Methods for Simultaneous Meta & Data manipulation in Blaise

Marien Lina, Statistics Netherlands

1. Introduction

From the beginning of the Blaise development, there is an old saying going “no data without meta data”. If you have data without a data description then it is obvious that you in fact have nothing at all. Without a meta data file Blaise does not know what the structure and the meaning of the data is and you will not be able to use the data within Blaise.

Until relatively recently the meta data of Blaise and the data of Blaise were in principle two separate worlds. Although Blaise can handle data only when the meta data is available, the handling of meta data and of data by the user have been separated in different parts of the Blaise system. The Blaise data model language and the Manipula language are primarily equipped for handling of the data in Blaise and the Cameleon language is primarily equipped for handling of the meta data in Blaise.

With the release of the Blaise API in 2001 it became possible to combine meta data and data in one application. This offered many new (and often unforeseen) possibilities. There is one ‘major’ drawback: you need IT skills and access to IT tools to be able to use the Blaise API.

Lately with the release of Blaise 4.8 a new possibility has become available to combine meta data and data. By using Manipula, Blaise 4.8 now also offers in the basic system a way to combine meta data and data. This paper describes the background of combining meta data and data in Blaise and how this can be used.

2. The Blaise and the Manipula language

The Blaise and Manipula languages have primarily been designed to handle data rather than meta data.

The Blaise language describes the structure and the relations in data. You describe blocks, types, fields, routing, relations, computations, external lookups etcetera with as main purpose to collect and validate data based on the specification. The Blaise language in itself has no way to access the meta definition actively during the collecting and validation process. In other words: despite the fact that field names and question texts are visible in the DEP on your data entry screen, there is no possibility to retrieve meta information from the meta file and use it in the rules. For example, you cannot ask for the field name or the field type of a certain field and assign it to a string field.

The Manipula language also aims at data handling. The language is great for many things like import/export, selections, sorting etcetera but within Manipula it is not possible to actively use the meta specification of the data of the files it is processing. Because of that it is for instance not possible to write a generic Manipula setup that produces a report that gives a list of answered questions showing both the question text and the given answer.

3. The Cameleon language

To read meta data from a Blaise BMI file and to write it in another format, something else than the Blaise and Manipula language was needed. In Blaise 2 the setup generator was introduced, replaced in later Blaise versions by Cameleon.

Cameleon uses setups specified in the Cameleon language. It enables you to present meta data in another format. This may be for documentation or for alternative data descriptions, suitable for converting Blaise data to external statistical packages and databases. Cameleon retrieves meta information from the BMI, manipulates its appearance to a specific form and puts the result in a text output file. The Cameleon language is not simple and it requires some skill to get the most out of Cameleon. Luckily numerous Cameleon setups are shipped with Blaise. They serve as examples for creating new setups.

There is an analogy between a Cameleon setup and a style sheet. Both are used to reshape in a generic way that what they process. A Cameleon setup reshapes meta data and a style sheet reshapes XML documents. You can read more on this topic in the paper "On the use of XML in the Blaise environment" by Jelke Bethlehem and Lon Hofman, proceedings of the International Blaise Users Conference in Kinsale in May 2000.

4. Combining data and meta data: the Blaise API

With Blaise 4.5 (2001) the Blaise Application Programming Interface (API) has become available. It is shipped as part of a separately licensed product called the Blaise Component Pack (BCP). The Blaise API enables you to address the data and meta in one application. The advantage is clear: with the Blaise API you can combine the Manipula and Cameleon functionality into one application. You can use the Blaise API in any programming language that supports COM technology (the Component Object Model from Microsoft).

Both the DEP and Manipula can easily interact with components using the Blaise API. Within the DEP these components can be called via the menu, via the alien procedure mechanism or via the alien router mechanism. The components can also be linked to a user defined type.

In short, the Blaise API makes it possible to create your own applications and address data and meta data in the same application. Based on the API it is in principle possible to develop applications that replace systems that combine Manipula and Cameleon applications.

Note that using the API requires knowledge of modern programming languages, it requires programming skills, and it requires access to advanced system development tools. This may be a barrier for the Blaise programmer or survey designer.

5. Introduction of API functionality in Manipula.

With the API in mind two developments took place for Blaise 4.8. Firstly, meta data access functionality has been added to Manipula, and secondly, the communication between the rules and Manipula has been made possible, thus

offering Manipula procedures as an alternative for DLL calls and ActiveX calls in the rules.

There where two reasons for doing this. The main reason is related to the requirements imposed on the development of Basil, a new tool in Blaise 4.8 that allows for 'tailored user interfaces' in a CASI setting. Basil applications should be able run on machines that have no Blaise components installed (so no API access) but they should still be able to do most things that are only possible when using the Blaise components. The second reason is to make it possible for qualified Blaise programmers to extend the system without having to go through programming using IT tools.

Most requirements for Basil could be met only when Manipula and Basil would work closely together and when Manipula was able to access the meta data of the data model. The generic solution that was implemented for Blaise 4.8 is based on a flexible, event driven connection between Basil, the rules and Manipula setups.

6. Calling Manipula procedures in the rules

Running Manipula from within the DEP was first made available in Blaise 4.7. From within the menu of the DEP you are allowed to call a Manipula setup and in that setup you can have access to a copy of the form that is being handled by the DEP. To get access to that form, you need to use the file setting `INTERCHANGE=TRANSIT`.

In Blaise 4.8 the implementation of `INTERCHANGE` has been extended in such a way that you now can have access to the actual form that is in the DEP session. To achieve this you need to use the file setting `INTERCHANG=SHARED`. Accessing the actual form instead of a copy is not only faster but makes it possible to give you access to much more information like the actual errors present in the form and routing status information without having the execute the rules again.

Blaise 4.8 also makes it possible that at all places where you are allowed to call a method of a component you now also can call a procedure in a Manipula setup. This makes it possible to call Manipula from within the rules by using alien procedures and alien routers and this makes it possible to use features in the rules that were not available previously.

Below is an example of a small Blaise data model ASurvey that uses an alien Manipula procedure ComputeC.

```
DATAMODEL ASurvey

PROCEDURE ComputeSideC
PARAMETERS
  IMPORT A, B: REAL
  EXPORT C: REAL
ALIEN('MySetup.msu', 'ComputeC')
ENDPROCEDURE

FIELDS
  SideA, SideB "Enter value": 1..20
  SideC "Side C will be:": 1.41..28.28
```

```
RULES
  SideA   SideB
  IF (SideA>0) AND (SideB>0) THEN
    ComputeSideC(SideA, SideB, SideC)
  ENDIF
  SideC.show
ENDMODEL
```

When values have been entered for SideA and SideB, then the alien procedure ComputeSideC is called and that procedure then calls the procedure ComputeC in the Manipula setup *MySetup*. It computes the value for C and returns it to the field SideC.

```
PROCESS MySetup

PROCEDURE ComputeC
PARAMETERS
  IMPORT A, B: REAL
  EXPORT C: REAL
INSTRUCTIONS
   C:= SQRT(A*A + B*B)
ENDPROCEDURE
```

It is just a simple 4.8 example to show that you can address Manipula procedures in the rules. Within those procedures you have access to all the richness of Manipula thus allowing to write to files, display a tailored dialog box, call other applications like another data entry session etcetera. This can now all be done using the Blaise basic system tools.

7. Retrieving meta data in Manipula

For meta data handling, a relevant new extension in Blaise 4.8 is the introduction of methods to retrieve meta data with Manipula. Like the Blaise API, a Manipula setup can now access the meta data of a file that it is using. You can access all meta data for the fields and types, like field names, question texts etcetera, but you can also access meta data like error information and routing status.

In the Manipula setup you can loop over all fields and blocks (recursion has been introduced in Manipula to make this easier). For every element (for example a field) in the loop you could print the field label, the question text, etcetera. In the Blaise basic system this could until now only be done by using Cameleon. In fact, many (although not all) things you can do in the Cameleon language are now also possible in a Manipula setup in Blaise 4.8.

An example of a meta producing Manipula setup is included in the Blaise 4.8 samples folder *demo48\ManipulaMeta*. The main setup of this example is called *producemeta.man*. The snippet below shows a part of the source code:

```
WriteLine('Field Size:' + InputFile1.GETFIELDINFO(MyFieldName,'SIZE'))

WriteLine('Attributes:' + InputFile1.GETFIELDINFO(MyFieldName,'ATTRIBUTES'))

WriteLine('Field Type Name:' +
 InputFile1.GETFIELDINFO(MyFieldName,'FIELDTYPENAME'))

WriteLine('Local Type Name: ' +
 InputFile1.GETFIELDINFO(MyFieldName,'LOCALFIELDTYPENAME'))
```

The setup gets meta information form the BMI and writes it to an output file. The file method `GETFIELDINFO` has various keywords that can be passed as a string parameter, like `'size'` that returns the size of the field. Note that validity of those keywords is checked at run time. So beware of typos! There are a number of other methods available for retrieving meta information like `GETMETAINFO` and `GETTYPEINFO`. These methods open the way to create integrated systems in Manipula that used to be a combination of Cameleon and Manipula setups. Manipula setups could eventually replace Cameleon setups.

8. Accessing meta information in the rules

By combining the access to meta information within Manipula and the use of Manipula procedures within the rules, the meta data becomes also accessible within the rules. The example below shows how to get the name of an enumeration category and assign it to a text field.

First the Manipula procedure:

```
PROCESS GetMeta
USES MyMeta 'GetMeta'

TEMPORARYFILE MyData:MyMeta

PROCEDURE GetCategoryName
PARAMETERS
  IMPORT FieldValue: INTEGER
  IMPORT TypeName: STRING
  EXPORT CategoryStr: STRING
AUXFIELDS
  Indx: INTEGER
INSTRUCTIONS
  FOR Indx:= 1 to VAL(MyData.GETTYPEINFO(TypeName,'CATEGORIES.COUNT')) DO
    IF
MyData.GETTYPEINFO(TypeName,'CATEGORIES['+STR(Indx)+'].CODE')=STR(FieldValue)
THEN
    CategoryStr:= MyData.GETTYPEINFO(TypeName,'CATEGORIES['+STR(Indx)+'].NAME')
      EXITFOR
    ENDIF
  ENDDO
ENDPROCEDURE
```

The procedure GetCategoryName in the Manipula setup receives the value of a field in the data model via the import parameter FieldValue. The value of the import parameter TypeName is used to access the meta information that belongs to the temporary file handle MyData using the method `GETTYPEINFO`. This method is used to determine the number of available categories, it loops over all available categories until the category code (again using the `GETTYPEINFO` method) matches the value of the import parameter FieldValue. The corresponding category name is returned to the rules using the export parameter CategoryStr.

The procedure GetCategoryName can be activated via the rules in a Blaise data model as an alien procedure.

```
DATAMODEL GetMeta

TYPE
  THP = (Man, Woman, Child, Otherwise)

PROCEDURE GetCatName
PARAMETERS
  IMPORT FieldValue: INTEGER
  IMPORT TypeName: STRING
```

```
  EXPORT CategoryStr: STRING
ALIEN('GetMeta.msu','GetCategoryName')
ENDPROCEDURE

FIELDS
  HousePos "Position in the household": THP
  CatString: STRING
RULES
  HousePos
  GetCatName(ORD(HousePos),'THP',CatString)
  CatString.SHOW
ENDMODEL
```

The example requires that the BMI used in the Manipula setup is the same as the data model that calls the Manipula setup. So the Manipula script needs to be prepared each time the data model structure changes. Blaise 4.8 also makes it possible to pass the BMI file as parameter to the Manipula setup. This allows you to re-use the prepared Manipula setup for any data model you have. It only requires two simple changes to the example. In the Manipula setup the meta identifier needs to be defined as a variable:

```
USES Meta (VAR)
```

And in the data model the alien call needs to be changes: it now needs to pass the name of the BMI file to the setup also. This looks as follows:

```
ALIEN('GetMeta.msu /KMyMeta=$dictionaryname','RetrieveCodeName')
```

At runtime `$dictionaryname` will be replaced by the name of the BMI file and by using the `/K` command line option of Manipula it is then passed to the setup linking it to the meta identifier MyMeta. Note that when you pass the BMI file as a parameter to the Manipula setup certain restrictions are imposed on the setup. You are for instance not allowed to access a field of a field directly but you will need to use the methods `GETVALUE` and `PUTVALUE` instead. The restrictions are checked during the preparation of the Manipula setup.

In the Manipula script you can also get access to all the data in the form that is in the DEP session that executed the rules. You have to use the `INTERCHANGE` setting to achieve that:

```
TEMPORARYFILE Data:Meta
SETTINGS
  INTERCHANGE=SHARED
```

9. Conclusions

While in older versions of Blaise meta data and data have been strictly separated in different parts of the system, new developments in Blaise make it possible to combine the retrieval of meta data and data in Manipula, and via the alien procedure also in the rules of a data model. This opens many new possibilities in the Blaise basic system for instance to create systems in Manipula that process meta data, without having to use Cameleon.