Paper 24-26

# Using WebAF for Data Entry and Editing in an Integrated Conservation Monitoring Data Management System

Scott E. Chapal, J.W. Jones Ecological Research, Newton, GA

## ABSTRACT

Providing data access to employees over an intra-net or to clients and constituencies via the Internet, necessitates the use of web technologies. However, the multitudinous options for web-enablement constitute a bewildering array of design choices, architectural alternatives, application servers, scripting methods and programming languages. In an environment where the SAS® system is the *de facto* data management platform, web-enablement using SAS/IntrNet® and AppDev Studio™ provides a deployment architecture that leverages SAS data structures and analytical capabilities.

The AppDev Studio application development environment is comprised of WebAF™, a Java visual application builder, and WebEIS™, a Java based OLAP reporting tool. A proposed advantage of such an Integrated Development Environment [IDE] is the ability to efficiently create data access interfaces (perhaps even by technicians who are not highly trained programmers) while reducing overall development effort. This is one of the expectations with which this project was approached.

Manual data entry applications require a balance of input validation [QA] and ease of entry for the operator, typically using a 'form' interface. In this paper, the design and deployment of Java based Data Entry applications for conservation monitoring is described.

## INTRODUCTION

The Joseph W. Jones Ecological Research Center is dedicated to the pursuit of long-term ecological research resulting in focused conservation guidelines and education materials. The Center has devised a Conservation Monitoring plan which will encompass the entire 28,000 acre Ichauway property, an ecological preserve on which the center is housed. This monitoring will be directed to assess biodiversity, natural variation, and ecosystem processes on Ichauway's ecosystems. The ecosystems include Longleaf Pine-Wiregrass, isolated and riparian wetlands, riverine, agricultural and other land types. Monitoring will be conducted at approximately 400 grid points site wide. The monitoring data will measure parameters which assess ecosystem structure, forest health, soil type and physiography, faunal surveys and other measurements.

### DATA MANAGEMENT FOR MONITORING

Preparation for the monitoring program has been arduous, and the need to implement a well designed data management system, in order to avoid restructuring and scaling problems later, was well understood. The monitoring is projected to continue for several decades at a minimum. Data management structures, organization and programs will need to persist beyond transition in personnel, hardware system changes and software cycles.

Reviewing other similar monitoring projects in government or academic institutions reveals that there are few examples of data management which include efficient, well designed and *integrated* Data Entry. Many of these existent monitoring programs use a varied assortment of software, cobbled together in a loosely knit aggregate. The components might include an RDBMS, spreadsheets, various analytical programs and other desktop applications.

The research and monitoring conducted at the Jones Center utilizes the SAS system for data processing and analysis. SAS can provide most of the data management and analysis necessary, and by extending user interfaces for data entry and reporting, an integrated solution can result. The development of the SAS web-enablement products has been accelerating over the past several years, so that now data can be accessed using several different web technologies. The intent of the data management for the monitoring program is to make data interaction available via the intra-net, and to extend the publication of some results, graphics and reports to public web servers as well.

### DATA ENTRY FAUX PAS

Perhaps no subject conveys boredom and solicits yawns as effectively as *Data Entry*, and even the process of creating data entry applications is broadly perceived as ho-hum. Maybe so, but data entry is critically important for first-level data validation in the data processing cycle. A typical data entry application normally involves the translation or interpretation of a "form" into its electronic equivalent, providing input constraints.

Data entry is often relegated to a tool such as a spreadsheet or a personal database such as MS Access®. More often than not this is because of the lack of development overhead to "get started" with a data entry task. While a spreadsheet *can* be created to do validation and apply formats, this is rarely done by end users and is not easy to secure or to maintain. Although Access is more capable in this respect, ultimately the data need to be imported into SAS for management and analysis. There are specific data entry tools such as Easy Entry™, but this is a terminal-based application which creates ASCII files and incurs the import requirement.

Traditionally, in a SAS environment, a data entry form would be deployed as a SAS/AF application. However, for this project, there was no pre-existing SAS/AF code or experience. Additionally, FRAME applications are perceived to be anachronistic in the browser-focused web era. For these and other reasons, Java and web-enablement was chosen as the direction for the project, under the assumption that this would be a viable platform into the future. Essentially, that meant shifting resources from SAS/AF development to java development.

In a web application, partitioning the logic between application and server is a key design conundrum. The partitioning depends on the specific architecture and the particular deployment technology used: applet, servlet, JSP, etc. With server-side approaches, the partitioning of aspects of logic on the server(s) is also a consideration. For data entry applications, data validation logic is a prime example. There are several techniques which can be utilized to enhance data quality from user input in a data entry form:

- Field value range checks
- Look up tables [Reference data]
- Advice
  - Help screen
  - Format Examples
- Format enforcement [Masking]

Another technique applied in FSEDIT or SAS/AF is to use PROC FORMAT to create custom informats and return _ERROR_ for invalid data . This effectively builds the constraint into the dataset, rather than in the form. As a design principle, this means that changes to the acceptable values can occur simply in the dataset and the entry application can remain unchanged.

## DATA ENTRY IN JAVA

Although several web-enablement approaches are available (even within SAS technologies), Java is widely considered the language of choice because it is portable, secure, and supports the "thin client" paradigm. Additionally, the object oriented development potential of Java, embodied in the Java Beans™ framework, promises cross-platform solutions and code re-usability.

Java applets can provide a high level of interactivity with the user and the components available can be combined to create powerful interfaces for data entry. Initially, development was undertaken using the WebAF GUI to generate applets. Since there were no pre-packaged entry validation or "Input Masking"[1] methods in the com.sas java classes, a catalog of validation SCL was created to do this. The applet then uses the Remote Object Class Factory [ROCF] to instantiate SCL objects from the VALIDATE.VALIDATOR catalog and validate the value of each variable under consideration. The message, returned from the SCL object via ROCF, is then parsed in the applet: If all of the variables pass validation, commit occurs[2].

```
getmsg:
method msg $ burnplan $;
msg='OK';
burnplan=burnplan;
if length(burnplan)=0 then msg='Must enter a value for';
burnplandot=substr(burnplan,1,1);
if length(burnplan)=1 and burnplandot in ('.')
then msg='OK';
else if length(burnplan)^=9
then msg='Invalid Format for Burn Plan ID. \\
    Please enter Burn Plan ID in MMDDYY### format. \\
    MM = Month of the Year 01-12. \\
    DD = Day of the Month 01-31. \\
    YY = 2 digit Year.  \\
    ### = Burn Unit Number where # is a number 0-9.';
else do;
date=input(substr(burnplan,1,6),mmddyy6.);
chkdate=datepart(date);
if chkdate=. then msg='Invalid Date Portion of Burn Plan ID. \\
   Please enter data in MMDDYY### format where \\
   ### = Burn Unit Number and # is a number 0-9.';
plan1=substr(burnplan,7,1);
if plan1 not in (0,1,2,3,4,5,6,7,8,9)
then msg='Invalid Burn Unit Portion of Burn Plan ID (Position 7).\\
     Burn Unit must be Numeric. \\
     Please enter data in MMDDYY### format where ### = \\
     Burn Unit Number and # is a number 0-9.';
plan2=substr(burnplan,8,1);
     ...
end;
endmethod;
```

A "Commit Record" button sends data from a textField to SAS for validation with SCL. The SCL returns error messages that are displayed in a textArea message display box as it sequentially validates each field.

```
textArea1.setText("All Fields are OK");
String msg1=sclBurnplan.getMsg(textField1.getText());
if ( msg1.equals("Must enter a value for")
     || msg2.equals("Must enter a value for"
     ...
) )

{ textArea1.setText("All fields must have valid entries");
}
else
{
    if (! msg1.equals("OK"))
        {
            area1AddReturn(msg1);
            textField1.setFocus(true);
        }
}
else ...
```

In this manner, the program loops through all variables, validating each defined value, then checks the return string for validation success of all required variables.

```
    String commit=textArea1.getText();
    if (commit.equals("All Fields are OK"))
```

Also, the key variable (in this case "burnplan") is verified as to whether it currently exists in the Data Set. Burn Plan ID must be unique for a successful commit.

```
String objburnplan=textField1.getText();
String newobjburnplan=objburnplan;
oldobjburnplan=("");
objcondvaluesDataSet.setWhere("burnplan=" + objburnplan);
try
{
   Object[] colvalues= objcondvaluesDataSet.getColumn(1);
   for (int i=0; i<colvalues.length;i++)
   {
        oldobjburnplan=(""+colvalues[i]);
   }
}catch (Exception e) {System.out.println(e);}

if ( !oldobjburnplan.equals(""))
{
   textArea1.setText("");
   String m1=("Please check the Burn Plan ID.\\
               Burn Plan ID must be a UNIQUE value");
   String m2=("Burn Plan ID "+newobjburnplan+" is NOT a unique\\
               value in the Burn Objectives and\\
               Weather Conditions Data Set.");
   String m3=("This value MUST be changed for a Commit\\
               to be Successful.");
   textArea1.append(m1+"\n"+m2+"\n"+m3);
   textField1.setFocus(true);
}
```

The users login is retrieved from the connection and a datetime stamp is appended to the record. This creates a unique record identifier which identifies who entered the data and when it was done.

```
burnCondDataInterface1.setFormattedUSER(connection1.getUsername());
burnCondDataInterface1.setFormattedCOMMITD(textField26.getText());
burnCondDataInterface1.setFormattedCOMMITT(textField27.getText());
// Commit record to datset
burnCondDataInterface1.commit();
// Write message to textbox
textArea1.setText("Commit of data is COMPLETE");
```

Using the Audit Trail facility (an update feature in v8 data sets) would be another alternative for this functionality.

Validation can also be done in java rather than in SCL. This has the consequence of being portable to non-SAS situations. Why would that be important? In many situations it's not, but in this monitoring program, the need for applications which can be deployed in the field necessitate a hand-held computer without connectivity to the server.[3] A disadvantage is that the code must be written! However, with this field data collection requirement, coding some validation in Java is potentially more reusable in these small devices than SCL would be.

```
public class javaCharacterValidation
{
 public String errorCode;
 public javaCharacterValidation()
 {
 }
 public String javaCharacterValidation(String character, int howlong)
 {
  errorCode="True";
  checkcharacter:
  {
   if ( !character.equals("") )
   {
    int i, len = character.length();
    StringBuffer dest = new StringBuffer(len);
    for (i = (len - 1); i >= 0; i--)
    {
     if ( Character.isDigit(character.charAt(i)) || len > howlong )
     {
      errorCode=("Field must contain CHARACTER data <= "+howlong+".");
      break checkcharacter;
     }
    }
   }
  }
  return errorCode;
 }
 public String getErrorCode()
 {
  return errorCode;
 }
}
```

The validation methods could then be made available to all applets, or as discussed later, servlets or JSP's. After prototype data entry applications are built, the long-term objective is to generate a library of java classes or beans which can be rapidly assembled into future applications as our needs change. The validation code can also be encapsulated in a class or bean either by using the "InformationBean Wizard" or by hand coding.

Although the WebAF IDE is useful in setting up, designing and compiling projects from a graphical interface, the interaction of components with each other and with SAS data requires manual coding of the event handling. Seemingly trivial actions require significant Java programming knowledge.

For example, a common form control device is to use a combo box to create a list of choices. However, it is often frequently desirable to initialize the combo box with a list from a "reference" data set instead of "hard-coding" the values in the applet itself. That way, the list can be altered over time without requiring changes to the applet. Also, descriptive labels which make up the choices in the combo box can be indexed to numeric or coded values which are entered into the data set. Thus, the following is necessary in the postInit section of the applet to initialize the combo box:

```
try { com.sas.sasserver.datasetinfo.DataSetInfoInterface\\
      dsinfo = (com.sas.sasserver.datasetinfo.DataSetInfoInterface)
  __rocf.newInstance(\\
   com.sas.sasserver.datasetinfo.DataSetInfoInterface.class,\\
   connection1);
  dsinfo.setDataSet("burn.origin");
  String[] values = dsinfo.getVariableUniqueValues(2);
  choice14.setInitialItems(values);
  originvaluesDataSet = (DataSetInterface)\\
    __rocf.newInstance(DataSetInterface.class,connection1);
  originvaluesDataSet.setDataSet("burn.origin");
} catch (Exception e) {System.out.println(e);}
```

A feature often desired in data entry is the ability to review the data once it has been entered. A data set may contain code information (often a counter/number representation of character description information). This coded data does not necessarily mean anything to the user, but is useful during data analysis. In order to scroll through the data, the selected item in the combo box (containing descriptive information) needs to correspond to the coded value in the primary data set. To do this an event listener needs to be added to the Navigation Bar. When the next, previous, first and last buttons are clicked, a method is triggered that performs a lookup (what number is in the primary data set) and sets the selected item (viewed item) in the combo box to the corresponding descriptive information.

```
public void primaryobjectives() {
 String objdesc1 =\\
  burnObjectives_BurnConditionsDataInterface1.getFormattedOBJCODE1();
  objvaluesDataSet.setWhere("objcode=" + objdesc1);
  try {
   Object[] colvalues= objvaluesDataSet.getColumn(2);
   for (int i=0; i<colvalues.length;i++) {
    String val = ""+colvalues[0];
    choice2.setSelectedItem(val.trim());
   }
  } catch (Exception e) {System.out.println(e);}
 }
public void navigationBar1ActionPerformedHandler12\\
      (java.awt.event.ActionEvent event) {
  //  Highlight an item in choice components 2-5 based on the number
  //  in the corresponding textField.
  String command = event.getActionCommand();
if (command.equals("NEXT") | command.equals("PREVIOUS") |\\
   command.equals("FIRST") | command.equals("LAST")) {
   primaryobjectives();
  }
 }
```

During data entry the user chooses an option from the combo box list. A corresponding code value needs to be inserted into the data set. Again, this is done because the code does not mean anything to the user so the combo box provides them with a list of options that are meaningful. Often data analysis or management needs to use numeric or coded values rathen than descriptive character data.

To set the number value in the data set to the corresponding descriptive item in the combo box, a “look-up” is performed and then the cell is set in the data set to the value received from the look-up.

```
public void choice2ItemStateChangedHandler8\\
(java.awt.event.ItemEvent event) {
  //  Set textField number based on selected
  // item in corresponding listBox.
  String objcode1=choice2.getSelectedItem();
  objvaluesDataSet.setWhere("objdesc='" + objcode1 + "'");
  try {
   Object[] colvalues= objvaluesDataSet.getColumn(1);
   for (int i=0; i<colvalues.length;i++)  {
   burnObjectives_BurnConditionsDataInterface1.setFormattedOBJCODE1\\
   (""+colvalues[i]);
   }
  }catch (Exception e) {System.out.println(e);}
 choice3.setFocus(true);
 }
```

As these few examples illustrate, behaviors in the interface require somewhat involved programming sequences which must be manually coded.

### BROWSER WARS

Write once run anywhere was Java’s original claim to fame. Its no news that the reality is different when browsers are involved. Browsers have differing characteristics, and even different versions of the same browser have functional differences. If that weren’t enough, controls (widgets, whatever) are rendered in a platform dependent way[4]. Presumably this is a manageable problem, but it has been a sobering lesson that applet controls designed on a Windows WebAF development PC, don’t necessarily display in predictable ways on other platforms and browsers. This is one of the problems inherent in deploying Java applets on the web, and one of the reasons that server-side approaches might be an attractive alternative. Although browser interaction effects are inevitable, the negative impacts may be reduced through prudent application design. Writing for specific browsers, versions or platforms is possible, but undermines the neutrality which Java intends to provide.

### SERVLETS AND JSP

The strengths of Server-side Java programming have been enumerated elsewhere, but obviously a basic motivation is to position the application on the server rather than on the client in order to: 1) eliminate applet download, 2) simplify deployment and 3) leverage the modular re-usability of EJB components. Servlets, in combination with Java Server Pages can achieve the separation of presentation from content which is advocated by the Model-View-Controller (MVC) design pattern approach. While ostensibly this is true, when developer resources are very limited and specific (Bean) components are not yet written, the move to servlet/JSP can introduce more complexity into the development process. Thus for this monitoring program, the applet vs. servlet/JSP choice for data entry applications remains uncertain. The interactivity provided by an applet is an asset which may be difficult to achieve with JSP.

## CONCLUSION

WebAF provides the Java classes and framework with which to construct effective data entry applications to interact with SAS. However the IDE does not obviate the need for skilled development. In order to achieve an effective data entry project with basic quality assurance and data validation features, knowledge of SCL and/or Java is necessary.

Running WebAF 1.2 applets in a browser requires a specific version of Java Plug-in, 1.1.3. This Plug-in dependence reduces the universality of the solution and can complicate deployment. Another issue has been the lag time between JDK improvements and the availability of those features in AppDev Studio, notably Java 2 enhancements such as Swing.

AppDev Studio and IntrNet are tools that can be used effectively to extend existing SAS data resources to web interfaces. The integration of data entry and editing functions using Java and other web technologies to extend the SAS system has the potential to simplify and streamline the development and maintenance of the data management system *in the long run*. The initial requirement for investment of resources in Java skills and development is substantial, however, and should not be under estimated. Server and data resources can be made more effective by extending web interfaces to staff, client constituencies and public web servers. Further, this could be a proof of concept for the forestry, ecological science and land management communities.

## Notes

[1] Input Masking functionality is provided in the JMaskField interface, but this is based on javax.swing classes, which were not implemented in WebAF 1.2.

[2]Double backlash notation \\ represents wrapped code lines.

[3] Supplementing the Data Entry applications developed in WebAF, are intended companion applications for handheld devices to be used by technicians in the field. Because of the potential for interference in the forest, there will be a limitation to un-connected devices which can be synchronized, such as the Palm. Radio modem access or

other connected approaches will be tested in the field as technologies improve. These wireless applications could embody logic similar to their networked counterparts, but there would necessarily be differences in design and other considerations.

[4]Netscape 6 appears to have overcome this platform dependent behavior somewhat.

## REFERENCES

Duguay, C. (1999). Jmaskfield, *Java Developers Journal* **4**(1).

Eckstein, R., Loy, M. & Wood, D. (1998). *Java Swing*, The Java Series, O'Reilly.

Franklin, G. & Jensen, A. (2000). Integrity constraints and audit trails working together, *Proceedings of the Twenty-Fifth Annual SAS® User Group Conference*, SAS Institute, Inc.

LaChapelle, C. & Gagliano, T. L. (2000). Migrating your SAS/AF® FRAME applications to the web, *Proceedings of the Twenty-Fifth Annual SAS® User Group Conference*, SAS Institute, Inc.

Morgan, D. & Province, M. (2000). Building a better data entry application using PROC FSEDIT, *Proceedings of the Twenty-Fifth Annual SAS® User Group Conference*, SAS Institute, Inc.

Neward, T. (2000). *Server-based Java Programming*, Manning.

## ACKNOWLEDGEMENTS

The Robert W. Woodruff foundation provides funding to the Joseph W. Jones Ecological Research Center to conduct research, education and conservation initiatives. The Ichauway property is owned by the Woodruff foundation.

## CONTACT INFORMATION

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

Scott Chapal
Joseph W. Jones Ecological Research Center
Ichauway, Inc.
Rt. 2 Box 2324
Newton GA. 31770
scott.chapal@jonesctr.org