

RE: Controlling Modal Dialogs (Solution)

from
the window. At this point the modal WebBrowser sets the event that tells
the calling browser that it can continue and the modal WebBrowser is closed.

Digest of the code:

in the constructor, each browser ObjectForScripting object is initialized:

```
public CustomWebBrowser(BrowserRemotingProxy proxy)
: base()
{
// Force the Document to be created
this.DocumentText = "<html><head></head><body></body></html>";
this.IE.RegisterAsBrowser = true;

m_ObjectForScripting = new ScriptingCallableObject(this);
this.ObjectForScripting = m_ObjectForScripting;
}
```

The ScriptingCallableObject is the object that implements the
showModalDialog and close methods.

```
[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class ScriptingCallableObject
{
public ScriptingCallableObject(CustomWebBrowser browser)
{
m_WebBrowser = browser;
}

#region IHTMLWindow2 Members
public object showModalDialog(string dialog, object varArgIn,
string varOptions)
{
object retval = null;
try {
ShowModalDialogEventArgs eventArgs = new
ShowModalDialogEventArgs(dialog, varArgIn, varOptions);
m_WebBrowser.OnShowModalDialog(eventArgs);
retval = eventArgs.ReturnValue;
}
catch (Exception ex) {
// Never throw exception back to script land!!!
}
return retval;
}

public void close()
{
try {
m_WebBrowser.OnClose();
}
}
```

RE: Controlling Modal Dialogs (Solution)

```
catch (Exception ex) {  
    // Never throw exception back to script land!!!  
}  
}  
#endregion  
#region Private DataMembers  
private CustomWebBrowser m_WebBrowser;  
#endregion  
}
```

Each browser sinks the DWeb interface and the implementation of the DocumentComplete event is as follows:

```
#region DWebBrowserEvents2Implementation class  
public class DWebBrowserEvents2Implementation : DWebBrowserEvents2  
{  
    public DWebBrowserEvents2Implementation(CustomWebBrowser browser)  
    {  
        m_Browser = browser;  
    }  
}
```

```
#region DWebBrowserEvents2 Members
```

```
[DispId(259)]  
public void DocumentComplete(object pDisp, ref object URL)  
{  
    if (URL.ToString() != "about:blank") {  
        IHTMLDocument2 doc2 = m_Browser.IE.Document as  
        IHTMLDocument2;  
        // override the showModalDialog function  
        doc2.parentWindow.execScript("window.showModalDialog =  
function(dialog, varArgIn, varOptions){ return  
window.external.showModalDialog(dialog, varArgIn, varOptions); }; }",  
"javascript");  
        if (m_Browser.m_isModalDialog) {  
            doc2.parentWindow.execScript("window.close =  
function(){ window.external.close(); }", "javascript");  
            PutPropertyToIDispatch(doc2.parentWindow,  
"dialogArguments", m_Browser.m_ModalDialogVarArgIn);  
            //PutPropertyToIDispatch(doc2.parentWindow,  
"dialogTop", m_Browser.m_ModalDialogVarArgIn);  
            //PutPropertyToIDispatch(doc2.parentWindow,  
"dialogLeft", m_Browser.m_ModalDialogVarArgIn);  
            //PutPropertyToIDispatch(doc2.parentWindow,  
"dialogHeight", m_Browser.m_ModalDialogVarArgIn);  
            //PutPropertyToIDispatch(doc2.parentWindow,  
"dialogWidth", m_Browser.m_ModalDialogVarArgIn);  
            //PutPropertyToIDispatch(doc2.parentWindow,  
"returnValue", m_Browser.m_ModalDialogVarArgIn);  
        }  
    }  
}
```

RE: Controlling Modal Dialogs (Solution)

```
}
```

When showModalDialog is called from script, the call is directed to the ScriptingCallableObject.showModalDialog which in turn calls the WebBrowser derived object CustomWebBrowser.OnShowModalDialog().

```
protected virtual void OnShowModalDialog(ShowModalDialogEventArgs  
eventArgs)  
{  
eventArgs.ReturnValue = this.ShowModalDialog(eventArgs.Dialog,  
eventArgs.VarArgIn, eventArgs.VarOptions);  
}
```

```
private ManualResetEvent m_ModalDialogDone = null;  
private bool m_isModalDialog = false;  
private object m_ModalDialogVarArgIn = null;  
private string m_ModalDialogVarOptions = null;  
private object m_ModalDialogReturnValue = null;
```

```
internal object ShowModalDialog(string dialog, object varArgIn,  
string varOptions)  
{  
CustomWebBrowser modalBrowser =  
Program.wrapperForm.CreateNewBrowserAndTab();  
modalBrowser.InitAsModalDialog(varArgIn, varOptions);  
modalBrowser.Navigate(dialog);  
while (!modalBrowser.m_ModalDialogDone.WaitOne(100, false)) {  
try {  
Application.DoEvents();  
}  
catch (Exception ex) {  
}  
}  
return modalBrowser.m_ModalDialogReturnValue;  
}
```

```
private void InitAsModalDialog(object varArgIn, string varOptions)  
{  
m_isModalDialog = true;  
m_ModalDialogDone = new ManualResetEvent(false);  
m_ModalDialogVarArgIn = varArgIn;  
m_ModalDialogVarOptions = varOptions;  
IHTMLDocument2 doc2 = this.IE.Document as IHTMLDocument2;  
IHTMLWindow2 win2 = doc2.parentWindow;  
win2.opener = win2;  
}
```

When script in the modal WebBrowser calls the close() method, the call is directed to the ScriptingCallableObject.close() method, which in turn calls the CustomWebBrowse.OnClose() method.

RE: Controlling Modal Dialogs (Solution)

```
protected virtual void OnClose()
{
    IHTMLDocument2 doc2 = this.IE.Document as IHTMLDocument2;
    if (m_isModalDialog) {
        m_ModalDialogReturnValue =
        GetPropertyFromIDispatch(doc2.parentWindow, "returnValue");
        m_ModalDialogDone.Set();
    }
    doc2.parentWindow.close();
}
```

Notice the use of the functions `GetPropertyFromIDispatch` in the `OnClose` method and `PutPropertyToIDispatch` in the `DocumentComplete` handler. These methods are implemented as follows:

```
private static object GetPropertyFromIDispatch(object dispObject,
string propertyName)
{
    object retval = null;
    IDispatch winDisp = dispObject as IDispatch;
    Guid guid = Guid.Empty;

    tagDISPPARAMS gdispparams1 = new tagDISPPARAMS();
    gdispparams1.rgvarg = IntPtr.Zero;
    gdispparams1.cArgs = 0;
    gdispparams1.rgdispidNamedArgs = IntPtr.Zero;
    gdispparams1.cNamedArgs = 0;

    string[] rgpszNames = new string[] { propertyName };
    int[] rgDispId = new int[] { -1 };
    winDisp.GetIDsOfNames(ref guid, rgpszNames, 1, 0, rgDispId);
    object[] pVarResult = new object[1];

    if (winDisp.Invoke(rgDispId[0], ref guid, 0,
(int)Win32.InvokeCallType.DISPATCH_PROPERTYGET, gdispparams1, pVarResult, new
tagEXCEPINFO(), null) == 0) {
        retval = pVarResult[0];
    }
    return retval;
}

private static void PutPropertyToIDispatch(object dispObject, string
propertyName, object value)
{
    IDispatch winDisp = dispObject as IDispatch;
    IDispatchEx winDispEx = winDisp as IDispatchEx;
    Guid guid = Guid.Empty;

    tagDISPPARAMS gdispparams1 = new tagDISPPARAMS();
    tagEXCEPINFO gexcepinfo1 = new tagEXCEPINFO();
```

RE: Controlling Modal Dialogs (Solution)

```
IntPtr ptr1 = Marshal.AllocCoTaskMem(0x10);
Win32.VariantInit(new HandleRef(null, ptr1));
Marshal.GetNativeVariantForObject(value, ptr1);
gdispparams1.rgvarg = ptr1;
gdispparams1.cArgs = 1;
gdispparams1.rgdispidNamedArgs = IntPtr.Zero;
gdispparams1.cNamedArgs = 0;

int rgDispId = -1;
winDispEx.GetDispID(propertyName,
(uint)FlagsDispIDEx.fdexNameEnsure, out rgDispId);
try {
int rc = winDisp.Invoke(rgDispId, ref guid, 0,
(int)Win32.InvokeCallType.DISPATCH_PROPERTYPUT, gdispparams1, null,
gexcepinf1, null);
string text1 = null;
if ((rc == -2147352567) && (gexcepinf1.scode != 0)) {
rc = gexcepinf1.scode;
text1 = gexcepinf1.bstrDescription;
}
switch (rc) {
case 0:
case 1:
case -2147221492:
case -2147467260:
return;
}
if (text1 == null) {
text1 = String.Format("PutPropertyToIDispatch(object,
\"{0}\", {1}) with error {2}", propertyName, value, rc);
}
throw new ExternalException(text1, rc);
}
finally {
Win32.VariantClear(new HandleRef(null, ptr1));
Marshal.FreeCoTaskMem(ptr1);
}
}
```

The external COM declarations are as follows:

```
[ComImport, InterfaceType(ComInterfaceType.InterfaceIsUnknown),
Guid("00020400-0000-0000-C000-000000000046")]
public interface IDispatch
{
int GetTypeInfoCount();
[return: MarshalAs(UnmanagedType.Interface)]
ITypeInfo GetTypeInfo([In, MarshalAs(UnmanagedType.U4)] int iTInfo,
[In, MarshalAs(UnmanagedType.U4)] int lcid);
[PreserveSig]
int GetIDsOfNames([In] ref Guid riid, [In,
MarshalAs(UnmanagedType.LPArray)] string[] rgszNames, [In,
```

RE: Controlling Modal Dialogs (Solution)

```
MarshalAs(UnmanagedType.U4)] int cNames, [In, MarshalAs(UnmanagedType.U4)] int
int lcid, [Out, MarshalAs(UnmanagedType.LPArray)] int[] rgDispId);
[PreserveSig]
int Invoke(int dispIdMember, [In] ref Guid riid, [In,
MarshalAs(UnmanagedType.U4)] int lcid, [In, MarshalAs(UnmanagedType.U4)] int
dwFlags, [In, Out] tagDISPPARAMS pDispParams, [Out,
MarshalAs(UnmanagedType.LPArray)] object[] pVarResult, [In, Out] tagEXCEPINFO
pExcepInfo, [Out, MarshalAs(UnmanagedType.LPArray)] IntPtr[] pArgErr);
}
#endregion
#region IDispatchEx Interface
public enum FlagsDispIDEx : uint
{
fdexNameCaseSensitive = 0x00000001,
fdexNameEnsure = 0x00000002,
fdexNameImplicit = 0x00000004,
fdexNameCaseInsensitive = 0x00000008,
fdexNameInternal = 0x00000010,
fdexNameNoDynamicProperties = 0x00000020,
}
[ComImport, Guid("A6EF9860-C720-11D0-9337-00A0C90DCAA9"),
TypeLibType((short)0x1000)]
public interface IDispatchEx
{
[MethodImpl(MethodImplOptions.InternalCall, MethodCodeType =
MethodCodeType.Runtime)]
void GetDispID([In, MarshalAs(UnmanagedType.BStr)] string bstrName,
[In] uint grfdex, out int pid);
[MethodImpl(MethodImplOptions.InternalCall, MethodCodeType =
MethodCodeType.Runtime)]
void RemoteInvokeEx([In] int id, [In] uint lcid, [In] uint dwFlags,
[In] ref System.Runtime.InteropServices.ComTypes.DISPPARAMS pdp,
[MarshalAs(UnmanagedType.Struct)] out object pvarRes, out
System.Runtime.InteropServices.ComTypes.EXCEPINFO pei, [In,
MarshalAs(UnmanagedType.Interface)] IServiceProvider pspCaller, [In] uint
cvarRefArg, [In] ref uint rgiRefArg, [In, Out,
MarshalAs(UnmanagedType.Struct)] ref object rgvarRefArg);
[MethodImpl(MethodImplOptions.InternalCall, MethodCodeType =
MethodCodeType.Runtime)]
void DeleteMemberByName([In, MarshalAs(UnmanagedType.BStr)] string
bstrName, [In] uint grfdex);
[MethodImpl(MethodImplOptions.InternalCall, MethodCodeType =
MethodCodeType.Runtime)]
void DeleteMemberByDispID([In] int id);
[MethodImpl(MethodImplOptions.InternalCall, MethodCodeType =
MethodCodeType.Runtime)]
void GetMemberProperties([In] int id, [In] uint grfdexFetch, out
uint pgrfdex);
[MethodImpl(MethodImplOptions.InternalCall, MethodCodeType =
MethodCodeType.Runtime)]
void GetMemberName([In] int id, [MarshalAs(UnmanagedType.BStr)] out
```

RE: Controlling Modal Dialogs (Solution)

```
string pbstrName);  
[MethodImpl(MethodImplOptions.InternalCall, MethodCodeType =  
MethodCodeType.Runtime)]  
void GetNextDispID([In] uint grfdex, [In] int id, out int pid);  
[MethodImpl(MethodImplOptions.InternalCall, MethodCodeType =  
MethodCodeType.Runtime)]  
void GetNamespaceParent([MarshalAs(UnmanagedType.IUnknown)] out  
object ppunk);  
}  
#endregion
```

```
#region tagDISPPARAMS class  
[StructLayout(LayoutKind.Sequential)]  
public sealed class tagDISPPARAMS  
{  
    public IntPtr rgvarg;  
    public IntPtr rgdispidNamedArgs;  
    [MarshalAs(UnmanagedType.U4)]  
    public int cArgs;  
    [MarshalAs(UnmanagedType.U4)]  
    public int cNamedArgs;  
    public tagDISPPARAMS()  
    {  
    }  
}  
#endregion
```

```
#region tagEXCEPINFO class  
[StructLayout(LayoutKind.Sequential)]  
public class tagEXCEPINFO  
{  
    [MarshalAs(UnmanagedType.U2)]  
    public short wCode;  
    [MarshalAs(UnmanagedType.U2)]  
    public short wReserved;  
    [MarshalAs(UnmanagedType.BStr)]  
    public string bstrSource;  
    [MarshalAs(UnmanagedType.BStr)]  
    public string bstrDescription;  
    [MarshalAs(UnmanagedType.BStr)]  
    public string bstrHelpFile;  
    [MarshalAs(UnmanagedType.U4)]  
    public int dwHelpContext;  
    public IntPtr pvReserved;  
    public IntPtr pfnDeferredFillIn;  
    [MarshalAs(UnmanagedType.U4)]  
    public int scode;  
    public tagEXCEPINFO()  
    {  
        this.pvReserved = IntPtr.Zero;  
        this.pfnDeferredFillIn = IntPtr.Zero;  
    }  
}
```

RE: Controlling Modal Dialogs (Solution)

```
}
#endregion

#region Win32 Interop Import declarations
internal class Win32
{
[Flags]
public enum InvokeCallType
{
DISPATCH_METHOD = 0x1,
DISPATCH_PROPERTYGET = 0x2,
DISPATCH_PROPERTYPUT = 0x4,
DISPATCH_PROPERTYPUTREF = 0x8,
}

public const int INET_E_DEFAULT_ACTION = unchecked((int)0x800C0011);
public const int E_NOTIMPL = unchecked((int)0x80004001);
public const int S_OK = 0;
public const int S_FALSE = 1;

[DllImport("oleaut32.dll", PreserveSig = false)]
public static extern void VariantInit(HandleRef pObject);

[DllImport("oleaut32.dll", PreserveSig = false)]
public static extern void VariantClear(HandleRef pObject);
}

```

Adar Wesley

"Adar Wesley" wrote:

Hi All,

Sorry for the lengthy post. I am trying to explain the scenario as well as give all the details in order to minimize the need for questions and repostings...

I have a WinForms application that hosts the WebBrowser control. This application needs to work stand alone on a server (without enduser intervention). This means that it needs to automatically handle all new windows that are opened by the web site (web application). The web sites are predefined intranet web applications. However, I have no control over their implementation.

In order to trap new windows created by script calls to window.open, I

RE: Controlling Modal Dialogs (Solution)

impleteted

DWebBrowserEvents2 and sinked events by overriding CreateSink.

My implementation of the NewWindow3 method creates a new tab in my Form and a new browser control in that tab. This works fine.

I am now trying to solve the problem of controlling dialogs created by window.showModalDialog. My application needs to be able to control wether they are created or not, and if they are created to manipulate their DOM. After reading in several places that this cannot be done I went ahead and tried to

solve the problem. I have managed a partial solution so far. I managed to get

a callback function called in my application whenever a script calls window.showModalDialog. See implementation details below.

My question is, now that I have captured the call what to do with it.

I have three possible options, and my application will know at runtime which of

these options is required for a particular dialog.

1. Don't show the dialog and just return a fabricated return value as if the dialog

was shown. This is not a problem.

2. Let the dialog show and don't interact with it. This is usfull in a server application only if the dialog closes itself after a while. I know how to do this too.

I just call the IHTMLWindow2.showModalDialog myself.

3. Let the dialog show and interact with its DOM. This is my problem. The IHTMLWindow2.showModalDialog doesn't give me access to the dialog's DOM (as far as I know). I thought about the following workarounds.

Any other suggestions would be most welcome.

A. Use IHostDialogHelper.ShowHTMLDialog. This may be good if the last parameter to the function (IUnknown *punkHost) gives me access to the DOM somehow. I was unable to find any documentation about this parameter.

B. Host MSHTML myself for the dialog's HTML. I am not sure this would work if

the script in the dialog would access members of the IHTMLDialog interface

that is implemented by the window object created by IHTMLWindow2.showModalDialog.

C. Create a new tab and new CustomWebBrowser, as I do for window.open call, and have it navigate to the dialog's URL.

About this solution I am not sure how to handle the modality considerations.

Other tabs in my application need to be disabled. My implementation of

RE: Controlling Modal Dialogs (Solution)

showModalDialog needs to return only when the created browser exits. Again, what would happen if (when) the script calls items in the IHTMLDialog interface, like IHTMLDialog.returnValue.

Any help would be greatly appreciated.

Finally, here are the details of my implementation.

In my derived

```
class CustomWebBrowser : System.Windows.Forms.WebBrowser
```

In the constructor I set the ObjectForScripting as below. This object has an implementation of a method named showModalDialog, after assigning the ObjectForScripting property this method is available to script through the window.external.showModalDialog reference. Now all that remains in order to hook this method is to replace the script's window.showModalDialog.

This is done in the implementation of DWebBrowserEvents2.DocumentComplete method by executing javascript code that replaces the window.showModalDialog with 'function(dialog, varArgIn, varOptions){ return window.external.showModalDialog(dialog, varArgIn, varOptions); }'

See the main implementation details below.

```
public class CustomWebBrowser : System.Windows.Forms.WebBrowser
{
public CustomWebBrowser()
: base()
{
this.ObjectForScripting = new ScriptingCallableObject(this);
}
...
protected override void CreateSink()
{
base.CreateSink();
m_eventsImplementation =
new DWebBrowserEvents2Implementation(this);
eventsCookie =
new AxHost.ConnectionPointCookie(this.ActiveXInstance,
m_eventsImplementation,
typeof(DWebBrowserEvents2));
}
...

public class DWebBrowserEvents2Implementation : DWebBrowserEvents2
{
public DWebBrowserEvents2Implementation(CustomWebBrowser browser)
{
m_Browser = browser;
}
...
[DispId(259)]
public void DocumentComplete(object pDisp, ref object URL)
```

RE: Controlling Modal Dialogs (Solution)

```
{
IHTMLDocument2 doc2 = m_Browser.IE.Document as IHTMLDocument2;
doc2.parentWindow.execScript("if
(typeof(gpfm_OrigShowModalDialog) == \"undefined\") {
gpfm_OrigShowModalDialog = window.showModalDialog; window.showModalDialog =
function(dialog, varArgIn, varOptions){ return
window.external.showModalDialog(dialog, varArgIn, varOptions); }; }",
"javascript");
}
...
}
```

```
...
[ComVisible(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
public class ScriptingCallableObject
{
public ScriptingCallableObject(CustomWebBrowser browser)
{
m_WebBrowser = browser;
}

public object showModalDialog(string dialog, object varArgIn,
string varOptions)
{
// This implementation preserves the default behavior.
// but I can implemet whatever I want here.
IHTMLDocument2 doc2 = m_WebBrowser.IE.Document as
IHTMLDocument2;
object options = varOptions;
return doc2.parentWindow.showModalDialog(dialog, ref
varArgIn, ref options);
}

private CustomWebBrowser m_WebBrowser;
}
}
```

Adar Wesley