows operating system. You can have automation servers; programs that perform certain tasks and that provide a COM interface such that other programs, automation clients, can control the execution of those tasks.

During the last IBUC Blaise 4.5 was shown by Statistics Netherlands in beta as an Automation server being controlled by Ms-Excel, the automation client. By programming MS-Excel using Visual Basic for Applications both data and metadata could be extracted from a Blaise data set and to manipulate it using the Visual Basic for Applications programming language available in MS-Excel.

MS-Excel however is itself also an automation server. You can make Excel programmatically perform any task that you can perform interactively. You will need to study the Excel *object-hierarchy* to understand how to achieve particular tasks. The top-level *object* is the application object. In the example below the Excel application object is represented by the v_xls variable. You can see that we refer to cell A1 in the active sheet of the application as

```
V_xls.Activesheet.range[‘A1’].value := ‘Manipula controls Excel!’
```

The process of invoking Excel, opening a workbook, placing some data into a cell and saving the workbook is illustrated in the following code extract:

```
    …
    try
      v_xls := createoleobject('excel.application');
    except
      showmessage('Could not start MS-Excel');
      exit;
    end;
    v_xls.Visible := True;
    v_xls.workbooks.add;
    v_xls.activesheet.range['A1'].value := 'Maniplus Controls MS-Excel';
    v_xls.activesheet.range['A1'].font.size := 54;
    v_xls.activesheet.columns['A:A'].columnwidth := 100;
    v_xls.displayalerts := false;
    v_xls.workbooks[1].saveas('BlaiseFecit');
    v_xls.quit;
    …
```

It is taken from our second example DLL (MANIXLS.DLL), which only has a single procedure Makesheet. This procedure invokes Ms-Excel , makes Excel show itself, creates a new workbook, and places a text in the active sheet of the new workbook. After some formatting, it saves the spreadsheet under the name 'BlaiseFecit.xls'.

To be complete, below is the source code for the Maniplus script that drives the DLL:

```
PROCESS TestExcel

AUXFIELDS
  IRes : INTEGER

PROCEDURE MakeSheet
PARAMETERS
  EXPORTS Result:INTEGER
ALIEN('ManiXls','MakeSheet')

ENDPROCEDURE {Makesheet}

MANIPULATE
   MakeSheet(iRes)
   Display('Invoked excel, returncode ='+str(iRes),wait)
```

Using MS-EXCEL Pivot Table interactively.

Before we start automating tabulations using the PivotTable, it is useful to have an idea what steps are involved in the process of generating tabulation interactively.

We start this process with the generation of an Excel worksheet that contains the data to be tabulated in one of the worksheets. Because there are a few problems in importing Blaise data into Excel that we like to avoid we use a Cameleon setup we developed to load up the data.

Then the definition of the PivotTable can take place. This is actually a four-step process.

- You define the kind of data source
- You define the location and kind of the of the data
- Then you define the tabulation
- Then you define the location of the table in the workbook

To do this most easily you first select the columns that contain your source data, and then you activate the PivotTable command.

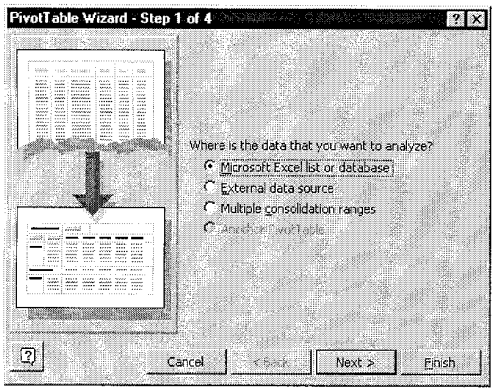


Figure 1 Interactive Pivottable Definition, Step 1

In step one you indicate that the data is obtained from an Ms-Excel list

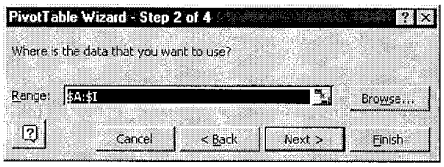


Figure 2 Interactive Pivottable Definition, Step 2

In step two you indicate the location of the data. Because you selected the columns already, this dialog is already filled in.

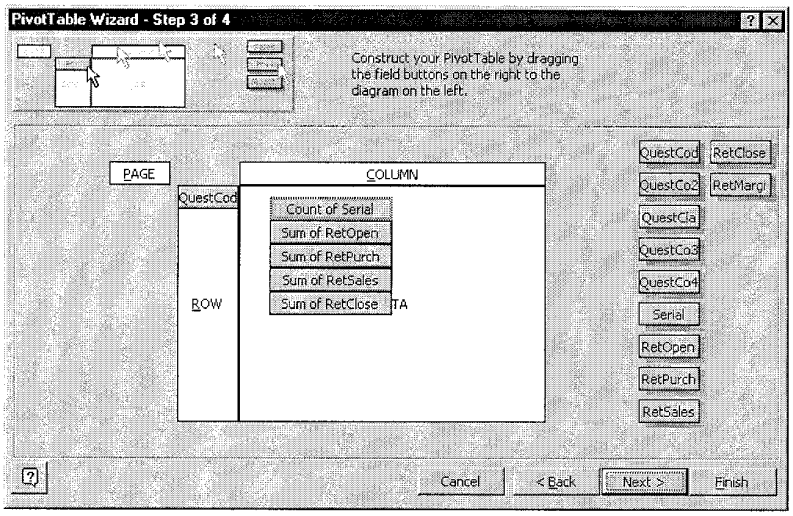


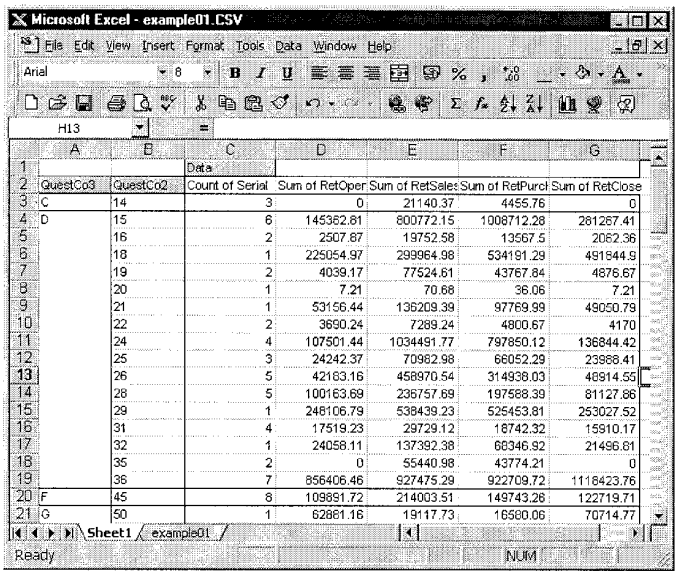
Figure 3 Interactive Pivottable Definition, Step 3

Step three is the most involved. You see a diagrammatic representation of the tabulation, and you see a list of all the fields that are available to you represented as buttons. The process of defining the tabulation is a matter of simply dragging and dropping the fields into the appropriate (Page, Column, Row and Data) areas. Subsequently you can set the properties of each of the fields in the tabulation by double clicking on the fields.

The result of our interactive definition of the pivot table is shown below. You can see that

- the layout of the table reflects our design preference of having the data fields in separate columns

- Each NACE 4 digit code is represented in a separate row, despite the fact that QuestCo3 and QuestCo2 are not enumerated fields.
- the Serial field is counted, not summed



	A	B	C	D	E	F	G
1			Data				
2	QuestCo3	QuestCo2	Count of Serial	Sum of RetOper	Sum of RetSales	Sum of RetPurch	Sum of RetClose
3	C	14	3	0	21140.37	4455.76	0
4	D	15	6	145362.81	800772.15	1008712.28	281267.41
5		16	2	2507.87	19752.58	13567.5	2062.36
6		18	1	225054.97	299964.98	534191.29	491844.9
7		19	2	4039.17	77524.61	43767.84	4876.67
8		20	1	7.21	70.68	36.06	7.21
9		21	1	53156.44	136209.39	97769.99	49050.79
10		22	2	3690.24	7289.24	4800.67	4170
11		24	4	107501.44	1034491.77	797850.12	136844.42
12		25	3	24242.37	70962.98	66052.29	23988.41
13		26	5	42163.16	458970.54	314938.03	40914.55
14		28	5	100163.69	236757.69	197588.39	81127.86
15		29	1	246106.79	538439.23	525453.81	253027.52
16		31	4	17519.23	29729.12	18742.32	15910.17
17		32	1	24058.11	137392.38	66346.92	21496.81
18		35	2	0	55440.98	43774.21	0
19		36	7	856406.46	927475.29	922709.72	1118423.76
20	F	45	8	109891.72	214003.51	149743.26	122719.71
21	G	50	1	62861.16	19117.73	16580.06	70714.77

Figure 4 Interactively Defined Pivottable

All that remains is to save the file as an XLS file in order to preserve the pivot table and to interactively modify the tabulation a little more for a pleasing layout.

Manitabs

The next step is to obtain a programmatic solution to our tabulation needs. The requirements are quite simple.

- We would want to be able to specify the tabulation inside Manipula and Maniplus scripts
- We want to be able to prepare the tables programmatically from within Manipula or Maniplus scripts., and save these as MS-Excel files.

As with many other programs, we will need to transfer the data between Blaise and Excel by means of an intermediate ASCII file. Excel makes importing delimited data particularly easy, therefore you will first need to export the data from a Blaise Dataset into a comma or a tab-delimited ASCII file. You can use the Manipula wizard for this.

Then you will need to write a setup that defines the tabulation. The natural tool for this are Maniplus scripts, although it is possible to define tabulations in the MANIPULA PROLOGUE section as well.

The main ingredient that you will need is the MANIPLUS.INC include file. This include file contains all the definitions of the Alien Procedures as well as a series of auxiliary variables that contain the relevant MS-Excel constant values.

To explain in detail the programmatic definition of PivotTables will lead us too far in the nitty gritty of OLE-Automation. Instead we will show you the MANITABS interface, and explain the crucial keywords that you need in order to programmatically define your own tabulations.

Usage of MANITABS

For the time being MANITABS will only read ASCII data. The user is required to specify the type of separator (comma or TAB separated), and the delimiter character. Because we chose not to complicate matters by involving Cameleon, you will need to specify the field names corresponding to the ASCII file in the Manipula/Maniplus sourcecode. You will also need to specify whether to use the Ms-Excel general format (for numbers) or the text format (for string data). Typically you would use a Manipula setup to prepare an abstract of the data file in ASCII (CSV or TAB separated), and subsequently call MANITABS.

MANITABS has only a few keywords. Because the Manipula DLL interface is implemented as ALIEN PROCEDURES, each MANITABS keyword is a procedure call.

As an example lets take an excerpt from a business survey that contains information on opening and closing stocks and sales and purchases. All of the enterprises are classified at NACE class, division and section level. The NACE codes are implemented as STRING types. Also the Nace section and division level codes have been implemented as a Classification type, and finally the section level codes have been implemented as an enumeration type.

We would like to make a very simple tabulation that shows opening stocks, purchases, sales and closing stock for each activity. The table would look like this:

NACE Section	NACE Division	Count of Units	Opening Stock	Purchases	Sales	Closing Stock
D	22	####.##	####.##	####.##	####.##	####.##
	24	####.##	####.##	####.##	####.##	####.##
….	….					

The source code extract shown below is a typical sequence used for preparing a tabulation.

Specification of input and output

You first specify input and output files:

```
InitializeConstants
aFilePath := 'D:\job\delphi3\xlspivotdll\'
InputFileName(afilepath+'example01.txt')
ExcelFileName(aFilePath+'TABLE01.XLS')
ASCIISeparator ( 2) {comma}
ASCIIDelimiter ( 1) {doublequote}
```

- The Initializeconstants procedure is defined in MANIPLUS.INC. It sets up the auxiliary variables that contain the Excel constants.
- The InputFilename procedure sets name and location of the ASCII file that contains the source data for the input file.
- The OutputFileName procedure sets the name and location of the Excel workbook that contains the tabulation.
- The ASCIISeparator and ASCIIDelimiter procedures establish the type of the ASCII inputfile. As mentioned earlier, it needs to be a delimited file type. You could also use a TAB delimited file.

Specification of file structure and tabulation

Then you simultaneously define the structure of the inputfile, as well as the bare bones definition of the tabulation.

```
Addfield('QuestCode'        ,2,0,0)
AddField('QuestCode02'      ,2,1,2)
AddField('QuestCodeSec'     ,2,1,1)
Addfield('Serial'           ,1,4,0)
Addfield('retopen'          ,1,4,0)
AddField('retpurch'         ,1,4,0)
Addfield('retsales'         ,1,4,0)
AddField('retclose'         ,1,4,0)
Addfield('retmargin'        ,1,0,0)
```

In designing MANITABS we have chosen to limit the number of procedures to a minimum, so there is a single procedure ADDFIELD that defines a field in the input file as well as determines if and how this field is tabulated. The ADDFIELD procedure has four arguments

1. The name of the field. You are free to name the field as you like, as long as it is unique.
2. The format specifier for the data. This is important. Normally Excel will import data in the ‘general’ format (1). This will give erroneous results if you are importing classification codes, as these will be interpreted as numbers instead. Here you can specify that you want such fields imported as ‘Text’ (2). You can also use this parameter to indicate the layout of date fields.

3. The kind of field in the tabulation. Fields can be either skipped (=0), a row field (=1) , a column field (=2), a page field (=3) or a data field (=4)
4. When you have several row fields or column fields it is important to establish the hierarchy between them. For instance if you would like to prepare a tabulation showing both section and division levels of a classification, you would want the section level to be shown is the outer field. You can use the fourth argument to indicate this sequence.

Specification of Tabulation Options

```
Datafieldoptions('serial'   ,0, Func.xlcount,'#,##0'); {count}
Datafieldoptions('retopen'  ,0, Func.xlSum,  '#,##0'); {sum}
Datafieldoptions('retsales' ,0, Func.xlSum,  '#,##0'); {sum}
Datafieldoptions('retpurch' ,0, Func.xlSum,  '#,##0'); {sum}
Datafieldoptions('retclose' ,0, Func.xlSum,  '#,##0'); {sum}
DatafieldOrientation(2)                                {columns}
ExcelVisible(2)
PivotFontSize(8)
PivotFontName('Arial')
```

Finally you can define some tabulation options.

- The DatafieldOptions procedure allows you to specify for each datafield how it should be presented (as a value, as a percentage of the total etcetera), what the aggregation method is (counting, sum, average, maximum, minimum etcetera. Note the use of the Func auxiliary variable to mimics constants), and what the presentation format should be. The chosen format here is no decimals, but with a separator between thousands.
- The datafieldorientation procedure allows you to set whether you would like the datafields to be presented in separate columns (=1) or in separate rows (=2).
- The ExcelVisible procedure allows you to specify whether you want to see Excel build the pivottable (=2) or not (=1)
- The PivotFontname and the PivotFontSize procedures set the font properties of the PivotTable.

Creating the Tabulation

Up until now there ha been precious little activity. We have provided the DLL with all the variable parameters that it needs to control MS-Excel. Excel itself has not been ‘invoked’ as yet.

```
Tabulate(iRes)
```

There is a single procedure, Tabulate, that changes all this. The Tabulate procedure will invoke Excel, and issue all the OLE-Automation commands to make Excel prepare the tabulation you have defined in Manipula. Tabulate takes a single parameter, iRes, which should contain the value 0 after Excel finishes. If it contains another value something has gone wrong.

The Clearfields procedure clears the definition from the DLL.

You can use MANITABS in both MANIPLUS and MANIPULA scripts. However MANITABS will feel at home in MANIPLUS scripts, because these do not require specification of INPUTFILE and OUTPUTFILE sections. Also, you might benefit from a CIF set-up that takes a hand in providing you with some additional Metadata, and would write out the fieldnames of the output files for you, as well as any calculated fields you might want to add.

After playing the Manitabs script we find that a pivot table has been prepared that is very similar to our design objective. We have separate columns for each of our data fields, we have succeeded in having the number of enterprises counted, but the other numeric data added together, and we have used a string field as a row field.

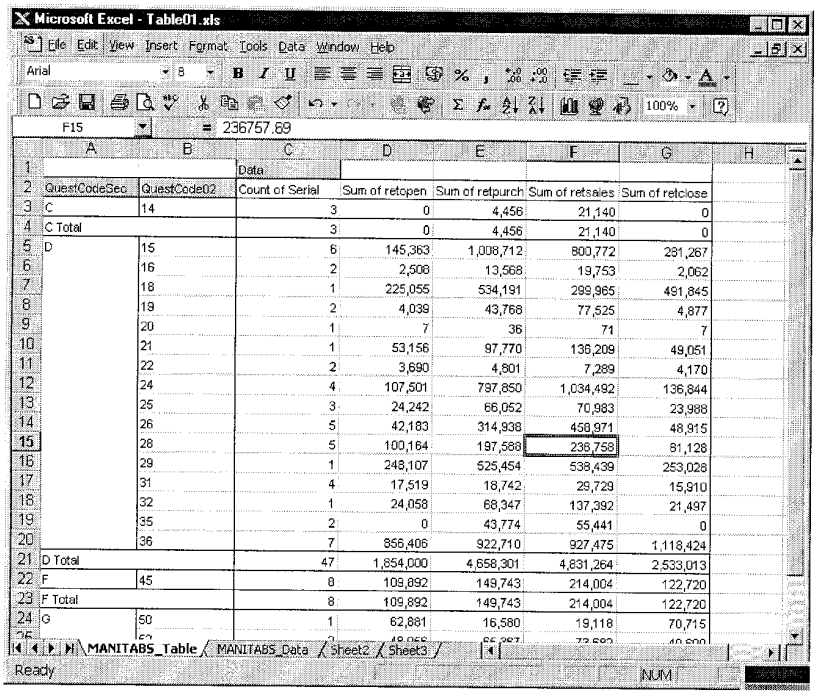


Figure 5 Tabulation prepared by MANITABS script TABLE01.MAN

Current limitations

Of course MANITABS as presented in this paper is intended as a working illustration of the benefits of the DLL – OLE-Automation technique. It is not a complete tabulation solution. Its most important limitations are currently:

- The need to prepare an ASCII file and specify the field names manually. This can be overcome by the preparation of an appropriate CIF file that reads the Metadata of a Blaise data model.
- The need to import the ASCII data into the Excel worksheet. This limits the size of the data file to around 65,000 records. However it is possible to bring the ASCII data into the pivottable using an ODBC link. This would be somewhat more complicated as it involves th definition of a ODBC data source definition as well as a data base structure (schema) file.
- Each field can only be used once in the pivot table. This is an obvious limitation that prevents for instance the same field being used twice as a data field. You might need this to present percentages as well as total values for instance. (ManiTabs). After you have defined your table in Manitabs you can of course easily add data fields using the Excel pivot table wizard interactively.
- You can specify only one tabulation per Maniplus script, because the script needs to terminate before the MANIPLUS.DLL is released and Ms-Excel is properly exited
- Because we use so-called 'late binding' in our OLE Automation controller DLL it is only possible to prepare tabulations with the English language version of MS-Excel. This is because the German, Dutch, French and other so-called 'localized' versions of MS-Excel use translated keywords for the various objects in Ms-Excel that are being controlled by Manitabs.
- The current MANITABS does not support datasets of more than 65000 records because we store the data in the worksheet itself. This is however not a limitation of MS-Excel Pivottable. You can establish ODBC datalinks to ASCII files, or you can distribute your data across various workbook pages to accommodate more cases. Pivottables are limited only in the amount of memory that you can give it. MANITABS could be reprogrammed to support either option.

A digression on Abacus

You might well ask why we would invest time and effort in developing a tabulation tool (or at least an interface into a tabulation tool) because Statistics Netherlands provides just such a tool for Blaise users. It is called Abacus.

As long as Abacus has been around it has been always somewhat separate from the other Blaise tools. I imagine an important reason for this is that it always was a completely interactive tool whereas the other Blaise products are programming tools requiring users to write datamodels and manipula set-ups.

Abacus has the great advantage over other tabulation tools that it understands Blaise metadata. It is therefore possible to prepare tabulations without having to create intermediate ASCII files. As we mentioned earlier, this is necessary for using Manitabs. Furthermore the process of defining tabulations in Abacus interactively is easy.

On the other hand Abacus seems to have a number of aspects that make it a less flexible package than you might wish. One important feature of Abacus is that it expects the row and column fields to be of an enumerated data type. If your row or column fields are of any other type all of the data is will be aggregated together in a single category, "Other". This is a disadvantage for Blaise users who have implemented classification types or external lookups with classification codes implemented as string type in their datamodels and would like to tabulate these.

Looking back at our earlier example datafile Example01.~bd Abacus would prepare the following tabulation.

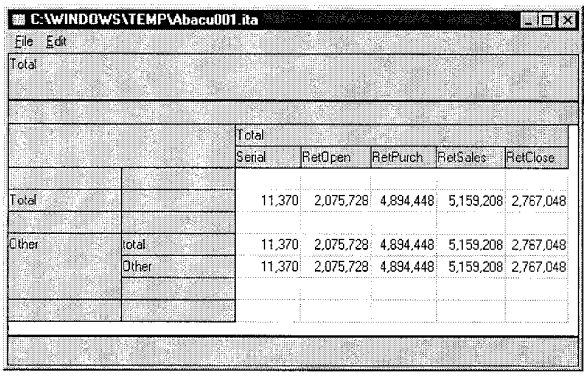


Figure 6 Output of Abacus on example01.~bd

Notice that all row field values are grouped together in the category "Other". Also notice that the values reported for the serial datafield are different from our earlier report. Abacus only calculates counts when you omit datafields and it sums the datafields if you specify data fields in the tabulation.

In order to have this tabulation the way I presented it earlier is possible after redefining the datamodel and after the introduction of an enumerated type for the NACE section level codes, and another enumerated type for the NACE division level codes. In order to have counts and totals in the same tabulation it is necessary to define a new field in the datamodel and fill it with ones (1). This has been done this in the datamodel Example02.BLA.

Example02.BLA starts of with two extensive defintions for enumerated types tQuestCodeSecEnum and tQuestCodeDivEnum, as shown below. With regards to tQuestcodeSecEnum we completed the enumerated field definition with descriptive information, and for tQuestCodeDivEnum we did not.

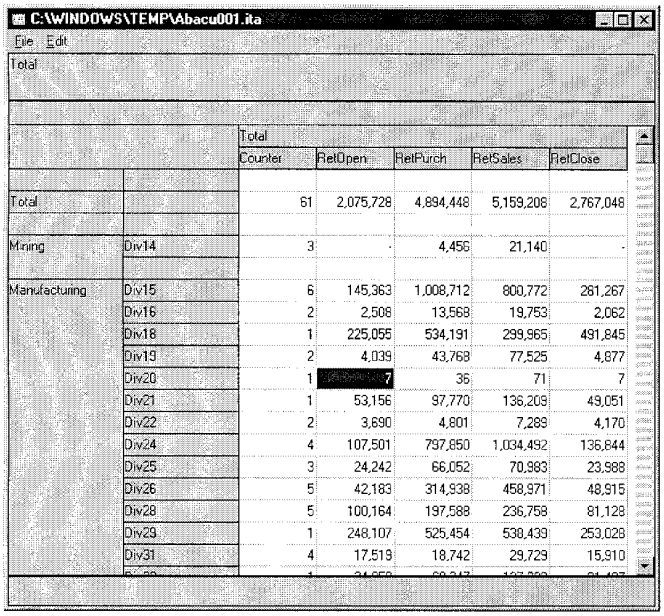
```
MANIPULATE  {Transform string data into enumeration}
case QuestCodeSec of
  'A' : QuestCodeSecEnum := SecA
  'B' : QuestCodeSecEnum := SecB
  'C' : QuestCodeSecEnum := SecC
  'D' : QuestCodeSecEnum := SecD
  … {some codes later}
  'Q' : QuestCodeSecEnum := SecQ
  endcase

  case QuestCode02 of
  '01' : QuestCode02Enum := Div01
  '02' : QuestCode02Enum := Div02
  '05' : QuestCode02Enum := Div05
  '10' : QuestCode02Enum := Div10
… {many codes later}
  '72' : QuestCode02Enum := Div72
  else
    QuestCode02Enum := DivUnknown
  endcase
  Counter := 1;
  OutputFile1.WRITE
```

Subsequently you would need to write a translation Manipula script that translates each individual value for the row and column fields into an enumerated value. Below we show you a typical extract for a manipulate section that recodes string data into an enumeration. It relies on a CASE/ENDCASE statement for each of the translations. We did not attempt to set up an enumeration and recode the four-digit NACE class codes as there are around 450 of those. Also notice the introduction of a

```
DATAMODEL Example02 "Example with enumerated fields and dummy counter  field"

TYPE
  tQuestCodeSecEnum =   (SecA (1) "Agriculture",
                         SecB (2) "Fishery"  ,
                         …  {many codes later}
                         SecQ (17) "Extra territorial organizations" )

  tQuestCodeDivEnum =   (       Div01 (1)   "Agriculture Hunting and Related",
                                Div02 (2)   "Forestry and logging",
                                …{many codes later}
                                DivUnknown (99))

FIELDS
   QuestCode    "NACE 4 digit Activity code"             : STRING[4]
   QuestCode02  "NACE 2 digit Activity code"             : STRING[2]
   QuestCodeSec "NACE Section level code"                : STRING[1]
   QuestCode02Enum  "NACE 2 digit Activity code as enumeration" : tQuestCodeDivEnum
   QuestCodeSecEnum "NACE Section level code as enumeration"        : tQuestCodeSecEnum
```

'Counter' field that obtains the value 1 in order for us to be able to include counts and totals in the same tabulation.

In Figure 7 below we show you the outcome of this tabulation. Notice that Abacus substitutes the category text for the category identifier (the section codes). Abacus does not print the category values.



		Total				
		Counter	RetOpen	RetPurch	RetSales	RetClose
Total		61	2,075,728	4,894,448	5,159,208	2,767,048
Mining	Div14	3	·	4,456	21,140	·
Manufacturing	Div15	6	145,363	1,008,712	800,772	281,267
	Div16	2	2,508	13,568	19,753	2,062
	Div18	1	225,055	534,191	299,965	491,845
	Div19	2	4,039	43,768	77,525	4,877
	Div20	1	[redacted]7	36	71	7
	Div21	1	53,156	97,770	136,209	49,051
	Div22	2	3,690	4,801	7,289	4,170
	Div24	4	107,501	797,850	1,034,492	136,844
	Div25	3	24,242	66,052	70,983	23,988
	Div26	5	42,183	314,938	458,971	48,915
	Div28	5	100,164	197,588	236,758	81,128
	Div29	1	248,107	525,454	538,439	253,028
	Div31	4	17,519	18,742	29,729	15,910

Figure 7 Output of Abacus after recoding

Conclusions

This paper set out to illustrate the power that the Alien Procedure/OLE-automation-DLL technique offers you. For this purpose we developed a proto-type tool, MANITABS, that allows you to define tabulations in Manipula scripts and prepare these tabulations in Ms-Excel. These tabulations, although programmatically defined, have the advantage of being completely open to interactive editing. The MANITABS therefore offers the best of both worlds, programmatic definition of tabulations and full user interaction with the underlying data.

In comparison with Abacus the method presented in this paper has some advantages. In particular enables you to include string, integer and classification data types to be included as row and column fields in tabulations. It also allows straightforward mixing of aggregation functions (Count, Sum, Stdev, Average, Max, Min). Pivottable's ease of use in modification of existing tabulations is quite superior. Finally it provides you with tabulations directly in MS-Excel format. Also, although we define the tabulations in Maniplus code, the resulting tables remain totally user configurable.

Abacus on the other hand has the important advantage of being able to access Blaise datafiles and metadata directly. It also has an elegant way of dealing with array questions. These advantages are mitigated somewhat because you are required to redefine your datamodel and write Manipula scripts to provide support for non-categorical data in tabulation row and column fields.

It is interesting to compare the Alien Procedure/Ole-Automation technique with the upcoming open Blaise architecture. In sofar as we know (We are not privy to any information about OBA, other than that which has been announced by SN) OBA will make Blaise into an automation *server*. This will allow anybody with more than a passing interest and knowledge of both Blaise and a second development environment (C++, Visual Basic, Delphi) to develop *applications* or *tools* that *control the Blaise system*. If you have invested on the other hand on learning Blaise and Manipula syntax, you may want to be able to control (parts of) other applications using the Manipula language or extensions thereof. Therefore the techniques outlined here are *complementary* to OBA.

On a final note, we were quite pleasantly surprised at the speed with which the MANITABS.DLL could be developed. Much of the facilities that Microsoft made available in Pivottable could be made available in Manipula within three days of work, and yield a tool that does not compare unfavourably with an already established tool, Abacus. The main point of the paper is that the Alien Procedure/OLE Automation approach is a high productivity approach. It can be quite gainfully harnessed to resolve other areas in which you might wish to extend the fundtionality of the Blaise system to suit (your) users needs.

References

1. Blaise Developers Guide, Chapter 5. Statistics Netherlands, 1998
2. Abacus Users Manual, Version 4.1, Statistics Netherlands 1999
3. MANWDLLO.PAS in \program files\statneth\blaise4.2\tools\dll\
4. Microsoft Office97 Visual Basic Programmers Guide, Microsoft Press 1997
5. Microsoft Excel 97 Developers Handbook, Wells & Harshbarger, Microsoft Press 1997
6. Delphi 4 Unleashed, Chapter 16. C. Calvert, SAMS, 1999
7. Excel by Blaise, a primer in Cameleon and Maniplus scripts, T.R. Jellema, NAMES B.V., Unpublished Course Material 1999
8. Maniplus Tutorial, T.R. Jellema NAMES B.V., Unpublished Course Material 1999