Paper 26-26

Walk Our Children to School: an Internet Data Management Application

Carol Martell, UNC Highway Safety Research Center, Chapel Hill, NC

ABSTRACT

This paper describes a complete data management and query system wherein data is collected, maintained and dynamically surfaced on the Web. The application integrates SAS/IntrNet® and SAS/GRAPH® with the SAS® email and ODS facilities. Even though all data is manually screened prior to release, the manual effort is minimal and the people screening and maintaining data are not SAS users. Good, old-fashioned data management experience teamed up with common sense can use the power of SAS to take care of the scene behind Web pages.

INTRODUCTION

The fourth annual National Walk Our Children to School Day occurred October 6, 2000. Children, parents and community leaders walked to school together to promote safety, health, physical activity, and environmental concern. Participation in this event to engender partnerships for change dramatically increased, as in prior years. The goal of enabling synergy among event providers and of quickly surfacing new information was realized with a dynamic Web application.

Having encountered difficulties using other application server products to manage data, our Web development group asked whether SAS could handle the requirements listed below.

Event providers would register online. As plans coalesced and more community leaders and organizations agreed to participate, registrants would update the event descriptions. Media would be alerted to consult the Web site to maximize coverage. All registrants would be prompted to update after the event to describe what actually happened in their community. All Web submissions of event information would be immediately available at www.walktoschool-usa.org pending ***manual*** screening. The project would be a litmus test of integrating Web and SAS developers. There would be two sets of clients needing special care: the event providers, who would register through the Web, and the Web developers, who would manage the data. SAS software provides a variety of components that make it possible to build a SAS application wherein the users remain completely in a Web environment.

Application Dispatcher handles registration data management. Manual data content screening is streamlined using email. Registrants are provided a userid and password for maintaining their records. A SAS/GRAPH drilldown map of the United States is the browser entry point for dynamic data queries, providing immediate access to new data. A dynamic Web page listing all registrants gives Web developers update access, the means to easily supply lost userids and passwords, and a simple way to communicate with individual registrants. A Web form provides a bulk email facility for general announcements to the registrants.

INITIAL REGISTRATION

The SAS Application Dispatcher broker accepts information from the registration form in **Figure 1** and executes a SAS program. The program creates a permanent SAS table containing only that registrant's observation and sends email containing those values to the Web developers for screening. The acknowledgement seen in **Figure 2** is returned to the registrant's browser window.

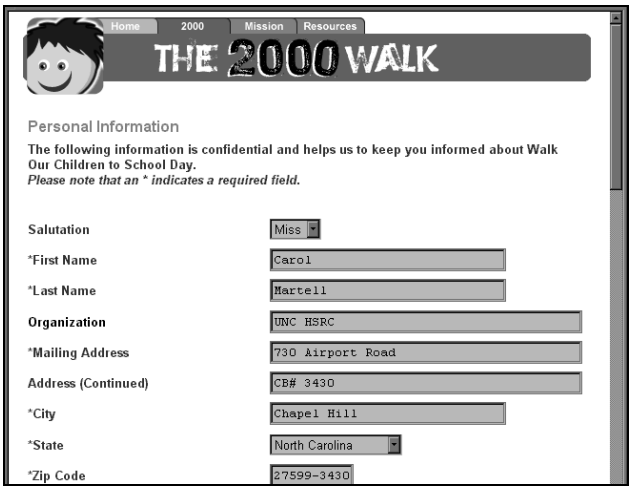

Figure 1

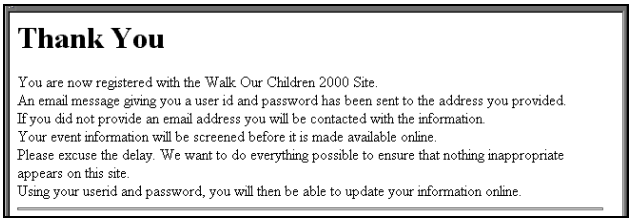

Figure 2

Userid and password variables are created in the program and are used as the SAS table name seen in **Figure 3**.

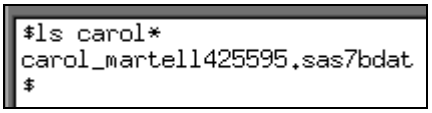

Figure 3

With Application Dispatcher, information typed into each input field in a Web form is passed to a SAS program. The program handling this form data is called *register.sas* and resides in a directory defined to Application Dispatcher as *myjobs*. This information is specified in the html source code as:

```
<input type=hidden name="_program"
value="myjobs.register.sas">
```

To illustrate how data is passed, the html source code for the first name field is:

```
<input type = "text" name = "first_name"
size = "30">
```

The registration form in **Figure 1** has 'Carol' in the blank for first name. The field name, *first_name*, is passed to *register.sas* as the macro variable *&first_name* having the value 'Carol'. The program can assign the macro variable's value to the SAS table variable for first name:

```
f_name=&first_name;
```

To enable manual screening, *register.sas* sends an email message to a project email account. The body of the message contains all the data from the registration form for screening as seen in **Figure 4**.

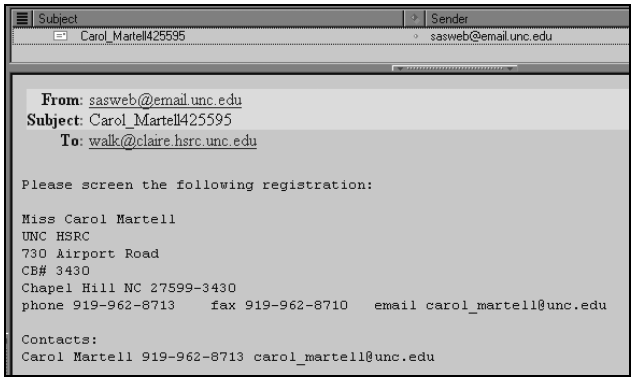

Figure 4

Sending email is accomplished using FILENAME and PUT statements. Following the syntax for our Unix host system, the fileref is:

```
FILENAME m EMAIL
'walk@www.walktoschool-usa.org'
SUBJECT="&userid&pwd";
```

PUT statements generate the message body:

```
FILE m;
PUT
'Please screen the following registration:';
PUT salutation f_name l_name;
```

MANUAL SCREENING

Content screening is carried out using email. A reviewer reads the email message to determine whether to keep or delete the record. The end of each message contains three hypertext links as seen in **Figure 5**. The URL for each link includes parameters equivalent to the fields in a Web form.

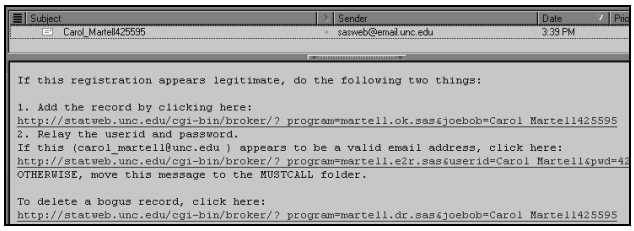

Figure 5

Each link executes a different SAS Application Dispatcher program. One link adds a valid registration to the main registration table. Another link deletes a bogus registration. The third link sends email to the registrant acknowledging registration and conveying the userid and password. These links allow the users screening data to quickly release it for Web queries. **Figure 6** shows part of the email message to the registrant conveying userid and password from our example registration.

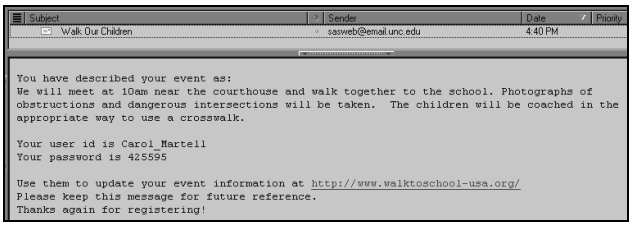

Figure 6

SURFACING DATA

The entry point for data presentation on the Web is an image map of the United States (**Figure 7**). Since the map indicates which states have registered events, it must be recreated when someone registers from a new state. When a record is added, if the state of residence was not in the main table, code is executed to recreate the image map with the new state highlighted.

Figure 7

Image maps are generated using SAS/GRAPH and the Output Delivery System (ODS).

```
ODS LISTING CLOSE;
FILENAME carol 'mypath';
ODS HTML FILE='us2.html' PATH=carol;
GOPTIONS DEVICE=GIF NOBORDER;
PATTERN1 COLOR=CXFFCC00 VALUE=MSOLID ;
PROC GMAP MAP=maps.us all DATA=states;
  ID state;
  CHORO j/
  COUTLINE=black CEMPTY=black
  HTML=st NAME="us" NOLEGEND ;
RUN;
ODS HTML CLOSE;
QUIT;
```

Highlighted states, those with registrants, are drillable. Every link in the map references the same htmSQL page with a state parameter to customize the resulting page. The links are set to the value of the variable named in the *html*= parameter in the *choro* statement. That variable was earlier constructed using SAS PROC SQL.

```
PROC SQL;
  CREATE TABLE states AS
```

```
SELECT
  1 AS j,
  STFIPS(state) AS state,
  'href="citylist.hsql?st='||state||'"'
     AS st
FROM
(SELECT DISTINCT state FROM r.registrants)
ORDER BY state;
```

The North Carolina link, for example, is the following:

```
citylist.hsql?st=NC
```

When clicked, the htmSQL page queries the registration table for all records from the state of North Carolina and return a list of communities with registered events (**Figure 8**).

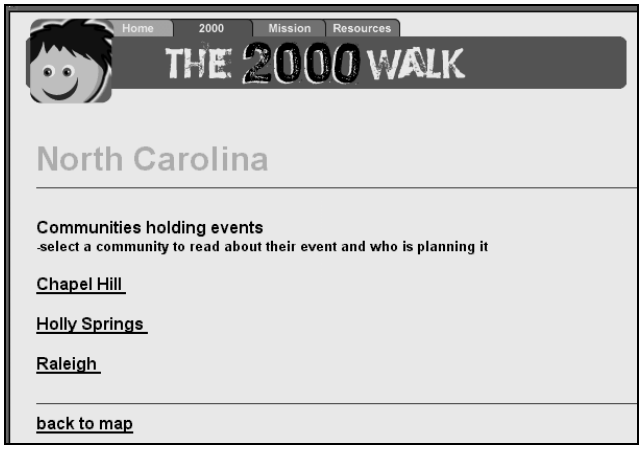

Figure 8

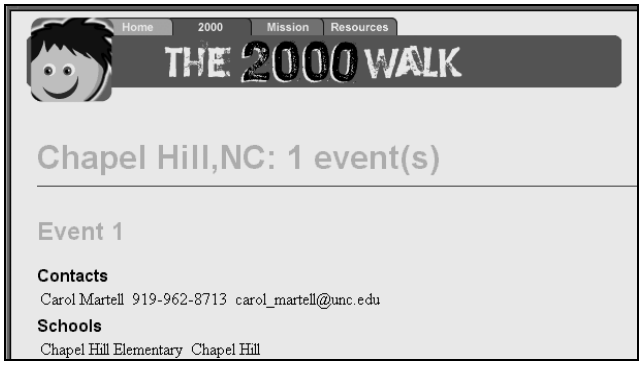

Figure 9

SAS htmSQL resembles other application server packages in that the code for the page contains one or more query sections with corresponding sections to display the query results for the browser window. The browser shows the results but not the query itself. The parameters passed in as well as the columns selected in the {sql} query are treated as macro variables syntactically referenced inside brackets: {&var}.

A SAS SHARE server provides access to the data and is identified in the query tag.

```
{query server="host.myserver"}
```

The query listing communities is complicated by the fact that a single registration can encompass events in up to five schools, which may be in different communities. The following htmSQL code selects a distinct list of communities in a given state.

```
{sql}
select distinct * from
(
 (select city1 as city,
   urlencode(trim(city1)) as cityp
   from wr.registrants where state="{&st}")
 union
 ...
 union
 (select city5 as city,
   urlencode(trim(city5)) as cityp
   from wr.registrants where state="{&st}")
)
{/sql}
```

The {eachrow} section formats the results of the query. Each community returned from the above {sql} section is html-encoded to be a link with appropriate parameters. This enables the browser to drill down to another level of information.

```
{eachrow}
 <p>
 <a href=getcity.hsql?sta={&st}&cit={&cityp}>
     {&city}</a></p>
{/eachrow}
{/query}
```

Clicking on Chapel Hill in Figure 8 drills down to reveal the event used in our illustrations. Getcity.hsql?sta=NC&cit=Chapel+Hill dynamically creates **Figure 9**.

REGISTRANT UPDATES

Registrants may update information using their userid and password in the login Web form seen in **Figure 10**.

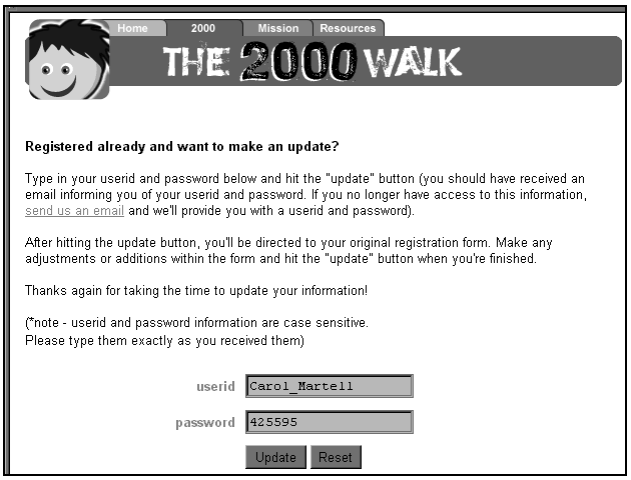

Figure 10

Clicking the update button executes an htmSQL query, which returns what appears to be the original registration form (**Figure 11**) with the current information already typed into the fields.

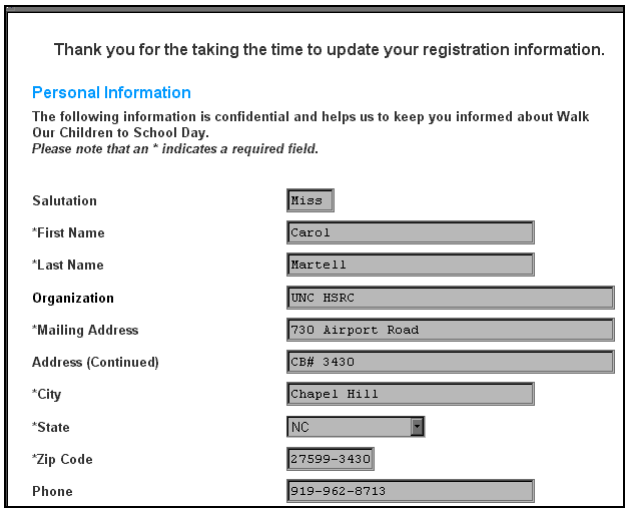

Figure 11

Populating the registration form with data from the registrant table using SAS htmSQL is simple. The html code for the original form is sandwiched into the {eachrow} section. The preceeding {sql} section selects the correct record. Macro variables containing values for that record are inserted into each corresponding input field using the *value* tag parameter.

```
{sql}
  select *
  from wr.registrants
  where userid="{&u}" and pwd="{&p}"
{/sql}

{eachrow}
  …
  <input type="text" name="first_name"
    size="30" value="{&f_name}">
  …
{/eachrow}
```

Processing registrant updates follows a path nearly identical to that for the original registration. Email reflecting the new version of the registration is sent to the reviewer, who again clicks an appropriate link to either accept or reject the changes. If accepted, an update data step replaces the old record.

MAINTENANCE

An htmSQL page (**Figure 12**) provides a single point of access for Web developers to perform several actions. The query lists all registrants, providing links to update, delete or send an email message to the registrant. Each link has parameters to select the specific record. The {eachrow} section from the htmSQL page to list all registrants follows:

```
{eachrow}
  <tr><td>{&st}</td>
    <td>{&ct}</td>
    <td>{&f_name}{&l_name}</td>
    <td><a href=
      "update.hsql?u={&userid}&p={&pwd}">
      <font color="#00FF00">update record
      </font></a></td>
    <td><a href=
      "remove.hsql?u={&userid}&p={&pwd}">
      <font color="#CC0000">delete record
      </font></a></td>
    <td><a href=
      "emailem.hsql?u={&userid}&p={&pwd}">
        <font color="#0000cc">email registrant
        </font></a></td>
  </tr>
{/eachrow}
```

Figure 12

The email link produces a customized Web form. This dynamic form uses SAS Application Dispatcher and the email facility to send a message to a specific registrant. For illustration, the form seen in **Figure 13** is completed and sent. **Figure 14** shows the message received by the registrant.

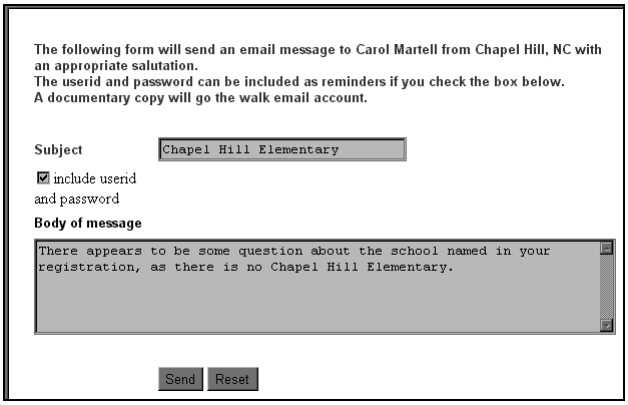

Figure 13

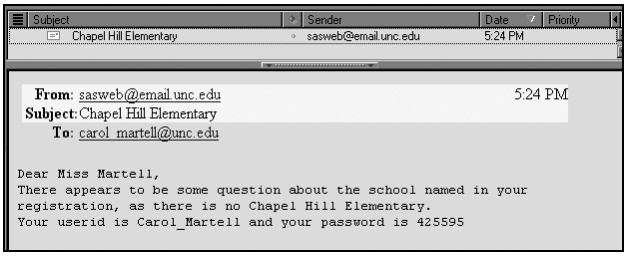

Figure 14

All correspondence is logged with email messages to the reviewer email account. Subject lines are constructed using userid and password so that when messages are sorted, all activity for a registrant appears together. The original registration subject has no suffix. Update subject lines contain the suffix 'update'. **Figure 15** shows a documentary email message and subject line with the 'msg' suffix to show there was individual correspondence.

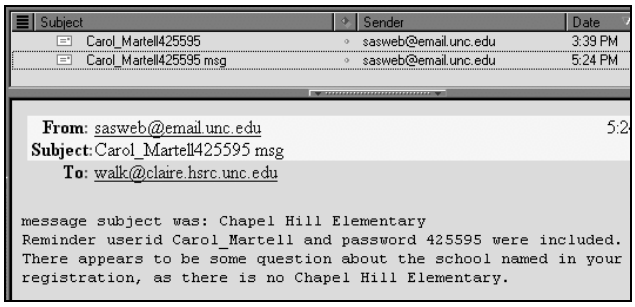

Figure 15

BULK MAIL

A bulk mail form (**Figure 16**) allows Web developers to send announcements and reminders to registrants. The users are encouraged to send and examine a trial message before clicking 'back' and changing the choice to 'all registrants'. Documentary copies of bulk email messages are sent to the reviewing mailbox with a subject line prefix of 'Bulkmail'.

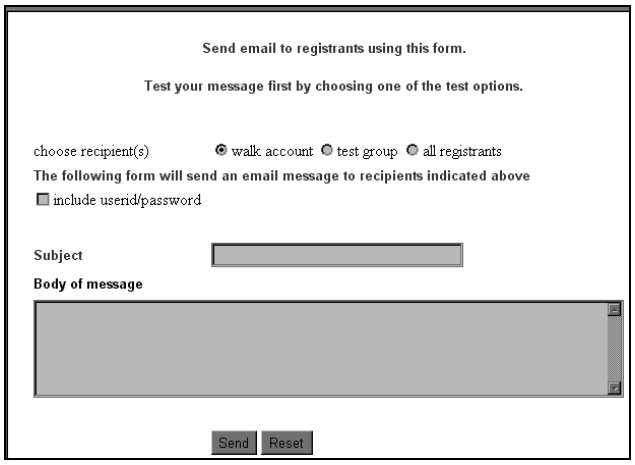

Figure 16

LIVE QUOTES

The truly exciting innovation for the project occurred on the day of the event. Event participants were encouraged to log on to the Web site after completing their walk to share their thoughts and read what others had to say (**Figure 17**). Whereas the event providers, and therefore registrations, numbered in the hundreds, the number of walkers would be in the hundreds of thousands in the United States alone. Since this was to be an International Walk to School Day special feature, involving seven countries, the email screening approach used for the registration data might not suffice to surface the data immediately. The solution used two SAS tables for the unscreened and screened quotes. The action parameter in the quote submission Web form pointed to an htmSQL page, which inserted the quote and associated data as a row into the unscreened data table. Another htmSQL page read the unscreened data, building a Web page containing a form for each table row, or unscreened quote (**Figure 18**). Each form provided edit capabilities and the option to accept or reject the quote. In either case, the quote was removed from the unscreened table. Every refresh of the browser would reveal only unscreened quotes. A mixture of published static pages and dynamic queries were used to surface the screened quotes to the Web (**Figure 19**).

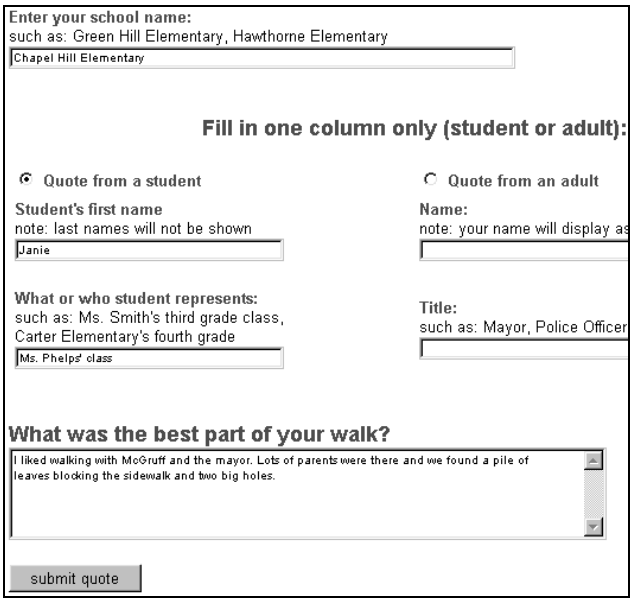

Figure 17

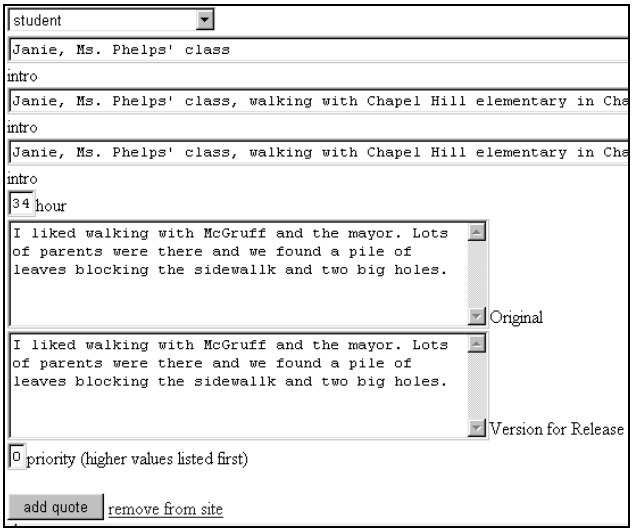

Figure 18

Figure 19

CONCLUSION

The dynamic registration data collection and retrieval system focuses around the registration table. The event providers generate new records and update existing ones. Web developers update or delete records. Data-triggered updates keep the image map current. Dynamic data retrieval occurs when visitors to the site drill down on the image map and subsequent pages, when registrants login to update their information, or when Web developers use the maintenance utility. Email serves as documentation in addition to providing a means to communicate with registrants. The live quote system leveraged the immediacy of the Web and htmSQL to collect and screen data with an absolute minimum bottleneck at the point of manual review. This application was built using SAS Application Dispatcher, htmSQL, SAS/GRAPH, the SAS Output Delivery System and the email facility. The only machine that actually runs The SAS System is the application server housing the data. Registrants, walkers submitting quotes and Web developers screening data do not need SAS running on their desktops because everything is processed using SAS/IntrNet technologies.

ACKNOWLEDGMENTS

The US Department of Transportation (DOT) provides funding for www.walktoschool-usa.org through the Pedestrian and Bicycling Information Center (www.walkinginfo.org).

CONTACT INFORMATION

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

Carol Martell
UNC Highway Safety Research Center
730 Airport Rd, CB# 3430
Chapel Hill, NC 27599-3430
Work Phone: 919-962-8713
Fax: 919-962-8710
Email: carol_martell@unc.edu