Paper 015-29

SCL Rides Again! Porting RESMENU to the Web

Michael L. Davis, Bassett Consulting Services, Inc., North Haven, CT
Ralph W. Leighton, The Hartford, Hartford, CT

ABSTRACT

The Hartford had a problem. SAS® Version 8 on OpenVMS did not support text-based SAS/AF® interfaces used by the Corporate Actuarial Reserving area's RESMENU application. The alternative of using SAS/AF Frame with X-Windows had unacceptable drawbacks. So the Reserving Automation Support area set out to migrate RESMENU to a web-based interface using SAS/Internet® and AppDev Studio™. But using the latter toolset to construct Java Server Pages (JSPs) proved to be more complex and difficult than anticipated.

Fortunately, The Hartford discovered a winning alternative: SAS/IntrNet and SAS Component Language (SCL). This paper will offer a tutorial on how to build SAS/IntrNet applications that feature data-driven (dynamic) selections using SCL, SAS/IntrNet sessions, and a touch of JavaScript. The balance of this presentation will show how these programming elements are successfully used to bring The Hartford's reserving model support application to the web.

This paper is organized into the following three parts:

- An introduction to the RESMENU interface and the problems that necessitated it to be rewritten
- A primer on SAS/IntrNet and the Application Dispatcher
- Discussion of the programming techniques used to rewrite RESMENU

PART 1: RESMENU AND THE PROBLEM OF CONVERTING TO THE WEB

1.1 GENESIS OF RESMENU

RESMENU was created in the mid-1990s as the key component in a major effort to modernize the Reserving area systems. As late as mid-1996, the reporting programs and the reserving models were written in FORTRAN. Reserving System data was stored in FORTRAN direct access files. User support consisted of direct involvement of Reserving Automation Support (R.A.S.) staff. Users forwarded requests to automation staff members who then ran programs to produce the reports. Year-end reorganizations typically took two of the staff 2.5 months to complete. The first quarter production was invariably late. Ten Reserving Analysts were served under this arrangement.

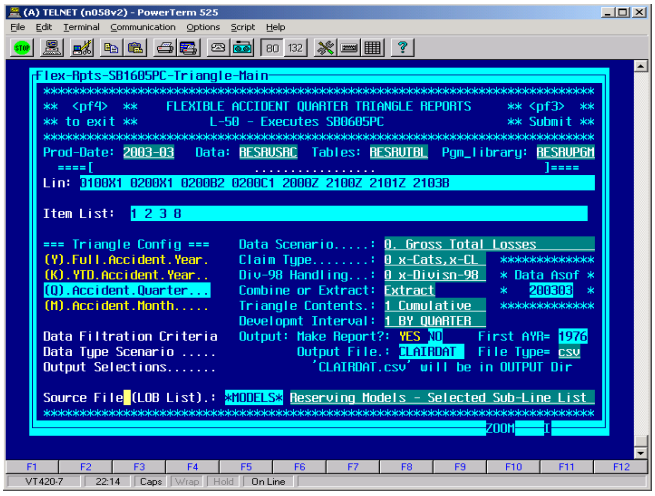


Figure 1.1
A Sample SAS/AF RESMENU Report Interface Screen
Flexible Triangles

In a nine-month period from August 1997 thru May 1998, R.A.S. staff totally rewrote the Reserving systems into version 6.12 SAS on OpenVMS. A key component of this rewrite was the creation of RESMENU, a SAS/AF application driver. This interface allowed the end users to submit their own model parameters and run the models by themselves. The use of SAS in RESMENU enabled the data capabilities to be greatly expanded.

Because of the limited processing horsepower of the DEC Alpha mid-tier then in use, a GUI interface for RESMENU (such as would be possible using SAS/AF FRAME) was not possible. So R.A.S. employed a "classic" text-based SAS/AF, using the older SAS/AF Program entries. These SAS/AF Program entries employ SCL. At that time, SCL was known as "Screen Control Language", but the acronym has since been recast as "SAS Component Language", starting with the Nashville Release (Versions 7 and 8) of SAS software.

In the RESMENU SAS/AF Program entries, SCL (Screen Component Language) provides the dynamic features needed for the menu activation and for the program interfaces that collect the user parameter choices. These selections are captured primarily in the form of Choice Groups and pop-up Selection Lists. The pop-up lists – and also the menus -- are what are called "Extended Table" SAS/AF Program entries. This RESMENU interface executes SAS programs in the form of macros to accomplish one task or another. Such a screen is shown as Figure 1.1 on this page. One of the Extended Table pop-up lists is

shown as Figure 1.2 below. As one selects from the extended list entry by hitting <enter> on the entry, the identification codes of the variate (claim financial statistic) populates the selection list prefixed by "sel". That list becomes one of the parameters passed to the report program making use of it.

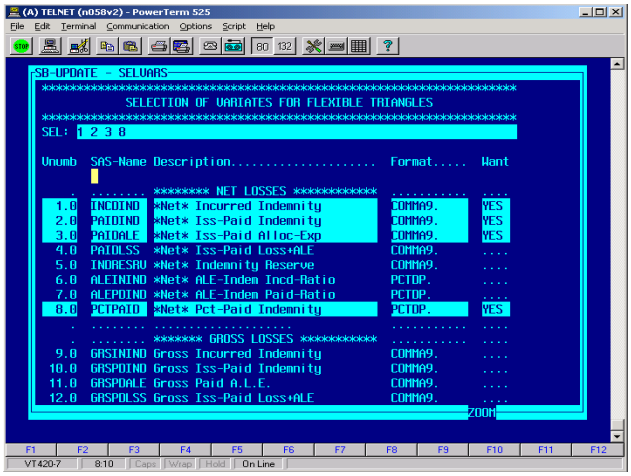


Figure 1.2
A Sample Extended Table Pop-up Selection List
Insurance Claim Statistics (Variates)

Despite having the text-based interface in an otherwise GUI world, RESMENU has worked out quite well, and it came to attract a sizable number of users from outside the Reserving area – people who have found the capabilities powerful and the system easy to use, despite the inability to employ the mouse! The current user base now numbers around 50, serving Reserving actuaries, their staff, planning consultants and also a number of insurance product pricing analysts, the latter having been attracted to the relative ease and speed with which data requests can be filled.

The scope of RESMENU goes far beyond service to the Reserving area analysts. The facility launches production processes that update or recreate the data and model results each month. R.A.S. staff uses RESMENU to handle application maintenance such as security, program library management and directory usage. With the introduction of RESMENU, the changes associated with reorganizations are completed in days rather than months.

One key feature of RESMENU, which we return to in our later discussions, is the "close coupling" of back-end data to the user interface. When a RESMENU user accesses a pop-up list -- such as a list of states or a list of sub-lines of business – the SAS/AF Extended list displays a SAS data set. The pop-up list depicted in Figure 1.2 is of this sort of interface.

1.2 THE PROBLEM AND SELECTED SOLUTION DIRECTION

With the introduction of SAS Version 8, a major problem arose for R.A.S. SAS dropped support for text-based terminal interfaces in directory-based operating systems such as Open VMS. SAS anticipated that customers would move to X-Windows (or similar) interfaces. But when R.A.S. staff tested a graphical terminal emulator, Excursion, with the RESMENU application, the result was ghastly. Even worse, each SAS window begat a separate taskbar button. Whereas users running under the text-based terminal emulator PowerTerm could run multiple SAS sessions, under Excursion, users were limited to a single session. Porting RESMENU to SAS/AF Frames under Excursion was considered and rejected. This alternative offered little benefit and yet required a major amount of development work to recreate the application.

So Reserving Automation Support decided to turn to a web-based solution. Using a web-based solution not only offered a more user-pleasing graphical interface, it would free R.A.S. from having to issue and support additional user IDs for the external users on the Corporate Actuarial Alpha mid-tier server. The following principles guided the RESMENU migration approach:

- **Use of a SAS Solution.** The entire RESMENU environment had been built using SAS software. There was a desire to stay with those products, despite the SAS decision to drop text-based terminal interfaces. Other Corporate Actuarial areas had developed web-enabled applications using non-SAS approaches. While each of these was successful enough in accomplishing the intended purpose, these alternative approaches also offered limitations and unappealing downsides.
- **Thin-Client.** The Reserving Automation area desired to avoid major Java programming. Installing SAS on desktop computers was not feasible.
- **Close Coupling of Tables to Front-End.** The solution should preserve this important feature of RESMENU, that user selections be drawn from SAS data sets.
- **Minimize Impact on Processing Software.** The reporting and data processing program structure (SAS macros) worked well. The need to make substantial modifications to processing programs was to be avoided.
- **Cross-Editing User Selections.** In some instances, values selected for one parameter were interrelated to or contingent on values of one or more other parameters. One selection might invalidate one or more other parameter choices. In the existing RESMENU, SCL code behind the screens prevented these incongruities. So the new system should do the same. As a related concept, Reserving Automation also wanted to preserve the feature of RESMENU where a series of selections from a list (for example, sub-line identification codes) were gathered and displayed on a text display.
- **Multiple End-User Sessions**: Users of RESMENU were accustomed to being able to run multiple concurrent sessions. R.A.S. was anxious for the replacement application to maintain this feature.
- **Convenient Development Environment.** R.A.S. staff hoped to be able to create a development environment with the same ease of use that they had enjoyed with SAS/AF program entries.

1.3 SELECTION OF APPDEV STUDIO (WEB/AF AND JSP) AND SET-BACKS USING IT

A formal consultation with SAS Institute technical staff resulted in a decision by R.A.S. to purchase SAS/IntrNet for the Alpha mid-tier server and AppDev Studio for the development desktop PCs. The license for AppDev Studio included licenses for SAS/IntrNet, webAF, and webEIS. AppDev Studio was licensed with the intention to reincarnate RESMENU as a JSP application. Licenses for Windows based SAS were included in the purchase arrangement, as these would be a necessary prerequisite for working with AppDev Studio and webAF.

As the R.A.S. unit started to work with the newly acquired product suite, it encountered two disappointments in short order:

- **Support for Close-coupling within webAF** JSP applications would require writing Java beans and some Java code, definitely more that the RAS staff had anticipated.. A problem was that the version of webAF that was then available to R.A.S. did not have the required interfaces available for "drag and drop" insertion of the interfaces needed for inputting the SAS data set tables in a JSP application.
- **Development Environment**: In general, the webAF development environment proved to be more challenging to work with than anticipated.

An interim solution was needed to start migrating RESMENU reporting functions, while the webAF generated JSPs approach was to be sorted out as a long-term solution.

1.4 AN ALTERNATIVE SOLUTION EMERGES

By late 2002, Reserving Automation Support was totally frustrated by lack of progress in mastering webAF, and in consequence the unit expressed great interest in a proposal offered to them by Michael Davis, Bassett Consulting Services. Michael advocated a different approach to meeting The Hartford reserving support area's immediate requirement to creating a web interface for the RESMENU application.

This approach would still use the AppDev Studio licenses, but instead of using webAF and JSPs, it would make use of SAS/IntrNet and SCL. SCL was available, since SAS/AF was a component of the SAS licenses acquired for the desktop computers. The SAS/IntrNet Application Dispatcher would both launch the reports and other programs requested by users as well as generate the HTML screens with which the user would interact. The browser interface screens would be written by SCL programs called by the Application Dispatcher.

Thus the actual user interface would be HTML, and it was anticipated that a modest amount of embedded JavaScript would be needed. "Application State" (retention of the user's previous selections) would be maintained by use of the SAS/IntrNet Application Dispatcher sessions. Because of the need to maintain application state, it would not be possible to use SAS's htmSQL to build the screens.

R.A.S. staff would need to acquire a comfortable knowledge level with basic features of HTML, HTML forms and (as implied above) a little JavaScript. As with the webAF JSP alternative, the approach would make use of the Apache webserver. For development and testing, Apache could be (and eventually was) installed on the area staff desktops.

To illustrate the feasibility of this approach to the R.A.S. unit, Bassett Consulting built and demonstrated a "proof of concept" application using the baseball example employed in the *Observations* article, "***SCL for the Rest of Us***". The application demonstrated that the combination of SCL and HTML could dynamically create a drop-down or scrollable selection list and it could also write the selected data to a comma-delimited value (CSV) file that could be imported into Microsoft Excel, thus duplicating what RESMENU currently did as a report service for its existing users.

The approach held great appeal to The Hartford. Reserving Automation Support had great familiarity with SCL as part of developing the RESMENU SAS/AF program entry processing logic. Similar to the webAF JSP approach, the Bassett Consulting approach was also server-centric and thin-client.

Of key importance to the client was the preservation of "close coupling" of the existing selection tables to the front end. In a manner paralleling the legacy RESMENU application, the new approach could access SAS data sets and convert them into selection lists by the SCL software. Finally, even with the change in approach, development would make extensive use of SAS run under Windows on desktop personal computers as the development environment.

PART 2: A PRIMER ON USING SAS/INTRNET AND THE APPLICATION DISPATCHER

2.1 HTML FORMS

The RESMENU application is started from a pure HTML screen. While this screen has been set up to start any of a number of applications, all SAS/IntrNet applications are launched by means of an HTML form. The HTML form sends both static parameters and user selections to a Common Gateway Interface (CGI) program, the Application Server. What kinds of static parameters are sent from the HTML form to the Application Dispatcher? For the RESMENU application, they include:

- location of the Application Broker
- service used to process the request
- name of the program that executes the request
- value of _debug dispatcher options

In the RESMENU application, the start-up screen is the only web page placed in a location where it can be seen readily by the browser.

The selection list on the start-up screen displays the functional descriptions. The values sent to the Application Dispatcher are the names of the SCL catalog entry to be executed. A small JavaScript routine displays the selected program in a window. Once the user hits the "submit" button, the Application Dispatcher executes the selected SCL program.

2.2 THE SAS APPLICATION DISPATCHER

The Application Dispatcher consists of two components: the Application Broker; and the Application Server

The web server launches the Application Broker each time a RESMENU user clicks on the "submit" button of a screen. As noted in Section 2.1, the HTML form tells the web server where to find the broker, what service should be used, and what program should be run.

The Application Dispatcher supports three types of services or Application Servers: socket, pool, and launch. For development and testing, the R.A.S. staff member starts and uses a socket service. For production, the intent is to use a pool service. A socket service is a continuously running SAS session. A pool service is one or more server sessions, started and stopped by a load manager, which monitors the activity level of the server sessions.

The Application Broker forwards information from input fields on the HTML form and any configuration information specified in the Application Broker configuration file. For socket service, the configuration file specifies the machine name (LOCALHOST on the developer's PC) and the TCP/IP port number to receive the request.

Programmers who have not done any web programming are often confused as to where things are located in order to get something to work. This was certainly true of author Ralph Leighton. On Ralph's computer, for example, LOCALHOST is actually mapped to the following directory:

C:\Program Files\Apache Group\Apache2\Htdocs

where **Apache2** is the web server location. In order to be easily invoked in Microsoft Internet Explorer or in Netscape, this subdirectory must be where the HTML file initiating the application, called StartUp04 in the example discussed in Part 3, must be placed. The rest of the application can reside elsewhere, since access to the application components is handled by the SAS librefs allocated to the Application Dispatcher. That flexibility is a major benefit of the approach.

Figure 2.1
A Sample PROC APPSRV Run

```
PROC APPSRV;
  Allocate  File RESRVPGM '\\CORPACT\\SASPGMS';          /*  RESMENU Programs    */
  Allocate  Library RESRVTBL   'RESRVTBL'   server=N058V2;   /* RESMENU Tables.. */
  Allocate  Library RESRVSRC   'RESRVSRC'   server=N058V2;   /* Source Data for Reports */
  Allocate  Library RALPHSCL  'RESRVCAT'   server=N058V2;   /* SCL Program Catalog Location. */

  Proglibs  RALPHSCL  RESRVPGM ;
  Datalibs  RESRVSRC  RESRVTBL ;
Run;
```

2.3 PROC APPSRV

The SAS Application Dispatcher executes a SAS program file that in turn executes a SAS Procedure called PROC APPSRV. This procedure is a component of SAS/IntrNet software, and Application Server sessions are invoked by APPSRV. On Ralph's computer, the location of the files that start the default socket service are in the SAS product directory structure:

C:\SAS\Intrnet\Default

For the purpose of understanding the RESMENU example (discussed in Part 3), we should note an important job that PROC APPSRV performs: allocation of the program libraries and the data libraries. This library allocation activity is very similar to the use of the LIBNAME and FILENAME statements in Base SAS. But there is an important difference in the way PROC APPSRV procedure distinguishes between program libraries and data libraries. The following types of statements need to be set up:

ALLOCATE FILE statements: these must be set up for any SAS Program Directories OR for any External File Directory sources, such as for csv or prn files used as input.

ALLOCATE LIBRARY statements: these must be established for any SAS Libraries, to get at catalogs or to access or create permanent SAS data sets. An important catalog for this discussion is that containing the SCL programs.

DATALIB statements: these specify the librefs defined in the ALLOCATE statements as applying to data sources, whether these be external file directories or SAS libraries.

PROGLIB statements: these specify those librefs defined in ALLOCATE statements as applying to program sources, whether these be SAS program directories or libraries containing SCL catalogs.

Thus the libref definition process has two stages to it. The ALLOCATE statements define the nature of the sources or target files. The DATALIBS and PROGLIBS statements define functional usage. Figure 2.1, on the previous page, illustrates, by way of example, the type of set-up required using SAS Connect to access programs and data on the Alpha Mid-tier.

In the example shown, the physical location of the SAS libraries is actually on the Alpha mid-tier. The syntax of the ALLOCATE statements is consistent with establishing a connection to them using SAS/CONNECT. (See section 2.7 below.) Below is the command sequence used to initiate the Dispatcher socket service and start up APPSRV on the PC:

→ **SAS System**
→ **Internet**
→ **Default Service**
→ **Start interactively**

As the session starts up, you briefly see the SAS graphic – and then a button for the Dispatcher appears on the task bar. Shutting the service down is a matter of right-clicking the task button and selecting the CANCEL option.

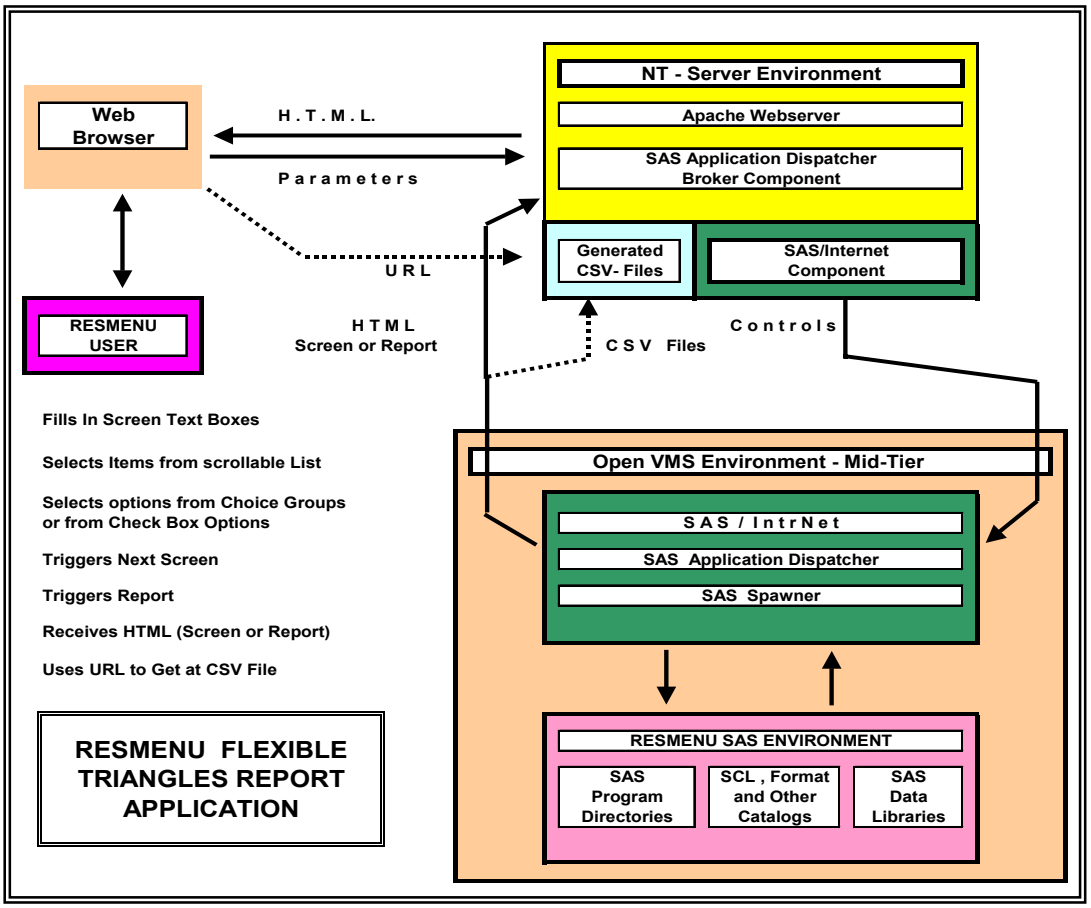


Figure 2.2

2.4 THE BROKER RUNS NON-VISUAL SCL

SCL is normally viewed as being associated with visual and dynamic displays, which is the case in SAS/AF RESMENU. But in this approach to web applications, the role of SCL is effectively reduced to being a batch programming language. The SCL

programs we will be discussing here have to know all parameters as execution starts. There is no way to feed to them replacement values for these parameters while the programs are executing. The reason is simple: there is no screen with which the SCL programs can interact. And this "batch role" has some ramifications on how the screens are set up for the application.

How are parameters passed to such a program? The Application Broker generates macro variables to pass name/value pair values to the SCL programs. The text boxes, checkboxes, radio buttons and other screen elements on the HTML form that calls the SCL program generate the name/value pairs.

One important difference between SAS macro programs and SCL programs should be noted. SAS Macro programs attempt to resolve the values for macro variables at execution time. By comparison, SCL programs attempt to resolve macro variables at compilation time. Since we want to defer resolution to execution time, the SCL programs should use the SYMGET, SYMGETC, and SYMGETN functions to get the values of the name value pairs.

What do these SCL programs do? The SCL programs create RESMENU screens, utilizing the following algorithm:

1) If it is building an HTML selection list, it reads a **control table** (as a SAS Dataset) into SCL lists for later use.
2) It writes HTML **header block** for the screen
3) It writes HTML **form header**. For the screen to activate a further function, it must have at least one HTML form.
4) It writes **form elements**. An obvious example is a "submit" button of sorts to activate the next function. But also part of this are the various parameters needed, if not by PROC APPSRV itself, at least by the SCL program creating the next screen.
5) It forms form **choice groups** and **selection lists**. The latter are in the form of HTML drop-down or selection lists. The former can be radio button clusters or checkboxes.
6) It write **closing tags** for each HTML form
7) Write **closing tags** for the HTML page

Part 3.0 of this paper will offer further details and examples.

2.5 OUTPUT TO _WEBOUT

Once an SCL program creates the HTML for a RESMENU screen, there is the small matter of returning the created screen back to the user's web browser. How does this take place? The Application Dispatcher creates a special fileref, called **"_WEBOUT**". This special fileref is actually a TCP/IP socket connection to the Application Broker. Sending output to **_WEBOUT** streams the SCL generated HTML screen back to the browser.

The **_WEBOUT** socket functions as a pipe, opened in an append mode. Thus, it is not possible to modify or overwrite what has already been written to **_WEBOUT**. As the Application Broker receives the component HTML lines of the screen from the SCL program, it does a quick consistency check on the HTTP headers and sends the results back to the Web server, which in turn streams the results back to the browser. Because of the streaming, results may begin to appear in the browser before the program has finished processing.

Because **_WEBOUT** is a fileref, there is an easy way for developers to inspect the output of SCL programs being tested. The SCL program can be altered to allocate **_WEBOUT** to an ordinary HTML file on a PC-accessible directory. This then can be inspected in Notepad or in an HTML editor such as FrontPage or Dreamweaver. The latter are especially useful if the screen has real-time interactions written in JavaScript. This redirection of output to a special output file can be set up as a macro parameter to the SCL program. In the RESMENU application the parameter is a "**SAVE_**" variable called **save_TESTAF,** which defaults to null in production, but is set to 1 in the AUTOEXEC.SAS on the developer's PC.

2.6 MAINTAINING STATE BY MEANS OF SESSIONS

One aspect of HTML often overlooked is that HTML pages are "stateless". After the user makes selections on an HTML form and clicks on a "submit" button, the Application Server resets itself. If a second request is made, the server remembers absolutely nothing of the first request.

Why is the stateless nature of HTML applications an issue? One of the RESMENU design principles is that user selections need to be maintained and be available for cross-editing and eventual passage to the report program. Fortunately, the Application Dispatcher supports a feature called "**Sessions**".

Sessions allow SAS/IntrNet to retain state between Application Broker requests. Within a Session, two types of information can be retained: macro variables and library members (data sets and catalogs).

How are Sessions implemented? The SCL program that writes the HTML page calls the APPSRV_SESSION function. Several macro variables, including **_sessionid**, are then set with the values of the current session. Subsequent SCL programs can be run in the same session. As each program runs, it picks up the macro variable "**_sessionid**". and passes it along to the Broker as an HTML tag (hidden variable).

How does the Application Dispatcher know which macro variables and library members to bind to the session? Any created macro variables prefixed with "**SAVE_**" are retained across requests. One such example was mentioned above: **save_TESTAF**. Recall that this one is used in the RESMENU application to determine what the SCL program should define as output for the html: "**_webout**" or a flat file. Such "SAVE_" macro variables can be created as names of scrollable Select Lists or Radio Button clusters or Check Boxes or Input items (hidden or text boxes). They can also be created using CALL SYMPUT in the SCL code.

In addition, library members in the **SAVE** libref are also retained during the Session. The SAVE library is thus the analog to the WORK library in ordinary base SAS programming using the SAS Display Manager. The values of these macro variables and library members are saved until the session expires due to browser closure or time-out.

2.7 EXPORTING RESULTS TO SPREADSHEETS

The current version of RESMENU creates all of its reports in the form of CSV files as an option. This is because the Reserving analytical staff exclusively uses Excel as their programming and analysis tool. In order to maintain this RESMENU functionality, the web version of RESMENU must be able to export data to non-SAS programs, preferably again in the form of csv files. In general, beyond the immediate needs of RESMENU, a common need is the ability to get SAS data into Lotus or into Microsoft Office applications such as Excel, Word and PowerPoint.

Comma-Separated Value (CSV) files are actually an excellent choice for exporting tables, since they can be generated simply in Base SAS. One quick way is to use the **DS2CSV** ("Data Set to CSV") macro, which is bundled with SAS/IntrNet. This macro was used in the baseball proof–of-concept model to produce the CSV version of the baseball team model's reports. A second alternative would be to use ODS custom tagsets to generate CSV or other compatible files. A third alternative is to build simple DATA Steps with the FILENAME definition pointing to the Browser. This last approach is used in the RESMENU report modules, since the Data steps were already in place as part of existing reporting software. All that was needed was to change the nature of the FILENAME statement to point to **"_webout**"

Although not used in RESMENU, for Reports to be exported to Microsoft Word, the ODS RTF destination ("rich-text") is a good choice. The ODS PDF destination renders files that can be viewed by Adobe Acrobat or printed with defined page breaks.

We should also mention Graphical Images. These were not (and are not) part of the immediate scope of RESMENU, since RESMENU report modules never did produce graphs. But, should such a need arise, these can be exported as GIF or JPG files, rendered in the browser via HTML image tags. The user can copy the image to the clipboard and paste it into Microsoft Office.

Generally, SCL programs generate the CSV, RTF, and PDF output files regardless of the user selections. Each file is given a unique, random filename. If the user needs those files, the user clicks on a hyperlink produced by the SCL program. The CSV, RTF, and PDF files are spooled to a directory on the web server, which is purged by a scheduled task each evening.

2.8 CONNECTING TO DATA

When RESMENU is actually run in production, all of the software – program libraries, SAS catalogs containing SCL and formats, and SAS data libraries – are actually on OpenVMS. Because the current SAS/AF RESMENU still has to access these, the data libraries and catalogs are actually Version 6.12. Clearly it is desirable to avoid duplicating these in the development environment on the PC. So SAS/CONNECT is used to enable the Desktop version of SAS to use the Open VMS data libraries on the Alpha mid-tier. Only format catalogs need to be cloned to the PC development environment.

We should note one thing about the format catalog. RESMENU makes very heavy use of user defined formats, mostly for table lookup. An important class of these formats is those which assign descriptions to sub-lines of business, to geographic states in the USA or to claim categories. Theoretically it should be possible to have PROC APPSRV make available such a format catalog in a manner analogous to the LIBNAME statement in ordinary Base SAS. Unfortunately we could never discover how to do this. As a result, each SCL program and each SAS program has as part of its up front code a LIBNAME statement assigning LIBRARY. The target reference, however, is very simple: it is is simply the SAS Data library containing the various tables, RESRVTBL, and (as shown previously) that data library is allocated by PROC APPSRV.

PART 3: PROGRAMMING TECHNIQUES – IN THE CONTEXT OF AN EXAMPLE

3.1 THE COMPONENTS AND SYSTEM ARCHITECTURE OF THE EXAMPLE

The discussion that follows is probably best appreciated in the context of a miniature of the actual "Flexible Triangles" reporting application built using the Bassett Consulting recommended approach. The Figure 3.1 (next page) depicts the flow of this miniature, and Appendix 2 shows a sample report, a so-called loss development triangle. The actual screens that are produced by the SCL and that the user interacts with are shown in Appendix 1.

There are four components screens used to collect and edit the user's parameter selections, these being

MAIN : a central screen used to access the other three screens and from which the report is launched after all parameters are collected.

SCENARIO : A screen called by MAIN in which the user sets choices using radio button choice groups. The exit from SCENARIO returns the user to MAIN.

SELSOURC : A screen called by MAIN in which the user makes a single selection from a list as to what cluster of sublines he wants to make his sub-line selections from – e.g. Personal Automobile or Standard Commercial Workers Compensation. The exit from SELSOURC returns the user to MAIN.

SELSUBLN : A screen called by MAIN in which the user makes his selections of sub-lines. The report program will make a version of its report for each such sub-line. An important wrinkle here is that the "sources" selection made above drives the actual table shown in this screen. The list of sub-lines selected here is a parameter to the report program. The exit from SELSUBLN returns the user to MAIN

All four of the above screens are actually produced by SCL programs, as explained in Part 2. The user enters this application from a simple HTML screen, noted as **STARTUP04** on the diagram. This causes the execution of the Application Dispatcher to run the SCL program **MAIN**, producing the MAIN screen. This particular screen **MAIN** actually has **four** HTML forms in it, three corresponding to the other screens (**SCENARIO**, **SELSOURC** and **SELSUBLN**) and the fourth corresponding to the final step, the execution of the SAS program making the report. Each form has a submit button on it. Each transition the user makes, from screen to screen -- or (in the last instance) from **MAIN** to the execution of the report program -- sends a request to the SAS Application Broker, which then executes, within the PROC APPSRV Session, the SCL program OR the report module

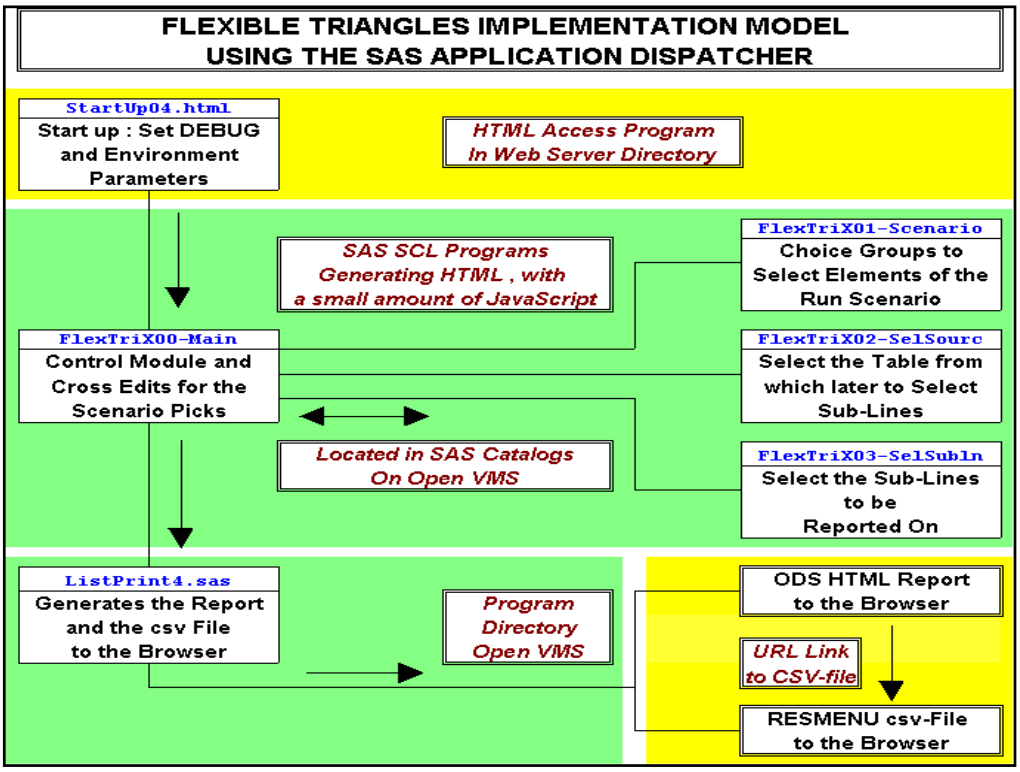


Figure 3.1

(a SAS macro). Since all executions have occurred within the same Session, the accumulated parameters to the report program, in the form of "SAVE_" macro variables, are all known to the report program, a SAS macro.

In each screen-to-screen transition, what the Application Dispatcher causes to be executed is the SCL program creating the target screen the user wants to move to: we noted this already in Part 2.0 of this paper. For example, suppose the user is in the **MAIN** screen and wants to get into the **SCENARIO** screen. To do this, he/she hits the SUBMIT button, labeled "Scenario", which actually resides in one of the HTML forms in **MAIN**. This form has, as the "Action=" target, the SAS Application Broker, and the HTML form contains an HTML INPUT entry whose name is "_program" and whose value is the text name of the SCL program which (when executed) will create the **SCENARIO** HTML screen.

Once the user is in **SCENARIO** and has made his/her choice group selections from the radio button clusters, he/she hits the SUBMIT button located in **SCENARIO**. This button resides in an HTML Form whose "Action=" reference is again the Application Broker and whose "_program" INPUT entry has as its value the SCL program creating the **MAIN** HTML screen. The radio

Figure 3.2

Program Flow for the SCL Programs Generating HTML Screens

Declarations - Any LENGTH and ARRAY statements

INIT:

Check that the Application Dispatcher Session is still open. (And in the control Module FlexTriX00-Main, if there is not a session, create a new session.).

Retrieve the values of any "**save_**" macro variables as SCL variables, using SYMGET. There are a lot of these, reflecting in the end the parameters that get passed along to the report program.

Open **"_webout**" the standard output file to the Web. In TESTAF mode, this will instead be a file to a PC Directory for debugging purposes.

Create the HTML <header> section of the screen and output same to "_webout". In the set up we have used, this is a LINK subroutine call.

Begin the HTML <body> section (titles, et al) and output same to "_webout" In the setup we have used, this is often part of the above subroutine.

Create the output screen. Start the HTML <form> and plaster out the non-interactive stuff at the top of the form. This is invariably a LINK subroutine call.

Add to the <form> the html for text, text boxes, hidden variables radio buttons, select lists, and so on. This may be fairly complicated. It may consist of one LINK subroutine call. Or it may be consist of several LINK subroutine calls. External routines might also be used.

Write out the html to create the Submit button, close out the form and close out the html. This is usually a LINK subroutine call.

Perform any end of html stuff such as saving "**save_**" variables when running under TESTAF. Close out the screen HTML <body>. Close out any files and delete any HTML lists.

RETURN;

Link Subroutine A

Link Subroutine B

button selections come back to this SCL program in the form of "save_" macro variables. And **MAIN** is regenerated with a screen display of the selected parameters.

Transitions from **MAIN** to **SOURCES** and **SUBLINES** and from them back to **MAIN** work similarly. The reason screen MAIN has four HTML Forms, rather than the single Form the other three screens have, is that MAIN needs an HTML form for each possible target selection of "next destination" the user can make, since we set the screen up with separate SUBMIT buttons for each function. Each transition is made with a SUBMIT button, and each HTML Form can support only one such button.

3.2 STRUCTURE OF THE SCL PROGRAMS

Some of the material in this section has already been discussed in general terms in Section 2.4. The purpose of this section is to describe the actual SCL programs as a basis for explaining programming techniques.

As we have already noted in Section 2.4, these SCL programs have no display screen. As catalog entries they have an Entry Type of "SCL", whereas, by contrast, the RESMENU SAS/AF entries have the Entry type "Program". Since the SCL programs have no display screens, all of the "main" part of their procedural SCL code is located in the INIT block, there being no MAIN or TERM blocks in these program.

The SCL modules, however, can have LINK subroutines called from code in the INIT block, and use of such subroutines is valuable in breaking such modules up into more manageable pieces as well as for isolating chunks of code that are used in more than one place. An example of the latter is forthcoming in section 3.7. The only exceptions to these "code location" rules are the non-executable declarations, like LENGTH or ARRAY statements, which can (and probably should) appear up front outside of the INIT block. The flow of such an SCL program is pretty much as shown in Figure 3.2:

3.3 CREATING THE HTML TO THE BROWSER

As explained earlier, the HTML code that is generated by the SCL programs is directed to a fileref called **"_webout**". Unless redirected to a file by using a FILENAME statement, the generate code appears in the user's web browser. In these programs, each fragment of html code is first placed into the buffer for **"_webout**" using the FPUT function and then it is written out using SCL function FWRITE. For example, suppose I want to create (using this approach) the HTML title line "Fonzie is a Good Basset Hound". As HTML this is:

```
<H2>Fonzie is a Good Basset Hound</H2>
```

and the following SCL sequence will generate this to the browser:

```
rc = Fput( fweb_out,  '<H2>Fonzie is a Good Basset Hound</H2>' );
rc = Fwrite( fweb_out)
```

In the above, "fweb_out" happens to be the SCL file-id for the SAS Filename "_webout", and the file is opened in SCL thusly with an FOPEN statement before use:

```
fweb_out = Fopen('_webout', 'O')
```

where "O" is for "output". After the creation of the screen is complete, the file is closed using an FCLOSE statement as per the statement:

```
fweb_out = Fclose ( fweb_out )
```

(As a safeguard, this curious syntax resets "fweb_out" to zero.) As a means of entering each and every HTML line constituting the screen, the FPUT-FWRITE sequence can yield some pretty repetitious and verbose SCL code, and so a number of steps have been taken to make the programs as readable as possible.

Figure 3.3
Creating the Header Area and Titles of the Screen - Using an SCL List

```
rc = Insertc ( htmlcode,  '<HTML>',  -1);
rc = Insertc ( htmlcode,  '<HEAD>',  -1);
rc = Insertc ( htmlcode,  '<TITLE> Flexible Triangle "Sources" Table </TITLE>',  -1);
rc = Insertc ( htmlcode,  '</HEAD>',  -1);
rc = Insertc ( htmlcode,  '<BODY>',   -1);
rc = Insertc ( htmlcode,  '<H3>FlexTri03-SELSUBL – Select Sublines </H1>',   -1);
rc = Insertc ( htmlcode,  '<H1>RESMENU Flexible Triangles</H1>',   -1);
rc = Insertc ( htmlcode,  '<H2>Sub-Lines Selection</H2>',   -1);
```

3.4 TAMING CODE CREATION USING SCL LISTS.

Readers not familiar with SCL list should consult the paper by Lisa Horwitz cited as a reference. In each of the SCL modules we create an SCL list to hold the lines of the HTML code; its name in this mini-application is **HTMLCODE** (which sort of indicates what we're going to do with it). The SCL List is created using the SCL MAKELIST command:

```
htmlcode = Makelist(  );
```

This statement appears at the top of the INIT block. Then the code is loaded into **HTMLCODE** using the INSERTC statements shown in the text box in Figure 3.3 above. This code display illustrates the loading in of the Top part of the HTML screen, with the entered display title: "RESMENU Flexible Triangles" on the first line and the title "Sub-Lines Selection" on the line below in smaller font.

Once a good "stash" of HTML lines has been created, they can be dumped out onto the web browser using the following DO-loop:

```
Do JJ = 1 to Listlen( htmlcode );
   rc = Fput(fweb_out, Getitemc(htmlcode, JJ)
   rc = Fwrite(fweb_out) ;
End;
```

The LISTLEN SCL function is used to tell the DO-loop how many entries are in the SCL list **HTMLCODE**. Once emptied, the SCL list must be cleared and this is done using the CLEARLIST function:

```
rc  =  Clearlist ( htmlcode );
```

At the end of the SCL program, the SCL List **HTMLCODE** should be deleted using the DELLIST function to prevent memory leakage:

```
htmlcode  =  Dellist ( htmlcode );
```

This curious syntax, with **HTMLCODE** as the target as well as argument, resets **HTMLCODE** to zero to help avoid accidental subsequent use of the SCL list-id.

Since the above DO-loop and CLEARLIST sequence has a good chance of being executed more than once, this little block of code is a good candidate for being set up as a LINK type subroutine in the SCL, a program technique we shall emphasise in other contexts.

Figure 3.4

Creating a Choice Group Set of Options as a Stored SCL List

```
CLMTYPE_list = Makelist( ) ;
  rc= Insertc( CLMTYPE_list,  '0 Losses, x-CL, x-Cats', -1);
  rc= Insertc( CLMTYPE_list,  '1 Catastrophe Losses',  -1);
  rc= Insertc( CLMTYPE_list,  '2 Continuing Litigation Losses', -1);
  rc= Insertc( CLMTYPE_list,  '3 Losses Ex-Continuing Litig',  -1);
  rc= Insertc( CLMTYPE_list,  '4 Total all Losses',  -1);

rc = Savelist( 'catalog',
               'RALPHLIB.ChoicGrp.CLMTYPE.slist',
               CLMTYPE_list,
               dummy_list,
               'Contents of Ralph List' ) ;
```

3.5 CREATE RADIO BUTTON CHOICE GROUPS USING SCL LISTS.

A second use made of SCL lists in the RESMENU application was to set up choice groups on the screens. These are set up as Radio Button clusters. This usage had two components to it:

- Creating the components of the Choice group as an SCL List and storing the list.
- Using the stored SCL List containing the components to actually generate the radio button sequence.

The SCL code used to create of the SCL list is shown in Figure 3.4 (above). Since more than one screen may use a particular choice group, a good ploy is to save the SCL list to a permanent catalog, from which it can be retrieved later. In point of fact, in RESMENU, the actual code that creates of the choice group options as an SCL List (as per Figure 3.4) is located in a separate SCL program, not in the SCL program that generates the radio button cluster. Thus you will observe that the bottom part of the code in Figure 3.4 empties out the SCL List as an entity into the catalog **RALPHLIB.SCLLIST**. The SAVELIST function is used for this purpose.

How the SCL List is then accessed to produce the Radio Button Choice Group is shown in Figure 3.5 (below). To retrieve a stored SCL list one uses the SCL FILLIST function. The DO-loop then sets up the Radio button cluster with a label at the top – in this case “Claim Type”. The DO loop then picks off each description, which also has the Claim Type Code value embedded as the first character – see the text options in Figure 3.4 above. As each description and code value is pulled off the particular radio button instance is created – i.e. the text for it is uploaded in the SCL List **HTMLCODE**.

Figure 3.5

Using a Stored SCL List to Create the Choice Group

```
CLMTYPE_list = Makelist() ;
rc = Fillist('catalog',
             'RALPHLIB.CHOICGRP.CLMTYPE.slist'
             CLMTYPE_list ) ;

rc= Insertc(htmlcode, '<P><B>Claim Type:</B><BR>',                    -1);
Do JJ = 1 To ListLen(CLMTYPE_list);
    CodeDescr = Getitemc(CLMTYPE_list, JJ);
    CodeValue = Substr(CodeDescr, 1,1);
     rc = Insertc( htmlcode,  '<INPUT Type="radio" Name="save_CLMTYPE" ', -1);
     rc = Insertc( htmlocde,  'value="'||CodeValue||'>'||trim(CodeDescr)||'<BR>',-1);
End;
```

3.6 JUDICIOUS USE OF COLOUR.

These HTML-generating SCL modules tend to be “wordy”. So to further make it easier to pick out the HTML code being constructed, we have incorporated into the SCL Programs color conventions. If you look over the examples of the preceding section in a coloured version of this document, you can see a little bit of this use of colour. The HTML code itself is green coloured, which allows a person viewing the SCL to see it clearly, even though it is buried in the INSERTC statements. In the SCL programs the following conventions are used:

Color	*Use*
Black	SCL code except where noted below
Blue	Comments
Red/Orange	Items that should stand out, such as section labels, RETURN statements, key data set and file names, LINK statements
Green	HTML code and messages "put" to SAS log
Pink	JavaScript

One VERY unfortunate aspect of the use of color in SAS/AF entries is that when the SCL program listings are printed to a color printer, SAS does not pass along the colors. The only way to get the colors in a printed format is via a screen print. And perforce such Screen prints are limited to what is shown in a single screen shot. In the code examples depicted in the previous sections of the paper, the colors had to be re-installed manually.

Coloring a shaded block of code is accomplished using the command Display Manager command COLOR MTEXT <*color*>. Clearly it would be quite tedious to keep typing this command each time we wanted to colour something, and so we make use of the following function key assignments to apply the colors quickly:

Color	*Assignment*	*Color*	*Assignment*
Black	<Cntl-E>	Magenta	<Cntl-I>
Green	<Cntl-G>	Red/Orange	<Cntl-D>
Blue	<Cntl-B>	Pink	<Cntl-J>

3.7 MODULES AND SUBROUTINES:

Another feature of the RESMENU web application is the breaking the SCL code into processing modules, and the SCL programs heavily reflect that feature in two ways:

INTERNAL : in the modules themselves, the programs are heavily broken up using so called "LINK" subroutines, and this step has helped maintenance as well as understandability. This type of subroutine usage was of tremendous assistance when the order of execution of functions was, in the case of two key modules, substantially altered.

EXTERNAL : separate programs and also subroutines which concentrate a commonly used sequence of code in one place. "Recyclable Code"

The latter direction merits explanation. We have already noted in section 3.5 that the Choice Group Creation process is actually in a program separate from the modules that use the option list. That is an example of an external program that locates the list creation process in a single spot. But external **subroutines** can also be used to good advantage for common generic processes, with parameters passed to them to fill in what is need for the particular use. The vehicle for such external calls to SCL subroutines is the SCL Call Routine CALL DISPLAY, whose first parameter references the SCL entry to be executed and whose subsequent parameters are those passed to the module by the calling routine. The latter are picked up in the called module in an SCL ENTRY statement at the top of the SCL code. Parameters can be character or numeric. One can pass arrays, but, as noted below, not SCL lists.

To see one spot where such a subroutine might be valuable, look again at Figure 3.3, which shows the creation of the header portion of an HTML screen. All the SCL programs that create the RESMENU screens are going to create a section very similar to the code shown. The only difference between one or another of these screen creation sequences lies in two places on the screen: the program name on the first title line; and the function description on the second title line. These two values can be passed into a call to a subroutine creating the top part of the code as parameters:

```
Call Display ( 'RALPHSCL.RESRVCAT.S01HDR.SCL'
               Fweb_out,
              "FlexTri03-SELSUBL – Select Sublines" , '
              "Sub-Lines Selection" )
```

where S01HDR is the name of the subroutine. This subroutine is actually going to have to create the HTML since we are not aware of any way to pass back and forth an SCL List like **HTMLCODE** as a parameter. Thus one of the parameters it must receive is the fileref pointer **fweb_out** pointing to **"_WEBOUT**".

We mentioned earlier the likely repetitious use of the code that empties and then clears out SCL List HTMLCODE, and we suggested it as a strong candidate for being isolated in a LINK subroutine. Here's what it might look like:

```
S00_DumpCode:
  Do JJ = 1 to Listlen( htmlcode );
      rc = Fput(fweb_out, Getitemc(HTMLCODE, JJ) );
      rc = Fwrite(fweb_out) ;
  End;
  rc = Clearlist( HTMLCODE );
Return;
```

But internal subroutines can serve a second purpose – to break up an otherwise lengthy and tedious module into bite-sized pieces, allowing the INIT part of the program to clearly show the program flow. Consider MAIN, whose full name is FlexTriX00-Main. In that module, there is a subroutine for each HTML form underneath that screen. Similarly SCENARIO (FlexTriX01-Scenario) has a subroutine for each Choice Group. Figure 3.6 shows, for this latter program, the resultant code in the INIT block itself after many of the processing details have been separated into separate routines. Note also the call (CALL DISPLAY) to the SCL module that creates the top end of the screen. See Figure 3.3 and the discussion below it.

Figure 3.6
Processing Flow of INIT Block in MAIN

```
/* Write HTML header  */
   ImageFile = 'http://'||trim(server_nm)||'/Ralph/logo.gif';
   Call Display ( 'RALPHSCL.RESCAT.S01HDR.SCL'
                  Fweb_out,
                 "FlexTri01-MAIN – Control Module" , '
                 "MAIN SCREEN" )

/* Create Dummy Form at top to Display the Debug Parameter  */
   Broker = 'http://'||trim(server_nm)||'/cgi-bin/broker.exe ';
   Link S02_FormStart ;

/* Perform the Cross Edits for the Current Choice Group Selections  */
   Link S03_CossEdits ;

/* Get the  Current Choice Group Selection Descriptions  */
   Link S04_OptionDesc ;

/* Create the Form for Scenario Choice Group Selections and Errors  */
   ProgramName = "Ralphscl.Triangle.FlexTriX01_Scenario_2.SCL";
   Link S05_ChoiceGrps ;

/* Create the Form for Selecting the "Sources" Table  */
   ProgramName = "Ralphscl.Triangle.FlexTriX02_SelSourc_2.SCL";
   Link S06_SourceTable ;

/* Create Form to Go and Select the Sublines  */
   ProgramName = "Ralphscl.Triangle.FlexTriX03_SelSubln_2.SCL";
   Link S07_SelSubline ;

/* Create the Form for Launchng the Report  */
   ProgramName = "RESRVPGM.SB0615WB.SAS";
   Link S08_MakeRept ;

/* Finish off the HTML for the Screen ..... */
   Link S09_CloseOut ;
```

3.8 CLOSE COUPLING : CREATING SELECTION LISTS FROM SAS DATASET TABLES.

An important feature of RESMENU is the direct connection of the user interface to tables (in the form of SAS Datasets) containing the item listing the user is going to select his/her choices from. We have called this feature "close coupling" of the tables to the front-end. It means that, whenever a table is changed, the revised version of the table is immediately available to all users without any special maintenance to the interface software. The actual update of such tables is actually a class of functions (applications) on the SAS/AF version of RESMENU, a class of functions available only to Reserving Automation Support staff.

In the context of the SAS/AF version of RESMENU, the selection mechanism presented to the user was a capture of the table as an extended list. An example of such a screen was shown in Section 1.0 In this environment, the user clicked on the entries he wanted and the selections populated a character string, which became a macro parameter sent to the report program. In the context of the SCL approach to a web interface, the actual selection mechanism presented to the user is going to be an HTML SELECT GROUP. The SCL Code inputs the SAS Dataset table and creates the scrollable Select Group. In the miniature RESMENU example, the subject of the current discussion, there are two such instances:

SELSOURC : in this the user selects a perspective on The Hartford's line of business structure – e.g. Personal Automobile Liability or Standard Commercial Workers' Compensation.

SELSUBLN : In this the user makes use of the selected perspective from SELSOURC and selects sub-lines of business from the listing associated with that perspective.

In Figure 3.7 (below), we show the SCL code used to create the **SOURCES** selection list from the SAS Dataset **RESRVTBL.SOURCES**. **SRCENAME** is a variable on the dataset carrying the sub-line perspective. The first character happens to be used to determine which one in a series of SAS Dataset tables -- whose names have the form **XB20LIX*** -- should be used as an input for later selection of sub-lines in a different screen. **SRCEDESC** is the label for the perspective that we want displayed in the Selection Group. The box will display eight entries at a time, consistent with the "SIZE=" setting.

Figure 3.7
Creating the "Sources" Selection Group from a SAS Dataset

```
/*  Open the SAS Dataset Table  */
   dsid_SOURCE = Open(  'RESRVTBL.SOURCES' , 'i' )
   Call Set( dsid_SOURCE )

/*  Create the Label Above the Select Group and Create the Select Group HTML Tag  */
   rc = Insertc( htmlcode, '<P><B>Select Source Table (i.e. Division):</B><BR>', -1)
   rc = Insertc( htmlcode, '<SELECT Name="SrcList" Size="8" >',                  -1)

/*  Load up the Select List from the SAS Dataset Table – create Options Tags */
   Do While (Fetch(dsid_SOURCE) = 0)
         FirstChar = substr(SRCENAME,1)
         TABLE     = 'XB20LIX'||FirstChar;
         rc = Insertc( htmlcode, '<OPTION Value=" ' | | Trim( TABLE ) | | ' " >' | |
                  Trim( SCRCDESC ) | |'</OPTION>', -1)
   End;

/*  Terminate the Select Group and Close the File  */
   rc= Insertc(htmlcode, '   </SELECT></P>');
   dsic_SOURCE = Close( dsid_SOURCE )
```

3.9 TO USE, OR NOT TO USE, JAVASCRIPT:

An issue encountered in fleshing out the current RESMENU sample application was whether to use embedded JavaScript code within a particular screen or whether the function otherwise served by the JavaScript could instead be done by the SCL code between screens as the next screen was built. If a requirement was to have "within-screen" linking of elements, the only solution path would be use JavaScript.

Here is an example. The module FlexTriX02-SelSourc – the construction of whose scrollable Select Group we see in Figure 3.7 – contains a text box displaying the filename of the selected table. Take a look at the screen print shown in Appendix 1.0. The selected file name is on the "options" as a parameter, but the only way to actually copy the file name to the display text box as the user selects an option is to use JavaScript. And in this case, the HTML module generated by the SCL code does have the small amount of JavaScript necessary to perform that data transfer. This script is below:

```
function FillSelectedSource(form) {
   var result = ""
   for (var i = 0; i < form.SrcList.length; i++) {
         if (form.SrcList.options[i].selected) {
                  result = form.SrcList.options[i].value
                  }
         }
   form.save_source.value = result
   //  alert("You have selected:" + result)
   }
```

Comparing this code with the SCL Code in Figure 3.7, you can see that variable "**SrceList**" is in fact the name given to the Select Group – it is the "object's" name. The Javascript function is triggered by "**onClick**" action in the Form tag. It simply scans the Select Group for the unique entry selected. It then copies this selected filename to a text box that displays the filename. The name of this text box is the "**SAVE_**" macro variable, **SAVE_SOURCE**, that will carry the value to the SCL program that builds the screen from which the user will select sub-lines. The "**options**" array in the JavaScript code is an overlay of the Select Group options. If you want to see what the screen looks like, turn to the second page in Appendix 1.

One final observation about the JavaScript routine: You will notice the disabled "Alert " statement, which is shown commented out. While the JavaScript was being debugged in FrontPage, the Alert was enabled, and it generated a check-point display as the JavaScript was executed.

In other instances, what might be handled as a within-screen real-time action can actually be deferred to occur between functions. This was the direction taken with what we call "Cross Editing". You may recall we mentioned that certain choice group

selections in SCENARIO might be inconsistent based on other selections made. The purpose of the cross-editing is to highlight the problem for the user and to prevent it or to possibly correct it. "Correcting" really means replacing the erroneous parameter value combination with a combination that does not have the error.

In SAS/AF RESMENU, the problem was handled on the spot within the Program entry as the user made his/her selections, and bad choices just did not occur, period. In the present web implementation, the decision was made not to go this route – because it would have meant some considerable JavaScript. Instead the editing is done in SCL. Specifically, the SCL regenerating the **MAIN** screen performs the cross-edits and makes the corrections of bad choices back to acceptable defaults. If errors are detected in this process, the HTML for **MAIN** shows text (in red) indicating which errors occurred. The user can re-enter the **SCENARIO** screen and make another set of selections and go back. Assuming these are OK the red-coloured error messages disappear. If the user instead enters one of the other screens and comes back the error messages also disappear.

Thus the second situation could have been handled in one of two ways: a JavaScript routine embedded in the generated HTML; or in the SCL generating the transition to the next screen. In this case, the SCL solution path was taken, for the very simple reason that it was a lot easier to code.

One other thing we should note. As should be clear, the embedded JavaScript is generated as part of the HTML. For readability in the SCL programs, JavaScript code is colour coded differently from the HTML – see section 3.6. This way it shows up in an manner easily found in the SCL module.

3.10 DEVELOPMENT ENVIRONMENT

In section 2.0, we touched on this point, but the issue is significant enough to revisit in a bit more detail here. The development and testing environment is actually a triad: desktop PC/SAS running under Windows; Microsoft FrontPage; and a web server running on the desktop personal computer. Depending upon the computer, the developmental web server is either Microsoft Internet Information Server (IIS) or Apache, and we are using the latter. Regardless of the web server used, the development computer is referenced as LOCALHOST on the browser.

The SCL catalog entries are developed the interactive SAS/AF BUILD procedure environment, on the development computer, which clearly requires a license for SAS/AF. In this BUILD development environment, the catalog entries are executed under TESTAF. All SCL entries can be tested using TESTAF, except those with SUBMIT CONTINUE blocks running SAS code. To fully test entries with SUBMIT CONTINUE blocks, one must execute the entry with a Display Manager "AF " command.

A switch (an SCL variable called **TESTAF**) is built into all of the application's SCL catalog program entries. **TESTAF** is set via SYMGET and macro variable **save_TESTAF** alluded to earlier. When the SCL entries are tested in on the desktop in either TESTAF mode or by using an AF command in the Display Manager, they will pick up a set value of "1" from the macro variable **save_TESTAF**, since the AUTOEXEC.SAS sets it thusly. In this case, the generated HTML is directed to an HTML file rather than to the user's web browser. This redirection causes generated HTML to be available as a file, which can be debugged using an HTML editor such as Microsoft FrontPage or Macromedia DreamWeaver. Reserving Automation Support happens to use FrontPage, since this program was available and convenient. An important use of FrontPage was to diagnose problems any embedded JavaScript. (With FrontPage you can set the scripting language sensitivity to either JavaScript or Visual Basic.) To test out a screen, we simply open the generated HTML file and check what is happening in the "Preview" window. Once we resolved any problems, the SCL program that creates the HTML is revised so as .to produce the corrected JavaScript.

The final piece of the triad is LOCALHOST, which enables the developer to simulate running the application as if it were posted to the production web server. For the Reserving Application, the web server is Apache with the mid-tier OpenVMS Alpha as the SAS/IntrNet Compute Server. This server contains both the data and the programs used by the Application Dispatcher.

The knowledge requirements for developers are familiarity with SCL and HTML. As we have seen, some knowledge of JavaScript is helpful. In the case of RESMENU, the application employs a modest amount of JavaScript to handle unavoidable situations where real-time response is needed to the user's interaction with a screen.

3.11 RUN-TIME DEBUGGING

As part of the model construction, the SCL programs make liberal use of "put" statements to indicate what is being worked on and they are used to dump out key values, such as any "SAVE_" macro variable values captured with SYMGET. In normal running, none of this output appears on the HTML. This is because, one of the system variables, **_debug**, is normally set to 0 (zero). Zero disables all of the special Application Broker debugging features. The variable **_debug** actually represents the sum of a series of binary toggle switches. When a switch is set, additional information is printed to the browser. For example, the _debug value of 131 represents the setting of three switches, 1 + 2 + 128. Switch 1 causes the Application Broker to print the values of all the system variables (name-value pairs) to the web browser. Switch 2 prints the Application Broker version number and elapsed run time. Switch 128 prints the SAS Log, along with the values generated by PUT statement embedded in the program. .

To expedite such debugging, the real version of **MAIN** actually has a choice group in it where the debug feature can be turned on, and this choice group shows up in the screen for **MAIN** in Appendix 1.0. Because of this, the code of the SCL module does not have to be touched to get a run displaying information hopefully usable in tracking down a run time errors.

CONCLUSIONS – THE "SCL APPROACH" TO WEB APPLICATIONS

For many readers, this may seem to be an odd-duck of a way to set up web applications. Without a doubt, there sure is a lack of "comin' thing" glitz about it. Indeed, as we have seen, in order to use the approach in a way that does not bury one in code, one must use planning and common sense. No "drag 'n drop" here. This is programming!!

Granted: vendor directions and emphasis are moving to the previously mentioned "comin' thing" world of object-oriented application software solutions. But for many of us, these approaches are not necessarily a panacea, even after we recover and move past the culture shock of the development environment. Clearly "Web application" does not in and of itself imply object-orientation. Object orientation is not a slam-dunk, for a web application uncouples viewer layer from model / controller. The SCL approach here essentially runs in batch and goes a long way to re-establish that kind of connection.

The approach is clearly not a solution for all parties nor is it really, in the case of RESMENU, a solution path for everything RESMENU currently does in the SAS AF environment. But it is a good solution path for its reporting functions, and as other solution paths are used elsewhere for other things, the fact that the solution path is client light bodes well for integrating this effort into a wider and larger one later.

REFERENCES

Davis, Michael (1998), "***SCL for the Rest of Us: Nonvisual Uses of Screen Control Language***", published in the Proceedings of the Twenty-Third Annual SAS Users Group International Conference, 23, 193-202. This paper covers a lot of applications of SCL beyond the traditionally seen use of SCL as the code that works underneath SAS/AF.

Horwitz, Lisa Ann (1998), "***Harnessing the Power of SCL Lists***" published in the Proceedings of the Twenty-Third Annual SAS Users Group International Conference, 23, 48-56. This is a great introduction to SCL lists, showing how to manipulate them, and the paper discusses a number of things the SCL lists are useful for.

Leighton, Ralph, "***Housekeeping Revisited: Managing the Application Environment***" published the Posters section of the Proceedings of the 2000 Northeast SAS User Group Conference, 2000, 548-557. ***Housekeeping Revisited***, discusses RESMENU, its philosophy, and its features. It discusses, among other things, the collection of techniques embedded in the system that make RESMENU very much self-documenting.

ACKNOWLEDGMENTS

SAS, SAS/AF, SAS/IntrNet, AppDev, and webAF are trademarks of the SAS Institute, Inc. of Cary, North Carolina.

Ralph wishes to thank Rob Russell who worked with him as co-developer of the RESMENU web Flex Triangle application. In addition a special thanks is due to the System Manager in Corporate Actuarial, Dennis Rihm, who provided tremendous technical assistance. In setting things up on the PC's and on the mid-tier, Dennis's help invaluable, and he was always available to give guidance to us and to work with the SAS Institute when problems arose involving SAS software.

Please note that the code examples supplied in this paper are designed only to illustrate the concepts being discussed and may need to be modified to work in other applications. The authors of this paper do not support modified code.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. You may contact the authors at
:

Michael L. Davis
Address: Bassett Consulting Services, Inc.
10 Pleasant Drive
North Haven CT 06473-3712
Phone: 203-562-0640 Fax: 203-498-1414
Email: michael@bassettconsulting.com
Web: http://www.bassettconsulting.com

Ralph W. Leighton
Address The Hartford Financial Services Group
c/o Corporate Actuarial Department
1 Hartford Plaza HO-1-02
Hartford CT 06115
Phone: 860-547-3014 Fax: 860-547-3606
Email: rleighton@thehartford.com (business)
rleighton@cox.net (home)

Appendices

Appendix 1 : RESMENU Flex Triangle Screens

MAIN : This is the screen that the user sees on entering the application. It activates screens SCENARIO, SELSOURC and SELSUBLN, where the user actually selects parameter values for the report. There is a submit button on the screen for each of these parameter setting routines. The fourth submit button submits the report for execution. One should also note the choice group in the upper part of the screen where the Debug option can be set (see section 3.11). A small JavaScript routine duplicates the selected value as a hidden variable on each of the HTML forms corresponding to the three parameter setting screens and the report generation.

SCENARIO : Here the user sets certain Loss Triangle Report options using choice groups. The choice groups are presented as HTML Radio Button clusters. The blocking of the choice groups is effected using HTML Tables. As noted in the text, there are some combinations of choices – such as "Division 98 Losses" and "Excess of Loss Scenario" – that do not happen to make sense in the business context of the application. Editing these situations is a function of the regeneration of the screen for MAIN when the user elects to return there by hitting the submit button.

SELSOURC : Here the user selects the line of business perspective, using the Scrollable Select Group generated from a SAS Dataset Table. The name of the table file containing the selected perspective appears in the Text Box above the Selection list and is returned as a parameter. Making this value appear consistent with user selection is a carried out using the JavaScript routine shown in section 3.9 of the paper. The submit button on this screen returns the user to MAIN.

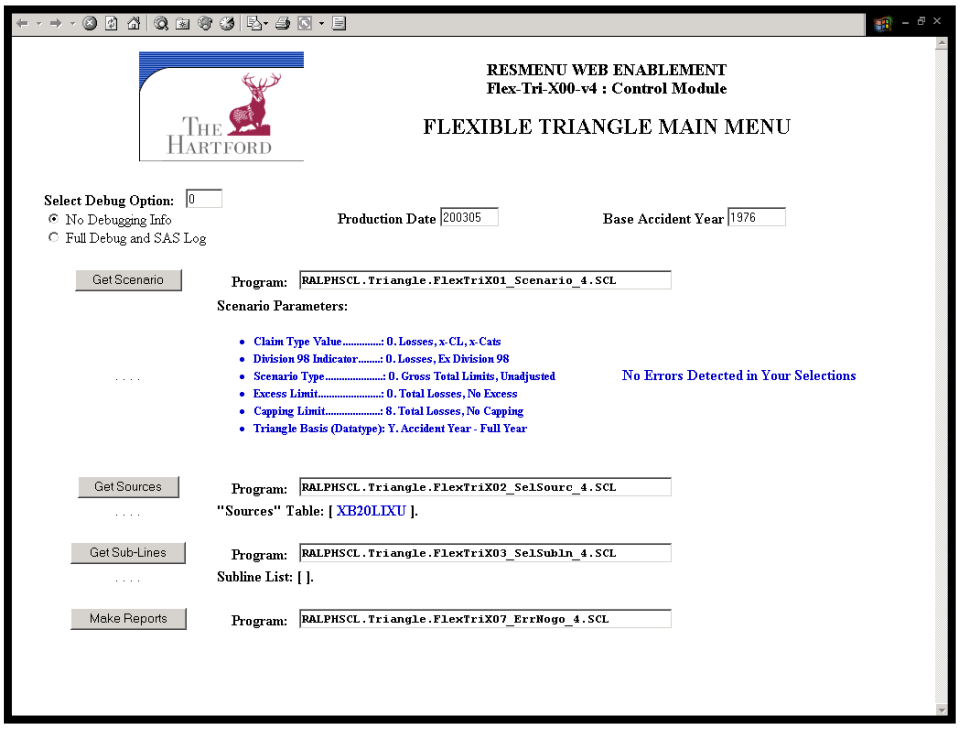
SELSUBLN : Here the user selects one or more sub-lines of business from the perspective selected in SELSOURC. The selection list is in the form of an HTML Scrollable Select Group and its source is a SAS Dataset table. In this screen, the user can select one or more sub-lined, and his/her selections populate the text box above the selection group. The selection process might have been set up as simply allowing multiple selection on the Scrollable Select Group itself. But we decided to try to perpetuate how the SAS/AF list worked. There, when a user clicked on a sub-line, he/she would see it appear in the list, appended to what was already there. If one hit on the same item more than once, the system would not create a second instance in the list. And the user could re-enter the selection routine and append more to the list already constructed. This somewhat complex set of requirements resulted in a non-trivial piece of Java Script in this screen to populate the text box and prevent duplicates. If a user wants to delete a select previously made, he/she can do that in the text box, which is editable. Like SELSOURC, the Submit button returns the user to MAIN.

Appendix 2 : Sample RESMENU Loss "Triangle" Report

Examples of Accident Year by Development Year Incurred Loss and Paid Loss Triangles are shown. The first of these examples originally appeared in the "Housekeeping Revisited" paper by author Ralph Leighton, cited in the References.

APPENDIX 1 : THE RESMENU FLEXIBLE TRIANGLE SCREENS

(Page 1 of 2)



MAIN

This is the main screen of the Flex Triangles application. This is where the user enters the application

The screen is generated by SCL Code.

Note the Submit Buttons. The first three bring up screens from which the user can select parameters governing the type of report to be delivered.

The fourth Submit Button actually submits the report itself.

The first Submit button brings the user into the SCENARIO screen shown below. On return, the SCL code recreating MAIN notes the parameter choices and displays an error notation. You can see the current values and also the fact that there are no errors in the users selections.

In addition, the choice of "source" and the sub-line selections made based on that choice are also shown near the submit buttons for the Screen.

Note that the programs to be executed are indicated in the text boxes next to the respective submit buttons for the functions. Since this is a prototype, having the information there helps in debugging problems.

SCENARIO

These are options in six categories and there are dependencies. Here are some of them

- For any of the scenario selections A thru C and 1 thru 3, excess and capped limit selections are meaningless.
- Excess limits only apply if you select scenario options 4. or 6.
- If you select a Division 98 indicator value other than 0, only scenarios A, B and 0 thru 4.are valid.

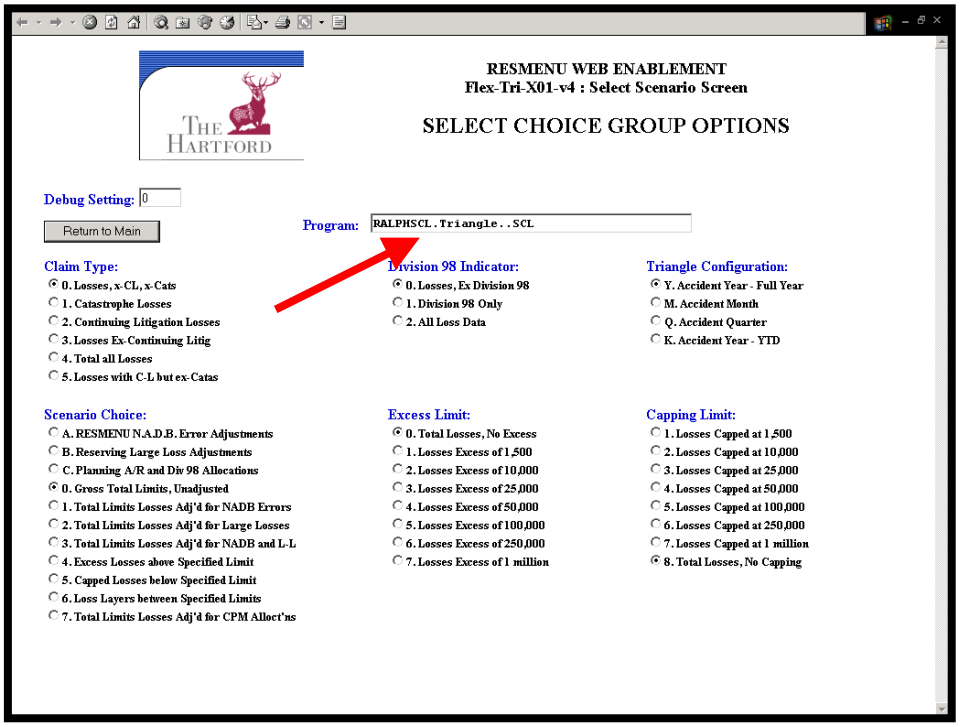
These relationships are checked in MAIN above when it is regenerated.

The radio button clusters are generated from stored SCL Lists.

Hitting the Submit button takes the user backs to main. You will note an error here: the program name is actually missing. It should be

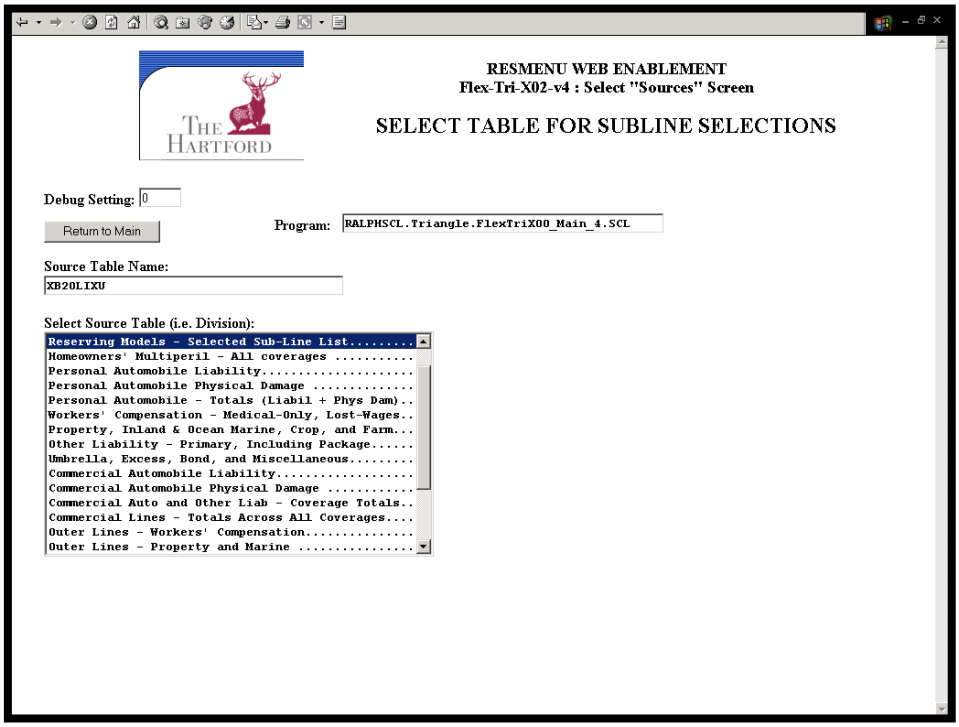
FlexTriX00_MAIN_4

for MAIN. This is why we display the program name in the screen.



APPENDIX 1 : THE RESMENU FLEXIBLE TRIANGLE SCREENS

(Page 2 of 2)



SELSOURC

The line structure is large – there are some 6,120 sub-lines from which to choose.

This enables the user to focus in on a perspective or subset of the company's business from which to choose sub-lines.

What the user selects here – a single choice – is used as the table in the selection of sub-lines. See the screen SELSUBLN below.

What you see here is what the code covered in the paper generated as a scrollable selection list.

As the user selects an option, the table name is filled in on the text box. This is done using the JavaScript, also covered in Section 3.9 of the paper.

Note that the Program name for MAIN appears correctly here.

SELSUBLN

The user employs this to select the various sub-lines.

As the user makes the selections, the entries appear in the text box about. That blank separated list is a parameter passed to the report program.

To undo an entry selected, the user need only delete the entry from the list.

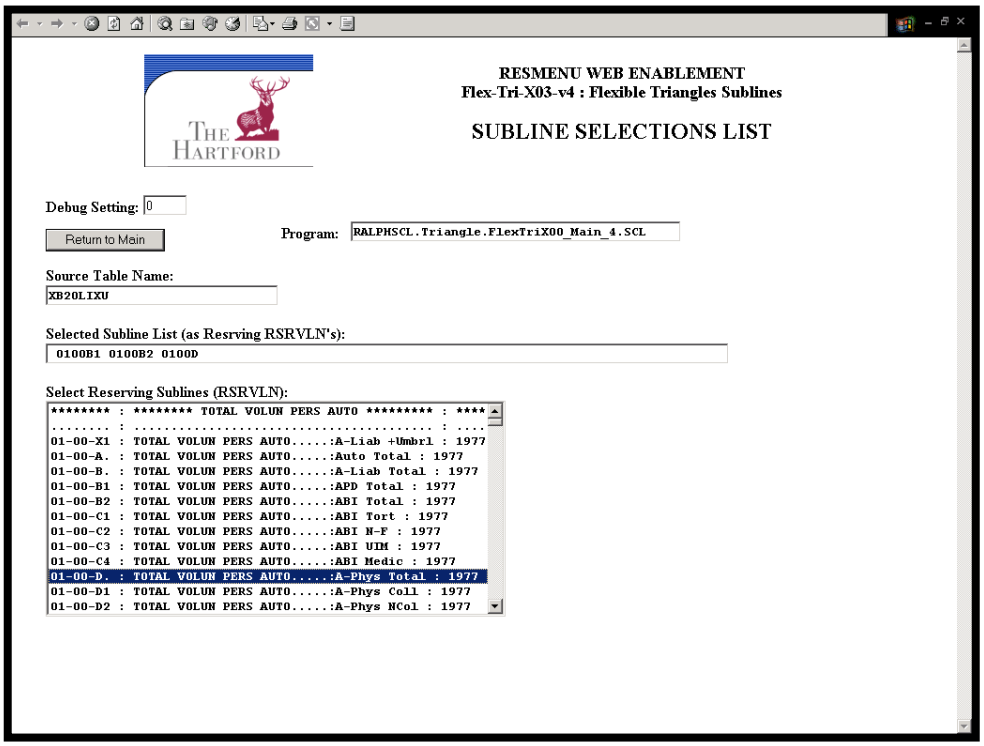
JavaScript Code behind the screen fills the list and avoids "separator" entries in the table that have asterisks. It also does not enter duplicates.

The list selected will be also displayed in MAIN.

The user can re-enter this screen and make more selections

The source table used is the one selected in screen SELSOURC above

In this screen the program name for MAIN appears correctly.



APPENDIX 2 : SAMPLE SIMPLIFIED REPORTS
FLEXIBLE TRIANGLE DISPLAYS FROM THE APPLICATION

75-21-K3 : Oil Storage Tank Multi-Peril
INCURRED LOSS TRIANGLE - as of 12-31-1998
<< Dollars in Millions >>

Devel Year	Accident Year 1989	1990	1991	1992	1993	1994	1995	1996	1997	1998
1	98.5	115.0	117.5	111.9	110.5	105.7	117.8	134.6	130.7	158.6
2	136.4	160.3	168.7	161.9	158.5	147.8	161.4	183.7	182.1	
3	153.0	181.4	190.2	188.9	178.6	170.7	183.7	209.8		
4	158.8	193.8	199.9	197.8	186.5	182.5	190.5			
5	162.2	195.8	204.0	199.7	188.6	180.7				
6	162.7	196.7	204.6	199.1	188.1					
7	163.8	196.6	205.1	199.3						
8	165.0	197.5	204.6							
9	165.1	197.3								
10	164.9									

AN INCURRED LOSS DEVELOPMENT REPORT

75-21-K3 : Oil Storage Tank Multi-Peril
PAID INDEMNITY LOSS TRIANGLE - as of 12-31-1998
<< Dollars in Millions >>

Devel Year	Accident Year 1989	1990	1991	1992	1993	1994	1995	1996	1997	1998
1	42.8	51.8	53.0	54.2	49.6	45.1	51.6	60.5	64.3	73.3
2	86.9	104.3	107.0	104.4	97.5	90.3	103.8	119.0	118.1	
3	119.0	141.0	142.3	141.9	136.1	130.6	143.8	161.8		
4	139.5	168.5	173.6	169.7	160.4	153.7	167.0			
5	150.0	181.8	190.4	186.3	176.4	165.4				
6	158.1	187.4	196.6	194.2	183.2					
7	160.6	193.3	200.2	197.4						
8	163.6	196.3	202.7							
9	164.5	196.5								
10	164.8									

A PAID LOSS DEVELOPMENT REPORT