Management of Classification Lookup Files

Fred Wensing, Softscape Solutions, Australia

1. Introduction

A key aspect of data collection is the correct assignment of codes to represent text entries which are given in the interview. Where the coding frame is structured or the list is long, classification is often done through a lookup action on an external file.

When a survey spans a time period it is important that the systems are set up to handle the inevitable updates to the classifications which result from new entries and other revisions.

This paper will discuss the management of classification lookup files, their creation, the application of version control and the way that systems can be set up to manage the transition from one release to the next.

2. Classification basics

This section describes the classification options available in the Blaise software. It is included here for the sake of readers who may be unfamiliar with the options.

In the collection of statistical information it is usual to codify the data. This puts order and structure into the presentation of results. In particular, it avoids the incidental alphabetical re-arrangement of data which can occur to output presentation if data is stored as character strings.

Blaise supports various methods for the assignment of codes to the answers given in a survey:

- Enumerated field type
- Classify method (using a classification field type)
- Lookup from an external file using alphabetical search
- Lookup from an external file using trigram search
- Combinations of the above

Some details in this section of the paper draw on material in the *Blaise Online Assistant* and the Sample code provided with the Blaise software. It is included here for completeness. Please consult the *Blaise Online Assistant* for more comprehensive information.

2.1 Enumerated field type

The simplest and most common method of classification is the enumerated field type in which the interviewer selects the relevant code, which relates directly to the answer given, from a category list which is visible on the screen.

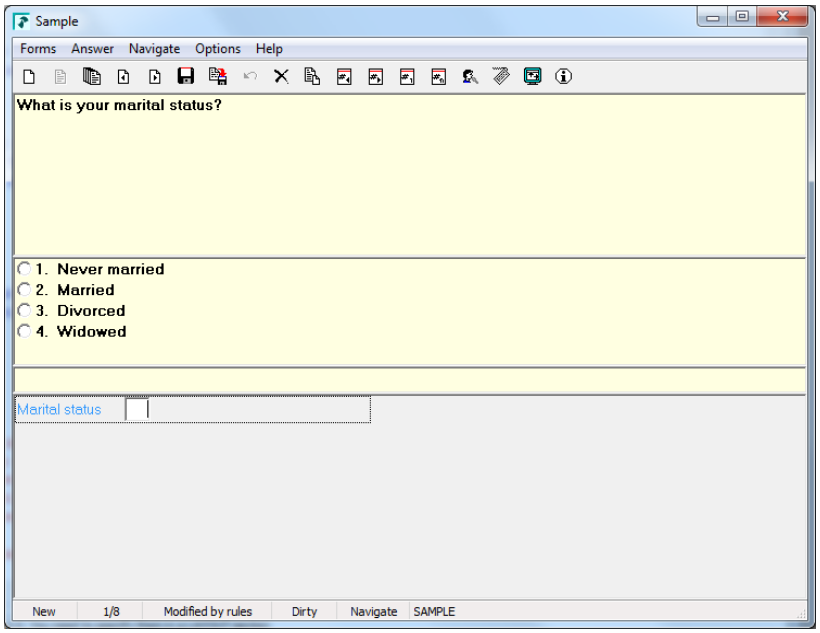
The field definition itself holds the classification. For example, in the following field definition:

```
MarStat "What is your marital status?"
      / "Marital status"
     : (NevMarr         "Never married"
       ,Married         "Married"
       ,Divorced  "Divorced"
       ,Widowed         "Widowed"
       )
```

The codes 1 to 4 are assigned for the four categories by default. A more explicit definition of the field reveals the classification:

```
MarStat "What is your marital status?"
      / "Marital status"
      : (NevMarr   (1) "Never married"
        ,Married   (2) "Married"
        ,Divorced  (3) "Divorced"
        ,Widowed    (4) "Widowed"
        )
```

When this question is encountered in the survey the list of categories shows on screen with relevant codes:



As you can see, by specifying the code number in the definition it is possible to assign a predetermined number to each category. The numbers do not need to be sequential and may contain gaps as can be seen in the following type definition for the response set of “Yes” (code 1) and “No” (code 5) which is often used in CAPI surveys:

```
TYPE
    TyesNo =
      (Yes  (1) "Yes"
      ,No   (5) "No"
      )
```

2.2 Classify method

Where the classification of an item involves a hierarchical code frame (For example: Occupation, Motor vehicles or Commodities) then Blaise provides a CLASSIFY method which will bring up the hierarchical code list for the operator to navigate through and select a corresponding entry.

To use this method it is firstly necessary to define the classification hierarchy to Blaise. This is done using a Classification type in which all the levels, codes and labels are defined. Essentially the Classification

type is a series of nested enumerations.

An example of a hierarchical classification of motor vehicles is given in sample code provided with Blaise and part of the classification (also shown in the *Blaise Online Assistant*) is reproduced here for your information:

```
TYPE
 Automobiles = CLASSIFICATION
 LEVELS
   Make, Model, Body_Style
 HIERARCHY
   American_Motors (1) = (
     Rambler_or_American (1) "AMER Rambler/American" = (
       _2dr_Sedan_or_HT_or_Coupe (2) "2dr Sedan/HT/Coupe",
       _4dr_Sedan_or_H (4) "4dr Sedan/HT",
       Station_Wagon (6) "Station Wagon",
       unknown_or_other_style (88) "unknown" ) ,
...
```

An American Motors Rambler that is a two-door sedan, hard top, or coupe would have a code of 1.1.2.

If the list is to be updated during the life of the survey then the DYNAMIC attribute needs to be specified along with the length of the corresponding code structure. For example:

```
TYPE
 Automobiles = CLASSIFICATION DYNAMIC[9]
 LEVELS
   Make, Model, Body_Style
 HIERARCHY
   ...
```

The Classification type can be defined within the Blaise datamodel for that survey although it could also be set up as a separate type library so that it can be used by multiple surveys. A separate type library would also be preferable if the content is likely to be updated during the life of the survey.

Since the syntax for coding frames with complicated descriptions can be awkward to type by hand, particularly for large hierarchical coding frames, it is recommended that you build the classification using a Manipula program which converts a text description of the classification (which may have been exported from a classification register). For details on how that is done and information about sample programs on the subject of hierarchical classifications consult the *Blaise Online Assistant* Contents under:

How to Use... / Programming Languages / Advanced Topics / External Files / Hierarchical coding

Once the classification type has been defined it can be used like any other type definition. For example:

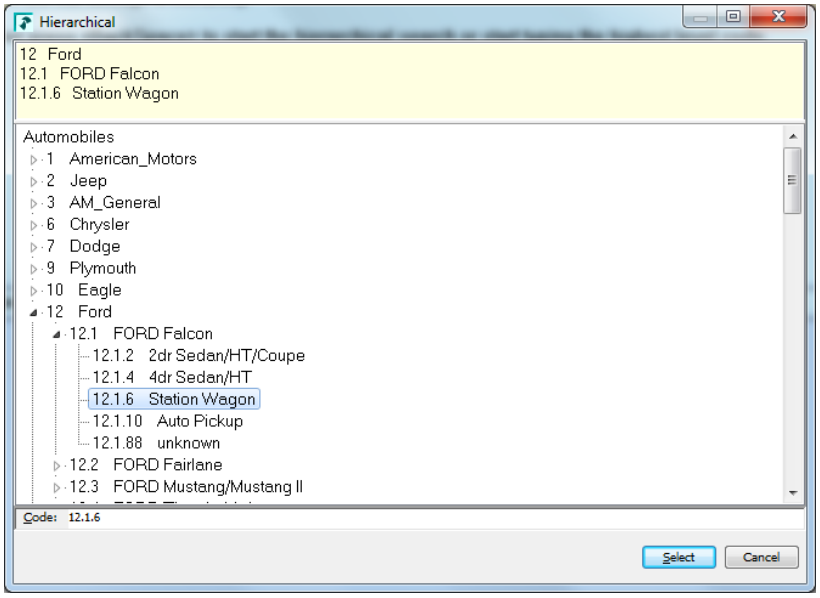
```
MyCar "Identify your car from the list" : Automobiles
```

To invoke the classification the field is placed on the path using the CLASSIFY method, rather than using the ASK method. For example:

```
MyCar.CLASSIFY
```

When the CLASSIFY method is invoked then the operator is presented with the hierarchy and then he/she navigates down the classification until the corresponding entry is located.

The following screen shows an example where the operator has located “Ford Falcon - Station Wagon” on the list:



Navigation through the hierarchy will return the code number for 12.1.6 for this entry.

2.3 Lookup from an external file using alphabetical search

When the list of categories is large it is not practical to set up an enumerated field. In this case it may be preferable to lookup an external file using one of the keys on that file. Use of an ordinary secondary key will result in an alphabetical list being presented to the operator.

For the external file to be accessible from the survey datamodel it is necessary to add a USES section to identify the external datamodel. For example:

```
USES
      mCountryList 'CountryList'
```

It is also necessary to add an EXTERNALS section within the relevant block to identify the external file:

```
EXTERNALS
      ExternalList : mCountryList ('CountryList',BLAISE)
```

The lookup is then activated in the RULES by associating the field in the lookup file with the field in the main datamodel. This is done with the piping symbol '|'. For example:

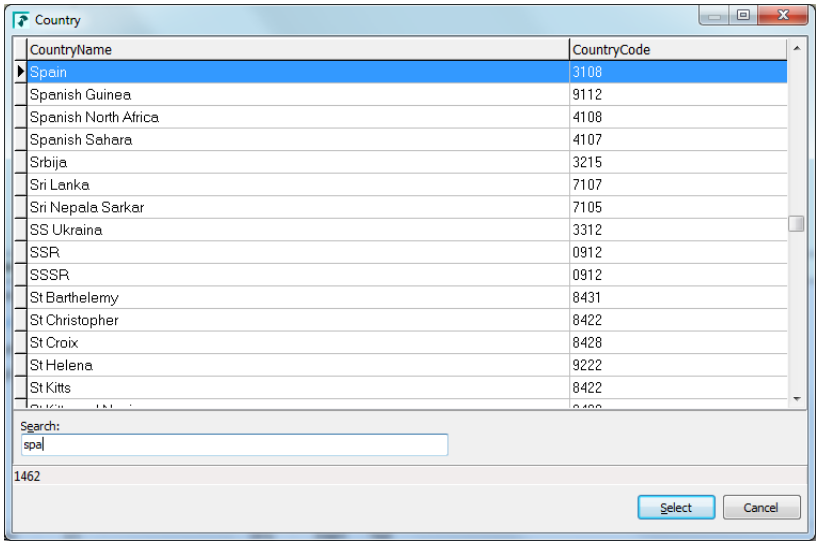
```
Country_of_birth | ExternalList.LOOKUP.Countryname
```

The lookup process will present the list of entries from the external file using the first or only secondary key. If the external file has more than one secondary key then search dialog will provide a radio button to enable the operator to switch between keys. If only one key is to be used in the lookup then the designated key can be added in the lookup instruction. For example:

```
Country_of_birth | ExternalList.LOOKUP(AlphaKey).Countryname
```

When the field is encountered in the survey then the lookup dialog will be presented on the screen. As the operator types the first few letters of the search string the lookup dialog will jump to that point in the list. The value which is returned by the lookup process is the content of the field named after the LOOKUP keyword, in this case **Countryname**.

The following screen shows the alphabetical lookup on a list of birth places after the first few letters of "Spain" have been typed in:



Once an entry has been selected then the identified field is returned to the survey data.

2.4 Lookup from an external file using trigram search

Blaise provides a more comprehensive search mechanism on text strings through the application of trigram indexing. Trigram indexing causes a text string to be indexed on all the subsets of three letters extracted from the text string. For example, the word Blaise consists of the trigrams #BL, BLA, LAI, AIS, ISE, and SE#, where # represents word boundaries.

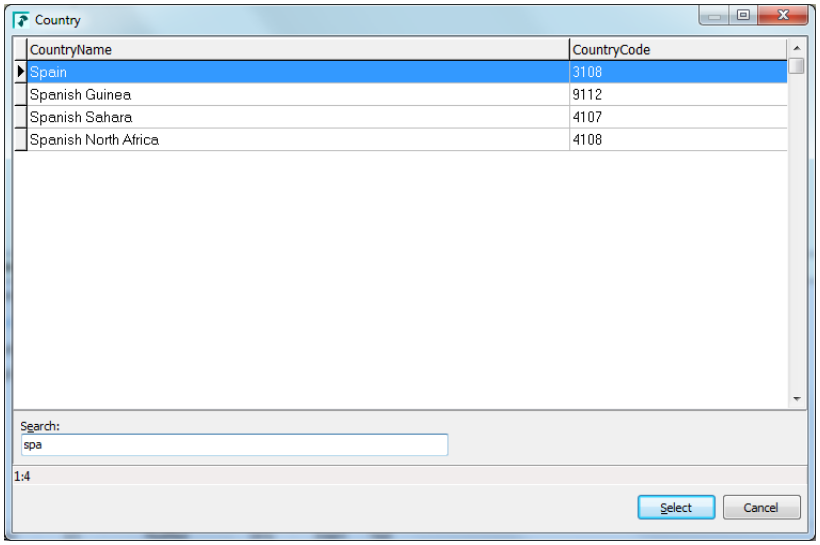
By using the trigram index in the lookup process Blaise will return to the screen all entries that match a defined number of trigrams found in the search text. The more text which is entered into the search field the more trigrams will be searched and the shorter the list of possible matching entries will become.

Because Trigram searching occurs on patterns of three letters it does not matter where those three letters appear in the indexed field. Trigram searching is therefore ideal for long descriptive texts.

The activation of lookup using trigram search is the same as for alphabetical search except that the designated key to be searched must be of type TRIGRAM.

For more details about Trigram indexing and setting or changing the parameters which control the search behaviour please consult the *Blaise Online Assistant.*

The following screen shows the trigram lookup on a list of birth places after the first thee letters of “Spain” have been typed in:



Notice that the list presented to the user is more refined than the alphabetical list.

2.5 Using a start value for lookup operations

The lookup process also allows for a start value to be passed into the lookup. That value can help the operator to narrow the search process. Where the start value is part of a Classification code then the subsequent lookups, whether alphabetical or trigram, are also filtered by that code.

2.6 Retrieving other fields from the external file

The value which is returned by the lookup process in the above two subsections is the content of the field named after the LOOKUP keyword, in this case **Countryname**.

In order to retrieve the values of other fields from the external file it is necessary to search and read the external file after the lookup has been done. For example, to obtain the country code for a selected country the following code can be included. For example:

```
IF ExternalList.SEARCH(Country_of_birth) THEN
    ExternalList.READ
    Country_Code:=ExternalList.CountryCode
ENDIF
```

2.7 Combining lookup methods

By combining the various lookup methods it is possible to provide considerable flexibility for any coding system.

3. Preparing Classification Lookup files

Most classification lookup files are created from classification data that is extracted from classification registers or systems. Such data is usually placed in a text file, which may be fixed-format or character-

separated, which can then be loaded into Blaise using a Manipula program.

Before loading the classification data into a Blaise file that can be used in lookup and/or search processes you need to consider and decide on the following.

3.1 Use of a hierarchical classification

Classification frames are often developed as a hierarchy because it provides structure and lends itself to convenient addition of new codes.

If a hierarchical classification is used then there is an option to make use of the hierarchical classification type in Blaise. The definition of a classification type in Blaise is somewhat complicated but an easy-to-use conversion program is provided in the Blaise Sample files. Consult the *Blaise Online Assistant* for more information.

Use of a hierarchical classification type will give access to the Classify method and allow the operator to navigate the hierarchy branches to make a selection.

Make sure that you add the Dynamic option if you expect the classification to be extended over the life of the survey.

If the Classify method is not to be used then the hierarchical code can be converted to a simple code string and assigned through the other lookup methods.

3.2 Which entries to put into the lookup list

To make the lookup list as useful as possible, the entries should be drawn from a list of synonyms linked to the relevant classification rather than from the classification titles themselves. For example, if “Great Britain” is the title for an entry in a list of countries it would be better if the lookup list contained not just that entry but also some synonyms like “United Kingdom”, “UK” or even “England”.

The more entries which are in your lookup list the better it will be for the coding process.

If you do not have a comprehensive list of synonyms then you can plan to update your list from time to time based on “Other – specify” entries that are extracted from your survey and sent to an expert for coding. Once a decision has been made about an appropriate code the newly coded description(s) can be added to the lookup list. Updating of code lists and coding unassigned entries is discussed in more detail later.

3.3 The types of indexes to provide

The decision to include a Trigram index in your code list will depend on how structured the coding is expected to be. Trigram indexes are best suited to coding where there is an unambiguous match between the text and its possible code (for example in a list of commodities or localities). Trigram searching generally ignores the structure of the coding frame.

For some coding frames which are highly structured (for example Occupations) then trigram searching may not be appropriate. You will need to decide this in consultation with classification experts.

An alphabetical index should generally be provided in the code list (as a Secondary Key) so that it can be used if desired.

For most applications, then, two Secondary Keys should be provided, one as a Trigram index and the other as an Alphabetical index.

3.4 Adding and retrieving other fields from the code list

The code list can be enhanced to provide other information that may be useful for the survey. These additional items could be things like edit tolerance levels for application in Signals or Checks, or weights for calculation of exercise indexes. It is particularly useful to provide this kind of data through the code list if it is expected that refinements or additions will be made during the life of the survey.

When other information is added to the code list then it will be necessary to add a Primary Key to the code list so that it can be searched. Once a matching external record has been located then it can be read and the values returned to the survey data.

3.5 Whether to add any special entries to the list during loading

For situations where a suitable entry may not be found in the code list, it may be appropriate to add a special entry, such as “Other – specify” (which would have a non-specific code number attached to it). The datamodel could be programmed to obtain a verbatim description for such cases. These verbatim entries could then be extracted from time to time for coding decisions to be made.

Another special entry which may be useful could be a “Delete me” entry. When this is chosen the datamodel could be programmed to remove the selection and its resultant code from the data. This may be the chosen solution to removing unwanted code selections because the delete key does not function on a field which is controlled by a Lookup. For example, the delete logic could look like:

```
{Ask third activity using trigram coder}
Activity_3|CodeFile.LOOKUP(TriKey).Description
{Remove activity if it starts with 'DELETE'}
IF POSITION('DELETE',Activity_3)> 0 THEN
    Activity_3:=''
    Code_3:=''
ENDIF
```

There are other solutions to the “delete” problem, such as adding a menu button to clear the field, but this method might be an easy one to implement.

3.6 Make adjustments for punctuation which may affect Trigram indexes

When the text strings which are used to create the lookup files contain punctuation such as commas, full stops, brackets etc. then they will affect the trigrams into which those strings will be indexed. Given that the operators will only be entering a few letters (and probably not any of the punctuation) then it is best to adjust the search texts so that the punctuation is separated from the words around it by a blank character or space. For example: a search string of “Food prep (cook, grill, bake)” would be indexed better if it was converted to “Food prep (cook , grill , bake)”. Alternatively, the punctuation could be removed completely.

The following procedure in Manipula applies such an adjustment to the texts before indexing:

```
PROCEDURE proc_FixString
  PARAMETERS
    IMPORT OldText : STRING
    EXPORT NewText : STRING
  INSTRUCTIONS
    NewText := OldText
    NewText := REPLACE(NewText,',',' ,')
    NewText := REPLACE(NewText,'(','( ')
    NewText := REPLACE(NewText,')',' )')
    NewText := REPLACE(NewText,':',' :')
    NewText := REPLACE(NewText,'/',' / ')
ENDPROCEDURE
```

Even though this kind of adjustment is made for the indexes, the original text is still retained and returned to the survey data file when the lookup is used. The following code illustrates this:

```
Activity|CodeFile1.LOOKUP(TriKey).Description
```

Where the Secondary Trigram Key (TriKey) is based on a field called **SearchString** which is a copy of the **Description** field that has been "fixed" using the above procedure.

3.7 Make adjustments for language/keyboard differences which may affect Trigram indexes

When preparing a multilingual instrument, it may be necessary to make adjustments to handle differences in lettering due to accents. In particular, those adjustments should be made so that it is possible for the operator to use an English/American keyboard to enter search texts without the accents.

For example, Spanish words commonly contain special letters such as 'ñ' and the Spanish language also employ accents which are placed over vowels. It is not easy to access these special letters on an English/American keyboard. In order to make trigram lookups in Spanish more readily accessible to the coder, these special characters are converted to (English) versions that do not have the accents.

For adjustments to Spanish texts, the above procedure is extended with the following extra lines:

```
    ...
    NewText := REPLACE(NewText,'Ñ','N')
    NewText := REPLACE(NewText,'ñ','n')
    NewText := REPLACE(NewText,'á','a')
    NewText := REPLACE(NewText,'é','e')
    NewText := REPLACE(NewText,'í','i')
    NewText := REPLACE(NewText,'ó','o')
    NewText := REPLACE(NewText,'ú','u')
ENDPROCEDURE
```

Once again, the Trigram key will make use of the modified text whereas the Lookup will return the original Description text.

3.8 Adding the code numbers to a Trigram index to increase functionality

Sometimes coders will become familiar with the code numbers used in the coding frame and so it may be useful to enable them to search on those as well. This can be enabled by appending the code number to

the search text used in the Secondary Trigram Key. For example:

```
SearchString := Description+' '+Activity_Code
```

Here also, the Trigram key will make use of the modified text whereas the Lookup will return the original Description text.

Using this technique can also provide a convenient search mechanism to locate all the entries coded to the same number.

4. Deployment of code list (lookup) files

Code list lookup files must be accessible to the systems in which they are to be used. When it comes to deployment, the main issues to consider are whether the lookup files are likely to be used by multiple surveys, how often they are likely to be updated and, in the case of network configuration, performance.

4.1 Distributed computer deployment

For distributed systems where each interviewer has his/her own notebook computer there are two sensible options for placement of the lookup files:

- in the same folder with the survey files
- in a common folder placed next to the survey folder

The first location is appropriate when the code list is unique to a survey and/or there are likely to be some updates during the life of the survey. Management of the installation tends to be easier because it just entails replacing the files in the survey folder with an updated set.

However, where code lists can be applied to multiple surveys then it is appropriate that you place all the code list files in a single folder which is parallel to the survey folders. For example, the following folder structure could be used:

```
C:\DATA\EXTERNAL
C:\DATA\Survey1
C:\DATA\Survey2
...
```

In this case, all the shared external lookup files would be placed in the EXTERNAL folder. This also has the advantage that there would be only one copy of each code list installed on the computer rather than one copy per survey.

If the code list files are always placed in the EXTERNAL folder (and parallel to the survey folder) then the file references to these external files can be easily managed using relative referencing. For example, the following Uses and Externals statements would locate the relevant country list in the EXTERNAL folder:

```
USES CountryList '..\EXTERNAL\CountryList_1_021'

EXTERNALS ExternalList : CountryList
   ('..\EXTERNAL\CountryList_1_021',BLAISE)
```

Using these references (involving **`..\EXTERNAL\`**) means that the survey datamodel will always find the external file(s) provided they are in a parallel folder called EXTERNAL.

4.2 Network system deployment

With network systems, the placement of code list lookup files should be done to provide the best performance response. Despite the improvements with network performance in recent years the best location for lookup files is still the local computer of the operator. This applies even if the survey data is to be held on a network location but it applies even more so if the lookup files (when placed on the network) would be being accessed by multiple operators at the same time. Local placement will give exclusive and speedy access.

Local deployment for network use, involves using a designated folder on the C drive where the lookup files can be placed. When the network system is invoked the local pathname can be supplied as a parameter to the Manipula program or Data Entry Program (DEP). This can be done using the relevant command line option or by using a Blaise Command Line Options file.

For example, the following Manipula statement invokes the DEP and sets the external search path to the C drive location (**C:\DATA\AllCoders**) using the command line option:

```
aResult:=fInterview.EDIT ('/K'+aIndic
        +' /E"C:\DATA\ALLCoders"'
        +' /X ')
```

or using a command line options file:

```
aResult:=fInterview.EDIT ('/K'+aIndic
        +' @SVY_System.ini'
        +' /X ') {make sure the /X option is after the Ini file}
```

where the options file (**SVY_System.ini**) could look like:

```
[DepCmd]
ExternalSearchPath=C:\DATA\ALLCoders
MenuFile=CAPIMenu.BMF
ConfigFile=AuditTrail.DIW
[ManipulaCmd]
ExternalSearchPath=C:\DATA\ALLCoders
```

Notice that the external search path is specified for both DEP and Manipula so that either program can find the external files.

Where an external search path is to be supplied at time of execution, the references to those lookup file within the survey datamodel are done without the folder names being mentioned. For example the Uses and Externals statements previously shown would now look like:

```
USES CountryList 'CountryList_1_021'

EXTERNALS ExternalList : CountryList('CountryList_1_021',BLAISE)
```

When installing the survey it will be necessary to place the datamodel definition files (*.bmi, *.bdm, *.bxi) for the lookup files in with the survey. The other alternative would be to set the Meta Search Path in the same way as the External Search Path by using a command line option (**/H)** or adding it to the options file.

5. Version control

It is almost inevitable that coding lists will be updated. It is therefore important to consider a system of version control so that you know which version of a coding list was used with a particular cycle of a survey. Two methods are suggested here.

5.1 Version number as part of the code list file name

A version number can be incorporated into the name of the code list. For example, for release 1.01 of the Activity coding list the suggested datamodel and file name could be Activity_1_01. Whenever a new release is made the version number is updated and therefore the file names are also updated.

By having the version number embedded in the name it is clearly visible to installers of the system which version is being used. Embedding the version number in the name also means that different versions of the same coder can be available at the same time (for different surveys or time periods).

This solution is appropriate for code lists which are shared between surveys and which do not change often. In this situation a typical survey would make use of a particular release of a code list, probably for the life of that survey. Of course the name of that code list file would be referenced in any Uses statement, and any Externals section of a datamodel so it would always be known which version was being used. For example, see the Uses and Externals for the country list above.

Using this method means that any application of the code list in Manipula or a Blaise datamodel will need to refer to the correct version by its name. If the current file is not found then the error message will name the file (with version number embedded) and it will be easy for the maintainers of the system to know which file is not in place.

Using this method does mean, however, if a correction is made to a code list and a new release is required then the file name(s) would be changed to match the new release number. It would be up to the managers of any affected survey to decide whether to adopt the new version and change and recompile any datamodels or programs. Provided that the structure of the datamodel used for the code list file did not change then the modified and recompiled survey datamodel will be compatible with the previous version.

Updating a code list while a survey is in progress does raise the issue of how to refresh the lookups and searches which were done in the earlier period of the survey. This is discussed further later.

5.2 Version number as part of the folder name

If a code list lookup file is expected to be changed a few times during the life of the survey then the version number can be embedded in the name of the folder where the external files are located. That way the name of the code files can remain constant and the system programs will not need to be modified or recompiled.

By having different folders for each release it will be necessary only to update the external search path used by the DEP or Manipula. In that case the use of a Blaise Command Line Options file would be advisable. Being a simple text file this would be the only part of the installation which would need to be updated or replaced whenever a new code list was released. The following is an example of a Blaise Command Line Options file used to operate the Physical Activity Recall (PAR) survey:

```
[General]
Version=1_053
Installer=Fred Wensing
SysFolder=H:\Blaisedata
CoderVersion=3_013A
CoderLocation=C:\DATA\PARCoder
Managers=4,9,52
[DepCmd]
ExternalSearchPath=C:\DATA\PARCoder\Version_3_013A
MenuFile=PARMenu.BMF
ConfigFile=AuditTrail.DIW
[ManipulaCmd]
ExternalSearchPath=C:\DATA\PARCoder\Version_3_013A
```

As you can see the folder containing the coder lookup files is named Version_3_013A. When a new release is made then another folder name would be used.

This INI file (called **`PAR_System.ini`**) is used to define various system settings and is interrogated by the Maniplus program which controls the survey on the network. The same INI file is passed to the DEP as a command line options file. Whenever a new coder is released the system is programmed to update this INI file to reflect the change of version number and the changes to the folder name(s). This would then flow into all the uses of the system.

6. A management system for Classification lookup files

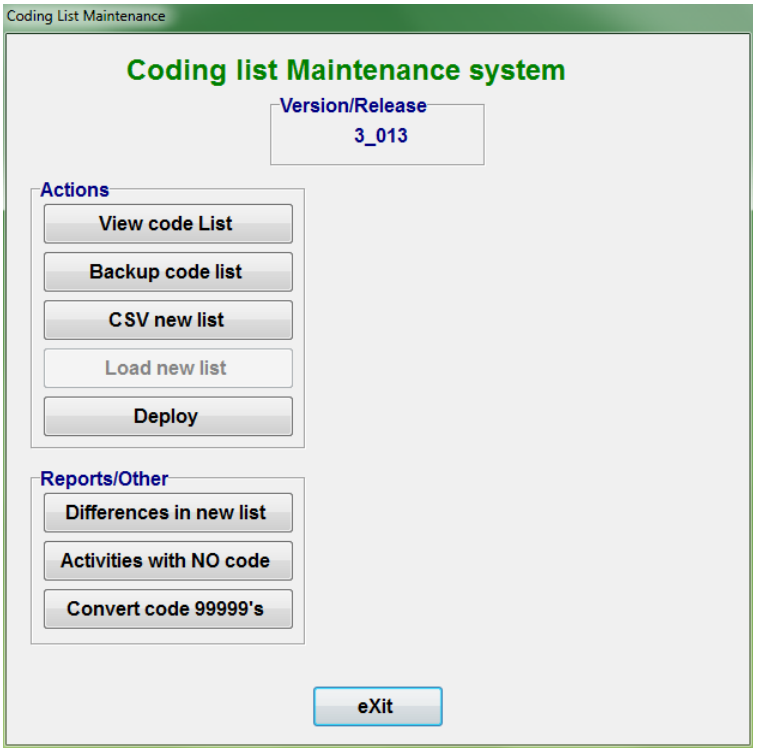
When code lists are likely to be updated during the life of a survey it would be appropriate to develop a management system to handle all the steps involved.

In a typical classification management system the following processes would need to be supported:

- knowing and updating the version control number
- backing up the current coder(s)
- extracting a list of entries from the survey data file(s) which could not be coded – that list would be sent to coding experts to make decisions and update their synonym list
- importing an updated list of synonyms into the classification management system
- obtaining a report on the code changes made in the updated list (in case there are unexpected changes)
- preparing the updated code list for deployment
- applying the updated coder to attempt to code those which were not coded previously

A system to handle these processes can be prepared using Manipula/Maniplus.

The following screen shows what such a system could look like:



Behind each of the buttons in this interface is a procedure or a Manipula program which carries out that step. The current version number is stored in an INI file which is interrogated by the system before being displayed on screen or used in other steps.

Maintenance tasks involve running the “Action” processes in order. The other processes can be run as needed.

6.1 Notes on the maintenance system implementation

The code list maintenance work is done in a folder which is separate from the current survey system even though the maintenance function is accessed from within the survey system.

The datamodel which defines the code list includes a field to record the code which a particular entry may have had in the previous release. The old code is obtained by checking whether an entry is in the previous release and noting what the code used to be. Storing both the current and previous codes makes it simple to produce a report of any changes that may have occurred. The differences report process also checks the backup code list for any entries that may have been removed before reporting the results.

In the main survey data file, entries that cannot be coded are “entered” as a literal string (using a button on the menu). Because those entries are not (yet) in the code list, the system is programmed to record the code number 99999 against those entries making it easy to identify them for listing and subsequent update when the lookup list has been improved.

The coding maintenance functions themselves do not interfere with the operations of the survey. Installation of updated code list files to their correct location (on the local computer of the operator) is triggered by the main system programs once the version number is updated.

6.2 Some technical details of the maintenance system implementation

Within the maintenance folder there is a copy of the current code list file and all previous releases of the code list. Each old release is named such that the suffix on the name matches the version release number.

The **View code list** button will open the current version of the code list using the Data Entry Program in the Browse Forms mode. That way progress with the update can be checked by the operator.

The **Backup code list** button runs a procedure that copies the current code list file to one which has the same name plus a suffix which matches the current version number (namely that shown on the screen). The system will only allow this button to be pressed once per release. Once the backup exists then the current version becomes the draft for the next release.

The **CSV New list** button opens a "Selectfile" dialog to locate a suitable CSV file to be loaded.

The **Load New list** button runs a procedure which loads the code list entries from the identified CSV file. During the loading process the text entries are "fixed" for punctuation and language issues as outlined in Section 3. In addition, as each entry is loaded, the backup copy of the code list is checked to see whether the entry existed previously and if it did then the old code number is copied into the "Previous code" field.

The **Deploy** button copies the current (now updated) code list files to a staging folder in the survey system where they can be accessed for installation later.

The **Differences in new list** button runs a Manipula program that reports on the changes in the current code list from the previously backed up copy. It identifies new entries, deleted entries and changes in the codes. The text report is written to a folder where it can be accessed later. That way all the changes between the different releases are stored for future reference.

The **Activities with No code** button runs a Manipula program which examines the records in the survey data file and produces a report of all the description which do not have a valid code (may be coded to 99999). That text report is also written to a folder where it can be accessed later.

The **Convert 99999's** button runs a Manipula program which examines the records in the survey data file and, for records that contain one or more entries coded to 99999, it applies the Checkrules method with the setting of CheckrulesUnchanged=YES. A report on records that have been "fixed" and the number of remaining non-coded entries is also produced and written to the same reports folder.

7. Handling some special classification issues

This section discusses some special issues that may arise with using external lookup files.

7.1 Refreshing searches following update of code list

When a code list has been updated it may be necessary to refresh the searches to ensure that any data based on the lookup entries is up-to-date.

If an affected record is to be reopened using the Data Entry Program then all the searches and retrieval of external data for that record will automatically be refreshed.

To ensure that all the searches are refreshed across the whole data file, however, it will be necessary to open every record and reapply the Rules. This can be done using a Manipula program but it will be necessary to apply the additional setting of CHECKRULESUNCHANGED=Yes. This is because the default checking process will only check blocks that have changed, and where external files are involved, such a change will not be known to the block. The following sample program from the *Blaise Online Assistant* shows how it can be done:

```
USES
    MyModel 'LFS98'
UPDATEFILE
    Lfs: MyModel ('lfsdata', BLAISE)
SETTINGS
    CHECKRULES = YES
    CHECKRULESUNCHANGED = YES
MANIPULATE
    Lfs.WRITE
```

7.2 Not losing data when the external code list file cannot be found

If the external lookup files are placed in a parallel folder, or on a folder on the local computer, then the potential exists for data to be lost because the search could not find the file. This is particularly possible when the survey files are moved around during processing.

One way to avoid this happening is to add a KEEP instruction to affected fields in the survey datamodel when the search is unsuccessful. For example:

```
IF ExternalList.SEARCH(Country_of_birth) THEN
    ExternalList.READ
    Country_Code:=ExternalList.CountryCode
ELSE
    Country_Code.KEEP
ENDIF
```

This technique is always valid when the same external file (**`ExternalList`** in this case) is used for both the lookup and the subsequent search/read. The assumption is that once a value has been obtained using the search (from the same unchanged file) then the result of the most recent successful search would still be acceptable.

This technique should not be used when the content of the lookup file has been changed during the life of the survey because the assumption may not hold true.

7.3 Use of ASCII file for small lookups and ease of updating the code list

Where code lists need to be updated often or tailored for particular uses then a simple solution may be to use an ASCII file to hold the code list rather than a Blaise file. In that case the Uses and Externals statements would look like:

```
USES Citylist 'Citylist'

EXTERNALS ExternalList : CityList('Cities.csv',ASCII(SEPARATOR=',')
```

where **Citylist** is a normal Blaise datamodel with primary and secondary keys defined. In this case the external file is comma-separated which is why the separator setting is added. For a fixed format ASCII file the separator would be left out.

Using this method, separate city lists could be provided to different interviewers, matching their local area. There would be no need to reload the data for a new or modified list. Blaise also loads the list into memory which is faster for the lookup. The only disadvantage is that the Trigram indexing is not available for lists that are loaded into memory.

7.4 Keeping track of alternate External lookup files in a multilingual survey

For multilingual surveys it is preferable that the lookups will be done using files corresponding to the active language. So for every lookup instruction there will be conditional logic to direct the operator to the relevant language lookup file. The problem is that if the active language is changed then all the searches in the datamodel, including those already performed, would be switched to the new language and many of them would fail. There is a serious risk of losing some of the data already collected.

To control the problems which can result from a change in the active language it is necessary to note the language for each lookup when it is performed and store it in a flag field. That flag can then be used to control which lookup file is searched and stop the possibility of using the incorrect lookup file (for each instance).

For example, the Physical Activity Recall Survey was developed as a bilingual survey with all questions and lookups being available in both English and Spanish. The operator can change the language at any time to use the one which is relevant for the respondent. The following logic is used to set the language flag for once instance of the Activity question (which uses trigram lookup on a list of activities):

```
{Set the language for the Activity coder}
Activity_Lang.KEEP
Activity.KEEP
IF (Activity=EMPTY) THEN
    IF Purpose=EMPTY THEN
        Activity_Lang:=''
    ELSEIF Activity_Lang=EMPTY THEN
       IF ACTIVELANGUAGE=ESP THEN
           Activity_Lang:='Sp'
       ELSE
           Activity_Lang:='En'
       ENDIF
    ENDIF
ENDIF
{show the language being used}
Activity_Lang.SHOW
```

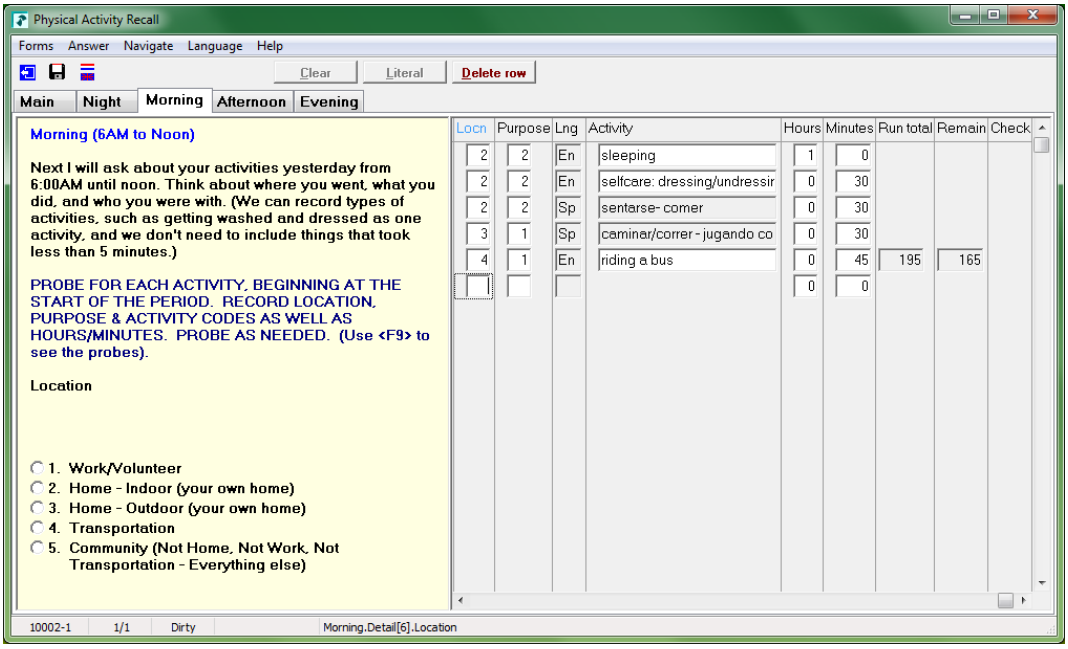
The language flag is checked and if it matches the active language then the lookup action is allowed, otherwise it is changed to just SHOW the results of the lookup when it was done in the other language:

```
IF Activity_Lang='En' THEN
    IF ACTIVELANGUAGE<>ESP THEN
        Activity|CodeFile1.LOOKUP(TriKey).Description
    ELSE
        Activity.SHOW
    ENDIF
ELSEIF Activity_Lang='Sp' THEN
    IF ACTIVELANGUAGE=ESP THEN
        Activity|CodeFile2.LOOKUP(TriSpanish).Descr_Spanish
    ELSE
        Activity.SHOW
    ENDIF
ENDIF
```

All follow-up searches of the external code list are also controlled by the language flag:

```
{Use the codefile to assign activity code number}
IF Activity_Lang='En' THEN
    IF CodeFile1.Search(Activity) THEN
        CodeFile1.READ
        Activity_Code := CodeFile1.Activity_Code
    ENDIF
ELSEIF Activity_Lang='Sp' THEN
    IF CodeFile2.Search(Activity) THEN
        CodeFile2.READ
        Activity_Code := CodeFile2.Activity_Code
    ENDIF
ENDIF
```

The following screen shows the effect of using this technique on the data capture process:



You can see the language used flag next to each activity text (obtained by lookup) and the (Spanish) entries which do not match the active language are displayed as SHOW text only. Those fields cannot be changed while the language is set to English. If the language is changed to Spanish then the English text entries would become SHOW text only and the Spanish text entries would be able to be changed.

8. Conclusion

This paper has covered a range of material associated with classifications and the use of external lookup files with the objective of explaining the processes involved and providing some techniques and solutions for the issues. Complimentary information and detailed technical descriptions may be found in the *Blaise Online Assistant*.