SAS/AF® and Internet Explorer:
Capturing Web Forms Data Entry in Thick Client Applications

Richard A. DeVenezia, Independent Consultant

ABSTRACT

This paper presents a new twist in data entry. The Microsoft Web Browser component is used to display a hypertext markup language (HTML) file containing a FORM element. Data submitted from the browser is accessed directly in SAS/AF without needing web servers or other middleware applications. HTML with a small amount of CSS is a great way to renovate tired looking input screens. You can add a touch of Javascript for dynamic adjustments prior to submittal.

WEB PAGES FOR INPUT

A large percentage of computing devices come with Microsoft Internet Explorer (IE) already installed. Additionally, many corporations have policies dictating its use. The IE installation includes an OLE COM object named “Microsoft Web Browser”. The object, or control, can be used in other Windows applications that function as an ActiveX host. SAS is one such application. You can take advantage of the IE installation by using the web browser to deliver inputs to your SAS/AF applications.

Some reasons for utilizing the browser:

1. Specification – The specifications of a project may require its use.
2. Design Team – Available personnel have expertise in web design and technologies.
3. CSS, Javascript – The technologies available to a web page designer allow newer tools and techniques to be utilized with an input form.
4. Development – Program experimental handling of inputs entered on production web forms.
5. Interception – Capture inputs destined for a remote server and using them for your own purposes.

A thick client is a SAS Display Manager System (DMS) session running a SAS/AF Frame application. A thick client can usually connect directly to data stores and often has a substantial level of trust within a systems infrastructure. A thin client is a web browser. The browser connects to a web server and is often delegated a low level of trust. Web browsing is ubiquitous.

The HTML author uses the FORM element to contain the named user interface (UI) elements and fields that encompass the input. When you submit a form the browser gathers the names and values of the inputs for transmittal. The METHOD tag dictates the way in which information is to be delivered to a web server in an encoded string comprised of **name**=**value** pairs delimited by ampersands (&).

GET and POST are two common methods for delivering inputs via hypertext transfer protocol (HTTP) from a browser to server. Inputs are better known as parameters in the web programming world.

GET parameters are sent via name value pairs following a question mark (?) after a pages Uniform Resource Identifier (URI).

```
http://www.google.com/search?hl=en&q=sas+"sn-015762"
```

The name value pairs portion of the URI is also known as the query string. Browsers typically limit a query string to a few thousand characters.

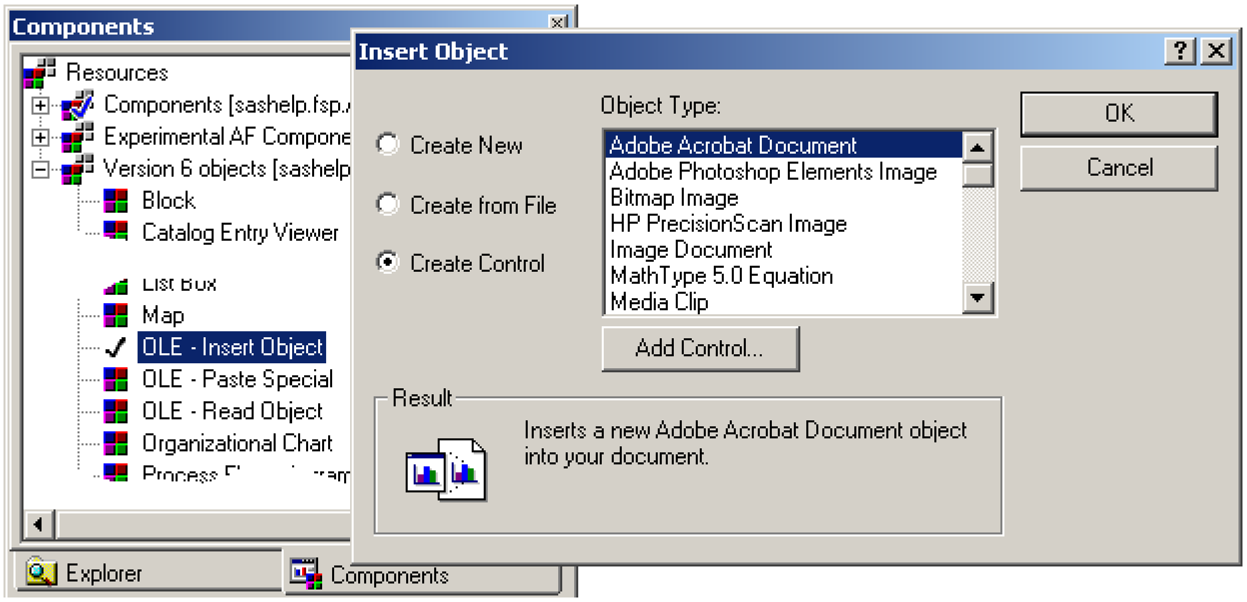
POST parameters are delivered in the same name value pair manner, however the content is passed to the server via a stream that is not visible to the person using the browser. The amount of data that can be sent by POST is essentially unlimited when compared to GET.

The sample URI has these parameters:

Variable		
Name	Value	Purpose
hl	**en**	Host language
q	**sas "sn-015762"**	Search terms

SAS/AF

SAS/AF software has been an ActiveX host since version 6.12. The ability to utilize ActiveX objects in SAS/AF Frames continues to this day. BUILD a new frame. The component palette will appear. Open the "Version 6 Objects" node in the palette and select "OLE – Insert Object":



Inserts a new Adobe Acrobat Document object into your document.

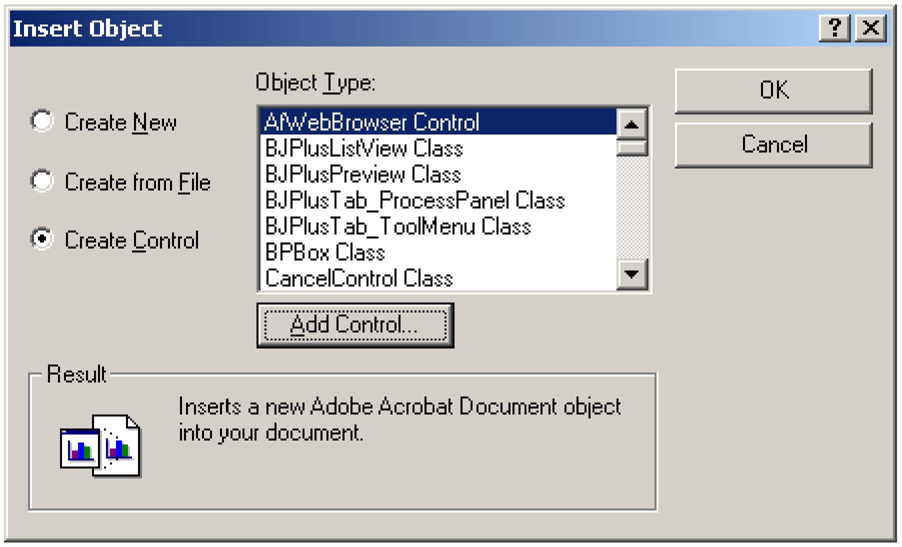
The "Insert Object" dialog is as old as version 6.12 and it can get out of sync with the real state of objects in the Windows system. In the image above, the "Create Control" selection was made, but the Object Types listed are still those of the "Create New" choice!

Experience teaches that when the "Add Control" button is clicked and an OCX in the folder "%windir%\system32" is selected, the dialog will update itself and show the correct list of insertable controls.

The author finds comct232.ocx reliably forces the update.

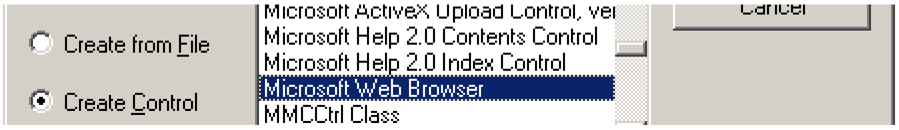
However, be prepared to select several OCXs before the dialog will update.

The SAS dialog for inserting objects is different from the Insert/Object dialog shown in a Microsoft Office application. The legacy SAS component may be raising an obsolete system dialog, which in turn could be causing the issues regarding the object type list.



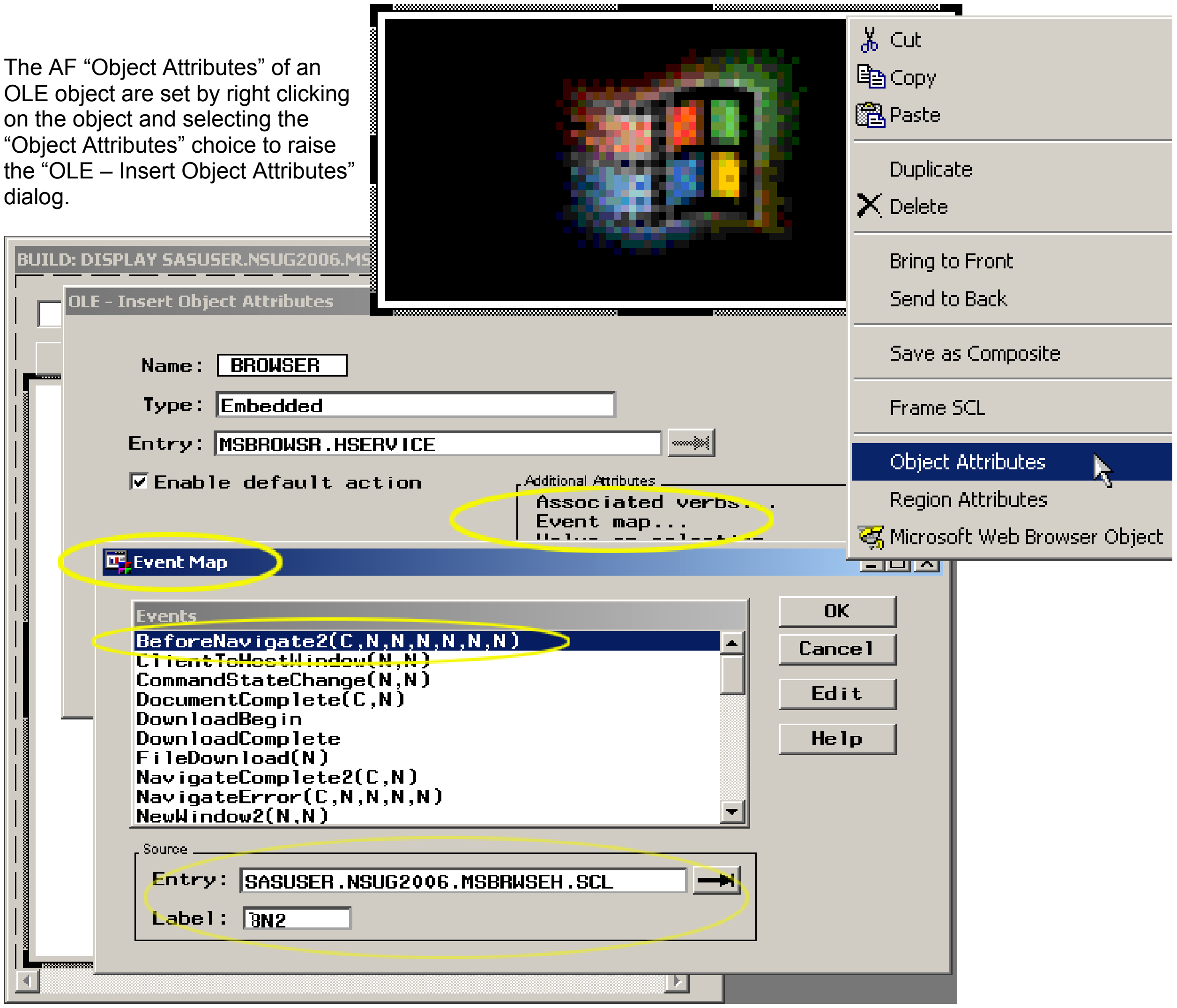
Inserts a new Adobe Acrobat Document object into your document.

Scroll down to the Microsoft Web Browser control and click OK.


Create from File
Create Control
Microsoft ActiveX Upload Control, ver
Microsoft Help 2.0 Contents Control
Microsoft Help 2.0 Index Control
Microsoft Web Browser
MMCCtrl Class
Cancel

What? It's not listed on your system? There is a reason for that. Newer versions of Windows do not register the "Microsoft Web Browser" component as installable, even though it still is. SAS Note SN-015762 discusses the "Microsoft Web Browser OLE Component is not readily available in SAS/AF" issue and has instructions on how to update your Windows registry.

The AF "Object Attributes" of an OLE object are set by right clicking on the object and selecting the "Object Attributes" choice to raise the "OLE – Insert Object Attributes" dialog.



Name: BROWSER
Type: Embedded
Entry: MSBROWSR.HSERVICE
☑ Enable default action

```
BeforeNavigate2(C,N,N,N,N,N,N)
ClientToHostWindow(N,N)
CommandStateChange(N,N)
DocumentComplete(C,N)
DownloadBegin
DownloadComplete
FileDownload(N)
NavigateComplete2(C,N)
NavigateError(C,N,N,N,N)
NewWindow2(N,N)
```

Entry: SASUSER.NSUG2006.MSBRWSEH.SCL
Label: BN2

The figure shows the name of the object has been changed to BROWSER and some event mapping has been configured.

OLE OBJECTS

Every ActiveX object has a set of properties, methods and events. Properties are used to configure the behavior of the object; they can also be used to exchange data between the object and the object user. Methods of an object are typically used to cause the object to perform action corresponding to some verb idea – think Run, Do, Collect, Open, Close. Events are signals raised by the object that other objects can listen for. When the signal is observed the listener can run some method in response.

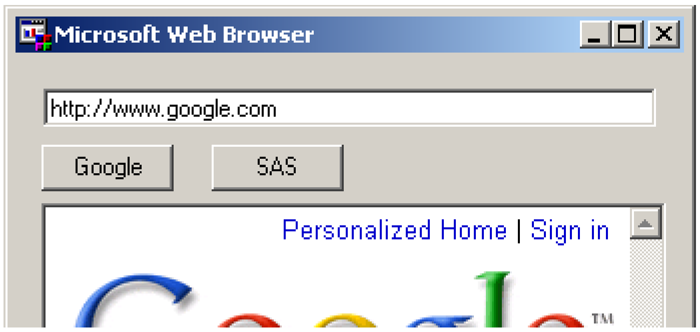
EVENT MAPPING

The “Event Mapping“ dialog is used to configure how the automatic listener, that is part of the “OLE – Insert Object” class, responds to events. In essence you are configuring a dispatch table that hands off or forwards the response action to an SCL method of your choosing.

MSBROWSER DEMO

A SAS/AF sample application demonstrating the “Microsoft Web Browser“ has been constructed and can be downloaded from the author's website. It is comprised of four parts:

1. MSBROWSER.FRAME – Layout containing a “Microsoft Web Browser”
2. MSBROWSER.SCL – SCL for frame
3. MSBRWS**EH**.SCL – **E**vent **H**andlers for events raised
4. MSBROWSR.HSERVICE – SAS System object containing information about an OLE object



```
init:
  declare Object oBrowser;  * o prefix is a nod to MS naming conventions;
  _frame_._getWidget ('Browser', oBrowser);
return;

URL: oBrowser._do ('Navigate', URL.text); return;

bmSAS:
bmGoogle:
  declare object w;
  _frame_._getCurrentWidget(w);
  URL.text = w.toolTipText;
  link URL;
return;
```

The buttons tool tip text is the web address to display when the button is clicked. The Microsoft Developers Network (MSDN) website (http://www.msdn.com) fully documents the “Microsoft Web Browser”. It is there that one learns the **Navigate** method is used to display a new page, and that the **BeforeNavigate2** event is raised before the control displays a new page.

EVENT HANDLER

```
BN2:
public method
  pDisp:num
  Url:char(1024)
  Flags:char(1024)
  TargetFrameName:char(1024)
  PostData:num
  Headers:num
  Cancel:num
;

  put 'BeforNavigate2 event handler';
  put pDisp=; put Url=; put Flags=; put TargetFrameName=;
  put PostData=; put Headers=; put Cancel=;endmethod;
```

The astute reader will note that the BN2 signature (N,C,C,C,N,N,N) does not match the signature listed in the Event Mapping dialog (C,N,N,N,N,N,N). This discrepancy arises due to the Microsoft types that comprise the true signature (*in the parlance of Visual Basic)*:

```
ByVal pDisp As Object, _
ByRef Url As Variant, _
ByRef Flags As Variant, _
ByRef TargetFrameName As Variant, _
ByRef PostData As Variant, _
ByRef Headers As Variant, _
ByRef Cancel As Boolean
```

SAS has to make a best guess at what SAS type the Microsoft types should be mapped to. The BN2 signature in the implementation was arrived at by iteration. Each of the argument types were modified and tested until the invocation ran without a halting run-time error.

Even though the BN2 method will run without halting, there are still OLE errors and the **PostData** is inaccessible. When the MSBROWSER.FRAME is run, these messages can appear in the log.

```
OLE: One of the parameters is not a valid type.
OLE: One of the parameters is not a valid type.
```

```
OLE Error: 80070057.  The parameter is incorrect.
OLE Error: 80070057.  The parameter is incorrect.
BeforNavigate2 event handler
pDisp=0
Url=http://www.google.com/search?hl=en&q=SN-015762&btnG=Google+Search
Flags=
TargetFrameName=
PostData=0
Headers=0
Cancel=0
```

```
OLE Error: 80070057.  The parameter is incorrect.
OLE Error: 80070057.  The parameter is incorrect.
BeforNavigate2 event handler
pDisp=0
Url=http://support.sas.com/techsup/unotes/SN/015/015762.html
Flags=
TargetFrameName=
PostData=0
```

```
Headers=0
Cancel=0
```

The GET parameters can be extracted from the accessible URL, but the lack of PostData and log showing OLE errors makes this solution highly undesirable. Is there some other solution?

ADAPTER OCX

Adapter: “Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces” [1] A correctly implemented adapter will not error when used in SAS/AF.

OCX: Once upon a time it was **O**bject Linking and Embedding (OLE) **C**ontrol E**x**tension, but is now known as ActiveX.

Before discussing the adapter, what features are really needed ?

1. Navigate to a page
2. Signal when a new page is to be navigated to
3. Retrieve GET and POST parameters *prior* to any actual communication between browser and server.

Corresponding implementation design:

1. URL property
2. OnBeforeNavigate event
3. Read-only Get related properties: GetCount, GetName[index], GetValue[index]
   Read-only Post related properties: PostCount, PostName[index], PostValue[index]

DEVELOPMENT

Delphi 2006 Professional was chosen for compiling the adapter OCX. The steps for preparing the project are listed here:

1. Create ActiveX Library “OleObjectsForSasAf.ocx”
    1. Create Active Form “AfWebBrowser”
        1. Layout Microsoft Web Browser (MWB)
        2. Add properties
            1. URL (read/write) – when set Navigate will be performed automatically
            2. GetCount (read-only)
            3. GetName (read-only, indexed)
            4. GetValue (read-only, indexed)
            5. PostCount (read-only)
            6. PostName (read-only, indexed)
            7. PostValue (read-only, indexed)
        3. Add event

1 Design Patterns – Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides; p.139

        1. OnBeforeNavigate

    4. Add event handler for MWB OnBeforeNavigate2

        1. Stores GET and POST parameters in TStringList for retrieval later via properties

        2. Raises forms OnBeforeNavigate (which SAS can respond to)

Details of the property and event implementation are beyond the scope of this paper – interested readers can download the Delphi project from the author's website.

INSTALLATION

Download and register the OCX. In the Windows Run box (Start-Run), issue this command

Open: **`regsvr32 OleObjectsForSasAf.ocx`**

The OCX was developed to automatically register itself an insertable component.

AFBROWSER DEMO

A SAS/AF sample application demonstrating the "AfWebBrowser" has been constructed and can be downloaded from the author's website. It it comprised of several parts:

1. AFBROWSER.FRAME – Layout containing an "AfWebBrowser". Returns SCL lists of inputs..

2. AFBROWSER.SCL – SCL for frame.

3. AFBRWS**EH**.SCL – **E**vent **H**andlers for events raised by the AfWebBrowser.

4. AFBROWSR.HSERVICE – SAS System object containing information about an OLE object.

5. GETWEBINPUT.SCL – Displays the AFBROWSER frame and logs the returned SCL lists.

FRAME CODE

The frame is passed the URL (page) to display and the SCL list that will receive the inputs entered in the FORM. The "OLE – Insert Object" object is named "Browser" and contains an AfWebBrowser. For OLE controls, the AF executor does not automatically run the object named section of code. Program flow is forced through the section by a method call in the OLE controls event handler.

The event handler, and thus the Browser section, will run once when the URL is initially displayed, and once when the user submits the web form. The browserCount variable is used to track this program flow.

```
entry
  inUrl:input:char
  outParameters:update:List;

init:
  declare Object oBrowser;  * o prefix is a nod to MS naming conventions;
  _frame_._getWidget ('Browser', oBrowser);

  rc = setNitemL (oBrowser, {}, 'GET');
  rc = setNitemL (oBrowser, {}, 'POST');

  oBrowser._setProperty('URL', inURL);
return;

Browser:
  browserCount + 1;
  if browserCount = 1 then return;

  if listlen (outParameters) >= 0 then do;
    get  = getNitemL (oBrowser, 'GET');
    post = getNitemL (oBrowser, 'POST');
    rc = setNitemL (outParameters, get,  'GET');
    rc = setNitemL (outParameters, post, 'POST');
  end;

  * after capturing the data, shut down the browser frame;
  * callee is responsible for deleting the SCL lists it receives upon return;
  call execcmd ('CANCEL');
return;
```

EVENT HANDLER

The “OLE – Insert Object” objects event handler copies the name value pairs of the URL inputs, captured by AfWebBrowser, to SCL lists presumed to be attached to the object. *Note: a further refinement would be the creation of a subclass that ensures the SCL lists exists.*

```
OBN:
public method;

  * _self_ is an OLE - Insert Object that contains an AfWebBrowser;
  *
  * presume _self_ already has two attached lists named GET and POST
  * for storing the inputs;

  Get  = GetNitemL (_self_, 'GET',  1,1,0);
  Post = GetNitemL (_self_, 'POST', 1,1,0);

  rc = clearlist (Get);
  rc = clearlist (Post);

  declare char(200) name value ;

  * Copy GET parameters to SCL list;
  _self_._getProperty('GetCount', count);
  do i = 1 to count;
    _self_._getProperty('GetName',  i-1, name); * Fetch pair from OLE object;
    _self_._getProperty('GetValue', i-1, value);

    rc = insertC (Get, value, -1, name); * Store pair in SCL list;
  end;

  * Copy POST parameters to SCL list;
  _self_._getProperty('PostCount', count);
  do i = 1 to count;
    _self_._getProperty('PostName',  i-1, name); * Fetch pair from OLE object;
    _self_._getProperty('PostValue', i-1, value);

    rc = insertC (Post, value, -1, name); * Store pair in SCL list;
  end;
  _self_._objectLabel(); * force program flow through object label in frame SCL;
endmethod;
```

TESTING – GETWEBINPUT.SCL

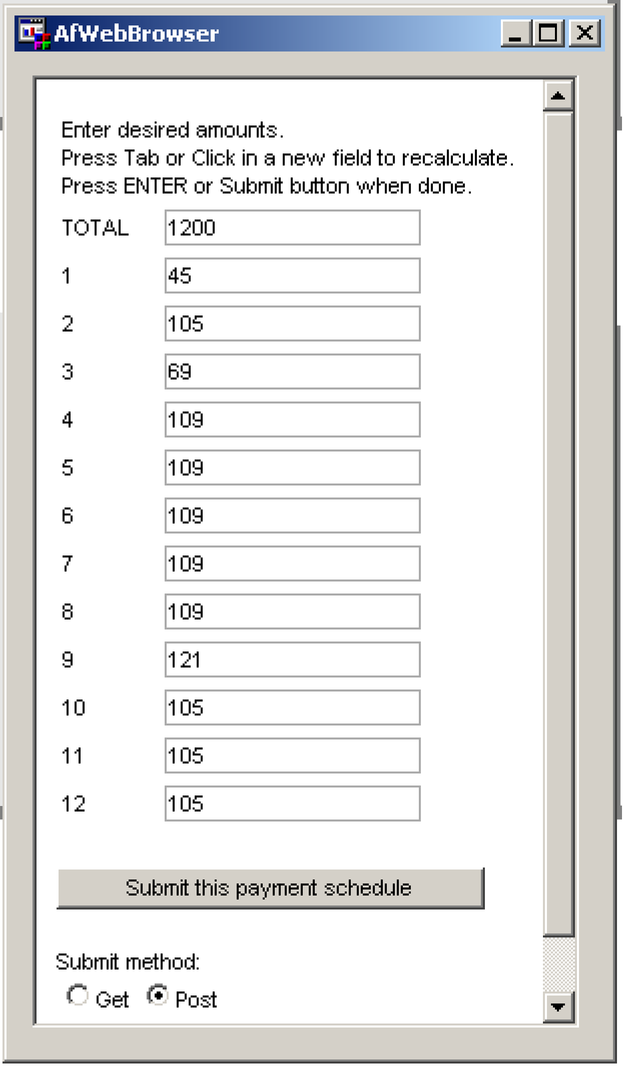
The AFBROWSER frame is called, passing in a test page to display, and the SCL list to receive the inputs. Putlist() is used to show the list in the Log window. Note: the page allows the submit method to be chosen. This is only for testing purposes.

```
init:
  declare List input = {};
  call display
  ( 'AFBROWSER.FRAME'
  , 'http://www.devenezia.com/papers/nesug-2006/Payments.html'
  , input
  );
  call putlist (input,'',0);
return;
```

TESTING – VERIFICATION

An examination of the log shows that the parameters received by the SCL application matches the inputs made in the web page.

```
(GET=()[18557]
 POST=(TOTALAMOUNT='1200'
       AMOUNT1='45'
       AMOUNT2='105'
       AMOUNT3='69'
       AMOUNT4='109'
       AMOUNT5='109'
       AMOUNT6='109'
       AMOUNT7='109'
       AMOUNT8='109'
       AMOUNT9='121'
       AMOUNT10='105'
       AMOUNT11='105'
       AMOUNT12='105'
       )[18567]
 )[5]
```



Enter desired amounts.
Press Tab or Click in a new field to recalculate.
Press ENTER or Submit button when done.

TOTAL	1200
1	45
2	105
3	69
4	109
5	109
6	109
7	109
8	109
9	121
10	105
11	105
12	105

Submit method:
Get [✓ Post]

CONCLUSION

SAS/AF frames have been able to use OCX libraries for many releases. The hosting of these libraries is not fully compliant due to Windows variable types that do not map to the SAS character or numeric types. Adapters can be written to overcome interfacing problems.

The paper demonstrates the specific case of an adapter that allows a frame to interact with the Microsoft Web Browser in an error free way. A stand alone SAS session that obtains user input from a web page is a new twist that can be put to creative uses.

The OCX discussed in this paper can be downloaded from the author's website. Visit http://www.devenezia.com and follow the link to Papers.

CONTACT INFORMATION

Richard A. DeVenezia
9949 East Steuben Road
Remsen, NY 13438
(315) 831-8802
http://www.devenezia.com/contact.php

Richard is an independent consultant who has worked extensively with SAS products for over fifteen years. He has presented at previous SUGI, NESUG and SESUG conferences. Richard is interested in learning and applying new technologies. He is a SAS-L Hall of Famer and remains an active contributor to SAS-L.

This document was produced using OpenOffice.org Writer.