Paper 33-27

Innovative Use of Web Application Technologies to Build a New SAS® Analytical Solution

Mark Lumsden, SAS, Cary, N.C.
David Ostiguy, SAS, Cary, NC

ABSTRACT

In setting out to develop a new SAS analytic solution with a Web-based user interface, we wanted to deliver a user experience that would be superior to that of typical Web application UIs, and approach the levels of ease of use and interactivity associated with installed applications. This paper describes the object-oriented dynamic HTML techniques that were employed to develop componentry that makes this possible in the *SAS® Process Intelligence* solution.

INTRODUCTION

In the early days of developing Web-based user interfaces, the limitations of the HTML (hyper-text markup language) and browser functionality meant that Web application developers had to make significant sacrifices in usability in order to obtain the important benefits of user convenience and reduced cost of ownership.

Within a few years, the introduction of dynamic HTML began to change this situation for the better. By enabling significant local processing, and making it possible for the interface to react locally to user actions, the user experience could be made much better. For example, many common user errors, such as not providing data in a required field, could be prevented, rather than have the user submit a form, wait, and then be told what was wrong and correct it.

However, while both Internet Explorer 4 and Netscape Navigator 4 had DHTML support that exposed a document object model, with properties and methods for its manipulation, no provision was made for either model to be directly extensible by developers – a key requirement for object-oriented software engineering. Instead, the client-side portions of Web applications using DHTML were implemented as stand-alone event handlers invoking large blocks of procedural code. The result was poor separation of page content from behavioral logic.

With the introduction of DHTML behaviors in Internet Explorer 5.5, we can now obtain the benefits of DHTML without sacrificing good software engineering practice. The object-oriented framework that they provide enables clean separation of page content from behavior.

OBJECT-ORIENTED PROGRAMMING (OOP)

JavaScript is the programming language most often used to drive DHTML interfaces. It has often described been described as being object based, but not object oriented. In fact, properly written JavaScript code can meet the canonical criteria for OOP.

- Abstraction
- *Extracting the essential details about an item or group of items, while ignoring the unessential details.* (Edward Berard)
- Encapsulation
- Enclosing all parts of an abstraction within a container
- Information Hiding
- Hiding parts of the abstraction.
- Hierarchy
- Abstractions arranged in order of rank or level (inheritance)

Given that this is so, the question of why such a practice has not been widely adopted arises. There are several reasons.

- Unlike Java, and like C++, JavaScript is a language that supports both procedural/functional and OO programming styles. The programmer is not forced to write object-oriented code.
- The syntax of class definitions in JavaScript differs from Java, and so is not obvious to the Java developer. (The syntax of class usage is nearly identical.)
- It is easy to start writing code procedurally, and then just keep going that way as the project grows. Then a changeover is too painful and typically doesn't happen.
- There is a vast Web of procedural JavaScript out there (most of it bad) and relatively few OO examples to follow.
- Until recently, it was not possible to use the base document object classes as a starting point, and extend them. So what is normally one of the great motivations for using OOP, building upon and leveraging an existing, powerful class library, was absent.

OH, BEHAVE!

By using behaviors, it is possible for the first time to extend any class of base document objects (e.g. <input>, <select>) with customized functionality to meet the needs of the user interface.

A SIMPLE EXAMPLE

The following very basic example demonstrates the XML-based syntax of an HTML component (HTC) file, the type of file that specifies behaviors. It defines two behaviors that are invoked as event handlers, resulting in the appearance of an object changing when the mouse pointer is over it.

```
<PUBLIC:COMPONENT>

   <PUBLIC:ATTACH
          EVENT="onmouseover"
          ONEVENT="Hilite()" />
   <PUBLIC:ATTACH
          EVENT="onmouseout"
          ONEVENT="Restore()"  />

   <SCRIPT TYPE="text/javascript">

   // These variables exist in the name
   // scope of the object
   var normalColor, normalSpacing;

   function Hilite()
   {
     // Remember the normal color
     normalColor  = currentStyle.color;
     // Remember the normal spacing
     normalSpacing=
          currentStyle.letterSpacing;

     // Make it red
     runtimeStyle.color  = "red";
     // Increase letter spacing
     runtimeStyle.letterSpacing = 2;
   }

   function Restore()
   {
```

```
      // Restore color and spacing to
      // their previous values
      runtimeStyle.color = normalColor;
      runtimeStyle.letterSpacing =
           normalSpacing;
   }

   </SCRIPT>
</PUBLIC:COMPONENT>
```

Behaviors are bound to document objects through style rules that apply to them and identify an HTC file. For example, this button:

```
<button
   class="hoverChange"
   onclick="window.alert('Hi!');">
   Press me for a greeting
</button>
```

…could be linked to the highlighting behavior above by the following rule:

```
button.hoverChange
{
   behavior: url(hoverHighlight.htc);
}
```

Trivial examples of behaviors such as these are easy to find on the Internet. Really significant examples are hard to come by. The purpose of this paper is to share some real-world examples of behaviors that do some heavy lifting in a real application.

ADVANTAGES OF BEHAVIORS

Behaviors provide several significant advantages for implementing a customized Web UI in an object-oriented fashion.

- Behaviors support code encapsulation with a simple declarative syntax .
- Behaviors provide a means for extending base object functionality consistently across a project or product family.
- Behaviors isolate code from content, resulting in cleaner, more manageable pages and helping to separate "programmer tasks" from "page designer tasks" on the development team.
- With behaviors, it is much easier to change the implementation when you think of a better way, or a new way becomes available, without impacting the consumers of the behavior.
- Through behaviors it is much easier to re-use code. (Procedural JavaScript is notoriously "page-dependent" and difficult to use outside of its original development context).

USE OF BEHAVIORS IN THE *SAS® PROCESS INTELLIGENCE* USER INTERFACE

The behaviors described below are drawn from *SAS® Process Intelligence*. These are just two of approximately ten HTC files used in the solution, each of which implements multiple behaviors in the form of event handlers and methods. The page shown below uses both of the examples.

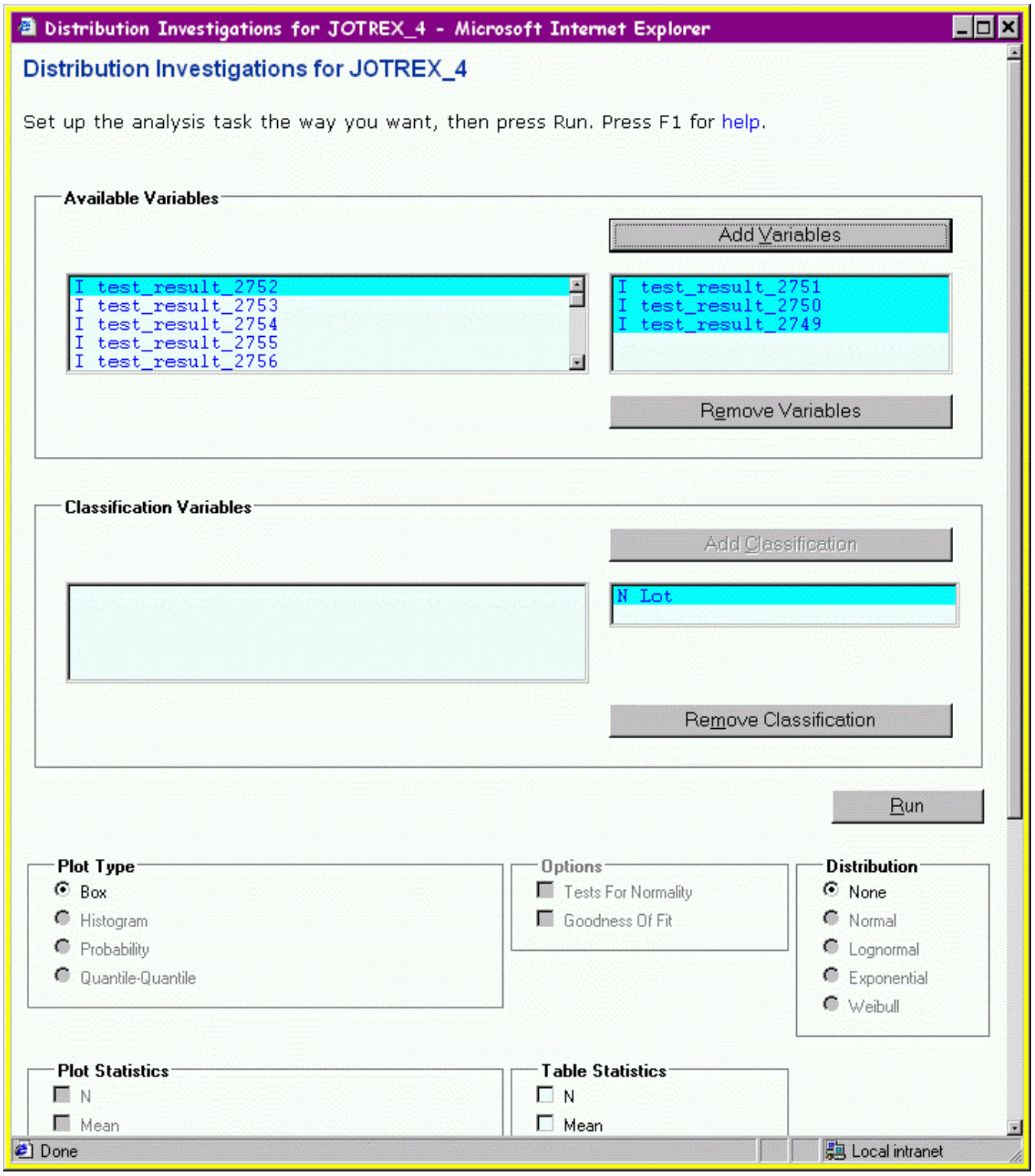

Figure 1

This page allows the user to set up a task called Distribution Investigations to be performed on a previously constructed data set. The following presentation elements and user interactions are mediated by the behaviors we will examine below:

SELF-POPULATING FIELD SETS

Introduced in HTML 4.0, field sets are very useful for visually associating a related set of elements in a form. The associated elements are enclosed in a labeled box, several of which are shown above.

By using the following behavior to automatically generate the markup for the desired content of a field set, much hand coding is eliminated and a consistent appearance always results.

This example also shows how to extend a class of document

objects (in this case the FIELDSET tag) with new methods.

```
<!--
          Generates lists of checkboxes or radio buttons, and supports disable() and
          enable() methods at the group level.  For sets of radio buttons, ensures that
          one is checked.

          How to use:

          <fieldset
                    label="statistics"
                    name="pk_s"
                    deferred
                    id="pk_s_set"
                    notify="statPickRequired"
                    type="checkbox"
                    choices="
                              box1:deferred;label text for box1:tooltip for box1;
                              box2:checked;label text for box2:tooltip for box2;"
                    >

                    (You can optionally include manual content here, such as instructions)

          </fieldset>

          If label is omitted, the fieldset is not enclosed in the typical border box.

          If deferred is specified, the entire fieldset is disabled (used for "future
          features").

          type may be "checkbox" or "radio"

          notify is optional and is the name of a function to call when selections change.
          This is useful to trigger voting behavior.

          The syntax for one control in the choices string is:
                    "name:pass-thru-attributes(optional);label-text:tool-tip-text(optional);"

          Mark Lumsden
          July 2001

          Copyright (C) 2001 SAS Institute, Inc. All rights reserved.

          Notice:
          The above copyright notice and this notice must appear in any whole or partial
          copy of this code and any related documentation. Permission to use, copy and
          modify the source code contained in this file is hereby granted, but is
          limited to customers of SAS with a valid license for the SAS software product
          with which this file was provided.  Except as expressly set forth herein, the
          terms of such license apply.
-->

<PUBLIC:COMPONENT
          LIGHTWEIGHT="true"
          NAME="fieldsetBehavior">

<PUBLIC:ATTACH
          EVENT="oncontentready"
          ONEVENT="init()" />

<PUBLIC:ATTACH
          EVENT="ondocumentready"
          ONEVENT="notifier()" />

<PUBLIC:ATTACH
          EVENT="onclick"
          ONEVENT="notifier()" />

<METHOD
          NAME="disable" />

<METHOD
          NAME="enable" />

<SCRIPT type="text/javascript">

// Default type is checkbox
var sControlType = element.type;

// Has notifier method been bound yet (if any)?
```

```
var bNotifyMethodResolved = false;

function init() {
   // Don't proceed with content generation if choices property not available
   if (element.choices == null) return;

   // Don't proceed with content generation if it has already been performed
   // Needed because modified content is remembered across page refreshes (!?)
   if (element.bProcessed) return;

   // Set processed flag
   element.bProcessed = true;

   // String in which we build up the control set
   var sTemp = "";

   // Get label
   var sLabel = element.label;

   if (sLabel != null) {
      sTemp += "<LEGEND>" + sLabel + "</LEGEND>";
   }
   else {
      // If no label, we don't want the default box border around the fieldset
      runtimeStyle.border = "none";
   }

   // Assign default id to fieldset, if not coded
   if (element.id == "") element.id = element.name + "_set";

   // Names of the choices
   var sChoices = element.choices;

   // Type of controls
   if (sControlType == null) sControlType = "checkbox";

   // Load them into an array for processing
   var aChoices = sChoices.split(";");

   var sHandle;
   var sId;
   var aControlText;
   var sToolTip;
   var sName = element.name;
   var sChecked;
   var sAttributes;

   if (element.deferred != null) {
      element.title = "These features are not available in this version";
      element.disabled = true;
   }

   // Write a checkbox or radio button for each choice
   for (var i=0; i < aChoices.length; i=i+2) {
      sAttributes = "";
      sHandle = aChoices[i];

      // Pass on deferred setting to children
      if (element.deferred != null) sAttributes += " deferred ";

      // Look for supplemental attributes
      aControlText = sHandle.split(":");

      if (aControlText[1] != null) {
         sAttributes = aControlText[1];
         sHandle = aControlText[0];
      }

      if (sHandle == "") sHandle = handle(aChoices[i+1]);

      sId = sName + "_" + sHandle;

      // window.alert(element.id + " " + sChecked);
      // Look for a tool tip (hover text)
      aControlText = aChoices[i+1].split(":");

      if (aControlText[1] == null)
         sToolTip = " ";
      else
         sToolTip = ' title="' + aControlText[1] + '" ';
```

```
      sTemp += '<input ' + sAttributes + sToolTip + ' name="' + sName +
         '" value="' + sHandle + '" id="' + sId + '" type="' + sControlType +
         '" label="' + aControlText[0] + '"><BR>';
   }

   // Insert generated content in front of current content (if any)
   element.innerHTML = sTemp + element.innerHTML;
}

// Notification to the element of how many contained boxes are checked
function notifier() {
   // Check to see if the element has a notify method specified for it
   if (element.notify == null) return;

   if (!bNotifyMethodResolved) {
      // Turn name of notification method into a reference to it
      window.eval("element.notify = window." + element.notify);
      bNotifyMethodResolved = true;
   }

   var iCount = 0;

   // Create an Enumerator for the elements
   var e = new Enumerator(element.children);

   // Count fieldset elements
   for (; !e.atEnd(); e.moveNext()) {
      if (e.item().checked) iCount++;
   }

   element.notify(iCount, element);
}

// Add a disable method to all fieldset objects
function disable(disableFlag) {
   // Default to disabling the form
   if (disableFlag == null) disableFlag = true;

   var fieldset = element;

   // Create an Enumerator for the elements
   var e = new Enumerator(fieldset.children);

   // Enumerate fieldset elements
   for (; !e.atEnd(); e.moveNext()) {
      if (e.item().tagName == "LABEL") {
         fieldset.children[e.item().htmlFor].disabled = disableFlag;
      }

      // Don't gray out the legend
      if (e.item().tagName != "LEGEND") e.item().disabled = disableFlag;
   }
}

// Add enable method
function enable() {
   element.disable(false);
}

</SCRIPT>
</PUBLIC:COMPONENT>
```

STATE MEMORY FOR CHECK BOXES AND RADIO BUTTONS

When a process engineer returns to a *SAS® Process Intelligence* form that she has used before, each check box and radio button appears in the state in which she last set it. All that is necessary to enable this automatic state memory for all check boxes and radio buttons in a page is for it to reference a style sheet containing a rule to associate the INPUT tag with a file containing the following behavior definitions. (INPUT types other than these are not affected in this example.)

```
<!--
          Behavior for all checkbox and radio button input elements.

                    State memory
                    Label generation
                    Voting on state of the control
                    Deferred property
                    Enable and disable methods

          Nothing special needs to be done to use state memory, besides associating
          this behavior with all INPUTs.
```

```
          To use automatic label generation, specify the desired label text like so:

          <input
                    type="checkbox"
                    name="mybox"
                    id="oBox1"
                    label="Please check this box">

          See the description of the voting framework (voting.js) for how components
          can cast votes on the availability of a fellow component in a form.

          A control can be marked as deferred to a later version.  Deferred means that the
          control will be disabled, and have a hover tip describing it as unavailable in the
          current version.

          <input
                    type="checkbox"
                    name="mybox"
                    deferred
                    id="oBox1"
                    label="Please check this box">


          Mark Lumsden
          July 2001

          Copyright (C) 2001 SAS Institute, Inc. All rights reserved.

          Notice:
          The above copyright notice and this notice must appear in any whole or partial
          copy of this code and any related documentation. Permission to use, copy and
          modify the source code contained in this file is hereby granted, but is
          limited to customers of SAS with a valid license for the SAS software product
          with which this file was provided.  Except as expressly set forth herein, the
          terms of such license apply.
-->

<PUBLIC:COMPONENT
          LIGHTWEIGHT="true"
          NAME="inputBehavior">

<PUBLIC:ATTACH
          EVENT="onclick"
          ONEVENT="save()" />

<PUBLIC:ATTACH
          EVENT="oncontentready"
          ONEVENT="init()" />

<PUBLIC:ATTACH
          EVENT="ondocumentready"
          ONEVENT="retrieve()" />

<METHOD
          NAME="disable" />

<METHOD
          NAME="enable" />

<METHOD
          NAME="vote" />

<SCRIPT type="text/javascript" SRC="../Util/voting.js">
</SCRIPT>

<SCRIPT type="text/javascript">

// Save the state of a radio button or checkbox
function save() {
   // Avoid needless saves at startup
   if (element.bInitializing) {
      element.bInitializing = false;
      return;
   }

   // id required as key to save state
   if (element.id == null) return;

   switch(element.type) {
      case "radio":
```

```
         // If radio button checked, save its id under its group's name
         if (element.checked) {
            window.document.setCookie(element.name,element.id);
         }

         break;

      case "checkbox":
         window.document.setCookie(element.id,element.checked);
   }
}

// Generate a label if requested
function init() {
   // Save title
   element.savedTitle = element.title;

   // Hand cursor for "clickable" controls
   if (element.type != "text") element.style.cursor = "hand";

   // Distinguish UI elements for deferred features
   if (element.deferred != null) {
      element.title = "This feature is not available in this version";
   }

   // Append label if requested
   if (element.label != null && !element.labeled) {
      if (element.id == "") element.id = element.name;

      element.labeled = true;

      element.insertAdjacentHTML("afterEnd",
         '<LABEL FOR="' + element.id + '" ID="' + element.id +
         '_label" TITLE="' + element.title + '"> ' + element.label + '</LABEL>');
   }

   // Distinguish UI elements for deferred features
   if (element.deferred != null) {
      element.disable();
   }
}

// Recall the state of a radio button or checkbox, and fire an event to "announce" it
function retrieve() {
   // id required to look up state
   if (element.id == null) return;

   // If already checked in the source code, act on that
   if (element.checked) {
      // Save state
      save();

      // Fire appropriate event, to trigger anything that might be listening downstream
      element.bInitializing = true;
      element.fireEvent("onclick");

      return;
   }

   var sFound;

   switch(element.type) {
      case "radio":
         // Look for memory of this radio button set
         sFound = window.document.getCookie(element.name);

         if (sFound == null) {
            // No memory of this set of radio buttons -- check & save this first one
            element.checked = true;
            save();
         }
         else {
            // Check it if it matches memory
            element.checked = (sFound == element.id);
         }

         // Fire appropriate event, to trigger anything that might be listening downstream
         if (element.checked) {
```

```
            element.bInitializing = true;
            element.fireEvent("onclick");
         }

         break;

      case "checkbox":
         element.checked = (window.document.getCookie(element.id) == "true");

         // Fire appropriate event, to trigger anything that might be listening downstream
         element.bInitializing = true;
         element.fireEvent("onclick");
         break;

      default:
         return;
   }
}

// Add a disable method to all input objects
function disable() {
   element.disabled = true;

   // Find label, if any
   var label = this.getLabel();

   if (label != null) {
      label.title = element.title;
      label.disabled = true;
   }
}

// Add an enable method to all input objects
function enable() {
   // Don't enable deferred elements
   if (element.deferred) return;

   element.disabled = false;

   // Find label, if any
   var label = this.getLabel();

   if (label != null) {
      label.title = element.title;
      label.disabled = false;
   }
}

// Find the label, if any, of a form element.  This relies on the convention of having the id of a
// lable being the id of the control with "_label" suffix.
function getLabel() {
   // Create an Enumerator for the form elements
   var e = new Enumerator(this.form.all);

   // Enumerate form children
   for (; !e.atEnd(); e.moveNext()) {
      if (e.item().id == this.id + "_label") {
         return e.item();
      }
   }

   // Not found
   return null;
}

</SCRIPT>
</PUBLIC:COMPONENT>
```

CONCLUSION

Early attempts at Web-based user interfaces had to make significant sacrifices in usability in order to obtain the important benefits of user convenience and reduced cost of ownership. The introduction of dynamic HTML made it possible to develop Web-based interfaces that were much more functional and interactive.

However, early attempts at adopting DHTML typically resulted in a poor result from a software engineering perspective: in-line event handlers and large quantities of difficult to maintain procedural code, often with poor separation of page content and behavioral logic.

With the introduction of DHTML behaviors, we can now obtain the

benefits of DHTML without sacrificing good software engineering practices. The object-oriented framework that they provide enables clean separation of page content from behavior. This separation will become even more important as we move toward interfaces where much of the content is generated based on XML.

REFERENCES

Behavioral Extensions to CSS
http://www.w3.org/TR/becss
DHTML Behaviors
http://msdn.microsoft.com/workshop/author/behaviors/overview.asp
HTML Component (HTC) Reference
http://msdn.microsoft.com/workshop/components/htc/reference/htcref.asp

ACKNOWLEDGMENTS

The author wishes to acknowledge the support and encouragement of the work described here by manager Jon Weisz and all of the members of *SAS® Process Intelligence* development team.

CONTACT INFORMATION

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

David Ostiguy
SAS
SAS Campus Drive
Cary, N.C. 27513

Work Phone: 919 531 0224
e-mail: David.Ostiguy@sas.com