Paper 235-25

Using SAS/IntrNet™ Software to Support Distributed Data Entry

Kathryn M. Trinkaus, Washington University School of Medicine, St. Louis, Mo.
Paul A. Thompson, Washington University School of Medicine, St. Louis, Mo.
J. Philip Miller, Washington University School of Medicine, St. Louis, Mo.

ABSTRACT

The advent of SAS/IntrNet™ has had significant impact on use of the SAS® System to support management of multi-center clinical trials. Using a combination of static and dynamically-generated HTML forms, JavaScript for data validation and user support, and the SAS System for form generation, data transfer and data management, a great flexibility can be obtained for data entry, output dissemination and communication with remote sites. Web-based data entry provides rapid, accurate and secure transfer of information, facilitates communication with dispersed clinical centers, and allows significant savings in equipment, training and support costs. Provided that the challenges of development time, fast, reliable access and secure data handling are surmounted, the capabilities of SAS/IntrNet™ are expected to become essential tools for management of large-scale, multi-center clinical trials.

INTRODUCTION

The Division of Biostatistics at Washington University School of Medicine acts as the data coordinating center (DCC) for numerous multi-center trials, including the Family Heart Study, HyperGen and HERITAGE. We anticipate initiating an additional six or more similar projects during the coming year, as well as serving the needs of multiple users involved with on-site trials and data registries. A role of a DCC is to establish a data entry process, install and maintain hardware and software, create data entry screens, and train and support data entry personnel in their use. Once data has been entered, it must be transmitted to the coordinating center, cleaned and verified, compiled into master data sets and summarized in reports. Some reports are scheduled at specific times for project monitoring and interim analyses. Others are generated on demand. To keep all aspects of this complex process running smoothly requires frequent, rapid and reliable communication between the center and dispersed clinical facilities.

While the extensive capabilities of SAS, especially SAS/AF, have provided an excellent basis for these activities in the past, the increasing size, clinical complexity and geographic dispersion of current multicenter trials require that more flexible and faster methods be used. At the same time, there has been no change in the basic need for accurate, consistently standardized, impeccably clean data presented in timely reports.

Web-based forms supported by SAS/IntrNet™ have become an essential tool in accomplishing these tasks. This has several real advantages. Users at a distance operate as thin clients, requiring only a PC and an up-to-date browser. An increasing number of users have knowledge of the Internet, reducing training costs. Error checking and data validation can be built into the forms using JavaScript, ensuring the quality of first-pass data and facilitating later data cleaning. Use of password protection, encryption and secure servers maintains the level of confidentiality needed when handling potentially sensitive medical information. SAS/IntrNet™ provides a high degree of accuracy in data capture and transfer, avoiding the uncertainties of special purpose CGI scripts. Meta-data are readily incorporated, facilitating data warehousing. All phases of the code are relatively easy to modify as trials undergo the reassessments and modifications characteristic of long-term studies. Data is transferred to the DCC rapidly and essentially without cost. Automated report generation and some special purpose data summarization can be carried out by the user and the results communicated immediately, removing the need for time consuming requests, special report printing and faxing or mailing of results. Best of all, much existing SAS code can be incorporated into this framework, so specially designed and trusted procedures can be used with minimal modification.

COMPONENTS OF THE DATA ENTRY SYSTEM

There are three components of the data entry process: 1) HTML code which creates web pages for data entry; 2) embedded JavaScript code for content validation and field formatting; and 3) SAS programs which receive the data, store it in SAS data sets and return the next data entry screen.

HTML: CREATING THE WEB FORMS

At the core of the data entry process, there are a series of screens that function as the project's data entry instruments. When setting up a project, these screens should drive the design of the data entry process, as they represent the information to be obtained. The most important decision lies in selection of data to be gathered. A well-designed page obtains all information about a particular aspect of data, while constraining itself to that domain and no others. Elements of page design such as inclusion of appropriate ranges, placement of items to facilitate entry, and other visual cues also improve data quality. Pages are often designed to physically resemble the paper forms used in the data collection. Web pages have some special characteristics that differ from paper forms, however, including a need to design them with related data in related locations. Designers should remember that browsers interpret web pages, and some of this is not under the control of the designer. In particular, pages may be used in a smaller configuration than they are designed in, forcing certain elements of the page to different locations.

Entry into the data entry system usually starts with a simple front page. On this page, the user is asked to enter a username and password and choose a task to be performed. For a given domain of data, certain standard operations are always needed, including options to: 1) add new observations; 2) edit existing data; 3) delete data; 4) list data; and 5) return to previous pages. Editing data and deleting cases are usually done in a two-step process, in which we 1) locate the observation and present existing values; and 2) complete the edit or deletion. These are done in two steps to enable the data entry person to verify the change.

These choices are usually offered on the main entry form as it is presented in “neutral position” at the first presentation (no values filled in). When the front page has been completed and submitted, the information is processed by a SAS program and a web form for the selected task is returned to the user. The complexity of the forms varies with the task selected. Adding a

new subject to the master list is carried out with a few data fields on a single form, on submission of which the user is returned to the front page to select another task. Entering data for a subject starts with the first data entry form, and can involve a complex HTML document.

A simple front page looks like this:

```
<HTML><HEAD>
<TITLE>Sample Project</TITLE>
<SCRIPT LANGUAGE="JavaScript">
</SCRIPT></HEAD><BODY >
<H1 ALIGN=CENTER>Sample Project</H1>
<H2>Select an activity.</H2>
<FORM ACTION="/cgi-bin/broker" METHOD="POST"
      NAME="CHOOSE"><P>
What would you like to do?
<TABLE ALIGN=CENTER>
<TR><TD><INPUT TYPE="radio" NAME="task"
         VALUE="0"></TD>
<TD>Add new study participant</TD></TR>
<TR><TD><INPUT TYPE="radio" NAME="task"
         VALUE="1"></TD>
<TD>Enter data on study participant</TD></TR>
<TR><TD><INPUT TYPE="radio" NAME="task"
         VALUE="2"></TD>
<TD>Edit data for a study participant</TD></TR>
<TR><TD><INPUT TYPE="radio" NAME="task"
         VALUE="3"></TD>
<TD>View reports</TD></TR></TABLE>
<INPUT TYPE="hidden" NAME="_PROGRAM"
       VALUE="~path.selectActivity.sas">
<INPUT TYPE="hidden" NAME="_SERVICE"
       VALUE="serviceName">
<INPUT TYPE="hidden" NAME="_DEBUG"
       VALUE="debuggingLevel">
<INPUT TYPE="submit" VALUE="Send Selection">
<INPUT TYPE="reset" VALUE="Clear Form">
</FORM></BODY></HTML>
```

Figure 1: HTML for the front page

In this page, we present 4 choices for action, and obtain a single macro variable (task=0,1,2,3) from the HTML form, which will lead to our next action. Returning a page to the user can be accomplished with a simple macro such as the following:

```
***** return page requested by user;

%macro sendpg;
%if (&task=0) %then %let f=enterNew;
   %else %if (&task=1) %then %let f=enterData;
   %else %if (&task=2) %then %let f=editData;
   %else %if (&task=3) %then %let f=viewReport;
%if (&task=0) %then %let h2=Enter new subject;
   %else %if (&task=1) %then %let h2=Enter data;
   %else %if (&task=2) %then %let h2=Edit data;
   %else %if (&task=3) %then %let h2=Gimme data;
%multprnt(flelist=f/j/k);
%mend sendpg;

%sendpg;
```

Figure 2: SAS program to process HTML results

This program uses a macro program multprnt to "publish" the next HTML page. This macro program is shown here:

```
%macro multprnt(flelist=);
…break flelist into separate items using %scan
…use fileref to point to each: f1, f2, f3…
data _null_;
file _webout;retain filex 0;
%do i = 1 %to &flcnt;
if (filex = &i-1) then do;
infile f1 lrecl=300 pad end=eo&i;
end;
if (eo&i) then filex=&i;
%end;
if _n_ = 1 then do; file _webout;
put 'Content-type: text/html';
put;
end;
input line $char300.;
line=trim(resolve(line));
put @1 line;
run;
%mend multprnt;
```

Figure 3: Macro multprnt-publishes web pages

In our approach to "publishing" web pages, little HTML code is written in SAS. Rather, macro program multprnt (shown above): 1) reads lines in as many files as are needed; 2) uses the RESOLVE SAS macro function to replace macro references with macro values; and 3) puts the resulting line out to the _webout dynamic page fileref. In this approach, macro references in the stored HTML code allow pages to be easily tailored WITHOUT EVER ACTUALLY LOOKING FOR STRINGS. NO SUBSTR! NO INDEX!

Using stored HTML code reduces the need for PUT statements as:

```
PUT 'code code "quoted code" code code';
```

Such constructions can be very difficult to write correctly. In particular, SAS requires quotes to surround strings in PUT statements. HTML requires additional quotes for other purposes. These two requirements can conflict. It is thus best to avoid the problem entirely, by not writing HTML in SAS, but storing pre-written code in HTML modules. In our shop, HTML which is written by SAS is often written by SAS macro programs, which enable the tedious details of HTML tag construction to be done by the macro program.

Modules of HTML code allow a flexible approach to the page composition task. A dynamic page can be composed on the fly, by queuing up HTML page modules in the appropriate order in the multprnt macro program. Many modules are sometimes required to compose a page. A disciplined use of files, with a well-designed plan for module naming, can result in an orderly, easy-to-maintain application. Remember, NO SUBSTR!!

Here is an example of an HTML file which would be sent back:

```
<HTML><HEAD><TITLE>Sample Project</TITLE>
</HEAD><BODY BGCOLOR=#C6EFF7>
<H1 ALIGN=CENTER>Sample Project</H1>
<H2>&h2 </H2>                                          A
<FORM ACTION="cgi-bin/broker" METHOD="POST"
      NAME="ADDNEW">
<P><TABLE><TR>
<TD>Study ID number:</TD>
<TD><INPUT TYPE="text" NAME="id1e"SIZE="4"></TD>
</TR>
<TR><TD>Relationship to subject:</TD>
<TD><INPUT TYPE="text"NAME="relation" SIZE="25"
     VALUE="&xrelation"> </TD></TR><TR><P></TR>B
<TR><TD>Vital Status:</TD>
    <TD><INPUT TYPE="radio" NAME="vs"
         VALUE="a">Alive</TD></TR>
<TR><TD><INPUT TYPE="radio" NAME="vs"
         VALUE="dead">Dead</TD></TR>
<TR><TD><INPUT TYPE="radio" NAME="vs"
         VALUE="unknown">Unknown</TD></TR>
</TABLE> <P>
```

```
Note:<INPUT TYPE="text" NAME="notes" SIZE="80">
<INPUT TYPE="hidden" NAME="_PROGRAM"
       VALUE="~path.addNew.sas">
<INPUT TYPE="hidden" NAME="_SERVICE"
       VALUE="serviceName">
<INPUT TYPE="hidden" NAME="_DEBUG"
       VALUE="debuggingLevel"><P>
<INPUT TYPE="submit" VALUE="Enter Data">
<INPUT TYPE="reset" VALUE="Clear Form">
</FORM></BODY></HTML>
```

Figure 4: HTML form with macro references

The HTML example obtains vital statistics about a person. Line A of the entry contains SAS macro reference &h2. The value for macro variable h2 is set in the SAS program previously shown. Use of pre-coded macro references can be used to 1) insert titles into pages; 2) insert pre-check selections to guide progress through pages; and 3) insert return values for editing to the data entry person. Use of the macro reference to return values for editing is shown in Line B. If xrelation is set to no value (%let xrelation=;), the value is blank; if xrelation is set to the current value (%let xrelation = father;), the value of the current entry is returned. For more general situations, the value of xrelation can be set by the current value of the variable relation, which is stored in the file being edited.

The use of the embedded macro reference is the key to efficient presentation of data for the editing process. The full editing process involves 1) identifying the observation to be edited; 2) converting all current values in macro variable values; 3) publishing a web page which incorporates these values into correct locations; 4) recovering the corrected values after the entry persons submits the web form; 5) converting the macro variable values into datatable variable values; and updating these values into the master datatable. With a properly prepared HTML page (modularly composed, of course), the actual presentation of such data are no more difficult than the presentation of a blank form.

Critical information (such as a subject identification number) may be passed as a hidden variable from page to page and echoed on each to help the data entry person track the task.

<INPUT TYPE=”HIDDEN” NAME=”_curval” VALUE=”3”>

As the data entry task moves from screen to screen, a series of such HIDDEN tags may be necessary to retain all necessary information. Standard packages of these hidden fields can be placed in HTML modules, and queued up as needed in the printing process. If there is a duplication of the name used on the hidden tag and the name used on an active FORM input object, the first one encountered becomes the default returned to SAS from the web page. Thus, the HIDDEN tag file should be placed at the end of the HTML page.

When the input fields have been completed, verified and submitted, the next screen is returned to the user until all screens have been completed. If the number of screens is large, it is not always feasible to complete data entry for a subject at one time. It is useful to include an option to start data entry at the beginning of any screen in the data entry stream for a given patient, appending or merging these data into the records already stored for that subject. This allows the user to terminate data entry at the end of any one of the intermediate screens and return at a later time to complete data entry for that subject. To edit data, the user specifies a subject ID number and a data field to be changed. The relevant screen is returned with data fields filled in.

On submission of the corrected form, the new information can be included in the database by any of the means provided by a SAS data step (as will be discussed below). In terms of feedback, a variety of methods using SAS techniques can be used to produce reports and output, which can then be published on the web.

JAVASCRIPT: VALIDATING AND FORMATTING

JavaScript combines the power and flexibility of an object-oriented programming language with a relative simplicity of syntax, making it ideal for the repetitive requirements of error checking and formatting without an excessively steep learning curve. While error checking can be carried out by SAS after data is submitted, client-side verification using JavaScript speeds the process by eliminating the communication time with the server and reducing the amount of information to be retyped by the user. The importance of screening impossible values from data entry, using well-designed JavaScript methods, cannot be overemphasized.

JavaScript has proven especially important when used to 1) compare values against standards; 2) test whether a required field has been entered; 3) perform tests based on conditions established by other input fields; and 4) perform tests of passwords and other entry needs using cookie methods. Other very useful aspects of JavaScript include clearing text fields, restoring default values after focusing on a text field, and so forth. The full JavaScript language is quite extensive, and cannot be discussed with any depth here.

Entries are usually validated in one of two ways. First, we may use various event handlers associated with each input object. In this way, JavaScript functions can be directly tied to data entry and specific data values. This has the advantage of doing data validation during data entry. It is often not appropriate when conditional validation is needed. Secondly, we may use a final pass through values after the form is completed and submission is chosen. This is often a good choice, as one data value is dependent upon another for validation.

Validation functions can be embedded using <SCRIPT> tags on the form being checked. Multiple <SCRIPT> tags are allowed. It is generally preferable to place these before the body of the form so that all necessary variables have been declared before input is accepted. Although some functions (*i.e.,* the validation of date formats) are carried out repeatedly, other validation functions are often unique to the particular form. For this reason, initial debugging and later maintenance may be more easily carried out if JavaScript functions are attached to each form.

In this case, a <SCRIPT> tag within the page contains declarations of all variables to be used in validation and an indicator variable which is set equal to 1. A JavaScript function then uses IF..THEN and IF…THEN…ELSE statements to check that all variables are correct. If any are found to be incorrect the indicator is set to zero, otherwise it remains set to 1. After responding to requests for corrections (using alert boxes, prompts, etc.), the user resubmits the form, executing the validation function a second time. This process continues until the indicator function remains equal to 1 and the validation function returns a value of true. At this point, the data are sent to the SAS program referenced by the hidden variable _PROGRAM.

If the number of forms is small and/or the validation procedures are repeated frequently, the JavaScript functions can be stored compactly in one or more text files with the .js extension. The <SCRIPT> tag of each form contains a reference to this file and a list of the functions to be executed with their parameter specifications. Although very efficient in some settings, this format can be difficult to debug, especially if the effect of each change to a function must be assessed for a large number of forms. In addition, more careful documentation is needed with this approach.

An example of storing validation functions separately from the HTML page is as follows:

```
<HTML><HEAD>
<TITLE>Sample Project</TITLE>
<SCRIPT LANGUAGE="JavaScript" src="http://a.js">
</SCRIPT></HEAD>
<SCRIPT language="JavaScript">
function checkForm(form) {
       return (vNum(form.id1e, 4, 1850, 2050));
       }
</SCRIPT>
<BODY BGCOLOR=#C6EFF7>
<H1 ALIGN=CENTER>Sample Project</H1>
<H2>Enter data for a study participant already
in the master list.</H2>
<FORM ACTION="/cgi-bin/broker" METHOD="POST"
NAME="formName">
<P>
<P><TABLE>
<TR><TD>Patients ID number:</TD>
<TD><INPUT TYPE="text" NAME="id1e" SIZE="4"
MAXLENGTH="4">
… additional variables to be entered …
</TABLE>
<INPUT TYPE="hidden" NAME="_PROGRAM"
       VALUE="~path.enterID.sas">
<INPUT TYPE="hidden" NAME="_SERVICE"
       VALUE="serviceName">
<INPUT TYPE="hidden" NAME="_DEBUG"
       VALUE="debuggingLevel">
<P>
<INPUT TYPE="submit" VALUE="Enter Data"
onClick="return checkForm(this.form)">
<INPUT TYPE="reset" VALUE="Clear Form">
</FORM>
</BODY>
</HTML>
```

Figure 5: HTML code using stored JavaScript

The file *a.js* is a plain text file containing only the validation function(s) to be used. In this case, the function vNum uses two general purpose functions called isSize and isInRange. Nesting functions in this fashion reduces the time involved in writing and debugging special purpose functions for each variable to be validated. *a.js* contains code such as the following (note that the code here is presented in a compact form, and would be more nicely formatted in the actual file):

```
var r1="Please enter a ";
var r2=" digit number in range:";

function isSize(data, size) {
if (data.length == size) {return true;}
else {return false;}
}
function isInRange(data, num1, num2) {
var i = parseInt(data)
if ((i >= num1) && (i <= num2)) {return true;}
else {return false;}
}
function vNum(str, size, num1, num2) {
var retv=0
if (isSize(str.value, size)) {retv=1;}
if (isInRange(str.value, num1, num2)) {retv=1;}
else (retv=0};
if (retv) {return true;}
else {alert(r1+size+r2+num1+"-"+num2);
      str.focus();return false;}}
```

Figure 6: JavaScript file a.js

Decisions concerning which fields to validate and with what degree of precision generally are made in consultation with scientific and data entry staff. These decisions vitally affect the nature and quality of data available for later analysis, as well as the process of data entry. Ideally, these decisions are complete before embarking on writing new or compiling existing JavaScript functions to carry out validation. Data entry personnel also are routinely included in discussion of how to return the results of validation. In some cases it is useful to return the result to the user as soon as he or she moves on to the next field, allowing critical fields to be checked at once. To do so with all fields may be perceived as an irritating interruption, in which case validation is more effectively carried out when the form is submitted

SAS PROGRAMS: BUILDING DATA SETS AND DYNAMICALLY GENERATING WEB FORMS USING MACRO PROGRAMS

After the web form is used for data entry and the data have been validated using the JavaScript validation element, the results are processed by a SAS program. Such programs are the heart of the web data entry process. The process of handling the code begins with the SAS cgi script. All values that are found on the screen are converted into macro variables, with names designated by the name fields. These may be converted in SAS values using standard methods for handling macro variables in SAS.

One great problem in the creation of the web data entry approach lies in balancing the competing problems of complexity of code with ease of maintenance. SAS web data entry methods are complex, as they include code in HTML for the screens, code in JavaScript for the validation/verification and code in SAS for the processing. Thus, the complexity of the maintenance problem is very great. This problem has been addressed in our shop using a series of macros that use meta-data macro variables and generalized macro-driven methods to do common tasks.

Meta-data is data about data. In our approach, we have defined macro variables that function as meta-data repositories. These macro variables indicate the name of the master datatable (xmset), variable names (vnm), lengths (vln), formats (fmt), informats (infmt) and other details. The unique statements needed during addition of values to the master datatable can be placed into a macro (code), and used on a common basis during updates. The macros are all indexed using a common letter, the datatable index letter or DIL. Thus, to select a datatable, its variables, formats for printing values and all other such important choices, a single letter, the DIL, is used to key into all of these other things.

Once a system of meta-data macro variables is established, it can be used to establish a series of macro variables that are linked to the datatable variables; that is, this is a macro which stores the names of additional macros. These macro names provide easy and STANDARD approaches for getting data from datatables into macro variables and from macro variable into datatables.

For an example of the use of meta-data macro variables, consider a case in which two small datatables are defined. In the first case, the DIL is “a”, and the data involve patient registration. In the second case, the DIL is b, and the data involve diet information.

User-defined:

```
%let _avnm=age name race hdsz patid;
%let _alng=3 $20 $1 8 8;
%let _afmt=2 $20 $racefmt 5.2 6;
%let _ainfmt=2 $20 $1 8 6;
%let _axmlib=FATLIB;
%let _axmset=SUBJECT;
%let _axpwd=fatboy;
```

```
%let _bvnm=patid prot carb satfat;
%let _blng=8 8 8 8;
%let _bfmt=6 best8 best8 best8;
%let _binfmt=6 8 8 8;
%let _bxmlib=FATLIB;
%let _bxmset=DIET;
%let _bxpwd=eatpeach;
```

System-defined:

```
%let  _amnm= _age _name _race _hdsz _patid;
%let  _amxnm=_xage _xname _xrace _xhdsz _xpatid;
%let _bmnm=  _patid _prot _carb _satfat;
%let _bmxnm= _xpatid _xprot _xcarb _xsatfat;
```

Figure 7: Meta-data macro variables

Using standardized macro programs, these meta-data macro variables can be used to generate master datatables, by indicating the choice of macro datatable using the DIL of a or b. Similar approaches are taken with macro programs which 1) check for the existence of an observation in a datatable; 2) insert an observation in a datatable; 3) remove an observation from a datatable; 4) print values from a datatable or 5) archive datatables during other processes. By using the meta-data macro approach, one can perform common tasks using common architecture and spend the important programming time on uncommon, unique aspects of the data management process.

The process of converting information from a macro variable representation to a datatable variable representation uses Macro Program m2v

```
%macro m2v(ltr=,dset=);
data &dset;
%do i=1 %to &&&ltr.cnt;
%let dv=%scan(&&_&ltr.vnm,&i);
%let mv=%scan(&&_&ltr.vnm,&i);
&dv=symget("&mv");
%end;
run;
%mend m2v;
```

Figure 8: Macro Program m2v: converts macro variables

The actual program in the system includes different specifications for numeric and character variables, and involves the use of informats where needed. However, the basic notion of the use of the meta-data macro variables is shown above. The process requires that the meta-data macro variables be global in scope, to ensure that all variables are fully available for all macro programs.

For addition of the new datatable containing the values of a single page, the system uses macro addobs. Figure 14 presents a greatly simplified version of addobs:

```
%macro addobs(ltr=,byvars=,dset=);
data &&_&ltr.mlib..&&_&ltr.mset;
        update &&_&ltr.mlib..&&_&ltr.mset &dset;
        by &byvars;
run;
%mend addobs;
```

Figure 9: Macro Program addobs—adds observation to datatable

This macro program demonstrates the savings of the system. The actual process of incorporating the new observation into the datatable is the most simple SAS program that could be written.

However, since the meta-data macro variables are used, their names need not be incorporated into the written code, and thus the need to write repetitive code to update yet another master datatable is greatly reduced. In this system, a task done once is done at all cases. Other macro programs exist to delete observations, check for the existence of observations, and do a variety of other tasks. The general tradeoff here is the usual one: a great amount of effort is involved in writing the macros, which enable a somewhat simpler use of the macros.

The final development to date has been the writing of shell macros. These are easily customizable shells of macro control programs. They include the standard set of operations (adding observations, editing observations, deleting observations, listing values, return to previous page) in a standard order, with standard macro references, and using standard values for options. This is a great help, as it places a structure upon a very difficult process. It is not a panacea, as the shell approach does not handle all problems. The general notion of the shell is a powerful one, as the shell can be refined and incrementally enhanced, thus leading to a continuous cycle of improvement.

The process of entry, editing, or deletion of observations involves sequential processing of a series of pages. Following the publication and submission of one page, the "next" page may be a subsequent web form, a report about the data just entered, a choice of "next" pages, depending on the values received, as in the case of task selection on the front page. In many cases, the next page is arbitrary; after a listing of values, the data entry may proceed to do any arbitrary task. During the edit process, the first page involves selection of the case to be edited, while the next page will involve the examination and alteration of the values. While the entry may be stopped at the point of the change of values, it is not reasonable to allow the process to divert to another option before the process is completed. Thus, the progress from screen to screen may need to be guided in some cases.

During the development phase, it may be helpful to debug the SAS program on a standalone basis. This can be done using the standard SAS system, with %LET statements used to create the name-value pairs which will be returned from the web form. The SAS macro variables created by the program can be written out to a file at the end. Once the SAS code is functioning as needed, the %LET statements can be deleted. Once the HTML, JavaScript and SAS code are functioning together, the Application Dispatcher-generated SAS log and messages written by the Application Dispatcher to the browser are the main sources of information concerning errors. Debugging is complicated if each Application Dispatcher session generates a single log file.

The process of returning HTML pages can be somewhat complex. However, careful segmentation of such pages into pieces can enable the process to be done somewhat more easily, and can enable the work in one area to be applied to others. The important aspect of HTML, that it is a mark-up language written in ASCII, enables the more general use of items.

SECURITY AND CONFIDENTIALITY

Managing potentially sensitive research data raises the issue of adequate security to maintain the necessary confidentiality. A variety of system-specific and SAS/IntrNet™ facilities can be used to prevent access by unauthorized persons. A secure web server with HTTPS protocol provides for encryption of information passing back and forth between the users and the server. We also require a username and password before the user can view the front page. The user then supplies a username and password when submitting information from the front page. These are checked, as is the identity of the computer as indicated by the automatic variable _RMTUSER (the IP address

of the computer at the time of the session), at the same time. Data quality is protected by requiring a password for read and write access to all SAS datasets created by the data entry process. Character strings that contain unbalanced quotes or other special character strings may cause problems for SAS. To prevent deliberate or accidental difficulties caused by such strings, the use of the SYMGET function for macro resolution and %SUPERQ function for handling the macro variable without resolution is recommend by SAS Institute, Inc.

The file generating the HTML front page must be accessible to the user's browser in a public HTML directory. In fact, the files generating front pages for several different projects may reside in the same public HTML directory. The remaining HTML and JavaScript files and all SAS programs used by any one project can be stored in their own directory, separate from other project-related files and from files relating to other projects.

DEVELOPMENT TIME

Designing, writing, debugging and maintaining the necessary code is a substantial effort for which sufficient well-trained staff members must be available. The application developer(s) typically spend a substantial amount of time in consultation with project staff. This time is spent in specifying content and appearance of forms, as well as determining which fields to validate and how thoroughly. At all stages of the process, access to debugging information is critical. Our experience suggests that it is helpful to design the web forms to the satisfaction of project staff before attempting to add and debug the JavaScript validation functions. Many HTML editors provide information on errors. The JavaScript console is also a useful source of error messages that refer to specific lines of HTML code. An editor that provides line numbers for the HTML code is essential to convenient use of this feature.

TRAINING AND MAINTENANCE

As familiarity with the Internet grows, training for use of web-based applications becomes easier. Use of the Internet is more and more widely regarded as a basic life skill, in contrast to a powerful and complex programming language, such as SAS, which may prove intimidating and confusing by itself. It is difficult to overstress the importance of consulting with data entry personnel as pages are developed, however. As noted above, much of the online documentation grows out of this process.

ACCESSIBILITY AND STABILITY

Our current web based data entry systems access SAS on the server by means of several constantly running, default SAS sessions, a process which is normally completed in a few seconds. Launching a new SAS session is offered as a slightly slower alternative. As the number of data entry systems and their users increases we anticipate that the number of concurrent SAS sessions will increase.

Stability of SAS/IntrNet™ on the server is also an issue. If new projects are tested using the same SAS sessions accessed by ongoing applications, downtime caused by programming errors adversely impacts projects already underway. All of the data entry personnel rely upon a functioning chain of programmed events to carry out their responsibilities, so any amount of downtime has a significant effect on project schedules. In addition, some processes must be completed within a specified period, such as patient enrollment in a clinical trial during a visit to a healthcare provider. When this constraint is severe, resulting in loss of subjects due to server downtime, hardcopy backups are provided so that the necessary functions can be carried out (e.g., a patient can be enrolled) even if the server is inaccessible for several hours. These data must be entered at a later time in order to be integrated into the project database.

When several projects are under development at one time separate, project-specific log files are needed timely identification of errors in the SAS portion of data entry system building.

CONCLUSION

Web-based data entry can provide rapid, accurate and secure transfer of information. It facilitates communication with dispersed clinical centers in a wide variety of data entry situations, local and remote. It can provide significant savings in equipment, training and support costs. Although establishing such a process can be challenging, the use of SAS/IntrNet™ significantly extends the capability of a DCC to meet the needs of large-scale, multi-center clinical trials.

REFERENCES

There are many sources, online and printed, for learning more about the interface between HTML, JavaScript and SAS/IntrNet™. Interested readers can surf to

http://www.biostat.wustl.edu/

which contains a list of resources that we use in development, under the link labeled “Web resources”. In addition, a fully operational “toy system” is available for examination and testing at that location.

ACKNOWLEDGMENTS

The authors would like to thank Jack Baty, Sarah Littlewood, Derek Morgan, Jim Schauer and Erich Schraer of the Division of Biostatistics, Washington University School of Medicine for helpful suggestions and continuing interest in refining web based data entry solutions. Institutional support for this project is provided by The Alvin J. Siteman Cancer Center and NIH/NIDDK grant #1UO1 DK56961-01.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Kathryn M. Trinkaus
Division of Biostatistics
Washington University School of Medicine
Campus Box 8067, 660 S. Euclid
St. Louis, MO 63110
Work Phone: 314-362-3686, Fax: 314-362-2693
Email: kimt@wubios.wustl.edu
Web:http://www.biostat.wustl.edu/~kimt