Paper 018-30

Building a Data-Driven Computer Assisted Interview Using SAS/AF®

Derek Morgan, Washington University Medical School, St. Louis, MO

ABSTRACT

Creating a respondent-usable survey using the SAS® System involves a large amount of developer time using traditional SAS System methods of data capture with an end-user interface, since it is directly linked to the size of the survey. In such an environment, each survey has to be custom-built and coded. This method was developed to bypass that and other obstacles related to end-user data capture. This method uses SAS datasets to implement program control, as well as provide questions and responses. A fixed series of SAS/AF frames interact with the user, yielding a consistent user interface. The frame code does not change between surveys—the datasets do. However, if the user needs to create a custom frame to handle the administration of a particular question, it is easily integrated into the existing method.

This method was initially developed in order to administer the diet history questionnaire (DHQ). The DHQ is a food frequency questionnaire developed by staff at the Risk Factor Monitoring and Methods Branch of the National Institutes of Health. It consists of 124 food items and includes both portion size and dietary supplement questions. The 124 food items comprise 404 questions with extensive skip patterns. However, there was no instrument available other than a paper form to administer this questionnaire. Traditional data entry methods using SAS/FSP or SAS/AF would have required a disproportionate amount of programming resources. This paper will use the prototype application to illustrate the development of the method and its basic features.

DISADVANTAGES OF THE TRADITIONAL SAS/FSP METHOD

Think about creating either an FSEDIT or a PROGRAM screen with four hundred seven fields. Think about typing the full text of each question, along with the possible responses for each question into the screen. The FSP application would have one labeled section of code for each field in addition to the INIT, MAIN, and TERM sections in a full-text, tightly controlled SAS/FSP application. The SCL code would be at least a thousand lines. A single SAS/AF FRAME hardly improves the situation. All of these methods can require a great deal of programmer time during maintenance, too. What happens if text in the questionnaire changes or a question is added or deleted? All of this adds up to a maintenance nightmare for something that already has a high development cost in terms of man-hours. While our traditional method[1] is still a valuable tool for small projects and questionnaires, the parameters of the DHQ clearly required an alternative method.

Another disadvantage is that FSEDIT screens are not re-usable. They may be recycled, but if anything changes from project to project, the screen will require some amount of reprogramming or redesign. Again, more man-hours spent in development. Part of the issue is rapid deployment of a series of questionnaires. It does not take very long to modify one questionnaire or even build a short one from scratch using FSEDIT. However, if you have a series of forms for data entry, or one of those forms requires a great deal of coding, FSEDIT begins to lose its attractiveness as a data capture tool.

DEVELOPING THE METHOD: SYSTEM ANALYSIS

The first thing we had to do is to look very carefully at the questionnaire to see what we were trying to emulate on the computer. When you go through the DHQ, the first thing that you notice is that the survey questions fall into distinct categories: "How often did you eat or drink {food item}?", "Each time you ate/drank {food item} how much did you eat/drink?", "Which of the following {food items} did you eat/drink? (Select all that apply)". In addition, most of the questions begin with the phrase, "Over the past 12 months." This commonality allows us to look a little closer at the types of responses that are expected.

Every response to each food question falls into one of two categories: single-response, or multiple-response. However, one set of single-response questions requires a different type of flow of control than the rest of the survey. The only other questions asked in the DHQ are three demographic questions, and of course, a variable will be needed to keep track of unique respondent identifiers. When you look at the survey in detail, you see that many questions use the same set of possible responses. We did a careful check, and discovered that there are 126 unique sets of single-response choices. There are seven unique sets of multiple-response choices, with the maximum number of responses for any multiple-choice question being 24.

The questions flow in a distinct order, with skip logic to bypass the quantity questions when foods are not consumed. The re-use of response categories and the similarities between questions allows us to factor the common elements, and that leads us to the possibility of using metadata for both questions and responses. Although the previously-noted exception requires special handling for its flow of control, it does use a common set of single-response choices, and therefore can still be driven by response metadata.

Both questions and responses are contained in their own tables. The question table (QDS) is what drives the application. It needs to contain all the information necessary to dictate flow of control, the set of possible responses associated with each question, and data capture information. The response dataset (RDS) is only responsible for the response text of each item and the values associated with that text.

METADATA TABLE 1: THE QUESTION DATASET (QDS)

Beyond the obvious (question text and corresponding response set), what do we need in this table to drive an application? First and foremost, we need an order, since the DHQ is an ordered questionnaire. While the question dataset has an ordering variable, it is not responsible for the survey's flow of control. In production, the ordering variable is only used to sort the dataset for production of hardcopies of the metadata in question order. By not using this ordering variable for the survey control, we allow for the insertion/deletion of questions without regard to their physical location in the table.

There is skip logic in the DHQ, so the application needs to know where it's going and what makes it jump. That means it also needs to know where it is, and where it would go if it the skip pattern is not invoked. We created a variable for the question number to identify each question, separately from its order. If we were to use the order variable, changing the questionnaire would require altering the order variable for all records subsequent to the added/deleted record, and re-sorting the QDS. By using this question number variable for survey control, the only records that need changing are those prior to and following the addition/deletion. Normal survey flow of control requires a variable to dictate the next question in the survey sequence. In addition, since this is to be a computer-assisted interview, allowing a respondent to go back and change an answer would be a good thing, so we created a variable to represent the question previously asked. Both of these variables assume that there is no skip logic, or that the skip pattern was not triggered. To perform the skip pattern requires two variables as well: one to define the response that triggers the skip pattern, and another to define the next question in the survey sequence when the skip pattern has been invoked. Together, these four variables define the entire flow of control within the survey from the QDS.

After the survey control problem was solved, we needed a way to tell the application where to store the response. We created a variable to hold the name of the variable in the data capture table. Multiple response questions are stored as yes/no variables; one per possible response. This allows us to use the variable prefix in the QDS, and avoid having to define each variable that would contain a possible answer, creating multiple records for a single question. There were two more variables added to the QDS related to application control. One variable indicates the single response in a multiple-response answer that will cancel any positive responses, e.g., “none of the above.” The second indicates whether the prefix phrase is to be displayed with the question text. Finally, we added a variable for text capture; it indicates whether the value entered in the text frame is to be stored as a character or numeric variable in the data capture table. This is the list of variables in the QDS resulting from our systems analysis of the survey.

TABLE OF VARIABLES IN THE QUESTION DATASET (QDS)

order	Num	4	Order that questions should appear (used for sorting ONLY)
q_no	Char	32	Question number (used by program)
q_text	Char	300	Question text as it is to be displayed
rcat	Char	32	Response category (R=single response, M=multiple-response, T=text field)
skip_to	Char	32	Question number (q_no) to skip to
skip_trigger	Num	4	Answer to trigger skip pattern
prev	Char	32	Previous question in sequence (q_no)
next	Char	32	Next question in sequence (q_no)
vname	Char	32	Dataset variable name associated with question
exc	Num	3	Exclusive answer in multiple-response category (ONLY used with M)
type	Char	1	Char or Num variable (for text (T) answers)
use_common_text	Num	3	Turn pre-text box on or off?

One difficulty that occurred in programming this method was figuring out exactly what was the last question asked. It cannot be assumed that the value defined in the QDS for the previous question is the last question that was asked, because the previous question in the standard survey sequence may have been skipped due to the survey's skip pattern. It would be confusing to many survey respondents if they had to press backwards more than once in order to find the last question that they answered. It turned out that the procedure to move backwards wasn't very complicated. It only requires two additional steps, and is still driven by using the question metadata.

1. Determine if the current question is a possible skip destination by checking to see if the current question is a value in the skip destination variable. If not, use the value in the "previous question" variable.
2. At this point, you know which record points to the current question as a skip destination. Get the question number for that record, as well as the values for the skip trigger and the data capture table variable name from it.
3. Test the value for that variable in the data capture table. If it is equal to the skip trigger, then go to the question number obtained in step 2, above. If not, use the value in the "previous question" variable.

This sends the user back to the previous question answered, not the previous question in the survey sequence, and it's all accomplished by using the variables in the QDS and the data capture table.

METADATA TABLE 2: THE RESPONSE DATASET (RDS)

The second piece of metadata we need is the response dataset. Where the QDS contains one record per question, the RDS has one record per possible response, and contains all possible responses for the entire survey. Records can be placed in any order in the RDS; there are two key variables that are used in WHERE clauses. The key variables are response class, and within each response class, response index. Response class groups responses together so the application can load all the possible responses for a given question, and response index provides not only the display order for the items in each response class, but is also the application's link to the data values themselves. The application only uses two more variables from the RDS. One contains the full text of each response as it should be displayed on the screen. The other is the actual data value to be written to the data capture table when a given response is selected. The remaining two variables are used to sort the RDS for hardcopy documentation purposes. Response class designations begin with the letter “R”if it is a single-response category, or the letter “M” to indicate that it is a multiple-response category. This tells the application which module is responsible for displaying the question and capturing the data. The list of the variables in the RDS is below.

TABLE OF VARIABLES IN THE RESPONSE DATASET (RDS)

rcat	Char	32	16	Response class
indx	Num	4	8	Response index
val	Num	4	12	Value to be written to data capture table
resp_text	Char	200	48	Response text to be displayed
rtyp	Char	1	248	Response type (R or M) (for documentation purposes)
rnum	Num	8	0	Response class number (for documentation purposes)

Together, the two metadata tables provide enough information to drive an application that conducts a survey. Now that we have the guts and fuel of the engine, all we have to do is to design and build a frame for it.

CREATING SAS/AF FRAMES FOR THE METADATA ENGINE

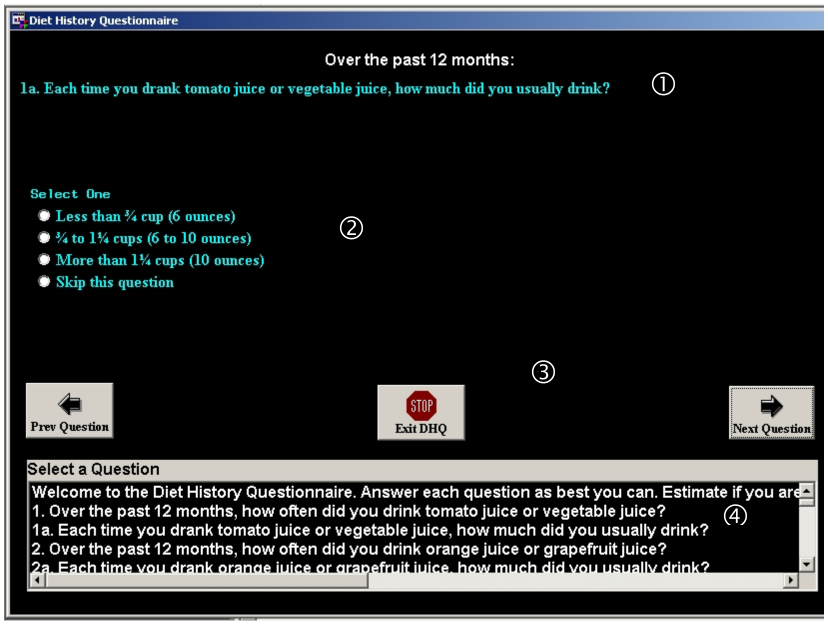
The first step was to visualize what we wanted to display. We needed a frame to display questions and their possible responses. Since we have two distinct styles of responses, it was easier to build two frames rather than to try using one frame with alternately disappearing objects. The next question to be asked is, "What, in addition to the question text and possible responses, do we need to show a person who is taking a survey?" Even though the application itself is going to handle the default flow of control for the survey, the respondent should be able to move forward and backward during the survey. Backward motion is critical so that a respondent can change a previous answer if necessary. The respondent should also have a way to leave the application gracefully before the survey is finished; longer surveys (and the DHQ takes at least an hour) may require more than one session. The ability to go directly to a specific question would be useful in an interviewer-administered or data entry from paper setting.

We worked on the frames one at a time; the single-response frame is incorporated into the main program module, while, after careful consideration, the multiple-response frame became two multiple response frames. One frame will display two columns of up to 12 response choices for a total of 24, while the other frame displays a single column of up to 12 twelve choices. This was done to provide a way of displaying longer responses using the full width of the screen. A utility text-response frame was also built. While it is not used in the DHQ application as distributed, it is used with the DHQ in our particular study in order to allow the entry of the date and the interviewer's certification number.

SINGLE-RESPONSE FRAME

The main control frame incorporates the administration of single-response questions. This was done because the majority of the questions on the DHQ were single-response. At some point, the application will be re-engineered to separate the administration of single response questions from the application control. It currently works fine in production, so this is a low priority.

SINGLE RESPONSE FRAME



Over the past 12 months:

1a. Each time you drank tomato juice or vegetable juice, how much did you usually drink?

Select One
- Less than ¾ cup (6 ounces)
- ¾ to 1¼ cups (6 to 10 ounces)
- More than 1¼ cups (10 ounces)
- Skip this question

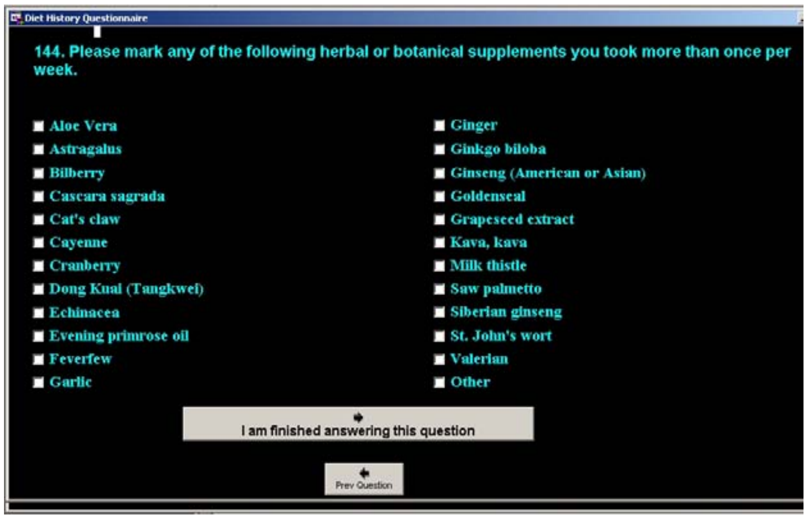
Select a Question

Welcome to the Diet History Questionnaire. Answer each question as best you can. Estimate if you are
1. Over the past 12 months, how often did you drink tomato juice or vegetable juice?
1a. Each time you drank tomato juice or vegetable juice, how much did you usually drink?
2. Over the past 12 months, how often did you drink orange juice or grapefruit juice?
2a. Each time you drank orange juice or grapefruit juice, how much did you usually drink?

This is what the single response frame looks like. The question text is at the top (❶). The question prefix is centered, and in white. It uses a different font than the rest of the frame to make it stand out visually. A variable in the QDS controls whether it is visible or not. The blue text in ❶ is the question text as specified in the QDS. The question area runs the entire width of the frame, but this particular question does not use the entire space. Item ❷ is the response box area. This area also encompasses the width of the frame, since there are several questions with more than one column of possible single-choice responses. The list that fills the radio box is generated dynamically from the RDS. The value of response category in each QDS record defines which records from the RDS are used to create the SCL list to be used. Area ❸ is the navigation area. It contains push buttons that cause the application to go to the next or previous question, and a stop button. The forward and backward buttons are only displayed if the survey context allows, that is, the last question shows no next button, while the first question shows no previous button. The combo box at the bottom (❹) is intended for administrative use only, and allows direct access to any question in the survey. It can be made invisible based on a toggle. In production for the general population, the toggle is a check box in the opening frame, but in our study, the toggle is based on the user's authorization level.

The elements required for multiple-response questions are the same as for single response questions, except that instead of getting the response from a radio box of mutually exclusive options, multiple answers are obtained from a list of checkboxes that can be independently checked. The respondent also must explicitly close the question. It is implicit that selecting one answer in a single-choice will cause the survey to advance; the question has been completely answered. In a multiple-response situation, there is no way of knowing how many checkboxes the respondent wants to select. Therefore, the respondent must take some sort of action to signal that all the desired choices have been checked.

MULTIPLE RESPONSE FRAME


Diet History Questionnaire
I am finished answering this question
Prev Question

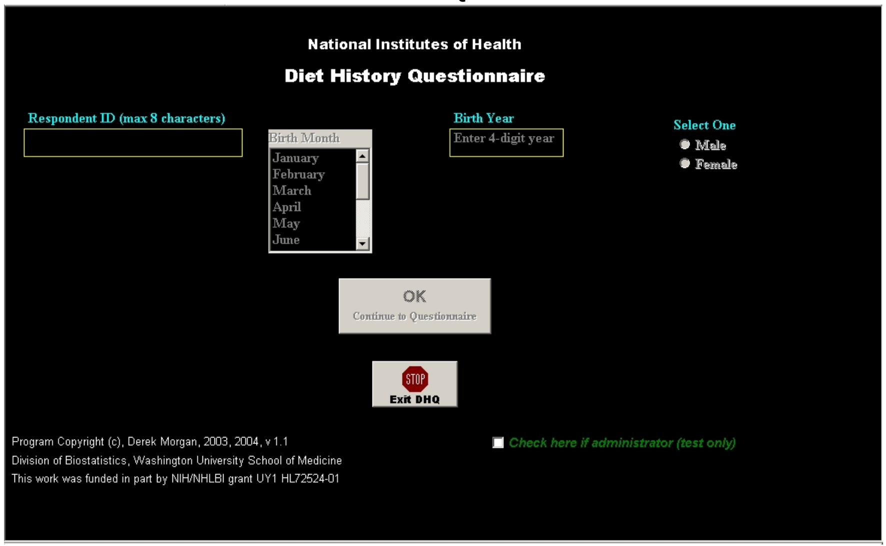
144. Please mark any of the following herbal or botanical supplements you took more than once per week.

Aloe Vera	Ginger
Astragalus	Ginkgo biloba
Bilberry	Ginseng (American or Asian)
Cascara sagrada	Goldenseal
Cat's claw	Grapeseed extract
Cayenne	Kava, kava
Cranberry	Milk thistle
Dong Kuai (Tangkwei)	Saw palmetto
Echinacea	Siberian ginseng
Evening primrose oil	St. John's wort
Feverfew	Valerian
Garlic	Other

This is the two-column, 24-checkbox version of the multiple response frame. There are twenty-four choices shown in the above screenshot, which also happens to be the maximum number of choices for multiple-response questions in the DHQ.

The two columns of 12 items are populated with a list from the RDS. The left column will be filled first, from top to bottom, and then the right. The question text from the QDS is at the top. Note that of the two push buttons at the bottom; "I am finished answering this question," is the preferred option as indicated by its size. The 12-choice, single-column version is identical in appearance except that there are only 12 checkboxes that fill the width of the screen. These three frames handle the bulk of the application, directed by the QDS metadata table, and using response information from the RDS. There are five other frames in the general application that serve specific functions. The START_DHQ frame is where the application starts. It obtains the necessary demographic information and is responsible for initializing the record for the respondent in the data capture table.

START_DHQ FRAME


National Institutes of Health

Diet History Questionnaire

Respondent ID (max 8 characters)

Birth Month: January, February, March, April, May, June

Birth Year: Enter 4-digit year

Select One: Male / Female

OK — Continue to Questionnaire

STOP — Exit DHQ

Program Copyright (c), Derek Morgan, 2003, 2004, v 1.1
Division of Biostatistics, Washington University School of Medicine
This work was funded in part by NIH/NHLBI grant UY1 HL72524-01

Check here if administrator (test only)

The TEXTRESP frame (mentioned earlier) is used to capture anything that a respondent types. It can handle both numeric and character data, up to the character limit in the SAS System. Both unformatted and formatted input are accepted, with the caveat that any formats and informats must be defined in the data capture table with the corresponding variables. The frame uses two text entry objects: one for numeric data and one for character data. They overlay each other in the frame, and the one that is not active is invisible and disabled. Which one is to be displayed is controlled by a variable in the QDS. TEXTRESP has one limitation, and that is there can be no skip logic associated with the text response.

There is also a custom frame in the prototype application. It asks the questions and obtains the responses for one specific set of DHQ survey questions whose flow of control cannot be handled by any of the main frames. It still utilizes both the QDS and RDS to display questions and capture data. The application easily accommodates custom frames as long as the custom frame adheres to a naming convention. IDIOTBOX is a utility frame that forces the respondent to confirm any change in an existing answer to a single-response question. Finally, there is the frame that generates the scoring file.

THE ENGINE

The procedure is surprisingly simple. The start-up frame creates the subject record in the data capture table if necessary and writes the demographic information that the user has entered. If the subject's record already exists, then it reads the information. It then calls the main control frame.

At initialization, the main frame uses the QDS to create the list that populates the direct question access list box, and decides whether it is to be displayed. The last initialization task is to figure out where to start the questionnaire. The data capture table (DCT) has a variable that contains the name of the current question. If it is blank or the questionnaire has been completed for that respondent, then the questionnaire starts at the beginning. Otherwise, it starts at the question stored in the DCT.

In MAIN (see appendix for flowchart), the previous and next buttons are turned on and off according to the question (first question, no previous button; last question, no next button.) If it is the last question, the frame terminates.

Now that the set-up is done, the survey can start in earnest. First, the QDS is opened using a WHERE clause based on the current question number, and the desired record is FETCHed (using CALL SET to make sure that all the variables in the QDS are available to the SCL program.) The QDS is then closed.

The next action to be taken is dependent upon the response category type. If it is multiple-response, text-response, or a custom-response, then control is passed to the appropriate frame. This can be one of four frames: one of the multiple-response frames, the text response frame, or a special frame to handle specific cases that cannot be taken care of by any of the existing response frames. Otherwise, control remains in the main control frame.

SINGLE-, NO-RESPONSE QUESTIONS AND THE CONTROL FRAME

Single and no-response questions are handled by the control frame. This isn't necessarily good programming practice, but a time-saving shortcut since this was developed as a part of a much larger production application with a very tight deadline. The first thing to happen is that the common text is turned on or off, triggered by the USE_COMMON_TEXT variable in the QDS, and then the question text is displayed. If the response category variable in the current QDS record has no value, then the response handling portion of the program is skipped. This allows "display only" questions, where the respondent is supposed to read text without answering before proceeding. The response radio box is then made invisible, control is returned to the application, and the respondent will have to press the “Next Question” button to proceed.

If a response is warranted, then the response radio box has to be displayed and populated. The RDS is opened with a WHERE clause that subsets records based on the response category variable from the QDS. An SCL list is created from the records in the RDS. One additional response option, “skip this question,” is added to all single-response questions. This is done by appending it to the SCL list in the frame code rather than by adding an extra record to each response class in the RDS. This SCL list then becomes the **list** property of the radio box.

Now that the response radio box is populated, we check for an existing answer from the data capture table. If there is one, applying a WHERE clause to the already-subset (by response class) RDS based on the value stored in the data capture table allows us to get the value of the index variable. The value of INDX becomes the **selectedIndex** property for the radio box, which is otherwise left blank. At this point, the RDS is closed and control is returned to the frame to await respondent action. When the respondent makes a selection, the application checks to see if the response is a modification of an existing response. If it is, the IDIOTBOX frame is called, and returns whether or not this response should be modified. If not, control is returned to the frame for a different respondent action.

If it is not a modified response, or the user has confirmed the modification, the response is processed. The RDS is opened with a WHERE clause that subsets the RDS on the response class and index value returned by the radio box. GETVARN() is used to obtain the actual value to be written to the data capture table. If the response was “Skip…”, the value to be written is a special missing value defined in the code.

The next step is to write the value to the correct variable in the data capture table. Only three variables are available to the data capture table when it is opened:

The ID, which is necessary to select the correct DCT record.

The LASTFLD variable, which tells the application where to re-start the survey if it is stopped.

The variable where the data should be stored. The name of that variable is obtained from the QDS by getting the value of VNAME.

The response radio box is then disabled to prevent any accidents. Finally, the next question is to be displayed is determined, based on the SKIP_TRIGGER, SKIP_TO, and NEXT variables from the QDS. If the response is not the same as the value that triggers the skip logic, or there is no skip logic (SKIP_TO=.), then the next question is represented by the value of NEXT. Otherwise, it is the value of SKIP_TO. If skip logic is triggered, then missing values are written to the DCT for all variables that corresponding to questions that have been skipped.

The loop iterates again until the final question. The entire flow of control for the survey is dictated by the metadata. There are no specific instructions inside the code about any specific field or survey question, except for the exit point. The value of Q_NO for the last question must be "fini".

MULTIPLE-RESPONSE QUESTIONS

Multiple-response questions are handled in a similar, although greatly simplified, manner. The main control frame determines which of the multiple response frames should be used by finding the number of responses for the response class. If there are twelve or fewer choices, it will use the single-column frame. Instead of using a single-answer radio box, either twelve or twenty -four check box components are used to create a single object in the frame. Because dynamic allocation is not available, the object is defined as having the maximum number of individual components for each of the multiple-response frames. Each check box component is disabled and invisible at frame initialization.

Both the QDS and RDS are opened within the multiple-response frames. The QDS provides the question text, response category, DCT variable name (which serves as the prefix of the DCT variable names in these frames,) and the index for any exclusive answer. An exclusive answer is an answer that locks out all other choices. For example, “None of the above” is an exclusive answer. Survey flow in the multiple-response frames is restricted to forward and back, and that is handled in the main control frame via a RETURN value.

The multiple response frame starts by receiving two parameters, question number, and subject ID when it is invoked with DISPLAY(). Question number is needed to use the QDS metadata, and subject ID is used for writing the data to the data capture table. The first thing that happens is that all of the check box components are assembled into a single object. This makes array-style referencing of the individual checkbox objects possible. Next, we get the information we need from the QDS by opening it with a WHERE clause and fetching the record. The QDS is closed, and the **label** property of the question-text text box object is set so that the survey question is displayed.

The next step is to get the information from the RDS. As with the single response-frame, it is opened with a WHERE clause to subset the RDS and restrict it to the records for the current response class. A DO-WHILE loop reads each RDS record. When a record is read, the program fills a corresponding check box (using the response index) with the response text as the **label** property of the check box object, enables the check box, and makes it visible. This loop also assembles a character string representing the variable list used for the questions, appending the loop counter to the variable name prefix obtained from the QDS. This character string is used when the data are written to/read from the data capture table. This does require that the variable names in the data capture table for a multiple-response question follow a strict naming convention (*variable-name**1**...variable-name**N***), but it’s a relatively small constraint for the convenience it provides. The DO-WHILE loop executes until no more records are found in the subset RDS, leaving any unused check boxes empty, inactive, and invisible. The final value of the loop counter is stored as the maximum number of responses, and the RDS is closed, having done its job.

Now the data capture table gets into the act. It is opened with access to the subject's record (based on ID, received as a parameter), the variables defined by the response string created during the RDS loop, and the current question variable. If the values for the variables corresponding to this question are not missing (i.e., the question has already been answered), the frame has to fill the respective check boxes before accepting any respondent input. The data control table remains open after this task is finished, because we will need it to write the responses before leaving the frame.

The MAIN section has to test for the exclusive answer. If there is an exclusive answer to the current question, then each time a box is checked, it may affect the contents of the other boxes. In other words, if "none of the above" is selected, no other boxes should be checked and vice versa.

Writing the data to the data capture table is done by using CALL PUTVARN() in a DO loop that runs from one to the number of maximum responses. This is currently DHQ-specific, writing 0 or 1, but could easily be made more general by putting the values from the RDS into an array as the RDS is processed at the beginning of the frame. The code that writes the data to the DCT is called from both control pushbuttons. The forward button returns 1 to the calling frame, while the back button returns 0. Both terminate the multiple-response frame, and the return value allows the calling frame to know which direction to go in order to get to the next question. The multiple-choice frames do not allow for skip logic.

TEXT RESPONSE QUESTIONS

Text responses are questions that require the user to type in text; therefore, they have no records in the RDS, but the frame's behavior is still dictated by the QDS. The TEXTRESP frame receives question number and subject ID as parameters from the main frame.

The QDS is opened, and the record for the current question is read. The variable TYPE in the QDS defines whether the data to be entered is character or numeric, and the maximum number of characters is obtained from the response category variable. Text responses are indicated in the QDS by the letter "T", followed by a number that represents the maximum number of characters that can be entered for this response. The question text is displayed, and the control buttons are made visible if warranted (no next button for the final question, no previous button for the first question.)

There are two identical, overlaid text box objects in the frame. Which one is active depends on whether the variable to be saved from this frame is character or numeric. The data capture table is opened, and any format or informat for the variable is obtained from there, along with the value (if there is one). The contents of the active text box are written to the data capture table with CALL PUTVARN() when the user clicks on either the forward or backward button. As in the multiple-response frame, this frame is closed when the user clicks on one of the control buttons. It also returns a direction to the calling frame, and does not allow for skip logic.

GETTING OUT

This is the only thing that is hard-coded throughout the entire application. The last question number in the survey must be "fini". This will end the application, close any open datasets, delete any SCL lists, and invoke the scoring program. This also occurs whenever the user presses the "stop" button.

UGLY SCL

This application uses SCL, and it isn't very pretty SCL at that. There is even one GOTO statement in the single-response/control frame. However, it was built as a part of a much more comprehensive X-windows data management application that went into production in July of 2003, and there were time constraints with its development. With that in mind, here is the code for the single-response/control frame.

```
/************************************************************************/
/* DHQ_CODE.SAS - Code for Dietary Health Questionnaire application     */
/* V 1.2 - D. Morgan 10 November 2004                                   */
/* Data sets used: DHQ_QUESTIONS (read-only) - Question table (QDS)     */
/*                 RESPCLASS (read-only) - Response table (RDS)         */
/*                 DHQ (read/write) - Output table w/subject data (DHQ) */
/*                                                                      */
/* Single- and no-response-questions are handled in this code.          */
/* Multiple-response and special requirements questions are handled in  */
/* their own frames.                                                    */
/*                                                                      */
/************************************************************************/
ENTRY id $ 8;  /* Passed from START_DHQ */

INIT:
CONTROL LABEL;
```

```
/* Make sure that all the necessary variables from both metadata datasets are available
to the PDV with CALL SET */
LENGTH q next prev q_no rcat qds_qno skip_to qds_skip2 $ 32 q_text $ 305 whereclz
       dsn $ 250 vname qds_vname $ 32 skip_trigger qds_skiptrig 4 resp_text $ 200
       indx val chkval 3 type $ 1 use_common_text 3 lastfld $ 32;

dsn = "sysds.dhq_questions";
qds = OPEN(dsn,'i');
CALL SET(qds);
/* Make list from QDS for manager's direct access box */
qlist = MAKELIST();
DO WHILE(FETCH(qds) NE -1);
   rc = INSERTC(qlist,q_text,-1);
END;
qlistbox.items = qlist;

/* What question should I start with? */
dsn = "clinic.dhq (KEEP=id lastfld WHERE=(id EQ '" || id || "'))";
_chk = OPEN(dsn,'i');
CALL SET(_chk);
tst = FETCH(_chk);
rc = CLOSE(_chk);
IF tst EQ -1 OR lastfld IN (' ','FINI') THEN
   q = "strt"; /* initialize to intro */
ELSE
   q = lastfld; /* Go to saved question */
/* show manager's box? Toggle goes here - can be anything */
IF SYMGETN('slevel') LT 40 THEN DO;
   qlistbox.visible = "No";
   qlistbox.enabled = "No";
END;
prevbutton.visible = "No";
prevbutton.enabled = "No";
/* No RETURN; statement - just run through */

main:
IF q EQ 'fini' THEN DO; /* time to go home */
   nextbutton.visible = "No";
   nextbutton.enabled = "No";
   dsn = "clinic.dhq (KEEP=id lastfld WHERE=(id EQ '" || id || "'))";
   dhq = OPEN(dsn,'u');
   rc = FETCH(dhq);
   CALL PUTVARC(dhq,2,'FINI');  /* Store next question indicator in DCT */
   rc = CLOSE(dhq);
   CURSOR exit;
END;
ELSE DO; /* show and enable next question button */
   nextbutton.visible = "Yes";
   nextbutton.enabled = "Yes";
END;

IF q EQ 'strt' THEN DO; /* at the intro, can't go forward */
   prevbutton.visible = "No";
   prevbutton.enabled = "No";
END;
ELSE DO; /* show and enable prev question button */
   prevbutton.visible = "Yes";
   prevbutton.enabled = "Yes";
END;
```

```
whereclz = "q_no EQ '" || q || "'"; /* set current question number */
rc = WHERE(qds,whereclz);
rc = FETCH(qds);   /* Get current question record from QDS */
rc = WHERE(qds,"undo");  /* got everything needed from QDS - undo where clause */
/* What kind of response do I have? */
trigger = SUBSTR(rcat,1,1);
IF trigger NOT IN ("R"," ") THEN DO;  /* not single-response or no-response */
   IF trigger EQ "T" THEN
      direction = DISPLAY('prgm.dhq_pc.textresp.frame',q_no,id,type);
   ELSE DO;
      IF trigger EQ "M" THEN DO;
         dsn = "sysds.respclass (WHERE=(rcat EQ '" || rcat || "'))";
         _chk = OPEN(dsn,'i');
         rc = VARSTAT(_chk,'indx','max',_ct);
         rc = CLOSE(_chk);
         IF _ct GT 12 THEN
            str = "prgm.dhq_pc.multiresp.frame";
         ELSE
            str = "prgm.dhq_pc.multir12.frame";
      END;
      ELSE    /* Custom Frame */
         str = "prgm.dhq_pc." || TRIM(LEFT(rcat)) || ".frame";
      direction = DISPLAY(str,q_no,id);
   END;
   SELECT(direction);
     WHEN(1) q = next;
     WHEN(0) q = prev;
     OTHERWISE
        GOTO exit;
   END;
   GOTO main;  /* Do not wait for a response to the current frame! */
END;

/* single response questions start here */

/* Show 'Over the last 12 months' if not redundant or a header */
IF use_common_text THEN
   lastyr.visible = "Yes";
ELSE
   lastyr.visible = "No";

dhqtext2.label = q_text; /*display question text */
/* If Q has responses, get them from RDS */
IF rcat NE ' ' THEN DO;
   /* create SCL list from RDS based on response category for this Q (from QDS) */
   resplist = MAKELIST();
   dsn = "sysds.respclass (WHERE=(rcat EQ '" || rcat || "'))";
   respclassdata = OPEN(dsn,'i');
   CALL SET(respclassdata);
   DO WHILE (FETCH(respclassdata) NE -1);
      rc = INSERTC(resplist,resp_text,-1);
   END;
   /* add skip option */
   rc = INSERTC(resplist,"Skip this question",-1);
   response.items = resplist; /* initialize radio box with response list */
   /* Get existing response for current variable (VNAME) if any */

   dsn = "clinic.dhq (KEEP=id " || vname || " WHERE=(id EQ '" || id || "'))";
   dhq = OPEN(dsn,'u');
   rc = FETCH(dhq);
   inval = GETVARN(dhq,2);
   rc = CLOSE(dhq);

   /* match response value with correct Index */
   SELECT(inval);
     WHEN (.M) response.selectedItem = "Skip this question";
     WHEN (.) response.selectedIndex = 0;
     OTHERWISE DO; /
   /* Augment existing WHERE clause with response value */
         if INVAL LE .z THEN
            whereclz = "val EQ ." || LEFT(PUTN(inval,'3.')); /* Adjust for special
missing values - WHERE clause will not work otherwise */
         ELSE
            whereclz = "val EQ " || LEFT(PUTN(inval,'3.'));
         rc = WHERE(respclassdata,whereclz);
         rc = FETCH(respclassdata,'NOSET');
         /* Get index for existing response */
         indx = GETVARN(respclassdata,VARNUM(respclassdata,'indx'));
         IF rc NE -1 THEN
            response.selectedIndex = indx;
         rc = WHERE(respclassdata,'undo');
      END;
   END;
   rc = CLOSE(respclassdata);
   /* make sure radio box is visible and active */
   response.visible = "Yes";
   response.enabled = "Yes";
END;
ELSE DO; /* rcat = ' ' , no response category for this Q, turn off response radio box
*/
   response.visible = "No";
   response.enabled = "No";
END;
RETURN;
```

```
response:  /* User clicks a response */
/* Is this a modified response? If so, Idiot prompt */
IF indx NE response.selectedIndex AND inval NE . THEN DO;
   confirm = DISPLAY('prgm.dhq_pc.idiotbox.frame',"Change This Answer?"," ");
   IF NOT confirm THEN  /* Not OK to change answer, get another response */
      RETURN;
END;
/* Special missing value for respondent skips */
IF response.selectedItem EQ "Skip this question" THEN
   val = .M;
ELSE DO; /* get correct value for selected response - subset RDS on response category
and index */
   dsn = "sysds.respclass (WHERE=(rcat EQ '" || rcat || "'))";
   respclassdata = OPEN(dsn,'i');
   CALL SET(respclassdata);
   whereclz = "indx EQ " || LEFT(PUT(response.selectedIndex,2.));
   rc = WHERE(respclassdata,whereclz);
   rc = FETCH(respclassdata,"NOSET");
   val = GETVARN(respclassdata,3);
   rc = WHERE(respclassdata,"undo");
   rc = CLOSE(respclassdata);
END;

/* write value to DHQ data set, only open w/3 variables, ID (for where clause), var
associated w/current Q, and next question to be answered */

dsn = "clinic.dhq (KEEP=id " || vname || " lastfld WHERE=(id EQ '" || id || "'))";
dhq = OPEN(dsn,'u');
rc = FETCH(dhq);
CALL PUTVARN(dhq,2,val);
/* determine next question to be answered and store in DCT*/
IF skip_to NE ' ' AND val EQ skip_trigger THEN
   CALL PUTVARC(dhq,VARNUM(dhq,'lastfld'),skip_to);
ELSE
   CALL PUTVARC(dhq,VARNUM(dhq,'lastfld'),next);
rc = UPDATE(dhq);
rc = CLOSE(dhq);
CALL SYMPUT('termed','Y');

/* Clear skipped variables */
IF skip_to NE ' ' AND val EQ skip_trigger THEN DO;  /* Skip pattern in effect? */
   clean = OPEN('sysds.dhq_questions');  /* Survey Control Information */
   cln_next = next;    /* Get next question in sequence */

/* Cleaning loop for skipped questions */
/*  Do this when not at next question to be answered  (next question in sequence ≠ skip
destination */
   DO WHILE(cln_next ne skip_to);
      where2 = "q_no eq '" || cln_next || "'";
      rc = WHERE(clean,where2);
      rc = FETCH(clean);
/* Get control variables for next question in sequence */
      cln_next = GETVARC(clean,VARNUM(clean,'next')); /* set next question in sequence
*/
      vnx = GETVARC(clean,VARNUM(clean,'vname')); /* variable name for next question */
      rsc_next = GETVARC(clean,VARNUM(clean,'rcat'));  /* Need response category for
multiple-choice questions */
      dsn = "clinic.dhq (KEEP=id " || vnx || " WHERE=(id EQ '" || id || "'))";
      dhq2 = OPEN(dsn,'u');
```

```
/* test for single/multiple response question */
/* If variable name does not exist in DCT, then multiple or no choice question.
      IF VARNUM(dhq2,vnx) EQ 0 THEN DO;  /* multiple- or no-response */
         rc = CLOSE(dhq2);
/* test for no-response question */
         IF vnx NE ' ' THEN DO;  /* Do this only if multiple-response */
            dsn = "sysds.respclass (KEEP=rcat resp_text indx WHERE=(rcat EQ '" ||
rsc_next || "'))";
            rds_next = OPEN(dsn,'i');
            rc = VARSTAT(rds_next,'indx','max',n_choice);  /* get number of variables
(possible responses) for question */
            rc = CLOSE(rds_next);
/*  vnx is the root of the variable name (e.g., q38a) for multiple choice questions */
            DO i = 1 TO n_choice;  /* create variable names by appending numeric suffix
*/
              IF SUBSTR(vnx,LENGTH(vnx),1) IN ('0','1','2','3','4','5','6','7','8','9')
THEN
                 str2 = TRIM(str2) || " " || TRIM(vnx) || '_' || LEFT(PUTN(i,'2.'));
              ELSE
                  str2 = TRIM(str2) || " " || TRIM(vnx) || LEFT(PUTN(i,'2.'));
            END;
/* open DCT with ID and multiple-choice variables */
            dsn = "clinic.dhq (KEEP=id " || str2 || " WHERE=(id EQ '" || id || "'))";
            dhq2 = OPEN(dsn,'u');
            rc = FETCH(dhq2);
            DO i = 1 TO n_choice;  /* Write missing values to multiple-choice variables
*/
               if VARTYPE(dhq2,i+1) NE 'C' THEN
                  CALL PUTVARN(dhq2,i+1,.);
               ELSE
                  CALL PUTVARC(dhq2,i+1,' ');
            END;
         END;
      END;
      ELSE DO;  /* Single/text response question, write missing value */
         rc = FETCH(dhq2);
         if VARTYPE(dhq2,2) NE 'C' THEN
            CALL PUTVARN(dhq2,2,.);
         ELSE
            CALL PUTVARC(dhq2,2,' ');
     END;
     rc = UPDATE(dhq2);  /* Update DCT */
     rc = CLOSE(dhq2);
     rc = WHERE(clean);  /* Clear where clause, prepare to test next variable */
   END;
   rc = CLOSE(clean);  /* All skipped data is now blank, close control dataset */
END;
 /* turn off response box - don't want any accidents */
response.visible = "No";

/* next question ? */
IF skip_to NE ' ' AND val EQ skip_trigger THEN
   q = skip_to;
ELSE
   q = next;
RETURN;
/*  that's it for controlling the survey, asking questions, and getting answers */
/* Process user interrupts and termination */
nextbutton:
IF next NE " " THEN DO; /* what's next? Ignore it if we're at the end */
   IF skip_to NE ' ' AND inval EQ skip_trigger THEN
      q = skip_to;
   ELSE
      q = next;
END;
RETURN;

prevbutton:
IF prev NE " " THEN DO; /* Go back unless at front */
   /* can I get to where I am (q_no=current Q) from a skip pattern? */
   dsn = "sysds.dhq_questions (WHERE=(skip_to EQ '" || q_no || "'))";
   aux1 = OPEN(dsn,'i');
   skipped = FETCH(aux1);
   IF skipped NE -1 THEN DO;    /* Yes, I can get here from a skip */
      /* Get all skip-related info for "skip-from" Q */
      qds_qno = GETVARC(aux1,2);  /* question number (potential "skip-from") */
      qds_skiptrig = GETVARN(aux1,6); /* skip trigger */
      qds_vname = GETVARC(aux1,9);  /* what variable should I check? */
      rc = CLOSE(aux1);

      /* did I get here from a skip pattern? Test DCT for skip trigger */
      dsn = "clinic.dhq (KEEP=id " || qds_vname || " WHERE=(id EQ '" || id || "'))";
      qck = OPEN(dsn,'i');
      rc = FETCH(qck);
      chkval = GETVARN(qck,2); /* get value from DHQ data */
      rc = CLOSE(qck);
      IF chkval EQ qds_skiptrig THEN /* you got here via skip logic */
         q = qds_qno;   /* Go to skip-from Q */
      ELSE
         q = prev;  /* Didn't get here by skip logic, go back 1 Q */
      RETURN;
   END;
   ELSE DO;  /* no skip involved - just back up 1 question */
      rc = CLOSE(aux1);
      q = prev;
   END;
END;
RETURN;
qlistbox: /* process manager's list box */
dsn = "sysds.dhq_questions";
aux1 = OPEN(dsn,'i');
here = FETCHOBS(aux1,qlistbox.selectedIndex);
IF here NE -1 THEN
   q = GETVARC(aux1,2); /* get q number of selected QDS record for program control */
rc = CLOSE(aux1);
RETURN;

TERM:
IF q NE 'fini' THEN
   _STATUS_ = "R";  /* keeps from terminating unless done */
RETURN;

exit:  /* close nicely  */
rc = CLOSE(qds); /* close QDS */
CALL DISPLAY("prgm.dhq_pc.dhq_filegen.frame",id); /* scoring */
_STATUS_ = "H";  /* Terminate application */
RETURN;
```

MULTIPLE-RESPONSE SCL

```
/* Get current question, DCT record identifier, set up return value */
ENTRY q_no $ 32 id $ 8 RETURN=NUM; /* Passed from main control module (DHQ) */

DCL OBJECT c{24};  /* create object for check boxes */

INIT:
CONTROL LABEL ALWAYS;
/* Make sure that all the necessary variables from both metadata datasets are available
to the PDV with CALL SET */
LENGTH q_text dsn str $ 300 resp_text $ 200 vname rcat $ 32 indx exc 3;
/* assign check boxes to object */
c[1] = chk1;
c[2] = chk2;
c[3] = chk3;
c[4] = chk4;
c[5] = chk5;
c[6] = chk6;
c[7] = chk7;
c[8] = chk8;
c[9] = chk9;
c[10] = chk10;
c[11] = chk11;
c[12] = chk12;
c[13] = chk13;
c[14] = chk14;
c[15] = chk15;
c[16] = chk16;
c[17] = chk17;
c[18] = chk18;
c[19] = chk19;
c[20] = chk20;
c[21] = chk21;
c[22] = chk22;
c[23] = chk23;
c[24] = chk24;

/* Open QDS for question #, text, DCT variable prefix, response category and exclusive
answer */
dsn = "sysds.dhq_questions (KEEP=q_no q_text vname rcat exc WHERE=(q_no EQ '" ||
       q_no || "'))";
qds = OPEN(dsn,'i');
CALL SET(qds);
rc = FETCH(qds);
dhqtext.label = q_text;
get_rcat = rcat;  /* RCAT is in the PDV and the DDV, and so it can be modified, need a
temporary RCAT for WHERE clause */
rc = CLOSE(qds);  /* Got everything we need from there */
```

```
/* Open response class dataset, initialize and enable check boxes and make them
visible*/
/* Create DCT variable list using value of VNAME as prefix */
dsn = "sysds.respclass (KEEP=rcat resp_text indx WHERE=(rcat EQ '" || get_rcat ||
"'))";
rds = OPEN(dsn,'i');
rc = VARSTAT(rds,'indx','max',n_choice); /* Get number of items in category */
CALL SET(rds);
i = 0;
DO WHILE(FETCH(rds) NE -1);
   i + 1;
   c[i].label = resp_text;
   c[i].visible = "Yes";
   c[i].enabled = "Yes";
/* Make variable names for each item - can be accessed later with a dash */
/* If prefix ends in a numeral, add underscore before appending index value */
   IF SUBSTR(vname,LENGTH(vname),1) IN ('0','1','2','3','4','5','6','7','8','9') THEN
      str = TRIM(str) || " " || TRIM(vname) || '_' || LEFT(PUTC(i,'2.'));
   ELSE
      str = TRIM(str) || " " || TRIM(vname) || LEFT(PUTC(i,'2.'));
END;
rc = CLOSE(rds);

/* Get existing responses (if any) - fill in checkboxes */
dsn = "clinic.dhq (KEEP=id " || str || " WHERE=(id EQ '" || id || "'))";
dhq = OPEN(dsn,'u');
CALL SET(dhq);
rc = FETCH(dhq);
DO i= 1 TO n_choice;
   IF GETVARN(dhq,i+1) = 1 THEN
      c[i].selected = "Yes";
END;
RETURN -1;

MAIN: /* Check for exclusive answer - must be in MAIN to execute every time a box is
checked */
DCL CHAR last;
IF exc GT 0 THEN DO;  /* Is there an exclusive answer for this question? (EXC=index of
exclusive answer obtained from QDS) */
   _frame_._getCurrentName(last);  /* Get name of selected checkbox (LAST)*/
   IF last EQ c[exc].name THEN DO;  /* Is selected checkbox the same as the exclusive
checkbox? */
/* Then de-select any other check boxes */
      DO i=1 TO n_choice;
         IF i NE exc THEN
            c[i].selected = "No";
      END;
   END;
   ELSE  /* If selected check box not exclusive answer, de-select exclusive answer */
     c[exc].selected = "No";
END;
RETURN -1;

TERM:
_STATUS_ = "R";
RETURN -1;
```

```
/* Write data to DCT */
writeout:
DO i = 1 TO n_choice;
   IF c[i].selected EQ "Yes" THEN
      CALL PUTVARN(dhq,i+1,1); /* hardcoded here, but can substitute value from RDS */
   ELSE
      CALL PUTVARN(dhq,i+1,0); /* Only if "Not checked" needs to have a value in DCT */
END;
rc = UPDATE(dhq);
rc = CLOSE(dhq);
RETURN -1;

/* User finished with frame, clicked "proceed" */
exit:
LINK writeout;
_STATUS_ = "H";
RETURN 1;

/* User finished with frame, clicked "go back" */
prev:
LINK writeout;
_STATUS_ = "H";
RETURN 0;
```

HOW LONG DID THIS TAKE TO DEVELOP?

Even with the fact that the method and prototype were built from scratch, development time was at the most equivalent to the amount of time that would have been spent doing it in a traditional SAS/FSP manner. This is true despite there being a wealth of existing SAS/FSP prototypes upon which the survey instrument could have been built. However, these prototypes are built for specific forms. They can be recycled, not reused (unless the form itself is being reused.) Considering the length of the DHQ, it's likely that the SAS/FSP method would have taken much longer. It definitely would have mandated an enormous increase in the amount of typing. The metadata tables were created by using base SAS to read ASCII files that were created by cut-and-pasting question and response text from the original DHQ Word document.

The design has already changed from its original state, which had a separate, custom frame for each multiple-response question. The single-column multiple-response frame was added by modifying the largest multiple-response frame. This was accomplished in less than two minutes (!), and the changes necessary in the control frame took less than ten, so this code is somewhat reusable.

Timing issues (when the metadata is opened, how long it stays open) have yet to be evaluated, and code clean-up (removing unused/unnecessary statements) still needs to be done. One of the improvements planned is to remove the single-response question handling from the control frame and put it in its own frame. This would leave just a controlling SCL entry that is responsible for initialization, handing response control to the appropriate response frame, and termination.

WHERE WE STAND NOW

The current version of this application as it is being used at the Washington University School of Medicine's Division of Biostatistics holds promise. It is currently accessed from our remote field centers with computers that have X-Windows software installed. They connect to a LINUX system located at the Division of Biostatistics through a VPN. This LINUX system has the SAS System installed, and they execute the DHQ on-line without paper forms, in real time as a part of a larger SAS-based data management application. Since the data are written as each question is answered, if the connection is lost, the only data that would be lost are the data from a question that was just answered, and there's only a very small window where that can happen. When the application is restarted for that person, it picks up where it left off as well, so there's no need to scroll through the questionnaire searching for the last question that was answered. There have been no response time or user issues, so we are confident enough to use the concept of remote data management with X-windows at the remote end, and a VPN connection to a local server for our next large-scale project. We are also planning to expand the use of the metadata method for data entry.

A generalized version of the DHQ application is available free of charge by sending an e-mail to the author. This version requires Base SAS for Windows, and is designed for the express purpose of administering the DHQ, as either a computer-assisted interview or an interviewer-adminstered computerized questionnaire. This is currently the only generally-available computerized method for capturing data from the DHQ. Instructions on how to modify the question and response datasets to customize the DHQ for the user's specific needs are provided. SAS/AF is only necessary if the user wants to create any custom frames.

There is also another Windows application available on request. The theoretical version of the application is designed for use in survey situation, and it has the added capability to randomize the display of answer choices for any given question. This will reduce any ordering effect within the survey. It is effected by adding a variable to the question dataset, which has a value of 0 or 1 for each question. The data-driven nature of the method is unaffected. In the DHQ, we subset the response dataset with a WHERE clause to get our responses for a question. In the survey application, a temporary dataset is created from that subset. It is then modified to contain a temporary index value and a random number. This dataset is sorted by that random number, and the temporary index value is derived from the sorted order of the answers. This version of the program uses that temporary index value, where the DHQ uses the absolute index value (INDX) in the RDS. The absolute index value is still used during the clearing of responses to skipped questions, so the RDS structure is the same.

Answer Randomization Code

```
   SUBMIT CONTINUE SQL;
       CREATE TABLE RESPCLASS AS
       SELECT * FROM sysds.respclass
       WHERE rcat EQ "&rcat";  /* SCL variables can be passed to the SAS System PREVIEW
buffer as macro variables that are resolved at run-time */

       ALTER TABLE RESPCLASS
        ADD USEINDX NUM
        ADD RAND NUM;

        UPDATE RESPCLASS
        SET RAND = RANUNI(0);
    ENDSUBMIT;

   dsn = "respclass";   /* Open temporary dataset, not sysds.respclass with a WHERE
clause */
   respclassdata = OPEN(dsn,'u');
   IF rndize THEN
      rc = SORT(respclassdata,"RAND");  /* Sort according to random number */
   CALL SET(respclassdata);
   useindx = 0;  /* Initialize temporary index variable */
/* create response list */
   DO WHILE (FETCH(respclassdata) NE -1);
      useindx = useindx + 1;   /* Index value used by frame – replaces INDX in DHQ
version except for clearing skipped variables */
      rc = INSERTC(resplist,resp_text,-1);
      rc = UPDATE(respclassdata);
   END;
```

WHAT'S NEXT?

Applications tools are currently being developed for the creation and administration of question and response data warehouses. The individual records in these warehouses will ultimately be manipulated by other SAS-based tools to create individual question and response metadata tables for surveys and data forms, reducing or eliminating the need for our beloved SAS/FSP method, and its related disadvantages.

Being frame-based, this application would be a natural for object-oriented programming. Unfortunately, I am not an OO programmer, and as is usual for production applications, I did not have the luxury of time to re-work the code to fit strict OO convention. If this application has no significant user and/or resource issues, then conversion to object-oriented standards should make it possible to create a web-based, metadata-driven data entry system with the SAS System. This has already been done at the Siteman Cancer Center at the Washington University School of Medicine in conjunction with the SAS Institute's Public Sector Group[2], so in some sense, this is reinventing the wheel. That system is more comprehensive, serving as a portal for total data management of many studies, whereas this is only designed to administer the data entry portion of a single study.

We are experimenting with porting this data-driven methodology to the WWW using htmSQL and SAS/IntrNet. htmSQL, and its dynamic, query-driven nature would seem to be a very good fit for a data-driven survey method, especially if the number of users at any given time is low. This work is only experimental since data capture/entry is only a part of our data management systems, and the development and web implementation of some of the other modules may adversely affect our ability to meet the deadline date for overall system rollout. In the meantime, development of the code continues to improve the method, allowing it to fit additional data entry scenarios we have encountered. This has already led to some enhancements in the QDS structure.

SUMMARY

The National Cancer Institute's Diet History Questionnaire is a long survey instrument that does not translate well into an application using SAS/FSP. The flexible nature of the instrument, as well as its length, make coding it as an FSP application impractical, expensive to develop in terms of man-hours, and difficult to maintain. A method and prototype have been developed using SAS/AF that uses metadata in the form of question and response datasets. This has significantly reduced development time while providing the capacity for true application re-use in the future. The prototype application can be used as a computer-assisted interview, or by trained interviewers in either a data entry or telephone/live interview environment, with a direct navigation feature that can be turned on or off according to the situation. By changing the metadata, the same application can be used to administer different questionnaires.

The data-driven nature of this method holds promise for rapid survey deployment using question and answer data warehouses, and tools for the manipulation of these warehouses. Surveys will be developed by creating question and answer datasets from their parent warehouses, instead of having to be constructed from scratch with SAS/FSP, or SAS/AF PROGRAM or FRAME entries. The first production prototype of this application has been described in detail here, and although it has performed well, there are several rough edges. The application is written entirely in SCL, and does not comply with object-oriented standards. Development of the method continues to enable it to fit a broader spectrum of data entry and survey scenarios.

REFERENCES:

Morgan D, Province M. “Building A Better Data Entry Application Using PROC FSEDIT,” *Proceedings of the Twenty-Fourth Annual SAS® Users Group International Conference*, SAS Institute, 1999, 365-374

Miller JP, Schauer, J, Baty JD, Littlewood S, Trinkaus K, Lill RM, Richards R. “Managing Clinical Trials with a SAS-Based Web Portal,” *Proceedings of the Twenty-Eighth Annual SAS® Users Group International Conference.* April 2003 <http://www2.sas.com/proceedings/sugi28/181-28.pdf>

ACKNOWLEDGEMENTS

This work has been funded by NIH grant UY1 HL72524-01.

CONTACT INFORMATION:

Derek Morgan
Division of Biostatistics
Washington University Medical School
Box 8067, 660 S. Euclid Ave.
St. Louis, MO 63110
Phone: (314) 362-3685 FAX: (314) 362-2693
E-mail: derek@wustl.edu

This paper, the prototype Windows SAS System application, and other SAS System examples and papers can be found on the World Wide Web at:

http://www.biostat.wustl.edu/~derek/sasindex.html

SIMPLIFIED FLOWCHART FOR MAIN IN DHQ.FRAME