Dynamic List Building Using Blaise

Barbara S. Bibb & Gilbert Rodriguez, RTI International

1. INTRODUCTION

Data collection instruments frequently ask redundant questions that collect data containing the same or similar information. For instance, enumeration questions might collect addresses for multiple people where each person may have the same address, or might have a different or alternate address. This paper addresses the problems and issues of building a dynamic list of respondent answers and using that list as a 'pick list' for answers to future questions in the instrument for the administration of that same case.

To reduce both interviewer error entering the data as well as respondent burden repeating the same answers over and over again, Blaise can be programmed to build a list of suitable answers maintained within each case. For subsequent times through these questions, the instrument will display a list of the previously entered answers. If the next respondent answer is on the list, the interviewer can simply pick the appropriate entry. If the respondent offers a "new" answer, that answer will be entered as data and added to the evolving list. That answer will become available the next time a question is asked where that set of categories is appropriate. This paper will discuss the programming necessary to build such lists.

2. GENERAL BACKGROUND

A survey is an efficient tool for collecting data from various pools of respondents. For collecting some types of non-sensitive data, a survey administered by an interviewer yields more accurate data than a self-administered survey. To increase the benefits of interviewer-administered surveys, respondent burden and the burden on interviewers should be kept to a minimum. To reduce interviewer burden, the amount of effort to enter an answer into a CATI instrument should also be kept to a minimum.

Some data collection efforts attempt to collect repeated, open-ended data. In some cases, several of the questions can have the same or similar answers. String data, repeatedly collected, can be tedious to key. Anything tedious to key is also very error prone.

Data entry of string data that is keyed multiple times cannot be easily compared. Any variation in spacing, spelling, and capitalizations will cause the comparison for a match to fail. By putting such data on a pick list, each occurrence of the entry will be exactly the same. This will create a more consistent set of string entries for each data record.

Examples of such data:

- Addresses associated with each household member
- Names of schools attended by members of a household
- Names (Last)
- List of medications

- Services received by members of a household
- Phone numbers
- Company names

The goal of list building code is to create a list of string responses within each administration of the instrument. This list is used as a tool for the particular household and particular questions being administered. This list is displayed for each question using the required set of similar responses. The interviewer can add a new response or pick a previously recorded response to answer the current question. The previously entered responses will store the data the same way it was originally entered. There will not be any variation in the string response from question to question where the answers are the same as a previous answer.

For example: Roster the children of a household. Collect data for each child enumerating the names of the schools each child attended during early childhood and elementary school.

> Child 1 attended Community Preschool, Mount Forest Elementary School and Bolin Creek Elementary School
>
> Child 2 attended River Ridge Elementary School and Bolin Creek Elementary School
>
> Child 3 attended Mount Forest Elementary School and River Ridge Elementary School.

The first time a question comes up asking for the school name the list would be empty. The only option would be to enter a “new school name”. The second time the question comes up, the first answer would follow the option to enter a “new school name”. The interviewer could either choose the name previously entered or enter a “new school name”. If the response is the previously entered name, the interviewer would save time and effort by choosing the answer rather than re-entering that same answer previously mentioned. The working list would be maintained by the program and additionally the correct responses are kept for each child.

3. SPECIFIC BACKGROUND

The particular code being discussed was developed by RTI International for the Bureau of the Census (BOC). It was implemented for the BOC QDERS (Questionnaire Design Experimental Research Survey) Project. For this survey, each household was rostered. Once the roster was complete a series of follow-up questions were asked about each household member and the possible alternate addresses they might have had. Data was stored on a household level and by person. Specific questions were asked to determine where each household member was actually living throughout the year. The answers to the address questions built the pick list for the subsequent questions. Multiple addresses occurred if any household member stayed any place other than the primary residence.

For example household members could have been away for a job, away at college, in the military, living in a joint custody situation, relocated, had second homes, etc.

In the following discussion and for the examples, all names, addresses, and other data elements are fictitious. They were inserted to provide realistic looking examples.

4. IMPLEMENTATION

4.1 Considerations

Before beginning to code, programming constructs were developed to handle the process of maintaining the address lists. There are numerous issues to consider before beginning. Thought was given to the block structure needed to handle the task. Considerations included:

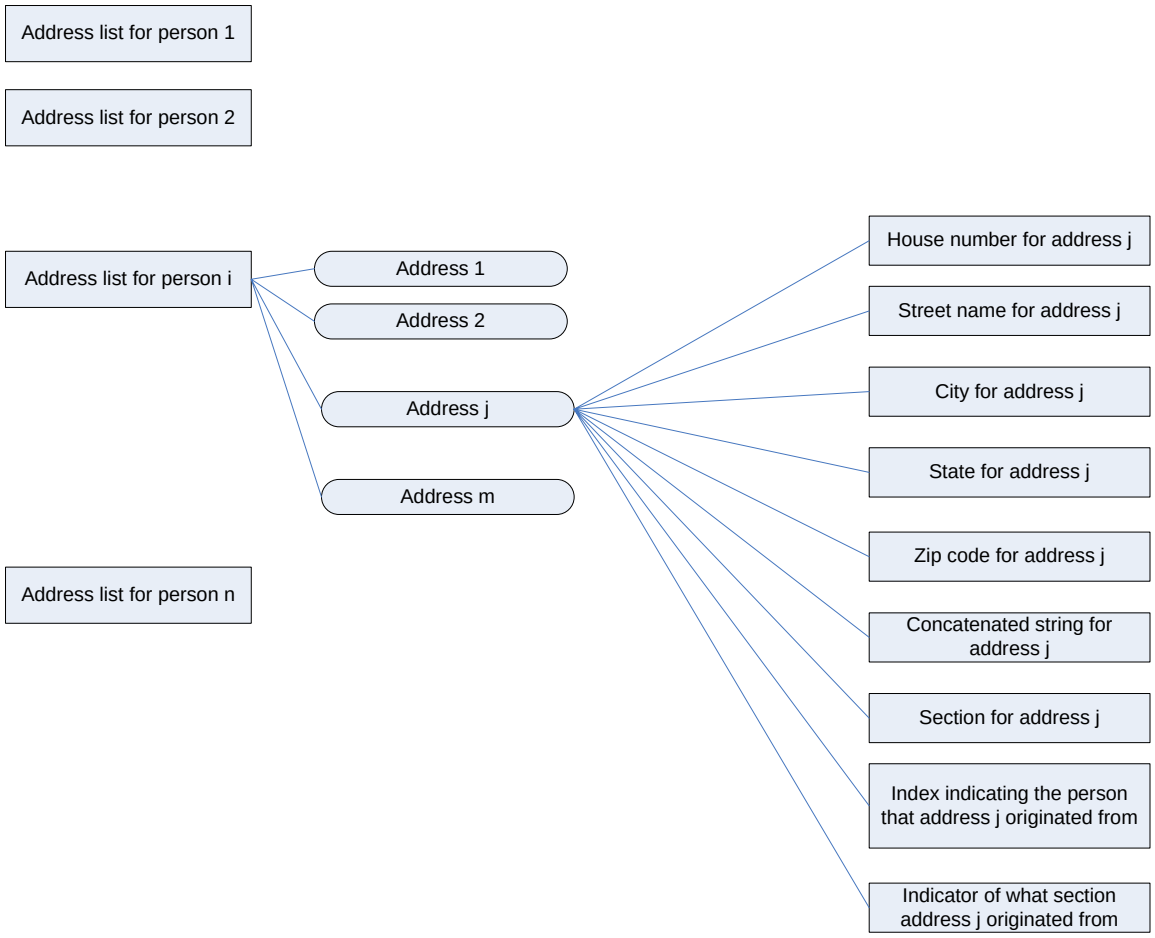
- the way the Blaise application re-evaluates the path throughout the interviewing process
- the number of points at which addresses could be added
- exceptions to the standard address collection block
- the handling of the counters to keep them in sync
- individual data storage for each household member
- global address collection array for unique addresses within the instrument

The complicated path structure and Blaise's variable assignment methods influenced the coding used to keep the lists of variables up to date and accurate. Counter variables can easily get out of sync, especially in a heavily nested block structure. Corresponding local variables were used to compensate for the possibility that the global variables might not remain accurate depending on the path through the instrument.

The standard address collection block was called using parameters. These parameters described where the data was coming from. Variables were kept so that the original source of each address could be tracked. These parameters decreased the reliance on global variables that could easily have gotten out of sync.

4.2 Coding design

A 1-dimensional block array indexed by the person in the household roster is defined at the top-level of the datamodel. Within the block array, array fields are defined corresponding to the different addresses for a particular person. These fields correspond to address number, street, state, zip code, and from which section of the datamodel the address originated. So, a list of addresses will be maintained for each roster member.



4.3 Implementation outline

The block bhhmemaddresses tracks unique addresses for a particular person.

```
block bhhmemaddresses
  parameters
  import persnum : integer

  fields
      numaddr : 0..15
      numaddrHN : array[1..15] of STRING[21]
      numaddrSTREET : array[1..15] of STRING[60]
      numaddrCITY : array[1..15] of STRING[20]
      numaddrSTATE : array[1..15] of STRING[3]
      numaddrZIP : array[1..15] of STRING[5]
      numaddrSTRING : array[1..15] of STRING[130] { concatenated address string }
      numaddrFIELD : array[1..15] of STRING[3] { indicator of where the address comes
from }
```

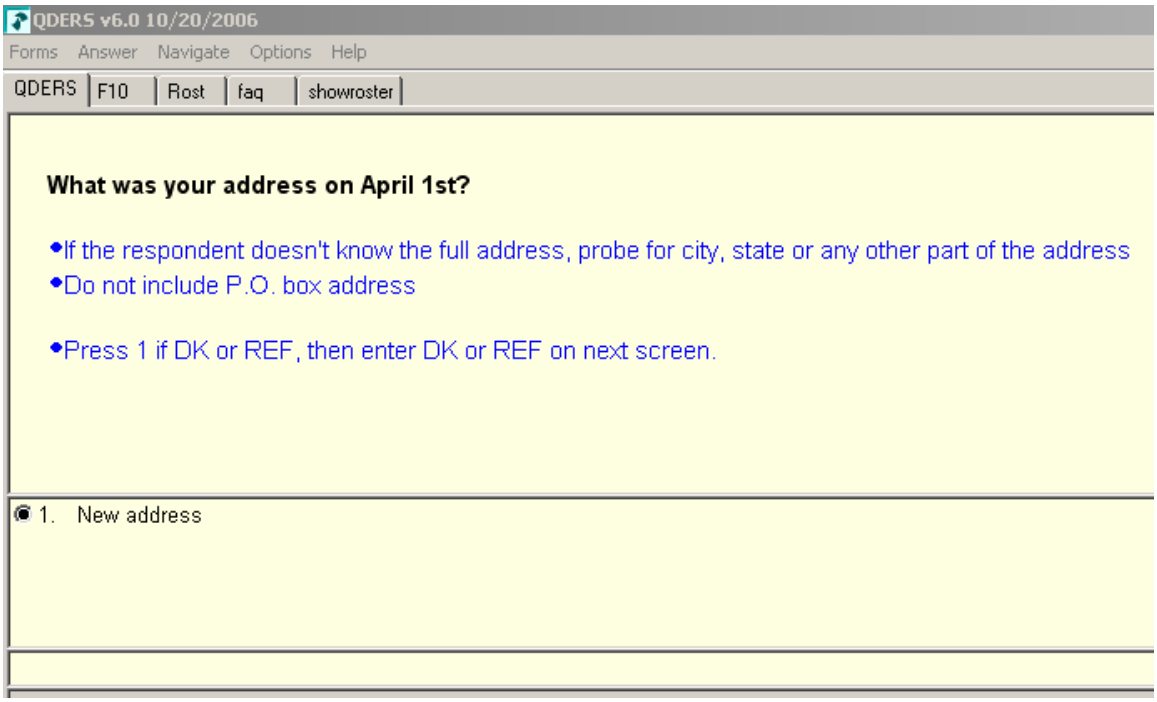
```
      numaddrROSTNUM : array[1..15] of 1..9    { person }
      numaddrROSTFIELD : array[1..15] of STRING[3]
 auxfields
      tnumaddrHN : array[1..15] of STRING[21]
      tnumaddrSTREET : array[1..15] of STRING[60]
      tnumaddrCITY : array[1..15] of STRING[20]
      tnumaddrSTATE : array[1..15] of STRING[3]
      tnumaddrZIP : array[1..15] of STRING[5]
      tnumaddrFIELD : array[1..15] of STRING[3]
      tnumaddrROSTNUM : array[1..15] of 1..9
      tnumaddrROSTFIELD : array[1..15] of STRING[3]
      tempaddresslist : array[1..15] of STRING[120]
 rules

     {initialize temporary arrays to hold the address values for this person}

     { for each address block in the datamodel for this person }

     if address block for this person is not empty then
        set temporary array elements corresponding to this address
     endif

     { In the address list for this person, only keep unique addresses and store them in the
address array }

     { tally the number of unique addresses and set the number of unique addresses for
this person }


endblock

DEFINE FIELD BLOCK ARRAY OF ADDRESSES FOR EACH ROSTER MEMBER

memaddresses : array[MAX SIZE OF HOUSEHOLD] of bhhmemaddresses


DEFINE GLOBAL FIELD AND ARRAYS FOR ALL UNIQUE ADDRESSES
THIS IS USED FOR DISPLAYING PREVIOUSLY-ENTERED ADDRESSES IN
ADDRESS QUESTIONS

numaddrall : 0..MAX TOTAL ADDRESSES

address_list : array[1..MAX TOTAL ADDRESSES] of string[120]
addrlist_hn : array[1.. MAX TOTAL ADDRESSES] of string[21]
addrlist_street : array[1.. MAX TOTAL ADDRESSES] of string[60]
addrlist_city : array[1.. MAX TOTAL ADDRESSES] of string[20]
addrlist_state : array[1.. MAX TOTAL ADDRESSES] of string[3]
addrlist_zip : array[1.. MAX TOTAL ADDRESSES] of string[5]
addrlist_rostnum : array[1.. MAX TOTAL ADDRESSES] of 1..9
addrlist_where : array[1.. MAX TOTAL ADDRESSES] of string[5]
```

IN THE RULES FOR THE MAIN BLOCK, PLACE THE FOLLOWING LOOP IN KEY LOCATIONS TO KEEP THE ADDRESS LISTS UP-TO-DATE.

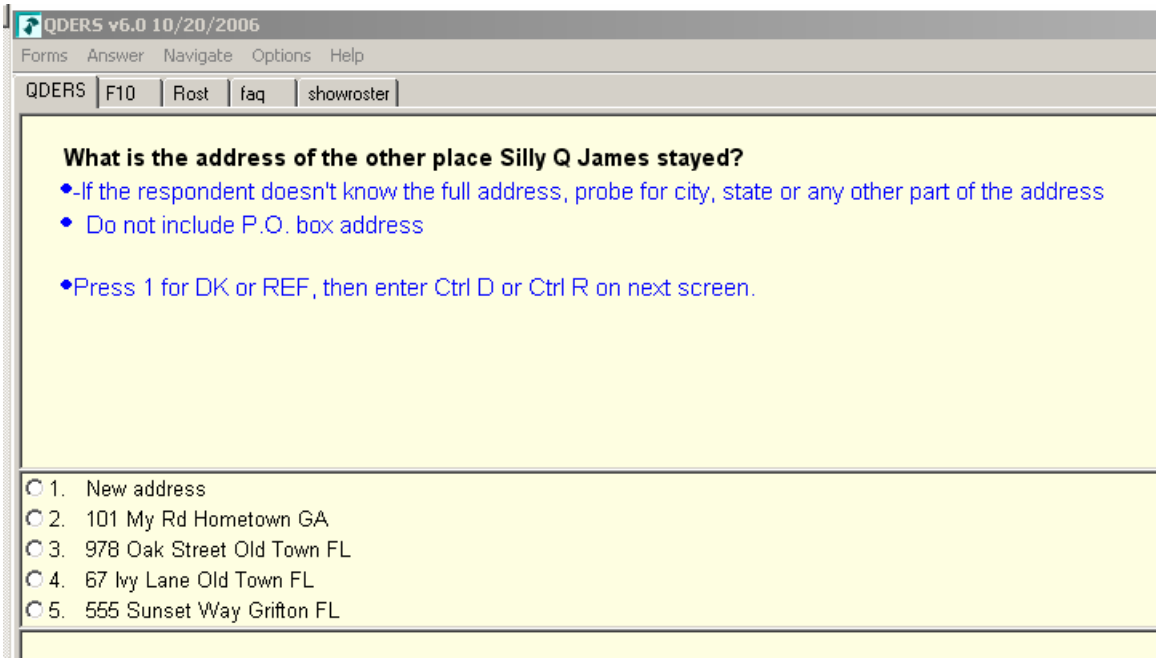
```
for i := 1 to NUMBER OF ROSTER MEMBERS
do
   memaddresses[i](i)
   memaddresses[i].keep(i)
enddo
```

In addition to the addresses for each roster member, a global address field array is maintained that contains the unique addresses. Associated with this global address field array is a block type that displays these addresses for fields that require an address to be entered.

In a field requiring an address, the field will have a type with the first response choice being "New address", and the remaining response choices showing the unique, previously-entered address strings, collected on the global address list. For the first potential address, the address list will be empty, so "New address" will be the only response option available.



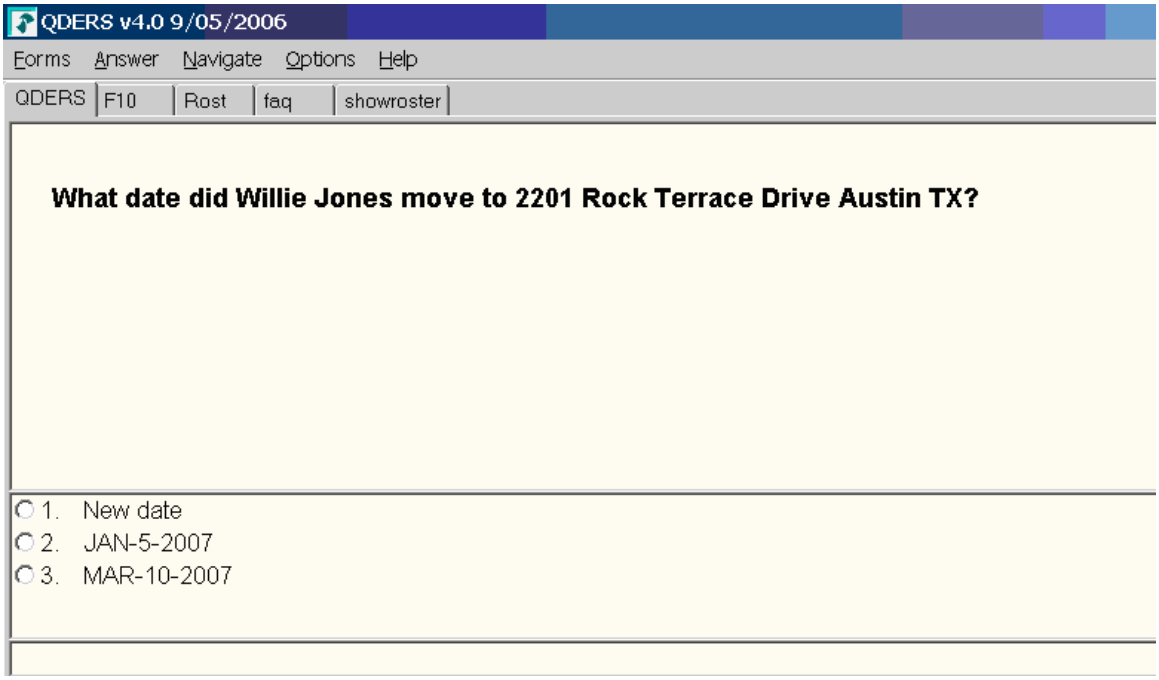
Here is an example screenshot displaying some previously-entered addresses.



If response category 1 ("New address") is selected, then follow-up fields are displayed for entering the various address fields. Once the address fields are filled in, then the address is checked for uniqueness against the list of all addresses. If the new address is unique, then it is added to the global address list and the global address count is incremented. The field values linked to the new address are set to indicate which address question and for which roster member this address came from.

If an existing address is selected, then the fields for the current address are set equal to the corresponding fields of the existing address.

In the BOC QDERS project a similar approach was also used for dates. This concept can be adapted to work with any other commonly-used list of responses.



5. CONCLUSIONS AND SUMMARY

The purpose of developing a dynamic list is to reduce both respondent and interviewer burden. Reducing burden increases accuracy and provides cleaner back end data. The methodologists at the BOC asked that the dynamic list approach be implemented for their QDERS project, specifically for the collection of household addresses and move dates. This concept can be applied to any open-ended data collection effort where enumerated categories are not readily apparent. This approach is most appropriate for string data but can also be used for other data, like the QDER date data.

Some observations for the QDERS project:

1. Not all address questions are asked in exactly the same way, and oftentimes special case code had to be inserted in the address list maintenance based on the address question. For example, if an address was a military address, the datamodel had to be able to handle if the address was onboard a ship.

2. Code for displaying addresses had to be repeated in some locations within sub blocks so that newly-added addresses would consistently appear in the response category address lists. This is due to how the Blaise Selective Checking Mechanism works. At the block level, if a new address was added to the global address list, then there were situations where the address would not immediately appear in the list of addresses until a re-evaluation of the rules was performed.

3. If the user backed up, then you would only want addresses that were entered up to that point to appear in a response list. This was done by suppressing addresses from appearing if the question from which the address came from comes after the current address question.

4. Due to nesting of block arrays within other block arrays, processing of data required special effort.

6. ACKNOWLEDGEMENTS

Parts of the research upon which this report is based were supported by the U.S. Census Bureau, and we would like to acknowledge members of the BOC Questionnaire Design Experimental Research Survey team for their help. The views expressed are those of the authors and not necessarily those of the U.S. Census Bureau. This report is released to inform interested parties of ongoing research and to encourage discussion.