

WYSIWYG Web Builder Extension Proxy

Ever since we've introduced the Extension Builder toolkit, we've been asked for (even) more control over the behavior of extensions.

This document describes a method of implementing your own extensions through COM. To make this possible we've created a 'proxy' extension which translates the WYSIWYG Web Builder extension interface (C++/MFC) to COM/ActiveX.

Notes:

This project is still experimental, it's not officially supported. Use at your own risk!
FOR ADVANCED USERS ONLY.



Your extension will always have at least two files:

- yourextension.wbx
The proxy extension, this file is the same for all extensions (a copy of template.wbx). Only the name will be different.
- yourextension.dll
The implementation of your extension. A COM/ActiveX DLL

If you use a programming language where the type library is not included in the DLL itself (such as .NET) you also must include the type library (yourextension.tlb).

Here's how it works:

1. WYSIWYG Web Builder loads 'yourextension.wbx' during startup
2. yourextension.wbx looks for yourextension.dll (or yourextension.tlb).
3. Next yourextension.wbx will load your dll into memory and forward all function calls to this DLL.

The following methods should be implemented (in the same order!).

HRESULT GetToolbarText([in, out] BSTR* text);

Get text to be displayed in the toolbox and menu. Maximum 25 characters.

HRESULT GetDescription([in, out] BSTR* text);

Get description of the extension to be displayed in the Extension Manager.

HRESULT GetToolbarBitmap([in, out] long* bitmap);

Get a handle (HBITMAP) to a bitmap to display in the toolbox. The bitmap should be 16x16 pixels. If bitmap is NULL, then the default image will be used.

HRESULT Load([in] BSTR data);

This function is called by Web Builder when the object is loaded (either from disk or undo/redo cache).

This must be used to serialize the data of your properties which have previously been saved using the **Store** function.

Developers should implement their own serialization method to load and store the data.

HRESULT Store([in, out] BSTR* data);

This function is called by Web Builder when the object is stored.

This must be used to serialize the data of your properties.

A common way to do this is by using XML, but it can also be another text format.

HRESULT GetRenderMode([out, retval] long*);

Specifies the render mode use for the extension.

Use '0' to display text only. The **GetToolbarText()** method will be called to request the text to be displayed.

Use '1' to render the HTML.

Note that only the <BODY> code (specified in **GetHtml ()**) will be parsed. HTML from the <HEAD> will not be interpreted by WYSIWYG Web Builder.

Use '2' to implement custom rendering. In that case the 'Draw' method must be implemented.

HRESULT Draw([in] long hdc, [in] long x, [in] long y, [in] long width, [in] long height);

Use this method if you want to implement your rendering.

HRESULT GetHeaderHtml([in, out] BSTR* html);

This code will be inserted between the <HEAD> tag.

This section of the HTML document is common for non-visible HTML code. Like file includes, scripts and style sheets.

HRESULT GetHtml([in, out] BSTR* html);

This code will be inserted between the <BODY> tag.

This section of the HTML document is common for visible HTML code.

HRESULT GetStartofPageHTML([in, out] BSTR* html);

This code will be inserted before the <HTML> tag. This section of the HTML document is common for server sided scripts like PHP or ASP.

HRESULT ShowProperties([in] long hWnd);

Show component's properties dialog.

Called when a user double clicks your component in Web Builder or selects Object Properties from the menu.

hWnd

Handle to the parent window.

**HRESULT CopyFiles(
[in, out] BSTR path,
[in, out] BSTR imagesPath);**

This function is called by Web Builder when the web page is published or previewed. Your component can use this to copy files used by your HTML code to the output path.

For example if you've implemented an image component, this is the place to copy the image(s) to the output path.

NOTE: When files are published to a FTP server, they are first copied to a local folder, so the path always contains a local path.

HRESULT Move([in] long x, [in] long y, [in] long width, [in] long height);

This function is called by Web Builder when the component has moved to a new position. You can use this to store the objects size and position which may be needed in your HTML code.

HRESULT UseDIV([out, retval] VARIANT_BOOL*);

This function is called by Web Builder to determine if it must use <DIV> tags to position your HTML object in the Web Page.

Return TRUE to let Web Builder handle positioning of your HTML object using <DIV> tags.

Return FALSE if you add the position attributes yourself or if you have an invisible object which doesn't require positioning.

The variables \$LEFT\$, \$TOP\$, \$WIDTH\$, \$HEIGHT\$ and \$Z_INDEX\$ are predefined variables. They will be replaced by the actual position and size of the object. These variables can only be used if 'Use DIV' is disabled!

HRESULT GetMinimumSize([in, out] long* width, [in, out] long* height);

Here you can specify the minimum width and height for the object in the workspace.

HRESULT GetAssetCount([out, retval] long*);

Return the number of assets your object uses.

For example if you implement a gallery extension, this should return the number of images.

For each asset, web builder will call **GetAssetFileName** to retrieve the filename.

HRESULT GetAssetFileName([in] long index, [in, out] BSTR* fileName);

Return the filename of the asset.

HRESULT SetAssetFileName([in] long index, [in] BSTR fileName);

Set the filename of the asset.

HRESULT GetURLCount([out, retval] long*);

Return the number of URLs your object uses.

For example if you implement a menu extension, this should return the number of links. For each URL, web builder will call **GetURL** to retrieve the URL.

HRESULT GetURL([in] long index, [in, out] BSTR* url);

Return the filename of the asset.

This method is used by Web Builder for updating internal links. For example if the user linked to 'page1' and it will be rename to 'page2'. Then GetURL will be called to check if a link 'page1' exists. Then SetURL will be called with the new name.

HRESULT SetURL([in] long index, [in] BSTR url);

Set the filename of the URL.

HRESULT SetSiteStructure([in] BSTR xml);

This method will be called each time the site structure (in WYSWYG Web Builder's Site Manager) changes.

The site structure will be passed in XML format. There is currently no documentation for the exact format.

You can use this to display a list of the available pages, so the user can select a page for an internal link.

In future versions of this document we will try to give a more detailed description of link management. For now we assume you are smart enough to figure it out yourself.

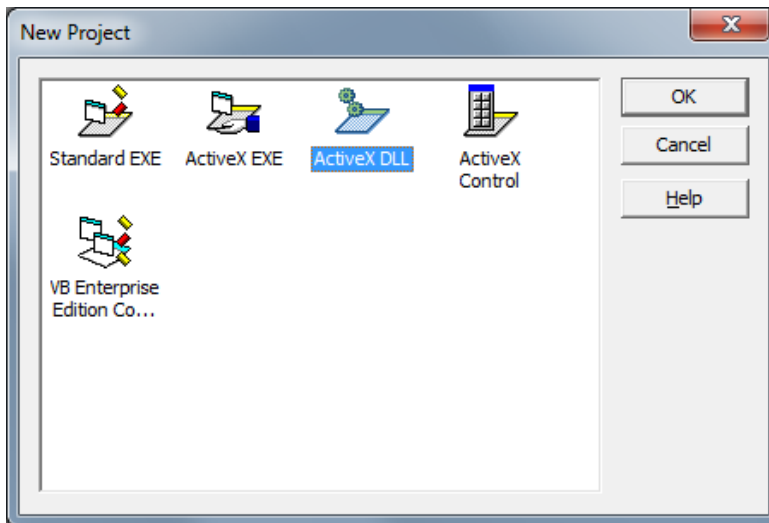
How to create a new extension in Visual Basic 6

Step 1

Rename template.wbx to newname.wbx, where 'newname' is the name of your extension. This file will be the proxy between your DLL and WYSIWYG Web Builder. It should have the same name (except for the file extension) as your ActiveX DLL!

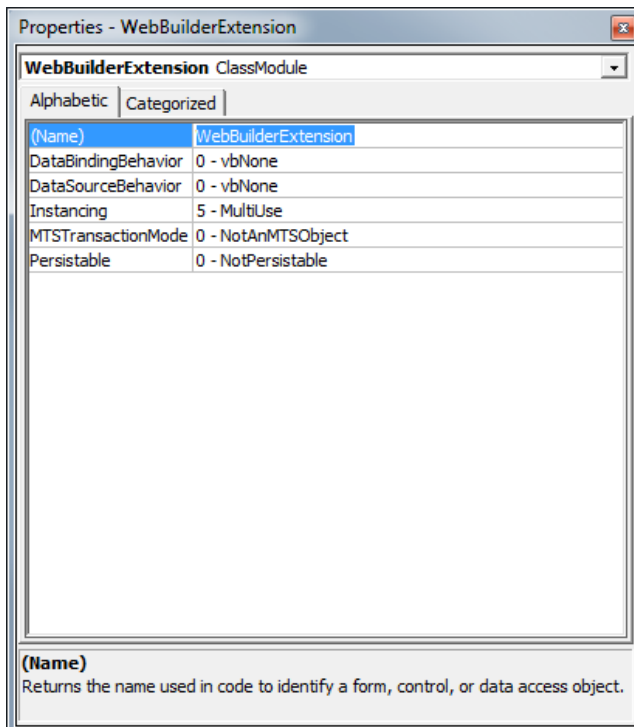
Step 2

Create a new project and select **ActiveX DLL**.



Step 3

In Properties rename 'Class1' to 'WebBuilderExtension'



Step 4

Copy/paste the following code. This way you can be sure you have implemented all methods:

```
Public Sub GetToolbarText(ByRef text As String)
    text = "My First Extension"
End Sub

Public Sub GetDescription(ByRef text As String)
    text = "This is my First Extension"
End Sub

Public Sub GetToolbarBitmap(ByRef bitmap As Long)
End Sub

Public Sub Load(ByVal data As String)
End Sub

Public Sub Store(ByRef data As String)
End Sub

Public Function GetRenderMode() As Long
End Function

Public Sub Draw(ByVal hdc As Long, ByVal x As Long, ByVal y As Long, ByVal width As Long, ByVal height As Long)
End Sub

Public Sub GetHeaderHtml(ByRef html As String)
End Sub

Public Sub GetHtml(ByRef html As String)
End Sub

Public Sub GetStartofPageHTML(ByRef html As String)
End Sub

Public Sub ShowProperties(ByVal hWnd As Long)
End Sub

Public Sub CopyFiles(ByVal path As String, ByVal imagesPath As String)
End Sub

Public Sub Move(ByVal x As Long, ByVal y As Long, ByVal width As Long, ByVal height As Long)
End Sub

Public Function UseDIV() As Boolean
End Function

Public Sub GetMinimumSize(ByRef width As Long, ByRef height As Long)
End Sub

Public Function GetAssetCount() As Long
    GetAssetCount = 0
End Function

Public Sub GetAssetFileName(ByVal index As Long, ByRef fileName As String)
End Sub

Public Sub SetAssetFileName(ByVal index As Long, ByVal fileName As String)
End Sub

Public Function GetURLCount() As Long
    GetURLCount = 0
End Function

Public Sub GetURL(ByVal index As Long, ByRef url As String)
End Sub

Public Sub SetURL(ByVal index As Long, ByVal url As String)
End Sub

Public Sub SetSiteStructure(ByVal xml As String)
End Sub
```

Step 5

Now implement your code...

Step 6

Compile the project. Make sure the name of the DLL is the same as the name you've used in step 1.

So you will end up with two files:

- newname.wbx (renamed template.wbx)
- newname.dll (your VB ActiveX DLL)

Step 7

Your extension is ready to be tested in WYSIWYG Web Builder!

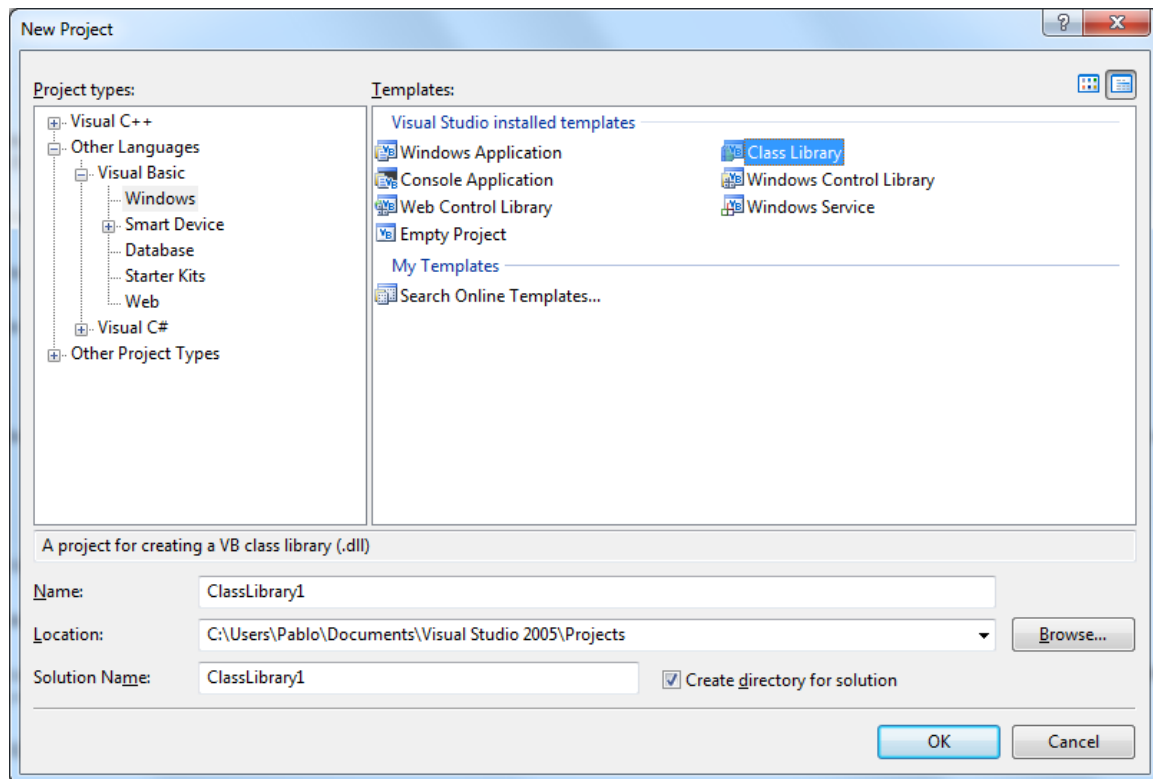
How to create a new extension in Visual Basic.NET 2005

Step 1

Rename template.wbx to newname.wbx. Where 'newname' is the name of your extension. This file will be the proxy between your DLL and WYSIWYG Web Builder. It should have the same name (except for the file extension) as your ActiveX DLL!

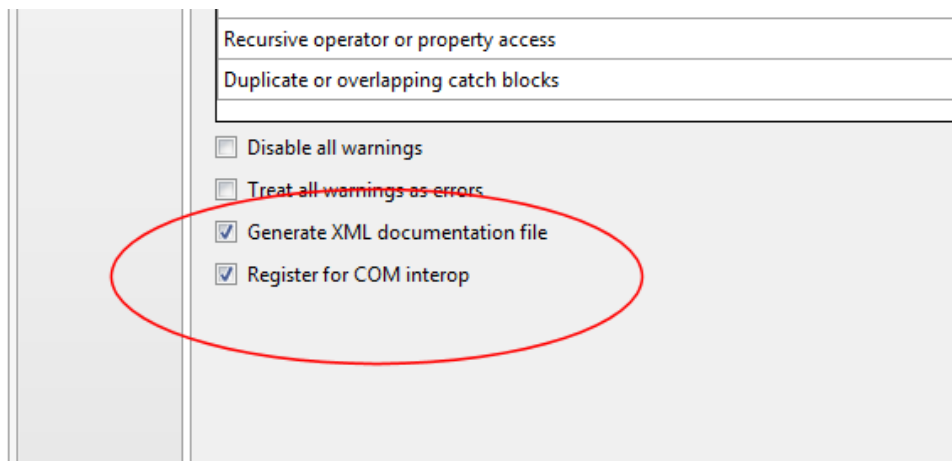
Step 2

Create a new project and select **Class Library**.



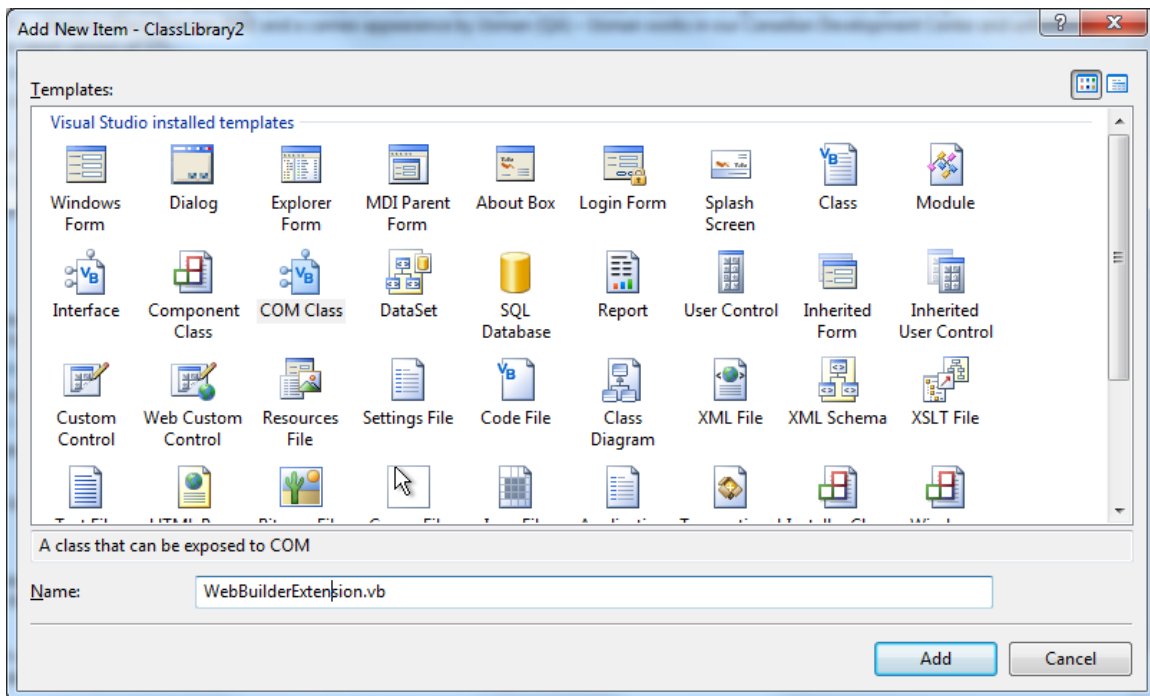
Step 3

In Project Properties make sure 'Register for COM interop' is enabled!



Step 4

Add a new COM class (Menu->Project->Add class->COM Class) and name it 'WebBuilderExtension.vb'.



Step 5

Copy/paste the following code inside the class. This way you can be sure you have implemented all methods:

```
Public Sub GetToolbarText(ByRef text As String)
End Sub

Public Sub GetDescription(ByRef text As String)
End Sub

Public Sub GetToolbarBitmap(ByRef bitmap As Integer)
End Sub

Public Sub Load(ByVal data As String)
End Sub

Public Sub Store(ByRef data As String)
End Sub

Public Function GetRenderMode() As Integer
End Function

Public Sub Draw(ByVal hdc As Integer, ByVal x As Integer, ByVal y As Integer, ByVal width As Integer,
ByVal height As Integer)
End Sub

Public Sub GetHeaderHtml(ByRef html As String)
End Sub

Public Sub GetHtml(ByRef html As String)
End Sub

Public Sub GetStartofPageHTML(ByRef html As String)
End Sub

Public Sub ShowProperties(ByVal hWnd As Integer)
End Sub

Public Sub CopyFiles(ByVal path As String, ByVal imagesPath As String)
End Sub

Public Sub Move(ByVal x As Integer, ByVal y As Integer, ByVal width As Integer, ByVal height As
Integer)
End Sub

Public Function UseDIV() As Boolean
End Function

Public Sub GetMinimumSize(ByRef width As Integer, ByRef height As Integer)
End Sub

Public Function GetAssetCount() As Integer
    GetAssetCount = 0
End Function

Public Sub GetAssetFileName(ByVal index As Integer, ByRef fileName As String)
End Sub

Public Sub SetAssetFileName(ByVal index As Integer, ByVal fileName As String)
End Sub

Public Function GetURLCount() As Integer
    GetURLCount = 0
End Function

Public Sub GetURL(ByVal index As Integer, ByRef url As String)
End Sub

Public Sub SetURL(ByVal index As Integer, ByVal url As String)
End Sub

Public Sub SetSiteStructure(ByVal xml As String)
End Sub

Public Sub New()
    MyBase.New()
End Sub
```

Step 6

Now implement your code...

Step 7

Compile the project. Make sure the name of the DLL is the same as the name you've used in step 1.

So you will end up with three files:

- newname.wbx (renamed template.wbx)
- newname.dll (your VB ActiveX DLL)
- newname.tlb (the type library file generated by .NET)

Step 8

Your extension is ready to be tested in WYSIWYG Web Builder!

Final notes

We've included 3 example projects:

- Visual Basic 6
- Visual Basic .NET 2005
- Visual C++/MFC.

The Visual Basic examples are just simple templates, they do not anything useful. The C++/MFC project is a more advanced example. Basically this is a stripped down version of the 'Random image' extension. It demonstrates data serializing, custom rendering and image management.

Important Note:

If you are going to create a new extension based on one of these projects then please make sure you COM GUIDs are unique!

Since we're dealing with COM, we will have to register the COM objects in the registry. For example by using regsvr32.exe. Unfortunately under Vista/Windows7 this will fail if you are not an administrator. So you will have to come up with way to install the object on the user's computer or provide instructions how to register the DLL.

Registering a COM DLL under Vista/Windows7:

1. Run the command prompt as administrator (Right click 'Command Prompt' in the Start Menu and select 'Run as administrator'.
2. Execute 'regsvr32 yourextension.dll'

Nothing of this document may be reproduced or copied in any form without the explicit written authorization of Pablo Software Solutions.

Copyright 2009 Pablo Software Solutions. All rights reserved.

<http://www.wysiwygwebbuilder.com>