vbwm.com original article

Mastering Internet Explorer: Using the Web Browser Control

By Michiel Meulendijk

The third article in the series, written by Michiel Meulendijk. This article provides practical information and code samples on how to implement the Web Browser Control in your application.

This article has been viewed 25615 times.

Pages: [1] [2] [3] [4] >>

Current Rating: (13 votes)

Article Assumptions

The following article applies to Visual Basic 6.0 Service Pack 4 and Internet Explorer 5.x. It assumes you have no experience using the Microsoft Web Browser control.

Another author, same subject

The first two instalments of these series (Programming Introduction and The Web Browser Control) have been written by Andrew Barfield. He, however, no longer continues writing articles in these series, and therefore I, Michiel Meulendijk, will finish Mastering Internet Explorer by adding several articles to the series.

Andrew started off with a good deal of detailed explanations about the various methods, events and properties the Web Browser control possesses. In this instalment, we'll use much of what Andrew taught to write a sample application that demonstrates the basic use of the Web Browser control.

Further on, we'll also focus on the various advanced functions that can be accessed through the ExecWB method.

Building a Browser

The best way to demonstrate and explore the functionality of the Web Browser control is to reverse-engineer the most famous application that uses it: Internet Explorer. In this article, we'll build our own highly customized browser; apart from adding basic functions to navigate through the Web, we'll also add some features that Microsoft should have implemented in their browser long ago, such as a pop-up blocker and a website lock.

Note that not every single line of code in the sample application provided with this article is covered. First **download the example project** so you can play around with it, then read along with the article.

The Basics

For starters, the basic function of a browser should be to navigate through webpages. With the Web Browser control, this is realized either through the Navigate method, or the Navigate2 method. For most functions, both methods do a decent job, but some flags may not work with the former method. Therefore we'll use Navigate2 in our example. Both methods support five parameters:

- *URL:* a string specifying the location of the document;
- *Flags:* a value that determines under more whether a page should be added to the history list, be read from the cache memory or be written to the cache;
- *TargetFrameName:* a string that determines in which frame the URL should be displayed;
- *PostData:* a value that specifies the HTTP POST data to send to the server;
- *Headers:* a value that specifies the HTTP headers to send to the server.

In our application, this provides us with the following code: The *BrowserNavConstants* enumeration is used to simplify the selection of flags. Instead of having to remember each and every value by its number, you can use the enumeration to choose from.

The WebNavigate sub performs the actual navigating to the URL. It can be called from anywhere within the code. If you use a standard textbox with a *Go!* button as navigational interface, the code necessary to browse would be this:

- MSDN: Web Browser Control Reference

```
Private Enum BrowserNavConstants
    navOpenInNewWindow = &H1
    navNoHistory = &H2
    navNoReadFromCache = &H4
    navNoWriteToCache = &H8
    navAllowAutosearch = &H10
    navBrowserBar = &H20
    navHyperlink = &H40
End Enum

Private Sub WebNavigate(URL As String, Optional Flags, _

        Optional Target, Optional PostData, Optional Headers)
webMain.Navigate URL, Flags, Target, PostData, Headers
End Sub
```

```
Private Sub cmdGo_Click()
WebNavigate txtURL.Text
End Sub
```

Now, we'll need some other basic functions to enhance our browser, such as a *back* button, a *forward* button, a *stop*, *refresh* and *home* button. The Web Browser control provides pre-made methods for these tasks and if we use simple toolbar buttons to perform them, our code would look like this:

```
Private Sub tlbMain_ButtonClick(ByVal Button As MSComctlLib.Button)
Select Case Button.Key
    Case "back"
        webMain.GoBack
    Case "forward"
        webMain.GoForward
    Case "stop"
        webMain.Stop
    Case "reload"
        webMain.Refresh
    Case "home"
        webMain.GoHome
    Case "search"
        webMain.GoSearch
```

```
End Select
End Sub
```

However, if you play around with this code in the example you'll notice that it doesn't work flawlessly. When you press buttons at certain times nasty errors will pop up. *Method 'GoBack' of object IWebBrowser2 failed*, for example, will appear when you press the *back* button without having navigated to a page to begin with. After all it's very logical that you can't go back in history if history has yet to be written. While you can easily bypass such an error by placing an *On Error Resume Next* statement above the code, it won't disable the button, thereby misleading the user of the application (he thinks he can go back in history while in fact he cannot).

To avoid these errors we'll have to disable the *back* and *forward* button when they can't be used. To do that, we'll need the *CommandStateChanged* event. This event will fire when the usability of the *forward* or *back* functionality changes. In other words, when you've reached the beginning or the end of the history list, parameter *Enable* will be *False* for either the *back* or the *forward* command. At the beginning of a session both commands will return *False*, in the middle both will be *True*.

```
Private Sub webMain_CommandStateChange(ByVal Command As Long, _
                                       ByVal Enable As Boolean)
Select Case Command
    Case 1 'Forward
        tlbMain.Buttons.Item("forward").Enabled = Enable
    Case 2 'Back
        tlbMain.Buttons.Item("back").Enabled = Enable
End Select
End Sub
```

When testing this code, you'll notice that the *back* and *forward* buttons will only be enabled when you can really navigate to another page, thereby preventing errors from spoiling the fun.


search
Enter keywords here.