Paper 155

SAS webAF™ for Java Application Development, a First Sip

Mickey Waxman University of Kansas, Lawrence, Kansas
Larry Hoyle University of Kansas, Lawrence, Kansas

ABSTRACT

SAS webAF is an integrated development environment for Java™ programs and applets. In addition to the standard components, it provides pre-built components that allow Java programs to access SAS data and procedures on a server from a thin client without the SAS system.

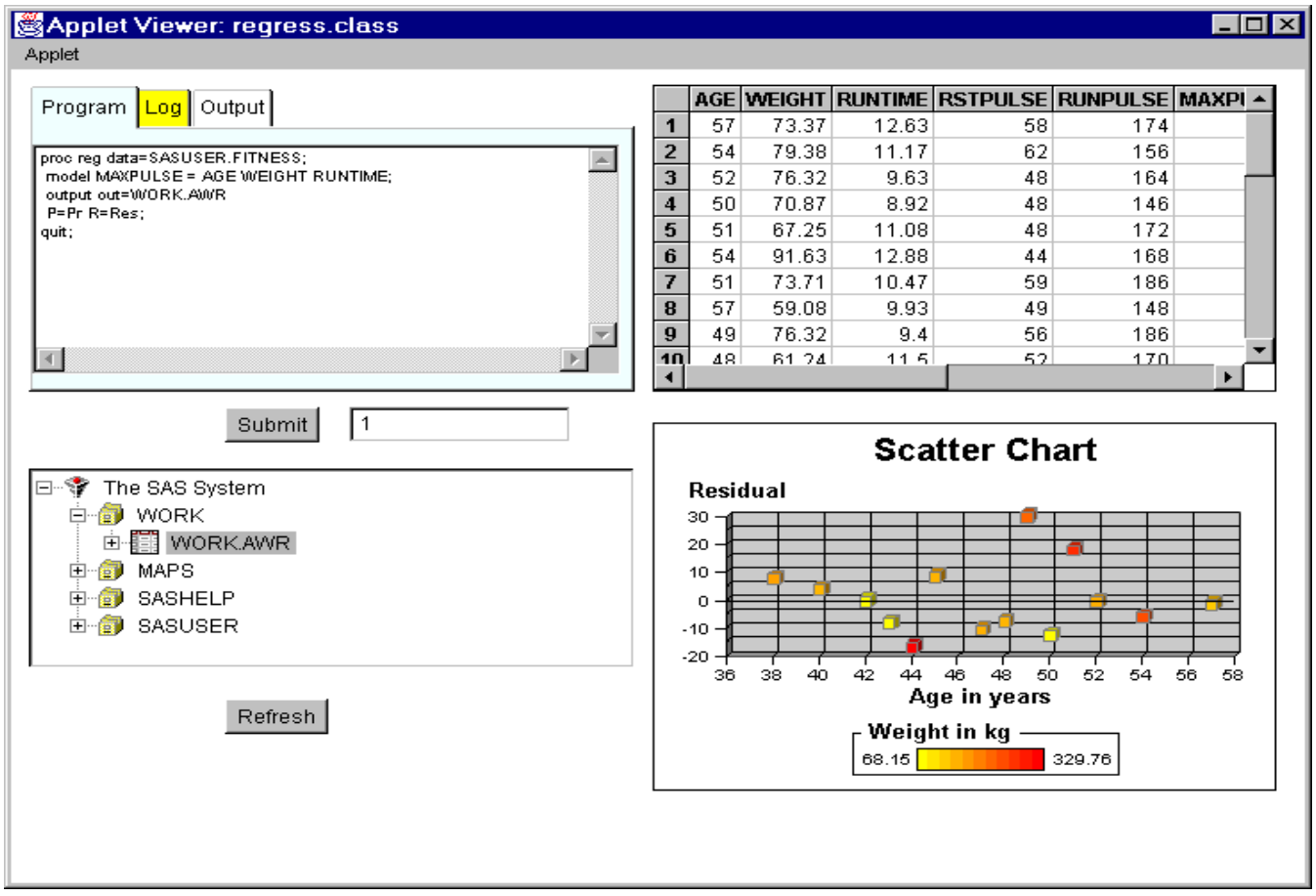
This hands-on workshop will serve as a first look at SAS webAF, demonstrating what it is and how it works. Participants will create a SAS-enhanced Java applet in a series of drag and drop exercises.

No previous knowledge of Java is required for this workshop.

INTRODUCTION

In this workshop we will recreate the application shown below. It can be used in a Web page or run as a standalone program. This Java program connects to a SAS server across the network and submits a regression procedure. It also retrieves a dataset with the predicted and residual values from the regression and displays them in a table and a graph.

The application will be recreated using Java **components** (JavaBeans™) which are supplied as part of the webAF package. Each of the visual elements in the figure below is such a component, placed in the project using the drag and drop capability of the webAF development environment.
The scatter plot, for example, is a component written by SAS Institute. It was dropped into the project and then connected to the rest of the project by setting its "datasetInterface" property. Once configured with this property, it was able to retrieve and display data.
Components are customized when designing an application by setting values for their **properties** (variables). The components are stored as a part of the application with those values retained. For instance, if the color of the submit button is set to blue while the application is being built, it stays blue until something else changes it.
Components also have **methods** (like subroutines - pieces of code), and can originate **events** (e.g. A button is clicked). The webAF development environment has tools for helping set up **event handlers**. In the example below, Clicking the refresh button fires the "actionPerformed" event, whose handler calls the tree view's "refresh" method.



THE WEBAF DEVELOPMENT ENVIRONMENT

The webAF development environment contains a number of collections of tools organized into toolbars and subwindows. This workshop will use just a few of those facilities, the Active Frame Window, the Component Palette, the Project Navigator, and the Output Window.

THE ACTIVE FRAME WINDOW

The subwindow on the right in the figure below is the Active Frame Window. It contains the view of the application's visual components. Non visual components are also added to the application by dropping them on this window. A right click on the components in this window allows selection of documentation about the component or setting object properties or event handlers. Tabs at the base of the Active Frame Window allow the programmer to switch between the application' source code and its run-time appearance.

THE COMPONENT PALETTE

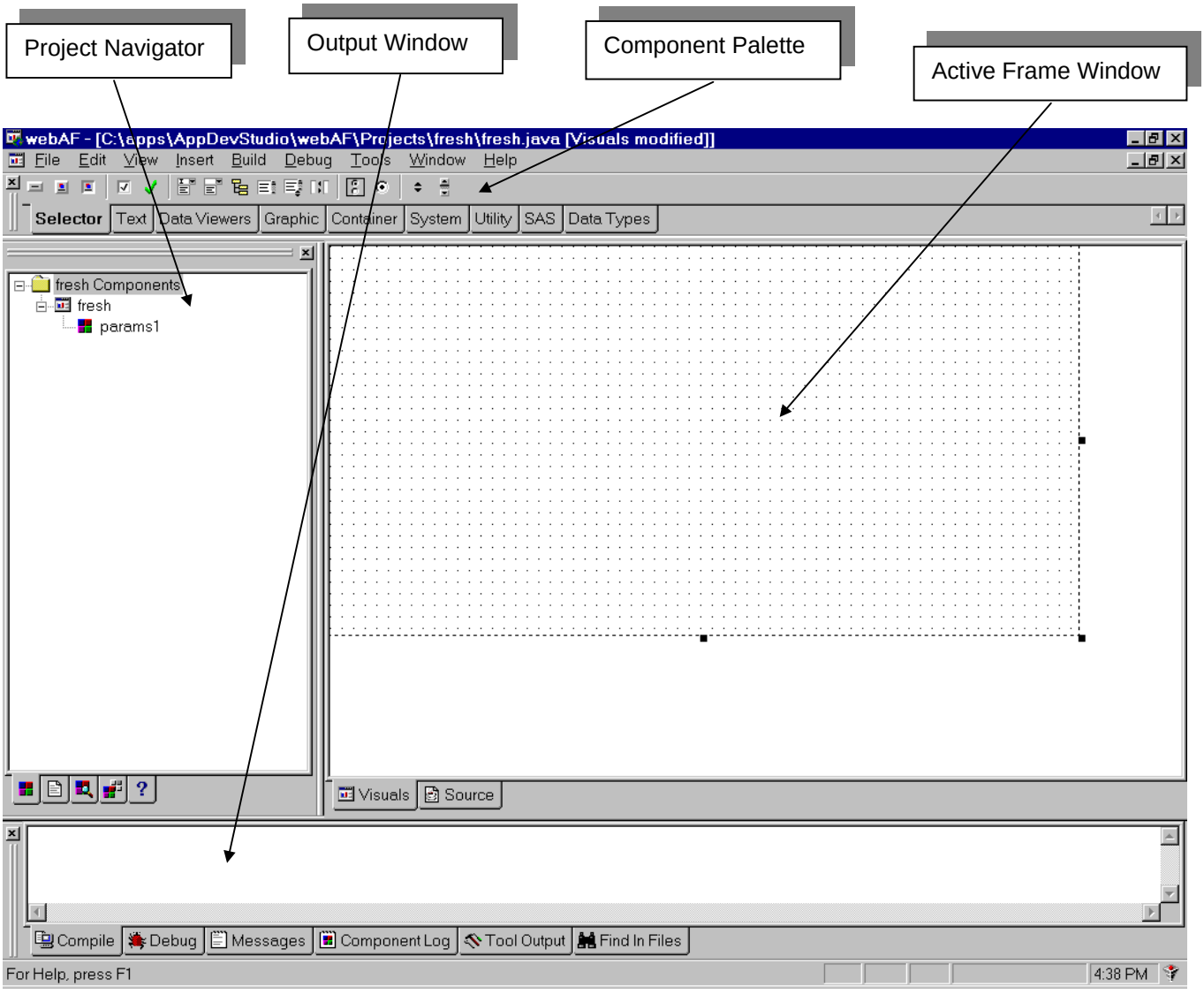
The tabbed bar just below the main menu bar in the window below contains the components bundled with webAF. These include many components based on Java's Abstract Windowing Toolkit (AWT), such as buttons, text boxes and the like. It also contains components designed to communicate across the network with SAS running on a server, as well as components which facilitate using SAS objects - datasets, libraries, catalogs, MDDBs, formats, procedures and so on. There are also graphical components - an image viewer and a chart component.

THE PROJECT NAVIGATOR

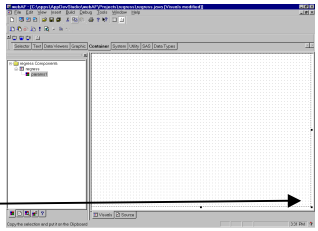
The subwindow on the left, the Project Navigator, contains an outline for tutorials and extensive help, as well as views of lists of the components, component properties, files, and classes in the project (application). Right clicking on the components and properties allows the user to set property values, design event handlers, and **link** the values of properties belonging to two different components. A link sets up the application so that when one property is assigned a new value, the other property is automatically assigned that value.

THE OUTPUT WINDOW

The bottom subwindow receives output from compiling and testing the application. Error messages, elapsed times, status messages and more appear here.



GETTING STARTED



Double click the webAF icon.

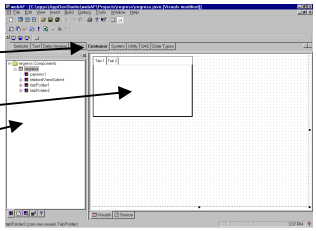
Go to **File -- New -- Project** and let's call this project ***Regress.***

Click **Finish.**

Click **View -- Output** to close the Output Window.

You may wish to resize the component (the Frame) in the Active Frame Window to fill the window.

PUTTING COMPONENTS INTO YOUR JAVA APPLET OR APPLICATION



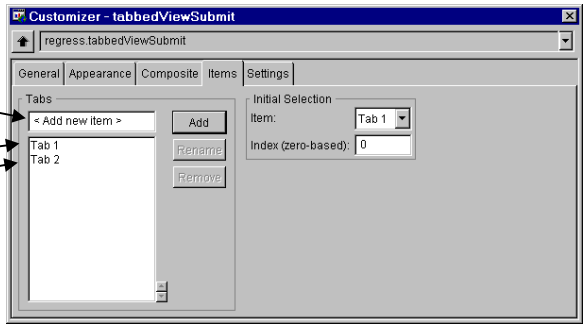
SETTING UP TABBED FOLDERS - A MINI SAS DISPLAY MANAGER

Click on the **Container** tab.

Find the **TabbedView** container on the Component Palette. Drag to the Frame's upper left corner.

Go to the Project Navigator window (left window), right click on **tabbedView1 --- Rename.**

Enter **tabbedViewSubmit** as the new name.



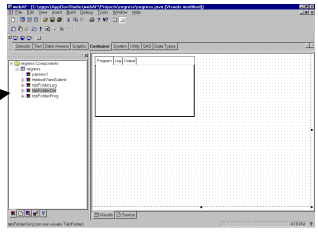
Right click on **tabbedViewSubmit** --- **Customizer.**

Click on **<Add new Item>** to highlight, enter **Output** as the new name and click **Add.**

Click on **Tab1** to highlight, type **Program** and click **Rename** button.

Click on **Tab2** to highlight, type **Log** and click the **Rename** button.

Close the Customizer window.

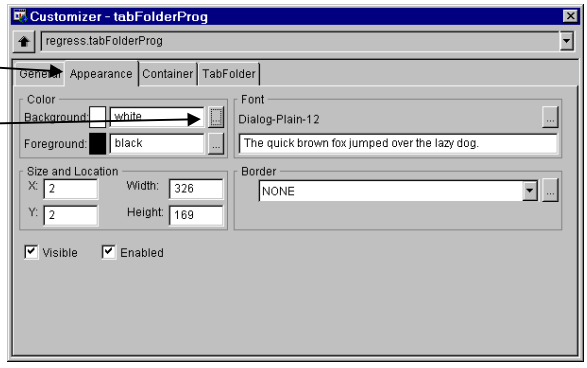


In the Navigator window:

Right Click on **tabFolder1** --- **Rename**, enter **tabFolderProg** as the new name.

Right Click on **tabFolder2** --- **Rename**, enter **tabFolderLog.**

Right Click on **tabFolder3** --- **Rename**, enter **tabFolderOut.**



Right click on **tabFolderProg** --- **Customizer. C**lick the **Appearance** tab.

Click on the **Color Background** ellipsis button ("**...**" on the right), select **cyan** and close the window.

Right click on **tabFolderLog** --- **Customizer,** click the **Appearance** tab.

Click on the ellipsis button ("**...**" on the right), select **yellow** and close the window.

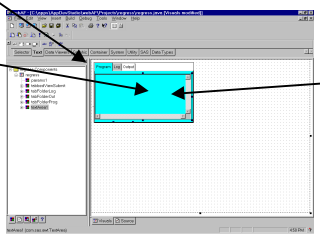
In the frame, click on the **Program** tab (to select). The tabbed folder should be cyan, if not, click the **Program** tab again.

Click on the **Text** tab in the Component Palette.

Find the **textArea** component, drag and drop it onto **tabFolderProg** (on top of the cyan folder in the Frame).

In the Frame stretch the new textArea component to fill the entire area below the tabs in the container.

In the Navigator window right click on **textArea1 --- Rename**, enter **textAreaProg**.



SAS programs entered in this textArea will be submitted to run on the remote server.

Click on the **Log** tab in the frame (to select). The tabbed folder should be yellow.

Click on the **text** tab in the Component Palette, drag and drop a **textArea** component onto tabFolderLog (on top of the yellow folder in the Frame).

In the Frame stretch the new textArea to fill the container.

Rename **textArea2** to **textAreaLog.**

The log from the submitted SAS program will appear in the Log tab.

Click on the **Output** tab in the Frame.

Click on **text** tab in the Component Palette.

Drag and drop **textArea** component onto **tabFolderOut.**

In the Frame resize the **textArea** as before.

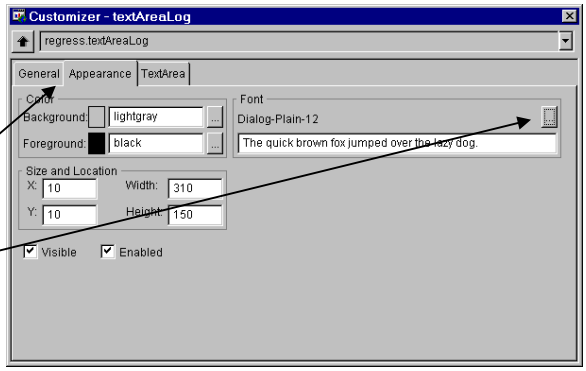
Rename **textArea3** to **textAreaOut.**

The output from the submitted SAS program will appear in the Output tab.

Right click on **textAreaLog** --- **Customizer**, Select the Appearance tab.

Click on the Font ellipsis ("**...**") button (far right edge of dialog box), change the font to 9 point.

Close the **Customizer** window.

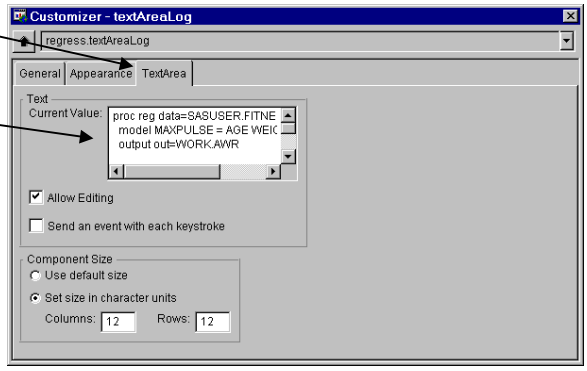


Repeat this step for both **textAreaOut** and **textAreaProg.**

Right click on **textAreaProg** --- **Customizer**, click on the **TextArea** tab.

Enter this text in the **CurrentValue** window:

```
proc reg data=SASUSER.FITNESS;
  model MAXPULSE = AGE WEIGHT RUNTIME;
  output out=WORK.AWR
  P=Pr R=Res;
run;
quit;
```



Close the **Customizer** window.

Click on the **Selector** tab in the Component Palette.

Drag a **button** (far left on Component Palette) to the Frame and center below the **tabbedView** component.

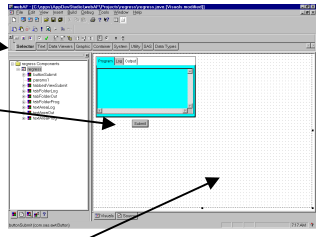
In the Navigator window, right click on **Button1** --- Rename it **buttonSubmit**.

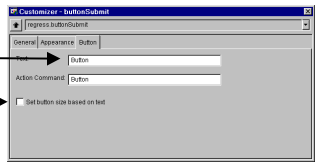
Right click **buttonSubmit** --- **Customizer**.

In the **Text** window, change “button” to "**Submit".**

Click (select) the **set button size based on text.**

Close the **Customizer** window.

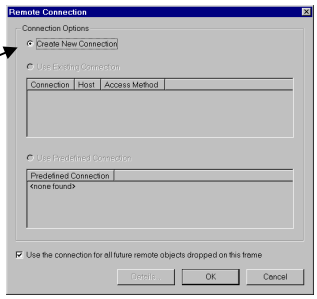




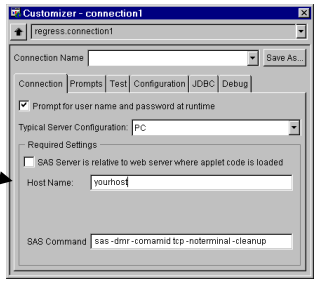
Click on the **SAS** tab in the Component Palette

Drag a **SubmitInterface** (far left on Component Palette) and drop onto the Frame ***but not on top of any component.*** A new dialog pops up, **create new connection** should be selected.

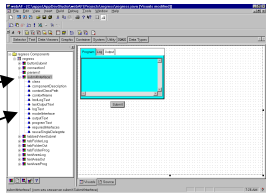
Click OK.



Right click on **Connection1. -- Customizer**. Set the host name to the address of your server. Click (select) **Prompt for username and password at runtime**.



CONNECTING THE COMPONENTS

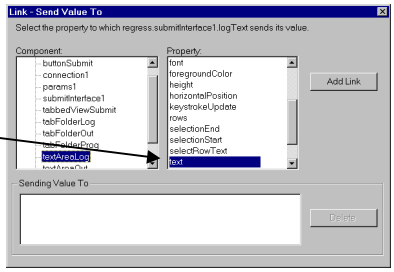


In the Project Navigator, click on the **+** to the left of **submitInterface1** to expand it.

Right click on **logText** --- **link** --- **Send value to.**

Click on **textAreaLog** --- **text,** click the **Add Link** button (on the right).

Close this dialog box.



Repeat the above three steps to link **outputText** to **textAreaOut** (in Navigator under **submitInterface1**).

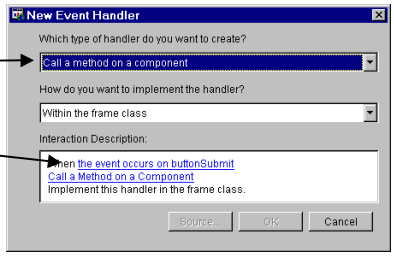
In the Frame, right click on the **Submit** button, click on **handle event.**

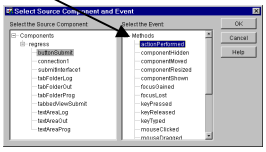
The first selection should read "**Call a method on a component**".

Click on "**the event occurs on buttonSubmit**".

In the **Select the Event** section (right section) click on **actionPerformed.**

Click OK.

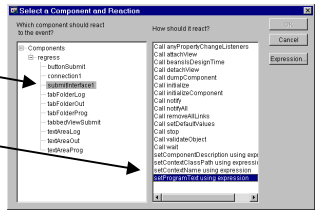




Click on "**Call a Method on a Component**" (New Event Handler Window- shown on previous page).

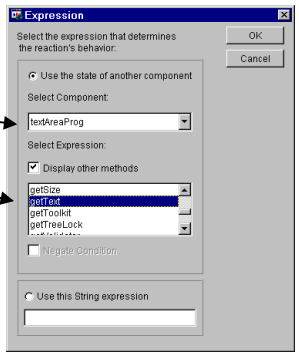
In the **Which method should react to the event?** section, click on **SubmitInterface1.**

In the **How should it React?** section, click on **setProgramText using expression.**



In the **Select Component** section (in the **Expression** dialog box), scroll down to click on **textAreaProg.**

In the **Select Expression section,** select **getText.**



Click OK in the **Expression** dialog box.

Click OK in the **Select a component and a Reaction** box.

Click OK in the **Event Handler** box.

SAVE YOUR WORK

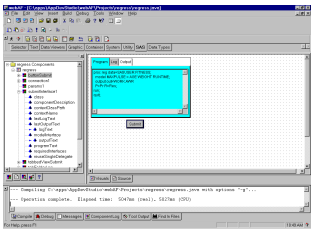
Go to **File --- Save Project.**

TESTING

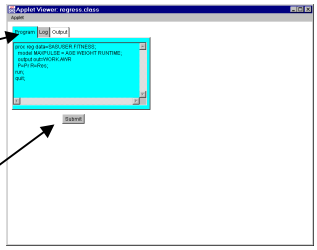
Click on **View** --- **Output.**

Click on **Build** --- **Compile File.**

If you see "operation complete" in the webAF Output Window and no error messages, then try **Build** --- **Execute.**



You should now have a working web application running in the AppletViewer window:



Select the **Output** tab and then the **Log** tabs.

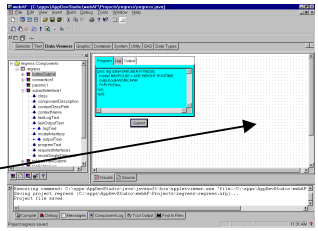
You will see an empty output window and a copyright notice in the log window.

Click the **Submit** button. You will see the log from the regression in the Log window.

The event handler for the Submit button sets the programText attribute of the SubmitInterface. The SubmitInterface sends the SAS code through the connection object to run on the server(RSUBMIT). The server runs the SAS program and sends back the log and output.

Select the Program window and delete the "RUNTIME" variable and change WORK.AWR to **WORK.AW** by deleting the final **R**. Press submit again and see the results of the second regression in the log and output windows.

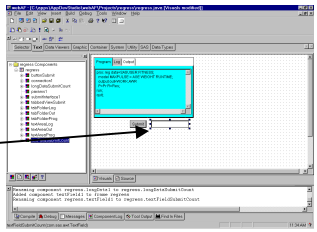
Close the **AppletViewer** window.



JAZZING IT UP - COUNT THE NUMBER OF SUBMITS

Click on the **Data Types** tab in the Component Palette and drag a **LongData** object into the Frame (not on top of anything).

In the Project Navigator, rename **longData1** to **longDataSubmitCount.**



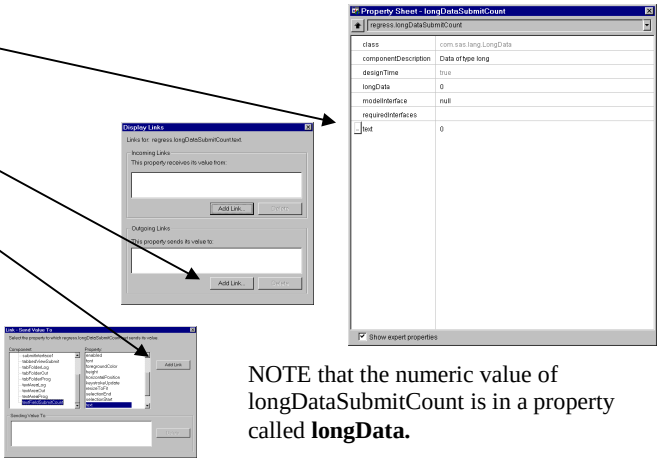
From the **Text** tab in the Component Palette, drag a **TextField** to the right of the **Submit** button. Rename **textField1** to **textFieldSubmitCount** (in the Project Navigator).

In the Navigator window, right click **longDataSubmitCount -- Properties**. Move the mouse over the word "text" (the text property of the object) and click the ellipsis button (**…**) that appears on the left.

In the **This property sends its value to** section (lower section) click on **Add link**.

Select **textFieldSubmitCount** --- **text**, click **Add Link** button.

Close the **Link Send Value To** box. Close the **Display Links** box. Close the **Property Sheet.**



NOTE that the numeric value of longDataSubmitCount is in a property called **longData.**

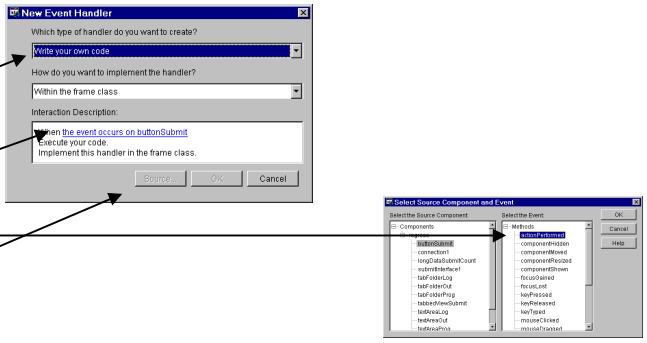
HERE WE ACTUALLY WRITE A LINE OF JAVA CODE

Right click on the **Submit** button and select **handle event**.

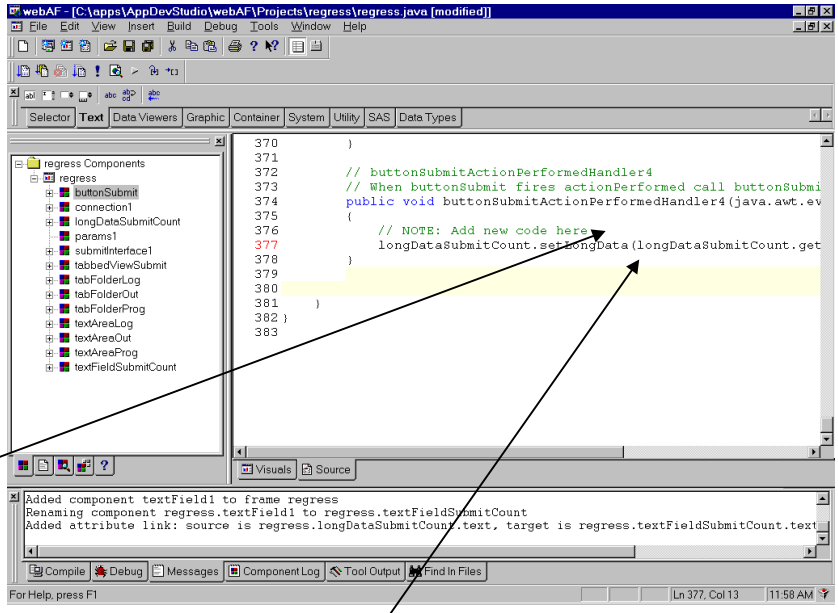
Select **write your own code** in the top box.

In the Interaction Description section, click on **the event occurs on buttonSubmit.**

In the **Select the Event** section, select **actionPerformed** and click OK.



Click on the **Source** button. Here we will type in a Java statement.



Place the cursor at the ***end*** of the comment line**:**

// NOTE: Add new code here

and hit <Enter> key

(scroll to find, if necessary. It will follow

public void buttonSubmitActionPerformedHandler1)

Type in the following:

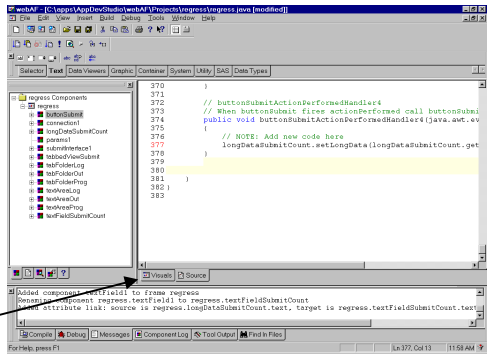
longDataSubmitCount.setLongData(longDataSubmitCount.getLongData() + 1);

Your application now counts the number of submits.

Try **Build** --- **Execute.**

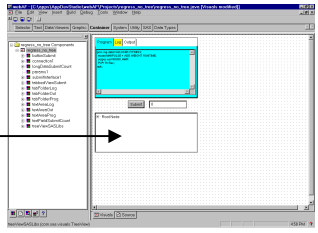
In the **AppletViewer** window, click the **Submit** button. Click it again and notice that the submit count increments.

Close the **AppletViewer** window.

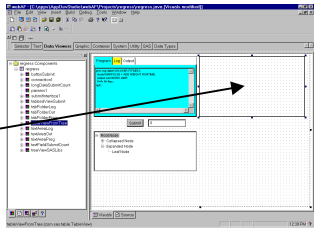


JAZZING IT UP - TABLES AND GRAPHS

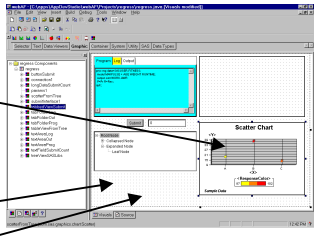
Click the **Visuals** tab under the Frame.



From the Selector tab in the Component Palette drag a **TreeView** component to below the submit button.
Rename it **treeViewSASLibs** (hint: go to Navigator window to do this).



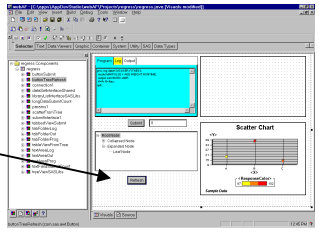
From the **Data Viewers** tab in the Component Palette drag a **TableView** component to the right of the **tabbedView**. Rename it from **tableView1** to **tableViewFromTree.**



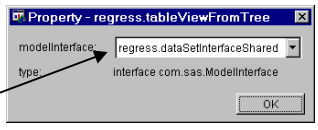
From the **Graphic** tab (Component Palette) drag a **Scatter** component to below the **tableView**. Rename it from **scatter1** to **scatterFromTree.**

From the **SAS** tab in the Component Palette drag a **DataSetInterface** component to an empty spot in the Frame. Rename it from **dataSetInterface1** to **dataSetInterfaceShared.**

Again from the **SAS** tab, drag a **LibraryListInterface** component to an empty spot in the Frame. Rename it from **LibraryListInterface1** to **libraryListInterfaceSASLibs.**

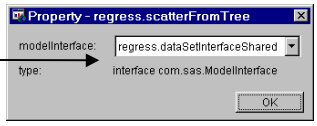


From the **Selector** tab (Component Palette), drag a button to below the **treeView** in the Frame. Rename it from **button2** to **buttonTreeRefresh**. Use the **Customizer** to change its text to **Refresh**.



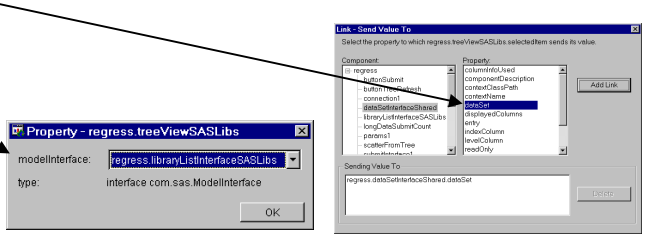
Expand the outline for **tableViewFromTree** (Hints: Navigator, +) and right click on the **modelInterface** property. Select **Set Value** and set to **regress.datasetInterfaceShared.**

Repeat the instructions in the preceding paragraph for **ScatterFromTree**.

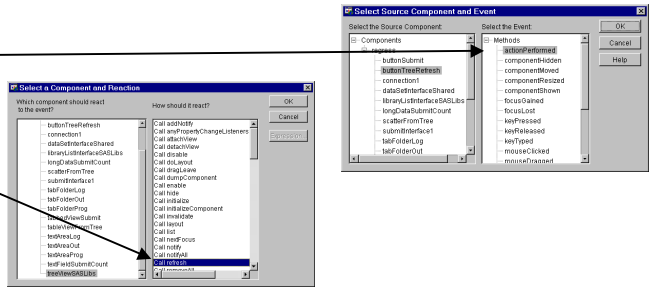


Expand the outline for **treeViewSASLibs** and right click on the **selectedItem** property. Select **Link** -- **Send Value,** select **dataSetInterfaceShared** and **Dataset**. Click **Add Link**. Close the **Link - Send Value To** dialog box.

Right click on the **modelInterface** property of **treeViewSASLibs.** Select **Set Value**. Scroll to **libraryListInterfaceSASLibs** and click OK on the Property dialog box.



Right click on the **Refresh** button, select **Handle Event**.
In the Interaction Description section (bottom), Click on **the event occurs on buttonTreeRefresh**.
Click **action Performed** and click OK.
Click **Call a method on a Component**.
Select **treeViewSASLibs**. Select **Call refresh**
Click on OK. Click OK on the **Event Handler** dialog box.

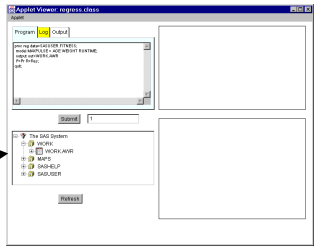


FINAL TESTING

The application is now complete. Click **Build -- Execute**.

In the AppletViewer window, click the **submit** button.

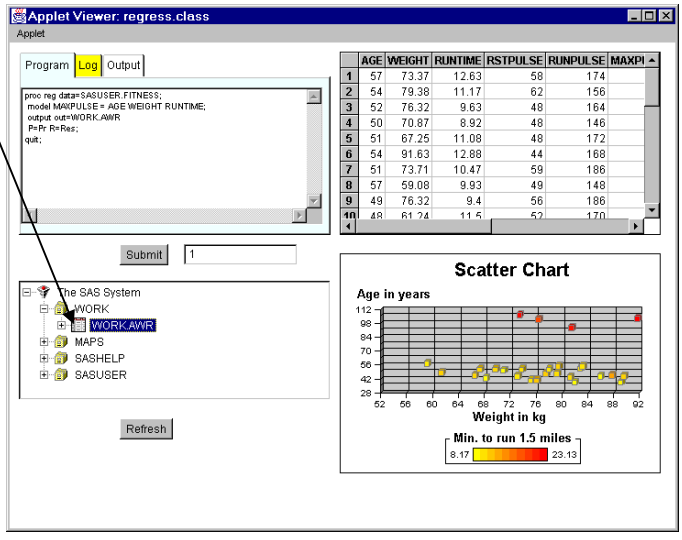
Expand the outline in the tree view by clicking on the '+" to the left of **The SAS SYSTEM** and to the left of WORK.



Click on **WORK.AWR**, the output dataset from the regression.

The table and graph become populated with data once the dataset is selected. Rerun the regression with different variables and a new output dataset. Click on the tree view's refresh button to make that dataset appear in the outline too. Choose it and see the table and graph change.

You might want to right click on the scatter plot and experiment with changing its properties at run-time.



CONCLUSION

An important feature of webAF which hasn't been shown here is it's ability to create Java components which, running on the client, mirror SAS/AF® components running on the server. This is accomplished with the **Remote Object Class Factory** (ROCF).

ROCF creates the Java component which can call methods on SAS/AF objects (written in SCL) running on the server. These SAS/AF objects can then call other SAS/AF objects or run other SAS programs. This architecture will be important to those with existing SAS/AF applications.

CAVEATS

This workshop's application was developed without the need to write any Java code. Many real applications will require writing Java code to supplement the SAS supplied components.

The application you have just created also accepts any SAS code the end user cares to submit to the server. This, of course, is not usually desirable.

The application shown in this paper was developed using the Release Candidate 1 version of webAF. Some graphical elements may change in the final release version.

TRADEMARKS

ACKNOWLEDGEMENTS

We wish to thank Emma Hoyle, age 9, who tested the accuracy and ease of use of the written instructions for this workshop. She successfully completed the project in about one hour.

CONTACT INFORMATION

Mickey Waxman mickey@ukans.edu
Larry Hoyle lhoyle@ukans.edu

A copy of this paper and the sample project will be available at:
www.ukans.edu/cwis/units/IPPBR/ksdata/sugi/sugi24/WaxmanHoyle24.htm