

A SAS® Approach to WEB-Based Surveys

Bernard R. Poisson, Statprobe Technologies

Abstract

The United State Air Force had a problem. Its existing methods of collecting information from its people were time-consuming, error-prone, and costly. Not any more! The Survey Engine, based on SAS®/IntrNet solved those problems once and for all! The Survey Engine provides the ability to host any number of Internet/Intranet surveys concurrently without in-depth computer or Internet knowledge or experience. All questions and response-sets are stored in SAS® data sets, independent from any display characteristics, giving them greater flexibility and reusability. This application provides a powerful, efficient, and cost-effective approach to gathering data from large groups of individuals.

Introduction

The U.S. Air Force Surveys Branch performs more than 3 dozen large-scale surveys on an annual basis, polling members to obtain their impressions on a wide variety of topics such as adequacy of compensation (pay, living conditions, etc), job satisfaction (having proper equipment, having adequate spare parts, adequate and safe working conditions), and others. Two principal goals of the data gathering efforts focus on determining what AF members believe senior leadership must do to make the USAF desirable enough to retain existing members and to attract high-school and college graduates as new members. The global nature of the USAF makes sampling of the total force a very difficult task.

In the not so distant past, surveys were administered using question booklets and scan-sheets. With the advent of increased software capabilities over the World Wide Web (WWW), the survey analysts assigned to the AF Surveys Branch conducted periodic reviews over the WWW of available survey applications on a routine basis in search of a better way of fulfilling their mission. Each application was thoroughly reviewed to ascertain if it was capable of coping with the rigorous demands placed upon it by professional survey construction and administration requirements. Although some applications possessed certain capabilities that made them an interesting possibility, they were also found to contain critical flaws that would render them practically unusable in the world of professional survey construction and administration. The primary capability lacking in most of these applications was inability to control program and question flow based on the respondent's responses. After several years of searching, the survey analysts and senior leadership concluded the most plausible solution was to obtain a military programmer/analyst with sufficient experience and knowledge to design and create the required application for them.

The resulting SAS®/IntrNet based survey application, the Survey Engine, written in SCL, has replaced all previous manual methods of data collection. Using this application, the operational efficiency of the AF Surveys Branch has increased by over 55 percent through decreased fielding and turn-around times. Operational costs of survey administration have been reduced by over 75%. More importantly, the survey experience leaves the respondent with a high degree of satisfaction through reduced frustration and time.

This paper will explain how this application came about and demonstrate how its use can help your organization, regardless of size, diversity, and geographic location.

Determining Program Requirements

In order to establish an organized idea of what the survey application was required to do, a list of program requirements was devised. These requirements came about through a series of many short informal meetings held between the two senior survey analysts and the programmer. The survey analysts brought 40+ years of survey experience and the programmer 18+ years of systems and programming knowledge to the table. Together, they established the requirements listed in Figure 1.

Questions must be reusable
Responses must be reusable
Responses must be independent of display characteristics
Additional Instructions capability needed
Response Format Options (Output)
Radio-Buttons (mutually exclusive)
Check-Boxes (mark all that apply)
Drop-Down List (single selection)
Drop-Down List (multiple selection)
Text-Box (single numeric value entry)
Free-Form Text field
Radio-Buttons with Large Response characteristics
Check-Boxes with Large Response characteristics
Check-Boxes with Large Response characteristics with branching
Multiple Drop-Down 'Rating' Lists
Multiple Drop-Down 'Rating' Lists with branching
Text-Box List (single numeric value entry for each)
Text-Box List (single numeric value entry for each with branching)
Responses must be validated to: disallow skipping of questions, ensure entry of values within allowable range, and to provide feedback to the respondent.
Respondent must be able to go back to previous questions, in case of errors or desire to change responses based on new questions
Respondent must be able to Stop & Resume a survey at a later time, continuing from the place where they left off.
Analysts must be able to 'track' respondents for demographic purposes and longitudinal/historical review
Must provide access control by restricting entry into surveys (limit entry to USAF Personnel, sample set, etc. only)
Prevent previous respondents from re-submitting multiple surveys
Follow-On Question and/or Skip-Logic required, based on respondent selection(s) to responses.
Must be able to administer multiple surveys concurrently
Must be coded so that future code maintenance is NOT required

Figure 1 – Program Requirements

Access Control & User Validation

An integral consideration of any survey, especially surveys administered over the WWW is that of access control. Sampling is used to derive a mathematically representative group of respondents, taking various strata into consideration. In order to maintain some sense of order, and to prevent the possible skewing of results, access to a survey must be restricted to only those individuals who fall within the sample group. This capability is provided through a small, independent module named CHKUSER.SCL, operating as a front-end to the Survey Engine. When a respondent attempts to enter a survey, they are greeted by the screen shown in Figure 2.

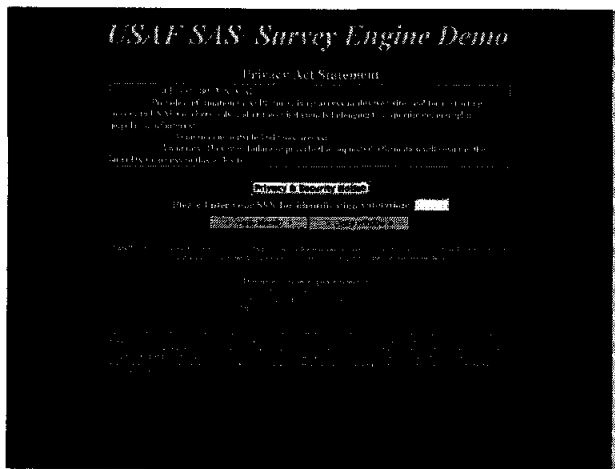


Figure 2 - Survey Login Screen

There are two distinct ways that a respondent may access a survey. These are:

- A URL to a given survey such as:
<http://surveys.af.mil/srvyon/login/demo.htm>
- A URL + PID (Personal ID) value such as:
<http://surveys.af.mil/srvyon/login/demo.htm?5349234234>

If the URL provided by the individual does NOT contain a PID value appended to the query-string, the individual must manually provide a valid Social Security Number (SSN) that is contained within the sample group of the survey. If the URL provided does contain a valid PID value appended to the query-string, JavaScript embedded within the login screen parses the value from the query-string and performs an 'auto submit' thereby effectively bypassing the individual's need to manually enter a valid SSN. In either instance, a look-up is performed to determine if a valid value was provided for authentication into the survey. If either of the values is not valid, the individual is redirected to a screen indicating an error condition. If the provided value is valid, the Survey Engine is invoked and survey administration begins.

Presenting Survey Information to the Respondent

The Survey Engine uses a standardized layout throughout the course of a survey to present the information to the respondent as shown in Figure 3.

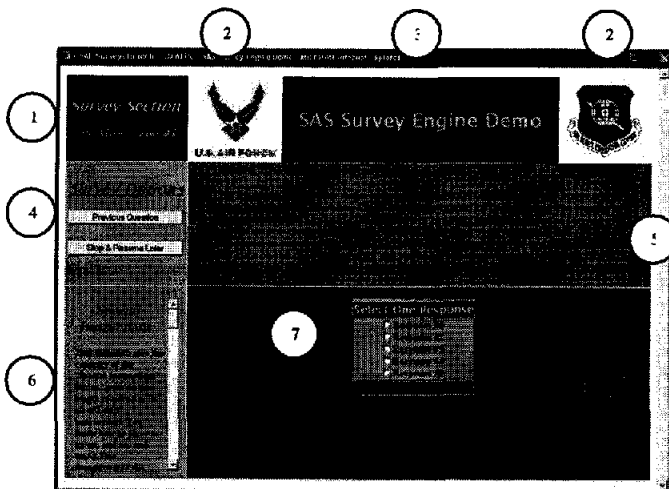


Figure 3 - Standardized presentation layout

Object 1: Section/sub-section designator. Allows surveys covering multiple topics to be broken up into multiple 'logical' sub-sections such as finances, job satisfaction, living conditions, career intent, etc.

Object 2: Graphic Banners: Allows customization of the output display.

Object 3: Survey Title: A data object containing the survey title.

Object 4: Survey Options: Control objects providing 'page back' and 'Stop and Resume' capabilities.

Object 5: Question Section: A data object containing the question to be asked.

Object 6: Special Instructions: A data object allowing the inclusion of additional information to be presented to the respondent to clarify a question, response, etc.

Object 7: Response Section: A data object containing the list of acceptable answers for the question presented.

All objects depicted above are static in that they do not change form or structure except for the Response Object (7). The response object can be dynamically represented in any one of 10 data-independent formats, depending on survey requirements.

The Response Object Formats: The response object formats are described in detail below:

The Radio-Button Response Format (Single Selection)

This response object uses radio-buttons for response selection. Radio buttons are mutually exclusive; therefore only one radio-button can be selected. The header of the response object instructs the respondent to 'Select One Response'. The application constructs as many columns, comprised of 10 radio-buttons + responses, as required to display the complete response list. The respondent need only click on the desired response and the application automatically advances to the next question. (Figure 4).

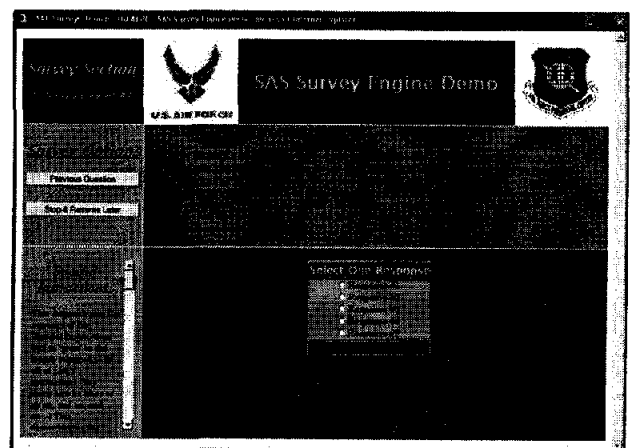


Figure 4 - Radio-Button Response Format

The Check-Box Response Format (Multiple Selections)

This response object uses check-boxes for 'Mark All That Apply' response selection(s). This implementation allows the selection of some or all responses, but not none. The header of the response object instructs the respondent to 'Check All That Apply'. The application constructs as many columns, comprised of 10 check-boxes + responses, as required to display the complete response list. Since more than one response can be selected, the respondent must click the 'Submit' button for the application to advance to the next question. (Figure 5).

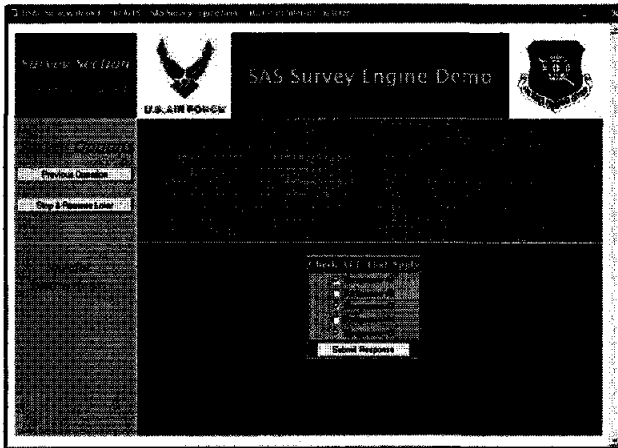


Figure 5 - The Check-Box Response Format

The Drop-Down List Response Format (Single Selection)

This response object uses a single Drop-Down List for response selection. This implementation of the Drop-Down List is mutually exclusive; therefore **ONLY ONE** item within the Drop-Down List can be selected. The header of the response object instructs the respondent to 'Select One Response'. The application constructs a **SINGLE** drop-down list containing as many selections/responses as required to display the complete response list. The respondent need only click on the desired response and the application automatically advances to the next question. (Figure 6)

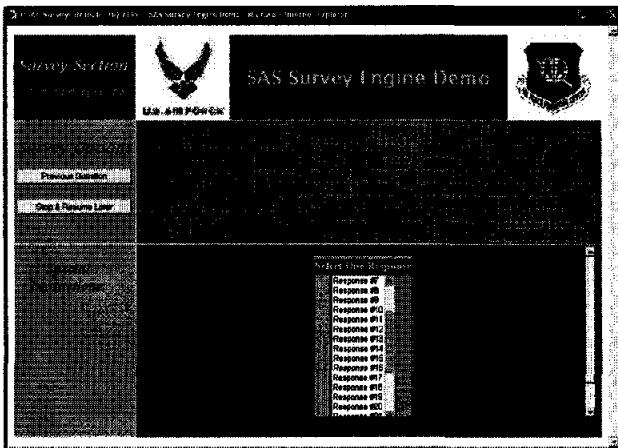


Figure 6 - The Drop-Down List Response Format

The Drop-Down List Response Format (Multiple Selections)

This response object uses a drop-down list for 'Mark All That Apply' response selection. This implementation of the Drop-Down List allows the selection of **some** or all responses, but **not none**. The header of the response object instructs the respondent to 'Select All That Apply'. The application constructs a **SINGLE** drop-down list containing as many selections/responses as required to display the complete response list. Since more than one response can be selected, the respondent must click the 'Submit' button for the application to advance to the next question. (Figure 7)

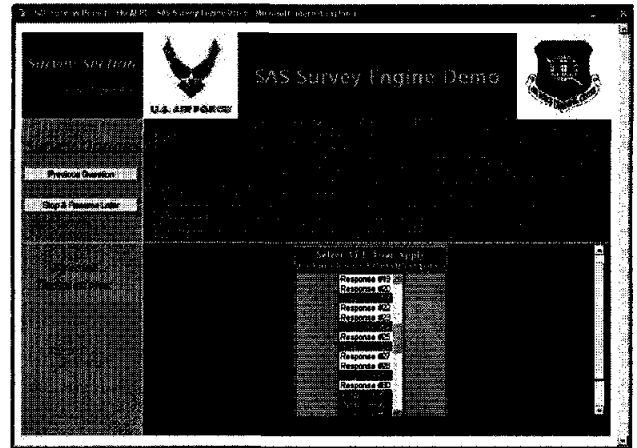


Figure 7 - The Drop-Down List Response Format

The Text Value Entry Response Format (Variable Length)

This response object uses a variable-length text-box for the entry of a numeric value entry such as 'Number of days TDY'. The header of the response object displays the range of acceptable values. The entry field is dynamically sized based on the number of characters required to enter the maximum value allowed for the particular response. Dynamically embedded JavaScript validates the value entered and prompts the user with an error/correction dialog if the value entered is not within the allowable range. The respondent must click the 'Submit' button for the application to advance to the next question. (Figure 8)

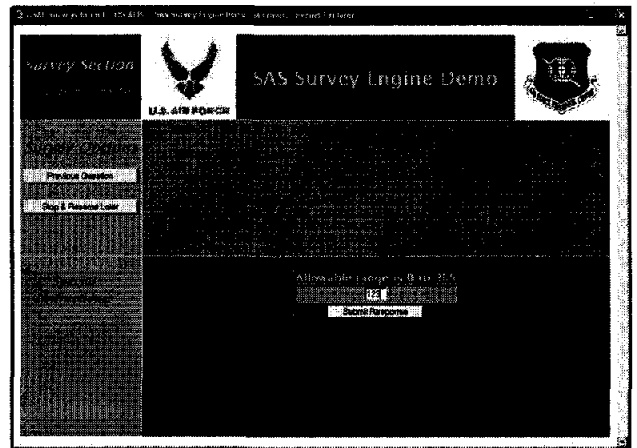


Figure 8 - The Text Value Entry Response Format

The Free-Form Text Entry Response This response object allows the entry of **FREE-FORM** text. Dynamically embedded JavaScript code monitors the respondent's progress and continuously displays '##### Characters Remain' in the response object header as the respondent types. The JavaScript code validates the number of characters typed and prompts the user with an error/correction dialog if the number of characters typed exceeds the allowable range. The respondent must click the 'Submit' button for the application to advance to the next question. (Figure 9)

Figure 9 - The Free-Form Text Entry Response Format

Figure 11 - The Check-Box Response Format

The Radio-Button Response Format (Single Selection + Large Response) This response object uses radio-buttons for response selection. Radio buttons are mutually exclusive; therefore **ONLY ONE** radio-button can be selected. The header of the response object instructs the respondent to 'Select One Response'. The application constructs a single column containing as many entries as required (with scroll controls) to display the complete response list. The respondent need only click on the response and the application automatically advances to the next question. (Figure 10)

Figure 10 - The Radio-Button Response Format

The Check-Box Response Format (Multiple Selection + Large Response) This response object uses check-boxes for response selection. This implementation allows the selection of **some or all** responses **but not none**. The header of the response object instructs the respondent to 'Check All That Apply'. The application constructs a single column containing as many entries as required (with scroll controls) to display the complete response list. Since more than one response can be selected, the respondent must click the 'Submit' button for the application to advance to the next question. (Figure 11)

The Check-Box Response Format (Multiple Selection + Large Response + Branching) This response object uses check-boxes for response selection and uses the respondent's selection(s) to control program flow through the survey. This implementation allows the selection of **SOME or ALL** responses but **not none**. The header of the response object instructs the respondent to 'Check All That Apply'. **Each response option** in the list is capable of branching to a **different/separate 'Follow-On'** question. Sequential program flow continues once all follow-on options have been completed. Since more than one response can be selected, the respondent must click the 'Submit' button for the application to advance to the next question. (Figure 12)

Figure 12 - The Check-Box Response Format

The Drop-Down 'Rating' list Response Format

This response object provides drop-down list containing predefined 'ratings/rankings' for the respondent to select from. Each response option 'Rated' by the respondent represents Selection+Rating. This implementation allows the selection of **some, all, or none** of the response objects. The header of the response object instructs the respondent to 'Rate Applicable Items'. The application constructs a single column of text items containing as many entries as required (with scroll controls) to display the complete response list. Since more than one response can be selected, the respondent must click the 'Submit' button for the application to advance to the next question. (Figure 13)

Figure 13 - The Drop-Down 'Rating' list Response Format

The Drop-Down 'Rating' list Response Format (+ Branching)
This response object provides drop-down list containing predefined 'ratings/rankings' for the respondent to select from. Each response option 'Rated' by the respondent represents Selection+Rating. This implementation allows the selection of some, all, or none of the response objects. The header of the response object instructs the respondent to 'Rate One or More response(s)'. The application constructs a SINGLE column of text items containing as many entries as required (with scroll controls) to display the complete response list. Each response option in the list is capable of branching to a different/separate 'Follow-On' question. Sequential program flow continues once all follow-on options have been completed. Since more than one response can be selected, the respondent must click the 'Submit' button for the application to advance to the next question. (Figure 14)

Figure 14 - The Drop-Down 'Rating' list

The Text Value Entry List Response Format (Variable Length)
This response object uses variable-length text-boxes for the entry of a list of numeric value entries such as 'Number of days TDY'. The header of the response object displays the range of acceptable values. The entry fields are dynamically sized based on the number of characters required to enter the maximum value allowed for each particular response. Dynamically embedded JavaScript code validates the entered values and prompts the user with an error/correction dialog if any of the values entered are not within the allowable value range. Since more than one response can be selected, the respondent must click the 'Submit' button for the application to advance to the next question. (Figure 15)

Figure 15 - The Text Value Entry List Response Format

The Text Value Entry List Response Format (+ Branching)
This response object uses variable-length text-boxes for the entry of a list of numeric value entries such as 'Number of days TDY'. The header of the response object displays the range of acceptable values. The entry fields are dynamically sized based on the number of characters required to enter the maximum value allowed for each particular response. Dynamically embedded JavaScript code validates the entered values and prompts the user with an error/correction dialog if any of the values entered are not within the allowable value range. EACH NUMERIC ENTRY in the list is capable of branching to a different/separate 'Follow-On' question. Sequential program flow continues once all follow-on options have been completed. Since more than one response can be selected, the respondent must click the 'Submit' button for the application to advance to the next question. (Figure 16)

Figure 16 - The Text Value Entry List Response Format

The Survey Completion/Termination Screen:
Upon completion of a survey, the respondent is presented with a completion/termination screen explaining what will be done with the data collected from them during the course of the survey. At this point the survey has been terminated and all data recorded. The respondent is provided a 'Close Window' button to facilitate closing of the survey browser interface. An example screen is shown in Figure 17.

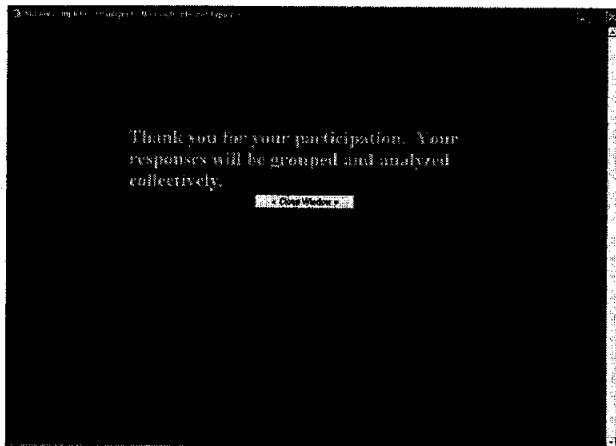


Figure 17 - The Survey Completion/Termination Screen

Behind The Screens – Making it all come together

Creating a survey engine capable of displaying any response using any one of several response formats came about from the fact that although the available topic matter of surveys may change and is unlimited, the ways in which responses can be successfully presented and formatted to create a workable survey is finite. Taking these factors into consideration, we created an adequate set of response object format styles to meet any future needs of the survey analyst. Taking advantage of the capabilities of today's web browsers, we can build powerful and truly dynamic surveys that can address any topic matter.

Data Sources

In order to have a truly dynamic application, one capable of literally changing with the needs of the organization, the application must obtain as much of its required operational information from the most dynamic of sources available to the computer. The most dynamic sources of data of course are data files. The survey engine acquires 99% of its operational data from four separate SAS® data sets. These are the QuestionsDB, ResponseDB, JumpTableDB, and AppCFG data sets. Their functions are as follows:

QuestionsDB: This local data set is where the survey engine obtains most the operational information required for a particular survey and is a subset of the Master Questions Database. The data obtained from this data set is comprised of the Question Text, Response Code, Response Format, Response Range, Skip-Logic Flag, Section Text, and Instruction Text. Using a combination of these elements, the application has the information it requires to dynamically generate a particular survey.

ResponseDB: This global data set is where the survey engine obtains ALL of the information required relating to ALL response-sets for ALL surveys. The data obtained from this data set is comprised of the Response Name and Individual Responses that comprise the response.

JumpTableDB: This local data set may or may not exist for a particular survey. If a survey does not require any response-dependent branching (skip-logic), then this file will not exist for the given survey.

AppCfg: This global data set is the 'master configuration file' used to prevent the need of maintenance on the survey engine's code. The data obtained from this data set is comprised various

operational parameters such as libnames for the Master QuestionsDB, ResponseDB and other data sets, WEB server name, path to HTML information screens, foreground/background colors to be used when generating the surveys, etc. The libname 'APPCFG' is the ONLY EXTERNAL item of information that the Survey Engine requires and is dependent upon for its successful operation. Armed with this information, the Survey Engine can obtain all other required operational parameters from the AppCfg data set.

JumpTable.slist: This is a dynamically created SAS® SCL list. If/When a survey contains a response object utilizing response-dependent branching; this list will be created on the fly and saved to disk in order to perform correct question sequencing. Branching represents a deviation from the normal sequential flow from one question to another. The existence of this list indicates to the survey engine that it must anticipate this deviation for this particular respondent. Additionally, the engine uses the contents of this file to repopulate the SCL list, which contains the question numbers represented in the deviation from normal flow through the next 'logical' question.

ResponseHistory.slist: This is also a dynamically created SAS® SCL list. The intent of this list is to create an audit-trail of each respondent's path through a survey. As the respondent proceeds through a survey, the number of each question referenced is appended to the existing list and the list saved to disk for future reference thereby creating a complete transaction history. Its intended purpose is to provide an EXACT reverse-path through questions visited by the respondent so that if/when the respondent selects the 'Previous Question' button one or more times, the Survey Engine will be able to correctly display the desired question along with the respondent's response.

The Survey Engine Module and its Routines

The Survey Engine's principal module is comprised of 13 separate CONTROL sections, and 16 FORMATTING sections each constructed as subroutines. Due to the reentrant nature of this application, most variables used are global and are not unique to a particular section of code. Each section of code was given a 'meaningful' name which clearly describes the functionality of the code contained within it.

The CONTROL sections are:

```
INIT
FIND_NEXT_QUESTION_NUMBER
GET_PREVIOUS_QUESTION
SAVE_RESPONSE
SAVE_HISTORY
SET_LOCKOUT
GET_PREVIOUS_RESPONSE
DISPLAY_NEXT_QUESTION
DISPLAY_MESSAGE
SET_LOCKOUT
WEBOPEN
WEBCLOSE
DEBUG
TERM
```

The formatting sections are:

```
FORMAT_RB
FORMAT_CM
FORMAT_DS
FORMAT_DM
FORMAT_TX
FORMAT_TF
FORMAT_RL
FORMAT_CL
```

```

FORMAT_DL
FORMAT_TL
ADD_CONTROL_PARAMETERS_FRAME
ADD_CONTROL_PARAMETERS_NOFRAME
IFRAME_RESPONSE_OPEN
IFRAME_RESPONSE_CLOSE
INNER_TABLE_OPEN
INNER_TABLE_CLOSE

```

Walking Through the Survey Process – The Operational Cycle

We will now step through the typical processing cycle of the Survey Engine and briefly describe the various processes involved within each of the CONTROL and FORMATTING sections encountered during the run. Please note that there are NO 'hard-coded' values incorporated into the Survey Engine code itself. Any required parameters and values are provided by the AppCfg data set and will be discussed in greater detail below. Pseudo-code will be used to simplify explanation and understanding of code functions and provide brevity.

Once validated into the system via the ChkUser module, a call is made to the Survey Engine's main processing module, POSTRESP.SCL. Respondent-dependent data values such as SURVEY_NAME, PID, PREVIOUS_QUESTION, NEXT_QUESTION and the like are passed to POSTRESP via an SCL list, PARAMS, as part of the program call.

INIT Section: This section is **unconditionally** executed and performs the following:

```

DEFINE required variable types and lengths
RETRIEVE data values from the PARAMS list and set like-named
variables
INITIALIZE control variable values
RETRIEVE control variable values from the AppCfg data set and
set like-named variables
EXECUTE 'LINK' calls to the other sub-routine sections contained
within POSTRESP

```

The sub-routines are called by the INIT section in the following order:

```

GET_PREVIOUS_QUESTION
FIND_NEXT_QUESTION_NUMBER
GET_PREVIOUS_RESPONSE
SAVE_HISTORY
SAVE_RESPONSE
DISPLAY_NEXT_QUESTION
RETURN

```

The remaining sub-routines within the POSTRESP.SCL module are called by these sections, as required, to satisfy the balance of control and function of the sections listed above.

GET_PREVIOUS_QUESTION Section: The function of this sub-routine is to replace/bypass the value provided by the FIND_NEXT_QUESTION section and enables the Survey Engine to logically 'reverse' its normal program-flow and revisit questions previously displayed to the respondent. This section is **conditionally** executed and is called **ONLY** when the PAGE BACK flag is set. This section is 'triggered' by the respondent's selection of the 'Previous Question' button on the survey screen and performs the following:

```

Create SESSION_HISTORY_LIST (SCL list)
IF SESSION_HISTORY_LIST.SLIST exists
    LOAD values from .SLIST entry identified by user's PID
    and SURVEY NAME
IF SESSION_HISTORY_LIST length > 0 THEN

```

```

SET NEXT_QUESTION to value of last item in
SESSION_HISTORY_LIST
DELETE last item from SESSION_HISTORY_LIST
SAVE updated SESSION_HISTORY_LIST.SLIST
ELSE
    SET NEXT_QUESTION to 1
RETURN

```

FIND_NEXT_QUESTION Section: This section determines the next question to be displayed to the respondent. This section is **conditionally** executed and is executed only when the PAGE BACK flag is not set. The determination is made using dynamically provided values and is the single most complicated section of code within the POSTRESP module. This section performs the following:

JT is a flag indicating that skip-logic will be used.

MATA is a flag indicating that the question will use 'mark all that apply' functionality.

RF is a flag indicating the output response format to be used in displaying the response object to the respondent.

```

IF RESPONSE_INDEX > 0 and JT='Y' and MATA='Y' THEN
    Open JUMPTABLE file
    Set WHERE clause to value of
    PREVIOUS_QUESTION
    Create JUMP_TABLE_LIST (SCL List)
    Insert PREVIOUS_QUESTION value into
    JUMP_TABLE_LIST
    LOOP
        IF Nth response is selected
            Insert nth value from JUMPTABLE data set
            based on numeric value of the respondent's
            response
    END LOOP
    Insert PREVIOUS_QUESTION value +1 into
    JUMP_TABLE_LIST
    SAVE JUMP_TABLE_LIST.SLIST

```

```

IF RESPONSE_INDEX > 0 and JT='Y' and MATA='Y' AND
RF='DL' OR RF='TL' THEN
    Open JUMPTABLE file
    Set WHERE clause to value of Nth sub-item of
    PREVIOUS_QUESTION
    Create JUMP_TABLE_LIST (SCL List)
    Insert PREVIOUS_QUESTION value into
    JUMP_TABLE_LIST
    LOOP
        IF Nth response of Nth sub-item is selected
            Insert nth value from JUMPTABLE data set
            based on numeric value of the respondent's
            response
    END LOOP
    Insert PREVIOUS_QUESTION value +1 into
    JUMP_TABLE_LIST

```

```

IF JUMP_TABLE_LIST.SLIST EXISTS THEN
    Create JUMP_TABLE_LIST (SCL List)
    Load JUMP_TABLE_LIST with values contained in
    JUMP_TABLE_LIST.SLIST

    JUMP_TABLE_LIST_LENGTH = length of
    JUMP_TABLE_LIST
    FIRST=value of 1st entry in JUMP_TABLE_LIST
    LAST=value of last entry in JUMP_TABLE_LIST

```

INTERNET, INTRANET AND THE WEB

Look for PREVIOUS_QUESTION in
JUMP_TABLE_LIST

If position returned <= JUMP_TABLE_LIST_LENGTH -
1 then

 NEXT_QUESTION=value of position+1 in
 JUMP_TABLE_LIST

If position returned = 0 then
 NEXT_QUESTION=NEXT_QUESTION+1

If position returned <= JUMP_TABLE_LIST_LENGTH
then

 NEXT_QUESTION=value of position in
 JUMP_TABLE_LIST +1 (normal resume
 point)

IF RESPONSE_INDEX > 0 and JT='Y' and MATA='N' THEN
 Open JUMPTABLE file and set WHERE clause to value
 of PREVIOUS_QUESTION
 Set NEXT_QUESTION to Nth value from JUMPTABLE
 data set based on numeric value of the respondent's
 response
ELSE
 NEXT_QUESTION=PREVIOUS_QUESTION+1

RETURN

GET_PREVIOUS_RESPONSE Section: This section retrieves
the respondent's response(s) to the current question from the
RESPONSE data set. The purpose of this section is to enable
the pre-select/re-selection of the respondent's responses
(remember) AFTER they have selected the 'PREVIOUS
QUESTION' button and revisited one or more questions. This
section performs the following:

Open RESPONSE data set and set WHERE clause to
respondent's PID value
Retrieve respondent's previous response from RESPONSE data
set
If response value = 0 then
 PREVIOUS_RESPONSE=""
RETURN

SAVE_HISTORY Section: This section tracks a respondent's
progress through a survey. The purpose of this section is to
enable the respondent to navigate BACKWARDS through
question(s) they have already responded to. This is
accomplished by incrementally modifying an SCL list and saving
the list to a .SCLIST entry for future reference. This section
performs the following:

IF NEXT_QUESTION > 1 THEN
 Create SESSION_HISTORY_LIST (SCL List)
IF SESSION_HISTORY_LIST.SLIST exists
 Load SESSION_HISTORY_LIST with values contained
 in SESSION_HISTORY_LIST.SLIST
IF SESSION_HISTORY_LIST length > 0 THEN
 LAST_HISTORY_LIST_ITEM=value of last entry in
 SESSION_HISTORY_LIST
IF PAGE_BACK flag not set and PREVIOUS_QUESTION <>
LAST_HISTORY_LIST_ITEM THEN
 Insert value of PREVIOUS_QUESTION into last
 position of SESSION_HISTORY_LIST
 Save SESSION_HISTORY_LIST to
 SESSION_HISTORY_LIST.SLIST
RETURN

SAVE_RESPONSE Section: This section records the
respondent's response(s) to the current question to the
RESPONSE data set. This section performs the following:

Open RESULTS data set
Set WHERE clause to respondent's observation
IF the question is a 'mark all that apply' type THEN
 Initialize MATA_RESPONSE variable to "
 LOOP
 Set index to maximum number of possible
 responses
 If Nth response is selected THEN
 Concatenate current response value to
 MATA_RESPONSE
 END LOOP
Save MATA_RESPONSE or RESPONSE to RESPONSE data set
Close RESULTS data set
RETURN

DISPLAY_NEXT_QUESTION Section: This section displays the
'next question' and dynamically generates and displays the
response object to the respondent. This section performs the
following:

QT contains the question-text obtained from the QUESTIONS
data set
JT is a flag indicating that skip-logic will be used.

MATA is a flag indicating that the question will use 'mark all that
apply' functionality.

RF contains the output response format to be used in displaying
the response object to the respondent.

RC contains the NAME of the RESPONSE SET to be formatted
and displayed to the user. Example: Agree007. Each response-
set indicates the number of entries within it. In the example
above, the response-set is an Agree/Disagree response-set and
contains 7 entries/values to be displayed to the user.

Open QUESTIONS data set
Set SURVEY_TITLE to value obtained from 1st observation
Set WHERE clause to NEXT_QUESTION
Load Question-Text
Close QUESTIONS data set

IF QT='END' THEN
 LOCKOUT='Y'
 Delete JUMP_TABLE_LIST.SLIST
 Link SET_LOCKOUT
 Display Thank you/Termination Message

Open RESPONSE data set
Load response-object entry/entries
Close RESPONSE data set

If RF is 'mark all that apply' type
 MATA = 'Y'
ELSE
 MATA = 'N'

IF 'PAGE_BACK flag set THEN
 Link GET_PREVIOUS_RESPONSE

CALL appropriate Response-Set formatting routine (an example
will be provided later)

Display static HTML containing global output display formatting
objects such as TABLES, FRAMES, and embedded JavaScript
for input validation

Output Question-Text
 Output Special-Instructions Text
 Output Response-Object (SCL List containing dynamically
 generated HTML/JavaScript)

RETURN

FORMAT_XX Sections: These 10 sections of code are responsible for dynamically generating the output response object (HTML and JavaScript) using the response-text obtained from the RESPONSE data set. The appropriate section is triggered by the value contained in 'RF' during the current execution cycle. We will discuss one of the simpler response-objects (Radio-Button) in order to avoid confusion. The compound-object types, dynamic objects comprised of one or more dynamically generated objects are too complex to illustrate within the constraints of this paper. That is unless you want a really, really BIG paper ☺. These sections generally perform the following:

```
Create RS SCL List (Container to hold dynamically generated
HTML/JavaScript)
Calculate # of columns in output display (N-times rows of 10
responses)
Link IFRAME_RESPONSE_OPEN (Create In-Line frame)
Generate dynamic header
Link INNER_TABLE_OPEN (provide structure for response
objects)
Link ADD_CONTROL_PARAMETERS_FRAME (add required
hidden variables to dynamic HTML to make selection re-select
Survey Engine & provide required operational/informational
parameters)
LOOP
Generate Nth response object
END LOOP
Generate 'Submit Response' or 'Re-Submit Response Button
Link INNER_TABLE_CLOSE
Link IFRAME_RESPONSE_CLOSE
RETURN
```

===== End Code =====

Once the referenced code sections have been executed and have performed their operations, communication between the SAS®/IntrNet server and the respondent's browser is terminated. All control information required for progression to the next question has been embedded within the display's response-object region thereby negating the need for continued communication between the server and respondent via sessions or other methods, thereby conserving resources on the server. The resultant display within the respondent's browser contains all the information required by the respondent to successfully understand and provide their response(s) to the question.

Conclusion

Professional survey construction and administration places tremendous demands on an application. The advent of the WWW proffers one of the most efficient and cost-effective methods of administering surveys to individuals located in even the most remote locations. The SAS®/IntrNet-based Survey Engine is the application of choice for any organization requiring a professional commercially available product capable of fulfilling its survey administration needs.

Contact Information

Bernard R. Poisson
 (801) 816-0137
BPoisson@Statprobe.com