

# Extension Builder for WYSIWYG Web Builder

## Introduction

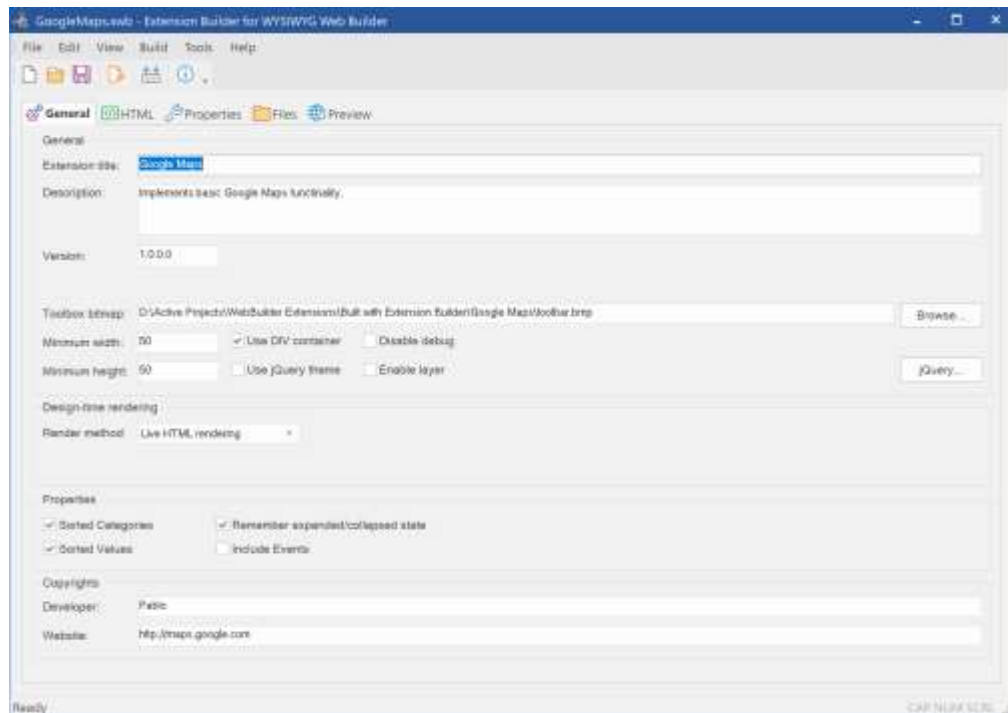
Extension Builder is a utility to build your own extensions for WYSIWYG Web Builder. It can be used to share code snippets, scripts, flash components and other web widgets in one single .wbx file that can be used by users in the same way as other objects in the WYSIWYG Web Builder toolbox.

Key features:

- Start building your own extensions within minutes without the need for an external programming tool.
- Convert your scripts or Flash applications into a WYSIWYG Web Builder object and share them with other users.
- Define properties to control the behavior of the HTML like text, colors, fonts, links and a special gallery option!
- All properties you define can be edited by the user through a generic dialog. You can also categorize the properties.
- 'Live HTML rendering', the results of the HTML will be directly visible within Web Builder for most extensions.
- Many types of image galleries can be created, using XML templates.
- All files will be embedded within the extension, so the output is only 1 file

Note: **This utility is for advanced users only. Some knowledge of HTML/XML is required!**

In Extension Builder you can define HTML code, attach additional files and specify properties to control the behavior of your object. In this document you will find a brief description of the available options and a few examples to help you get started.



## **Menu Options**

### **File Menu**

#### **New**

Creates a new extension project. All information that you input will be stored in a project file with the file type **.xwb**. Note that in order to save the extension in .wbx format you must use the 'build extension' option! This will generate a .wbx file (web builder extension).

#### **Open**

Opens an existing extension project.

#### **New extension wizard**

The wizard provides a step by step process for building a new extension.

#### **Import Third Party Add-On**

This is an experimental option, which can be used to import (Adobe) Muse Configurable Options Widget (.muco) files.

This option was added to give Muse Widgets developers the possibility to make their widgets available for WWB without having to start from scratch.

This tool attempts import all code sections and properties. When possible, the properties are converted to WWB compatible options. In some cases, you may need to fine-tune the imported code to optimize it for WWB.

Besides the ability to import .muco files, there is also limited support for .mulib widgets.

Known limitations:

- conditional properties are not supported
- translation is not supported. The import option will only import 'English'
- not all .mulib files are real widgets. Sometimes they are just page fragments/blocks. In that case it will not be possible to import the widget in Extension Builder.
- some widgets load their own version of jQuery. This reference should be manually removed, because it may cause conflicts with the built-in jQuery version.

#### **Save**

Save the active extension project.

#### **Save As**

Save the active extension project with a new name.

#### **Most recently used files**

A list of the most recently used files, select the filename to open the project.

#### **Exit**

Quits the application.

## ***View Menu***

### **Toolbar**

Show or hide the toolbar.

### **Statusbar**

Show or hide the status bar.

### **Application Look**

Specifies the theme of the user interface.

### **Show Tab Icons**

Specifies whether to show icons on main tabs.

### **Editor Font**

Allows you to set a different font for the code editors.

## ***Build Menu***

### **Build extension**

This option will compile the extension and create one single file that contains all files, code and properties. This file (with file type .wbx) can be loaded by web builder. The file will be saved in the same folder as your project file.

## ***Tools Menu***

### **Test XSL template**

This option can be used to test XSL templates for datasets, galleries and navigation menus. The **Input** field specifies the input XML data. The tool will automatically generate some example data for the selected **Type**, but you can also modify it, if needed. The **Template** field specifies the XSL template code. Use the **Transform** button to process the data/template. The result will be displayed in the **Result** field. See the examples later in this document for more details about XSL templates.

### **Set Extensions Folder**

Use this option to select the extension folder for WYSIWYG Web Builder.

If this folder is set then Extension Builder will make a copy of the extension to this folder after it has been built.

Normally the extension folder is located in:

My Documents\WYSIWYG Web Builder\system\extensions\

## General Page

### Extension title

This is the name that will be displayed in the toolbox of WYSIWYG Web Builder.

### Description

Description that will be displayed in the Extension.

### Version

Specifies the version number of the extension.

The version number consists of 4 numbers separated by dots: 1.0.0.0

Examples:

2.0.0.0       => Major new version

1.1.0.0       => Minor new version

1.0.1.0       => Bug fixes only

WYSIWYG Web Builder will use this version number when checking for updates if the extension is in the official extensions database.

### Toolbox bitmap

Specifies the bitmap displayed in the toolbox. The bitmap must be 16x16!

### Designed for

Specifies whether the extension was designed for desktop or mobile pages.

Desktop extensions will only be displayed in the toolbox for standard pages.

Mobile extensions will only be displayed in the toolbox for mobile pages.

### Use DIV container

If this option is enabled the HTML code will be inserted inside a DIV container with size and position information by WYSIWYG Web Builder. If this option is not enabled the HTML code will be inserted "AS IS", without size and position information.

### Use jQuery Theme

Enable this option if your extension uses jQuery themeroller. WWB will automatically include the theme selected by the user (in Page Properties).

### Disable debug

Prevents users from debugging your extension in WWB.

### Enable Layer

Enable this option to turn the extension into a container for other objects. This can be useful to create a layer or form-like extension. Use the predefined \$OBJECTS\$ variable to include the HTML of the objects in your code.

Example: `<form action="">$OBJECTS$</form>`

Notes:

- The \$OBJECTS\$ variable can only be used in code between `<body></body>` tags!
- This feature is only available in WYSIWYG Web Builder 11.5 and newer!

## jQuery

One of the new features in WWB8 is the built-in support for jQuery.

If your extension uses jQuery then you do not have to include any code for that!

Simply click the 'jQuery' button, select the jQuery modules required by your extension and Web Builder will automatically add the code and files for you!

Tip: If your project already includes references to one of these modules remember to remove the code to prevent duplicated entries in the HTML page. **For example: do not include references to 'jquery-x.x.x.min.js' in your code!**

## Render method

The render method specifies how WYSIWYG Web Builder will draw the extension in the designer workspace. For practical reason it's not always best to implement extensions as 'What-You-See-Is-What-You-Get' (live HTML rendering). So there for there are 3 different render methods to choose from.

- **Display text**  
This option will display a text. This is the quickest render method and it will not use many system resources. This text will not be used in the final HTML code.
- **Display image**  
This will draw a static image as a 'place holder' for your object. This image will not be used in the final HTML code.
- **Live HTML rendering**  
This will use the HTML code as specified in HTML->Code Between <BODY> tag to render the object inside the web builder workspace. This method may use many system resources depending on the HTML code. Also note that only the <BODY> code will be parsed. HTML from the <HEAD> will not be interpreted by WYSIWYG Web Builder. It's also possible to specify HTML code that will only be used for rendering. See 'Render HTML' for more details.

## Sorted Categories/ Sorted Values

Specifies whether property categories/values will be sorted alphabetically. If this option is not enabled then the categories will be displayed in the same order as they appear in the Extension Builder properties overview.

## Remember expanded/collapsed state

If this option is enabled then the property categories in the extension will have the same expanded/collapse state as in the properties overview. This can be useful if you do not want all categories initially opened so the user is not presented with all properties at once.

## Include Events

Enable this option if you want to include the 'Events' property (which displays the standard 'Events' dialog). Use the predefined \$EVENTS\$ variable to include the selected events in your code. Example: `<a href="#" $EVENTS$> ... </a>`

## Copyrights

Original developer and website of the HTML, scripts and files.

## HTML Page

### Before <html> tag

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.

### Between <head> tag

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 and style sheets.

### Between <style> tag (requires WWB 10.3.3 or higher)

This (CSS) code will be inserted between the <style> tag.

If you want to insert CSS code specifically for media queries/breakpoints then you can do that like this:

```
// other styles here...
// ...
@media $BREAKPOINT$
{
  .yourclass
  {
    left: $LEFT$px;
    top: $TOP$px;
    width: $WIDTH$px;
    height: $HEIGHT$px;
  }
  #ID$
  {
    width: $WIDTH$px;
    height: $HEIGHT$px;
    float: left;
  }
}
```

\$BREAKPOINT\$ will be replaced by the actual media queries. So the final code may look something like this:

```
@media only screen and (max-width: 320px)
{
  .yourclass
  {
    left: 0px;
    top: 0px;
    width: 250px;
    height: 250px;
  }
  #Extension1
  {
    width: 250px;
    height: 250px;
    float: left;
  }
}
```

The \$LEFT\$, \$TOP, \$WIDTH\$ and \$HEIGHT\$ variables will have the position/size values of the object in the current breakpoint.

### **\$(document).ready() script**

In this section you can place jQuery code that needs to be placed inside Web Builder's standard `$(document).ready()` handler. While JavaScript provides the `load` event for executing code when a page is rendered, this event does not get triggered until all assets such as images have been completely received.

In most cases, the script can be run as soon as the DOM hierarchy has been fully constructed. The handler passed to `.ready()` is guaranteed to be executed after the DOM is ready, so this is usually the best place to attach all other event handlers and run other jQuery code. **You must enable 'jQuery' for this option to work.**

```
<script type="text/javascript">
$(document).ready(function()
{
    // your code will be inserted here
});
</script>
```

### **Between <body></body> tags**

This code will be inserted between the `<body></body>` tags and inside the body container (if the page is centered). This section of the HTML document is common for visible HTML code.

### **After <body> tag (requires WWB 10.3 or higher)**

The code will be inserted directly after the `<body>` tag and **outside** the body container.

### **Before </body> tag (requires WWB 10.3 or higher)**

The code will be inserted directly before the `</body>` tag and **outside** the body container.

### **Render HTML**

This code will only be used during design time. If the 'Render method' is set to 'Live HTML rendering' then this code will be used to render the extension in the workspace. This option can be useful if you design an extension that will only work on a web server (for example if it required PHP). Instead of rendering the 'real' code, you can now render something that will 'look like' the final result.

The HTML code can include variables to allow the user to change attributes and other parts of the code. Variables can be connected to properties in the Properties Page.

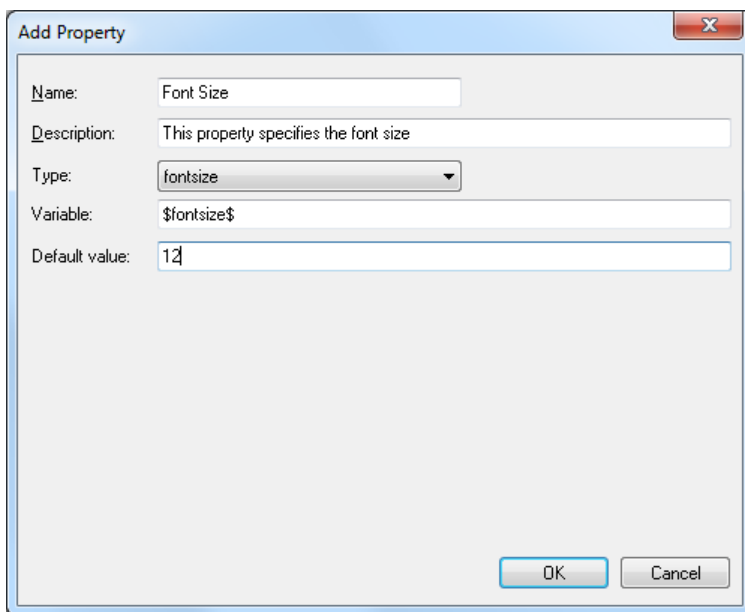
## Properties Page

In this section you can define the categories and properties of the extension. Properties should be connected to variables within the HTML code.

For example, the following HTML is entered in the **Code between <BODY> tag** section:

```
<font style="font-size:$fontsize$px">  
Hello World!  
</font>
```

In this code the font-size is defined as the variable **\$fontsize\$**. This variable can be connected to a property, which later can be modified by the user of the extension. In this particular example the property could be defined like this:



The screenshot shows a dialog box titled "Add Property" with a close button (X) in the top right corner. The dialog contains the following fields:

- Name:** A text box containing "Font Size".
- Description:** A text box containing "This property specifies the font size".
- Type:** A dropdown menu with "fontsize" selected.
- Variable:** A text box containing "\$fontsize\$".
- Default value:** A text box containing "12".

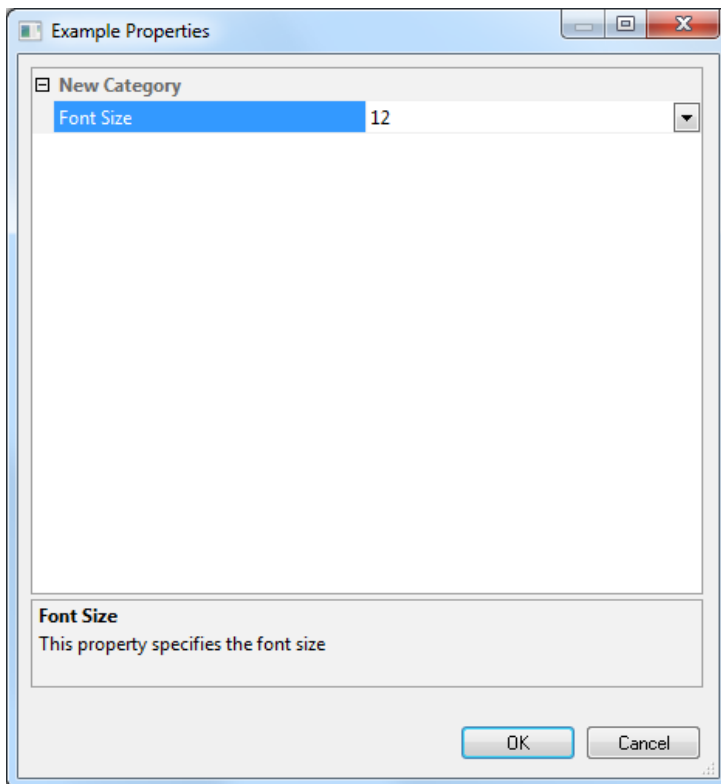
At the bottom right of the dialog are two buttons: "OK" and "Cancel".

### Notes:

- Properties can be dragged & dropped between categories.
- Properties can be re-arranged with the 'Move Up/Move Down' buttons.



In WYSIWYG Web Builder this property will be displayed like this:



If the user selects the value '14'. Then the resulting HTML code will be

```
<font style="font-size:14px>Hello World!</font>
```

The extension builder supports the most common type of properties:

- **bool**  
Boolean property with the options 'true' and 'false'
- **color**  
Color property, which displays a color picker.
- **dataset**  
Here you can define a group of properties that will (in combination with a XML template) allow the user to create a dataset. This can be useful for navigation objects (menus), media players or other advanced elements.
- **Edit**  
There are 4 edit field types:
  1. Single line, standard text input field.
  2. Multi line, useful to input multiple lines or HTML code.
  3. Rich text, allows the user to input formatted text (font, size, color, bold, italic and underline). Web Builder will convert the formatted text to HTML.
  4. HTML Textbox, this will launch the HTML Textbox extension to edit the value of the edit field, giving the user powerful editing options.  
<https://www.wysiwygwebbuilder.com/htmltextbox.html>  
The user must have the HTML Textbox installed, to be able to use it...

- **file**

A file selection field. The selected file will be published together with the HTML page.

The option **Publish this file to the user defined target folder**, copies the specified file to the target folder for the file type (as configured in Menu->Tools->Options->Preview & Publish).

You can also specify a 'fixed' output folder in '**Publish to folder**'. For example, in your set this value to 'pictures' then the selected file will always be copied to the /pictures folder (relative to the page location).

- **file collection**

This property will allow the user to select files to be published along with the extension. This can be useful if multiple files need to be published to a specific folder. For example thumbnail images of a gallery. The selected file names will not be stored in variable. The fields will only be copied to the output folder. '**Publish to folder**' can either contain a fixed folder name or a variable (so the user can specify the output folder).

- **fontname**

A font name picker

- **fontsize**

A font size picker

- **gallery**

An advanced image selection option with support for XML template based HTML/XML output (using <xsl:template>). See the examples at the end of this document for more details about this powerful option!

Enable **Generate thumbnails** to automatically generate thumbnail images. You can specify the thumbnail width (required), height (required), prefix (optional) and folder (optional). These can be hardcoded values or variable names (for example: \$thumbwidth\$, \$thumbheight\$, \$thumbfolder\$). By using variables you can let the user specify the size, prefix folder for the thumbnails.

- **navigation**

Use this property to display a navigation/menu editor dialog.

This property uses a XML template to generate navigation code based on the user input. See the examples later in this document.

This property supports two menu types:

- **list**

- **tree**

Please select the option which applies to your menu code.

**List** displays a list based menu editor. Only supports one level.

**Tree** display a tree based menu editor. Which allows the user to create multi level menu structures.

- **options**

An options menu to allow the user the select from multiple items.

Enable 'editable' if you want to allow the user to enter a value that is not in the list. If 'editable' is disabled then the user can only select items from the list.

It's possible to specify a 'display' value and a 'real' value in the 'Add option' window like this: (use ~ as separator)

```
Yes~123456  
No~987654
```

where the first value before '~' is the display value and the second value will be used as the real value. The user will only see 'Yes' (and not '123456') in the option list. If the user selects 'Yes', the actual value will be '123456'.

Note: the Default value should not include the 'real' value.

### Show / Hide Properties

This option makes it possible to show, hide, enable or disable other properties based on the selected value in the option property.

Value, specifies the value that will trigger the action. This must be one of the configured options. You can put an '!' in front the value to invert the condition. For example: '!true' will trigger the action for all selected values that are not equal to 'true'.

Action, specifies whether to enable, disable, show or hide the target property or category.

Property / Category, specifies the name of the property or category to hide.

*Note: this feature is available in WWB 15.1 and newer.*

- **numeric**  
Useful for collecting numeric values. You can specify the minimum and maximum value.
- **url**  
An URL input field with support for internal and external links.
- **theme**  
This option is deprecated. To make use of jQuery themes, please enable 'Use jQuery Theme' in 'General' options.
- **icon library**  
Adds support for icon libraries to the extension, like FontAwesome, Material Icons etc. Icon Libraries can be used in HTML code (for example as icon on a button), datasets and the menu property. The default value is usually 'FontAwesome', because this icon font is installed by default.  
The same icon library will be used for all icons in your extension.

To add support for icons in menus, simply add the 'icon library' property. There is no need to use the variable name of this property in your own code.

*Note: this feature is available in WWB 16.3 and newer.*

- **icon**

Adds an icon selection property to the extension.

Note: to support icons, the extension should also include the 'icon library' property!

By default, icons inherit the size and color from the parent element. You can control the size and color of the icon by adding the following CSS to your extension:

```
.$ID$-icon
{
    width: 32px;
    height: 32px
    font-size: 32px;
    color: #000000;
}
```

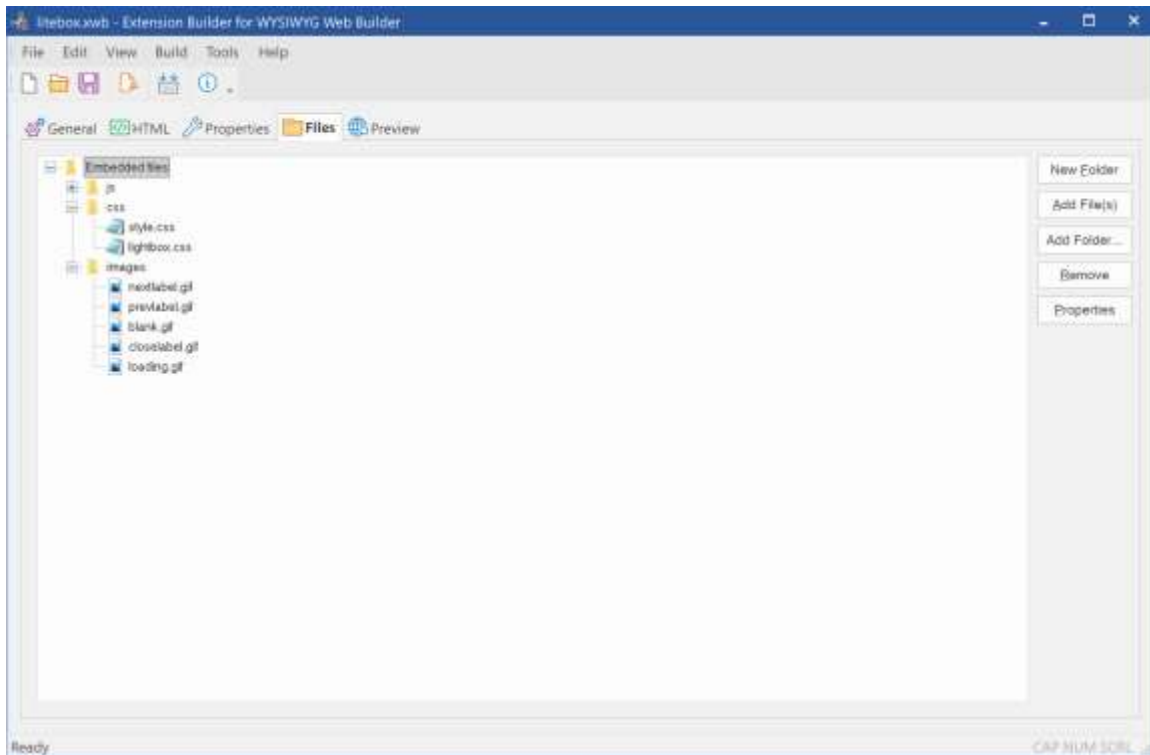
The icon functionality is demonstrated in these examples:

[https://wysiwygwebbuilder.com/extensions/xwb\\_icons.zip](https://wysiwygwebbuilder.com/extensions/xwb_icons.zip)

*Note: this feature is available in WWB 16.3 and newer.*

## Files Page

In this section you can select files that will be included in the extension. This can be javascripts, flash files, images etc. When the page is published all these files also will be published. Files can be organized in folders and you can also specify if files need to be processed before publishing. In that case the variables in the file will be replaced by the values of the matching properties.



### **New Folder**

Click this button to create a new folder.

### **Add File(s)**

Click this button to add one or more files.

The filename can also be a variable (**\$javascript.js**, **\$gallery.xml**, **file\$variable.txt**). In that case the user can control the name of the file through a property of the type 'edit'.

**Tip:** rename the file on disk before adding it to the list. So the input file must also be named \$javascript.js, \$gallery.xml etc.

### **Add Folder**

Insert a complete folder and all files it contains.

### **Delete**

Click this button to delete the selected item.

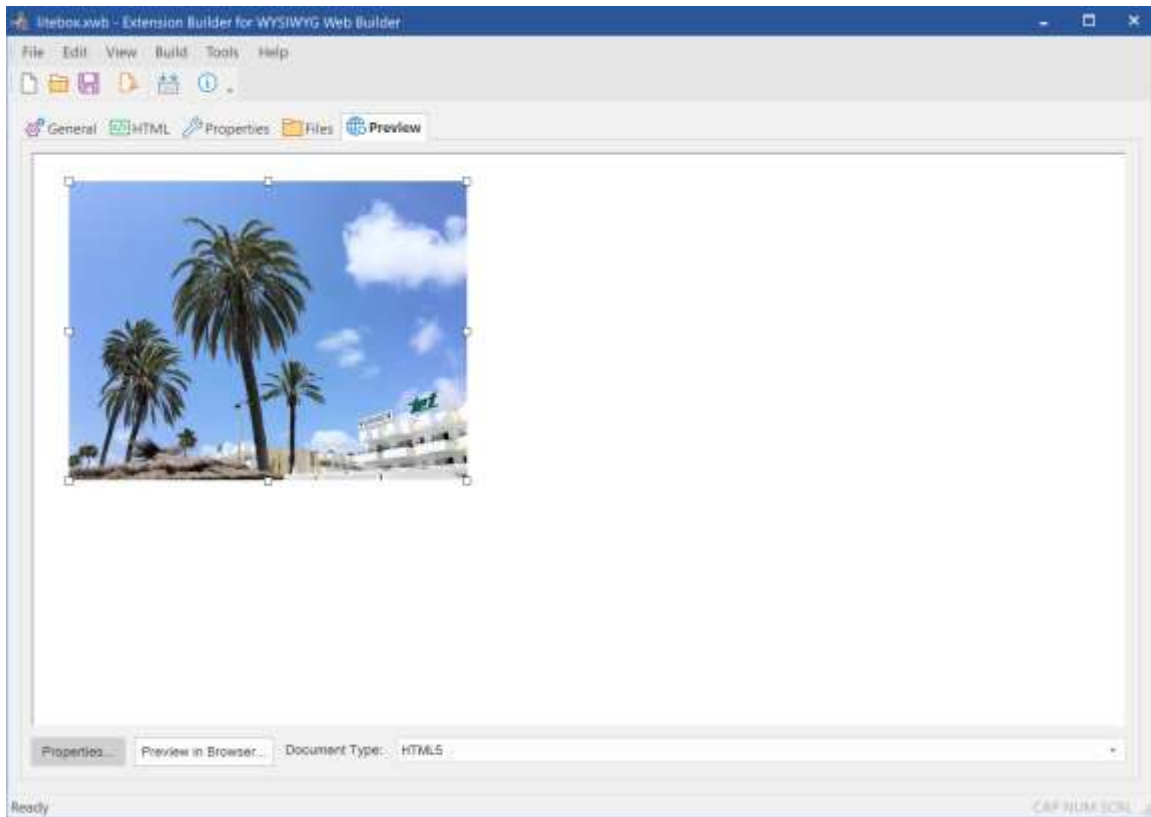
### ***Properties***

Click this button to view or edit the properties of the selected item.

Enable the option '**This file requires processing**' to replace variables with the current property values. For example if the color value in a HTML files needs to be controlled through properties, then you defined a property in the 'Properties page'.

## Preview Page

In this page you can preview the extension without having to start WYSIWYG Web Builder. It's possible to modify the properties just like in Web Builder and preview the results in the (default) browser. The values of the properties will be saved between sessions, so the next time you preview the extension you do not have to enter them again.



It's also possible to change the browser that is used for previewing. Click the **Preview in Browser** button and select 'Edit Browser List' to add/edit the list of browsers that can be used to preview the pages you are working on.

## Examples

In this section you will find a few examples to help you get started to create your first extension.

Note: the screenshots are based on an older version of Extension Builder.

### Example 1

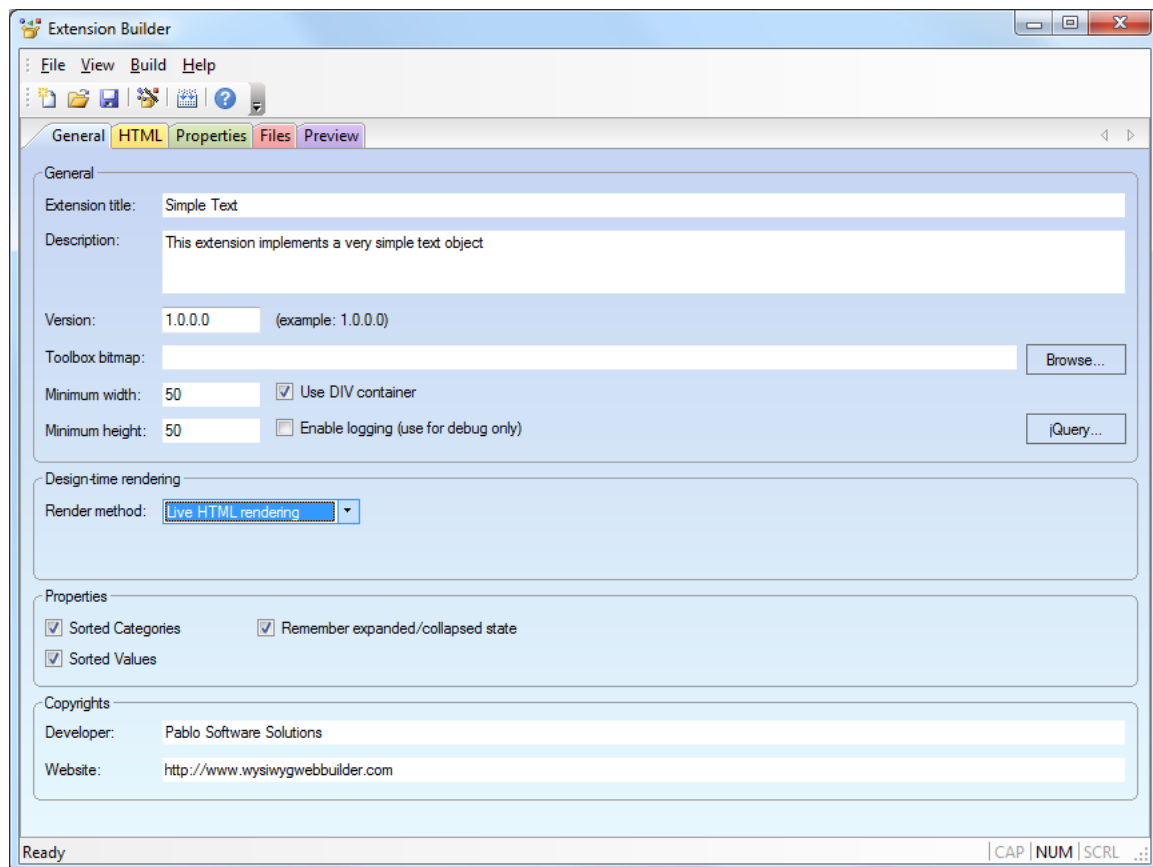
Let's start with a very simple example. This extension will do not much more than display a text. The text, font and color of the text can be modified by the user.

#### Step 1

Select Menu->File->New to create a new project.

Click the **General** tab to enter the general settings of the new extension.

- Set the Extension title to 'SimplyText'.
- Select a bitmap for the toolbar icon. If you leave this field empty the default icon will be used.
- Set the minimum width and height to 50.
- Enable 'Use DIV'
- Set the Render method to 'Live HTML rendering'.





## Step 2

Click the HTML tab. Now we will specify the HTML code for the extension. Copy/paste the following code to the **Code between <BODY> tag** field:

```
<font style="font-size:$size$px;font-family:$fontname$;"  
color="$color$">$text$</font>
```

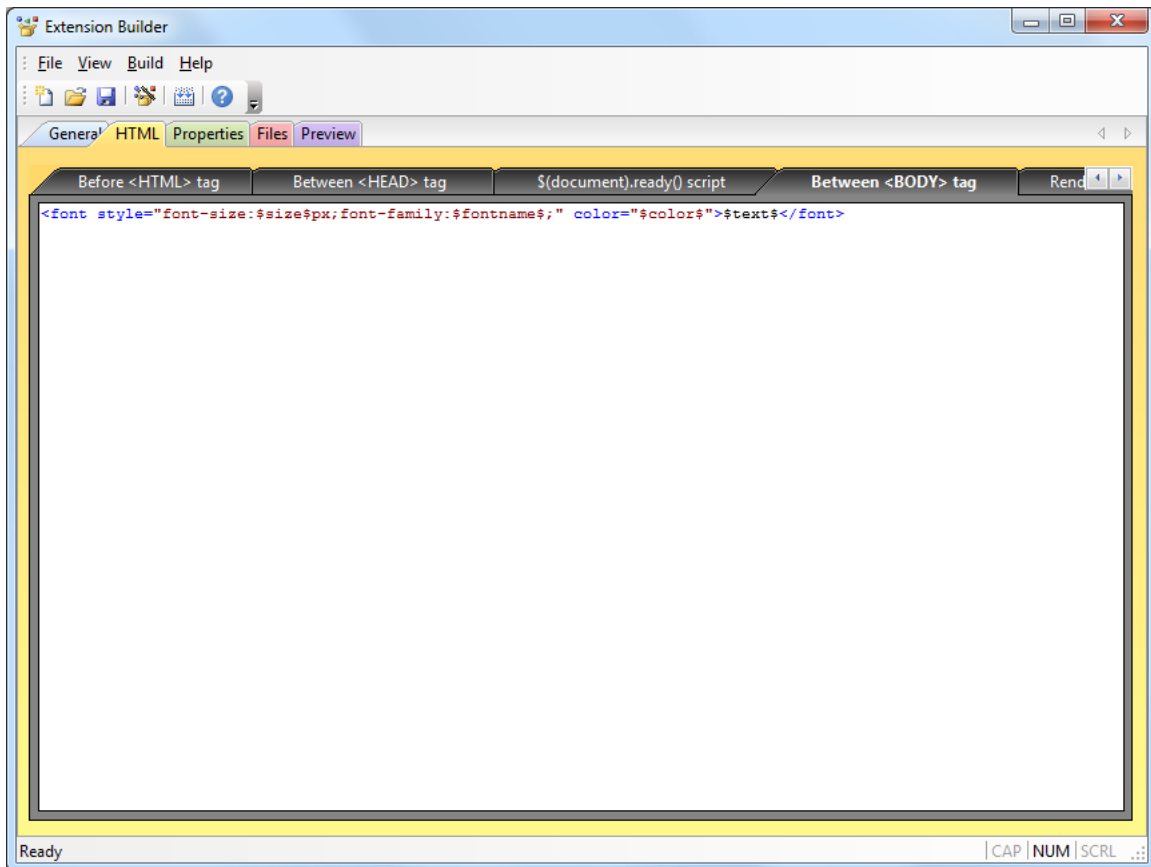
This simply piece of code has 4 variables:

**\$size\$**, specifies the size of the text.

**\$fontname\$**, specifies the font of the text.

**\$color\$**, specifies the color of the text.

**\$text\$**, specifies the displayed text.



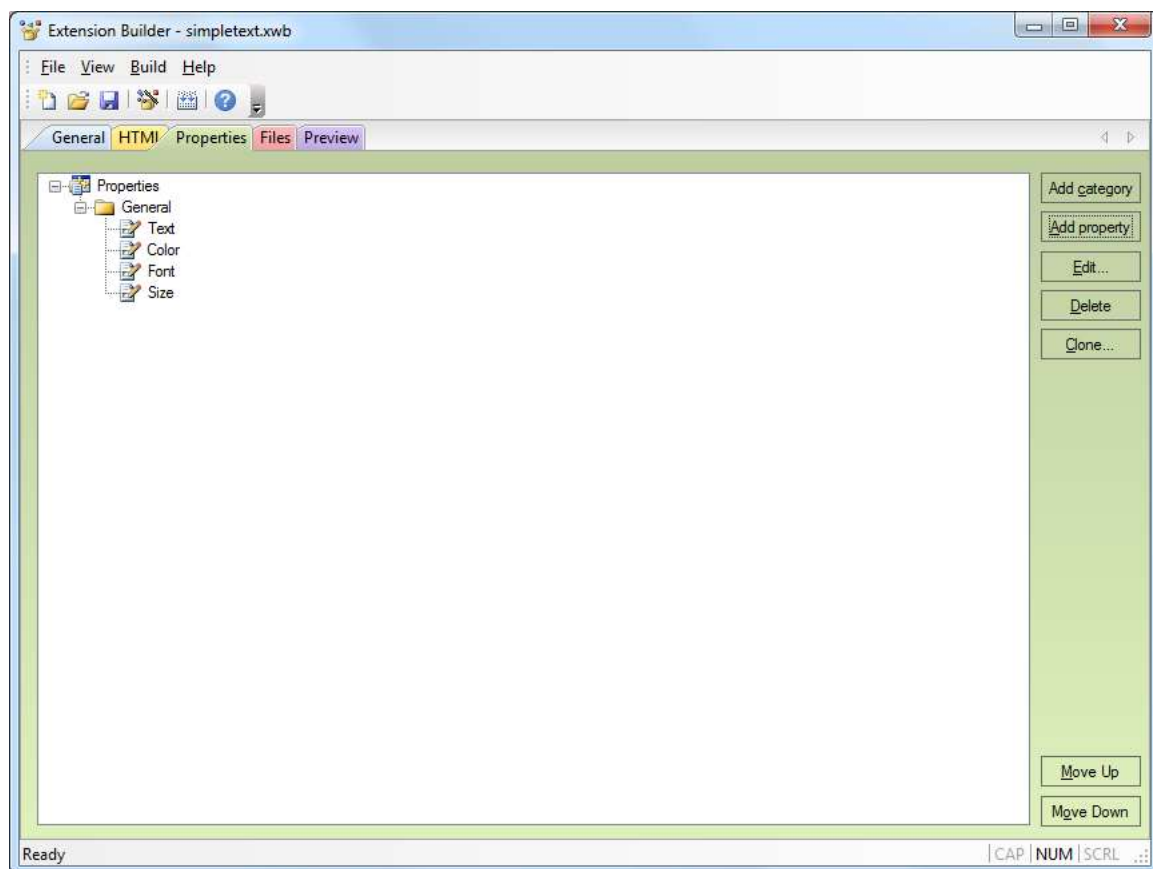
### Step 3

Click the Properties tab. This is where we will specify the properties which the user can modify.

For this extension we add a single category.

- Click 'Add Category', enter 'General' and click OK.
- To add properties to this category, make sure the text 'General' is selected before you click the 'Add Property' button.
- Insert 4 new properties and give them the following values:

Name	Description	Type	Variable	Default value
Text	Specifies the text	edit	\$text\$	Hello World!
Color	Specifies the color of the text	color	\$color\$	#000000
Font	Specifies the font	fontname	\$font\$	Arial
Size	Specifies the size	fontsize	\$size\$	12



### Step 4

Select Menu->File->Save As and save the project as **simpletext.xwb**

### Step 5

The final step is to build the extension so it can be used in WYSIWYG Web Builder. Select Menu-> Build->Build extension. A file called simpletext.wbx will be created in the same folder as the project file.

Congratulations, you have created your first extension for WYSIWYG Web Builder!

## Example 2

In this second example we're going to demonstrate the use of external files. We will create an extension based on a very cool effect by Christian Effenberger called 'Curl'.



### Step 1

First let's visit the website <http://www.netzgesta.de/curl/> and download the script. The file is called **curl.zip**. After you've downloaded the file you must unzip the files, for example in c:\javascripts\curl\.

Note that in this example we will only need the file **curl.js**

### Step 2

Select Menu->File->New to create a new project.

Click the **General** tab to enter the general settings of the new extension.

- Set the Extension title to 'Curl'.
- Select a bitmap for the toolbar icon. If you leave this field empty the default icon will be used.
- Set the minimum width and height to 50.
- Enable 'Use DIV'
- Set the Render method to 'Live HTML rendering'.

Note: the effect will not be applied to the image in design mode. You must preview or publish the page to see the effect!

### Step 3

Click the HTML tab.

Copy/paste the following code to the **Code between <HEAD> tag** field:

```
<script type="text/javascript" src="curl.js"></script>
```

#### Step 4

Click the HTML tab.

Copy/paste the following code to the **Code between <BODY> tag** field:

```

```

#### Step 5

Click the Properties tab. Now we will specify the properties the user can modify.

- Click 'Add Category', enter 'General' and click OK.
- To add properties to this category, make sure the text 'General' is selected before you click the 'Add Property' button.
- Insert 2 new properties and give them the following values:

Name	Description	Type	Variable	Default value
Image	Specifies the image	file	\$image\$	
Alternate Text	Specifies the alternate text	edit	\$alt\$	

#### Step 6

Click the Files tab. Here we will specify the files that will be embedded in the extension.

- Select 'Embedded Files' and click 'Add file(s)'
- Find 'curl.js' on your local disk. This is one of the files you've extracted in in step 1!
- Click 'Open' to add the file to the list.

#### Step 7

Select Menu->File->Save As and save the project as **curl.xwb**

#### Step 8

The final step is to build the extension so it can be used in WYSIWYG Web Builder. Select Menu-> Build->Build extension. A file called curl.wbx will be created in the same folder as the project file.

Note that the file 'curl.js' will be embedded inside the extension, there is no need to distribute the file separately.

The curl script supports many other features, but to keep this example understandable we've only implemented the basics. However it's easy to add more properties.

Let's say you want to control the size of the curl:

1. Change the code of **step 4** to  

```

```
2. Add a new property:  
Name: Size  
Description: Specifies the size of the curl  
Type: edit  
Variable: \$size\$  
Default value: 33

### Example 3

A popular type of extension is the image gallery. We've tried to come up with a generic way of implementing images galleries, but at the same time support many different types of galleries. We have done this by using XML templates

<xsl:template>.

In this example we will not create a real image gallery (yet), but we'll just give a demonstration how to use <xsl:template>. We will simply display the names of all selected images. In the next example we will use the same technique to build a real image gallery.

#### Step 1

Select Menu->File->New to create a new project.

Click the **General** tab to enter the general settings of the new extension.

- Set the Extension title to 'Image Info'.
- Select a bitmap for the toolbar icon. If you leave this field empty the default icon will be used.
- Set the minimum width and height to 50.
- Enable 'Use DIV'
- Set the Render method to 'Live HTML rendering'.

#### Step 2

Click the HTML tab. Now we will specify the HTML code for the extension.

Copy/paste the following code to the **Code between <BODY> tag** field:

```
$image_info$
```

Note that we've added only a single variable with the name `$image_info$`.

This will be used as a place holder for the actual HTML code which will be generated based on a XML template. See step 3.

#### Step 3

Click the Properties tab.

- Click 'Add Category', enter 'General' and click OK.
- To add properties to this category, make sure the text 'General' is selected before you click the 'Add Property' button.
- Insert 1 new property and give it the following values:

Name	Description	Type	Variable	Default value
Images	Specifies the images	gallery	\$image_info\$	

Because we have selected the 'gallery' type we must also specify a xml template. This template will tell WYSIWYG Web Builder what HTML/XML must be generated for the gallery. In step 4 we will create this template file.

#### Step 4

Now we will create the template for code generation. Open notepad (or another text editor). Create a new file and copy/paste the code below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
<table border="1">
<xsl:for-each select="GALLERY/IMAGE">
<tr>
<td><xsl:value-of select="FILENAME"/></td>
<td><xsl:value-of select="URL"/></td>
</tr>
</xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>
```

Save the file as 'imageinfo.xml'.

Since an XSL style sheet is an XML document, it always begins with the XML declaration: **<?xml version="1.0" encoding="UTF-8"?>**.

The next element, **<xsl:stylesheet>**, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).

The **<xsl:template>** element defines a template. The match="/" attribute associates the template with the root of the XML source document.

The content inside the **<xsl:template>** element defines the HTML to write to the output.

Everything inside the **<xsl:for-each>** element will be repeated for each image in the gallery. The **<xsl:value-of select="xxxxx">** element will be replaced with the actual properties of the current image.

The following values are available for the select attribute:

**FILENAME**

The filename of the current image.

**URL**

The URL for the current image (optional)

**TARGET**

The target for the URL (optional)

**TITLE**

The title for the current image (optional).

**DESCRIPTION**

The description for the current image (optional).

**WIDTH**

The width of the image (optional).

**HEIGHT**

The height of the image (optional).

**THUMB**

The name of the thumbnail image (optional).

This will only be included if 'Enable thumbnails' is on.

**THUMB\_WIDTH**

The width of the thumbnail image (optional).

This will only be included if 'Enable thumbnails' is on.

**THUMB\_HEIGHT**

The height of the thumbnail image (optional).

This will only be included if 'Enable thumbnails' is on.

When the page is published the resulting HTML may look like this:

```
<table border="1">
<tr>
<td>photo01.jpg</td>
<td>photo 01</td>
<td>This is the first photo</td>
</tr>
<tr>
<td>photo02.jpg</td>
<td>photo 02</td>
<td>This is the second photo</td>
</tr>
<tr>
<td>photo03.jpg</td>
<td>photo 03</td>
<td>This is the third photo</td>
</tr>
</table>
```

photo01.jpg	photo 01	This is the first photo
photo02.jpg	photo 02	This is the second photo
photo03.jpg	photo 03	This is the third photo

More information about xml templates is available online:  
[http://www.w3schools.com/xsl/xsl\\_templates.asp](http://www.w3schools.com/xsl/xsl_templates.asp)

### Step 5

Select Menu->File->Save As and save the project as **imageinfo.xwb**

### Step 6

The final step is to build the extension so it can be used in WYSIWYG Web Builder.  
Select Menu-> Build->Build extension. A file called imageinfo.wbx will be created in the same folder as the project file.



### XML template extra info

The XML template syntax is very powerful. It supports many additional features. Just to give you an idea, here are a few examples.

#### Example 3.1 - Sorting items.

Replace the code in step 4 with:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match='/ '>
<table border="1">
<xsl:for-each select="GALLERY/IMAGE">
<xsl:sort select="FILENAME"/>
<tr>
<td><xsl:value-of select="FILENAME"/></td>
<td><xsl:value-of select="URL"/></td>
</tr>
</xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>
```

The items will now be sorted on filename.

#### Example 3.2 - Filtering items.

Replace the code in step 4 with:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match='/ '>
<table border="1">
<xsl:for-each select="GALLERY/IMAGE">
<xsl:if test="contains(FILENAME, '.jpg')">
<tr>
<td><xsl:value-of select="FILENAME"/></td>
</tr>
</xsl:if>
</xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>
```

Only filenames with the .jpg extension will be displayed/used.

The complete documentation of the syntax is available on the Microsoft MSDN website:

[http://msdn.microsoft.com/en-us/library/ms256058\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms256058(VS.85).aspx)

## Example 4

Now it's time to create a more advanced image gallery...

We will use LiteBox (a modified version of LightBox v2.0) for this example.

### Step 1

You can download the script from this website: <http://www.doknowevil.net/litebox/>

Unzip the files to a folder on your computer.

### Step 2

Select Menu->File->New to create a new project.

Click the **General** tab to enter the general settings of the new extension.

- Set the Extension title to 'LiteBox'.
- Select a bitmap for the toolbar icon. If you leave this field empty the default icon will be used.
- Set the minimum width and height to 50.
- Enable 'Use DIV'
- Set the Render method to 'Display text only'.
- Set the display text to 'LiteBox Extension'.

### Step 3

Click the HTML tab.

Copy/paste the following code to the **Code between <HEAD> tag** field:

```
<link rel="stylesheet" href="css/lightbox.css" type="text/css"
media="screen" />
<script type="text/javascript" src="js/prototype-lite.js"></script>
<script type="text/javascript" src="js/moo.fx.js"></script>
<script type="text/javascript" src="js/litebox-1.0.js"></script>
<script type="text/javascript">
window.onload = function()
{
    initLightbox();
}
</script>
```

#### Step 4

Click the HTML tab.

Copy/paste the following code to the **Code between <BODY> tag** field:

```
$lightbox_images$
```

Note that we've added only a single variable with the name `$lightbox_images$`. This will be used as a place holder for the actual HTML code which will be generated based on a XML template. See step 5.

#### Step 5

Click the Properties tab.

- Click 'Add Category', enter 'General' and click OK.
- To add properties to this category, make sure the text 'General' is selected before you click the 'Add Property' button.
- Insert 3 new properties and give them the following values:

Name	Description	Type	Variable	Default value
Images	Specifies the images	gallery	<code>\$lightbox_images\$</code>	
Width	Specifies the thumbnail width	edit	<code>\$width\$</code>	100
Height	Specifies the thumbnail height	edit	<code>\$height\$</code>	100

Because we have selected the 'gallery' type we must also specify a xml template. This template will tell WYSIWYG Web Builder what HTML/XML must be generated for the gallery. In step 6 we will create this template file.

#### Step 6

Now we will create the template for code generation. Open notepad (or another text editor). Create a new file and copy/paste the code below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
<xsl:for-each select="GALLERY/IMAGE">
<a href="{FILENAME}" rel="lightbox[example]" title="{TITLE}"></a>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Save the file as 'lightbox.xml'.

Go back to the properties and configure this file for the 'Images' property.

### Step 7

Click the Files tab. Here we will insert the files from the lightbox zip package (see step 1).

The lightbox files are organized into 3 separate folders. Click 'New Folder' to add these folders to the embedded file structure:

**css**

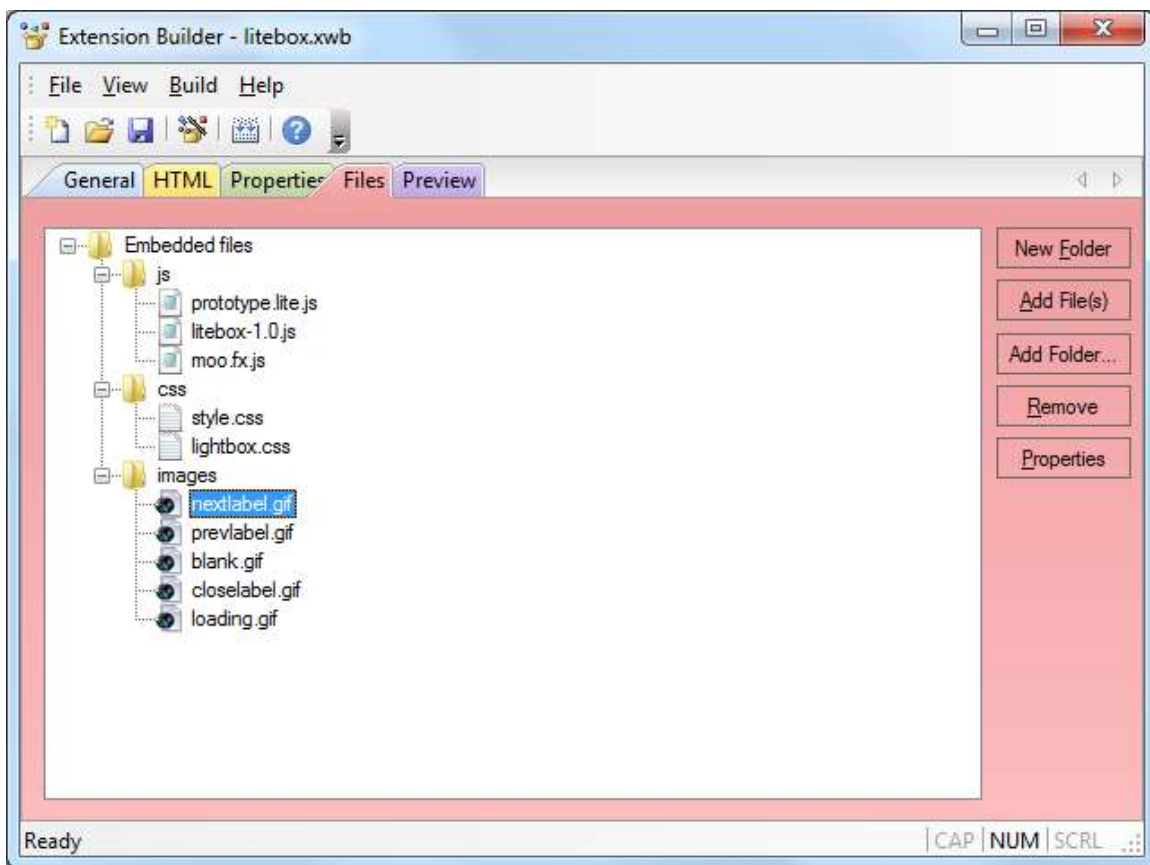
**images**

**js**

Add the file 'lightbox.css' to the css folder.

Add the files 'prototype.lite.js', 'lightbox-1.0.js' and 'moo.fx.js' to the js folder.

Add the files 'nextlabel.gif', 'prevlabel.gif', 'blank.gif', 'closelabel.gif' and 'loading.gif' to the images folder.



### Step 8

Select Menu->File->Save As and save the project as **lightbox.xwb**

### Step 9

The final step is to build the extension so it can be used in WYSIWYG Web Builder.

Select Menu-> Build->Build extension. A file called lightbox.wbx will be created in the same folder as the project file.

## Example 5

Here's yet another image gallery example: **Flash Carousel Slideshow** by Saverio Caminiti. This time we will let Web Builder create a XML data file. Usually such a file is used by flash based image galleries.



### Step 1

You can download the Flash Carousel SlideShow from this website:

<http://www.flshow.net/>

Unzip the files to a folder on your computer.

### Step 2

Select Menu->File->New to create a new project.

Click the **General** tab to enter the general settings of the new extension.

- Set the Extension title to 'Carousel'.
- Select a bitmap for the toolbar icon. If you leave this field empty the default icon will be used.
- Set the minimum width and height to 50.
- Disable 'Use DIV' (uncheck!)
- Set the Render method to 'Display text only'.

Set the display text to 'Carousel Extension'.

### Step 3

Click the HTML tab.

Copy/paste the following code to the **Code between <BODY> tag** field:

```
<div
style="position:absolute;left:$LEFT$px;top:$TOP$px;width:$WIDTH$px;height:$HEIGHT$px
;z-index:$Z_INDEX$">
<object width="$WIDTH$" height="$HEIGHT$" classid="clsid:D27CDB6E-AE6D-11cf-96B8-
444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#versio
n=9,0,0,0">
<param name="movie" value="Carousel.swf" />
<param name="flashvars" value="xmlfile=carousel.xml" />
<param name="quality" value="High" />
<param name="bgcolor" value="#FFFFFF" />
<embed src="Carousel.swf" quality="High" type="application/x-shockwave-flash"
pluginspage="http://www.macromedia.com/go/getflashplayer">
</embed>
</object>
</div>
```

#### Note:

The variables \$ID\$, \$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!

### Step 4

Click the Files tab. Insert the Carousel.swf file.

## Step 5

The carousel flash slideshow needs a XML configuration file to work properly. The configuration file contains the properties of the carousel and the images to be displayed.

Create a new file in notepad. Copy/paste the code below:

```
<slide_show>
  <options>
    <debug>>false</debug>
    <background>0xFFFFFF</background>
    <interaction>
      <rotation>mouse</rotation>
      <speed>180</speed>
      <view_point>mouse</view_point>
      <console>onClick</console>
    </interaction>

    <far_photos>
      <size>50</size>
      <amount>25</amount>
      <transparency>>false</transparency>
    </far_photos>

    <reflection>
      <amout>$amout$</amout>
      <blur>$blur$</blur>
      <distance>$distance$</distance>
      <alpha>$alpha$</alpha>
    </reflection>
  </options>

  $photos$
</slide_show>
```

This code has 5 variables. 4 of them are to control the reflection (\$amout\$, \$blur\$, \$distance\$ and \$alpha\$). If you like you can also replace all other static values with variables (like the rotation speed, far photo size etc).

The **\$photos\$** variable is a reference to a XML template that we will create in one of the next steps.

Save the file as 'carousel.xml' and insert this file to the list in extension builder. Select 'carousel.xml' in the list and click 'Properties'. Enable 'This file requires processing', so that the variables inside the XML file will be replaced with the user's selections.

## Step 6

Click the Properties tab.

- Click 'Add Category', enter 'General' and click OK.
- To add properties to this category, make sure the text 'General' is selected before you click the 'Add Property' button.
- Insert 1 property and give it the following values:

Name	Description	Type	Variable	Default value
Images	Specifies the images	gallery	\$photos\$	

Because we have selected the 'gallery' type we must also specify a xml template.

This template will tell WYSIWYG Web Builder what HTML/XML must be generated for the gallery. In step 7 we will create this template file.

- Click 'Add Category', enter 'Reflection' and click OK.
- To add properties to this category, make sure the text 'Reflection' is selected before you click the 'Add Property' button.
- Insert the 4 properties for the reflection:

Name	Description	Type	Variable	Default value
Amout	Specifies the reflection's amout	edit	\$amout\$	200
Blur	Specifies the reflection's blur	edit	\$blur\$	15
Distance	Specifies the reflection's distance	edit	\$distance\$	10
Alpha	Specifies the reflection's alpha	edit	\$alpha\$	75

### Step 7

Now we will create the template for code generation. Open notepad (or another text editor). Create a new file and copy/paste the code below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match='/ '>
<xsl:for-each select="GALLERY/IMAGE">
<photo href="{FILENAME}" target="{TARGET}">
<xsl:value-of select="FILENAME"/>
</photo>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Save the file as 'carousel.xml'.

Go back to the properties and configure this file for the 'Images' property.

### Step 8

Select Menu->File->Save As and save the project as **carousel.xwb**

### Step 9

The final step is to build the extension so it can be used in WYSIWYG Web Builder. Select Menu-> Build->Build extension. A file called carousel.wbx will be created in the same folder as the project file.



## Example 6

In this example we will demonstrate the use of the **dataset** property type. Just like the gallery this uses a XML template to control the HTML output. However in this case you will define the dataset structure. This can be useful when an extension requires the input of multiple items like media players or navigation menus.

Let's start with a simple example. We'll display a list of files (selected by the user) and a short description.

### Step 1

Select Menu->File->New to create a new project.

Click the **General** tab to enter the general settings of the new extension.

- Set the Extension title to 'File List'.
- Select a bitmap for the toolbar icon. If you leave this field empty the default icon will be used.
- Set the minimum width and height to 50.
- Enable 'Use DIV'
- Set the Render method to 'Live HTML rendering'.

### Step 2

Click the HTML tab. Now we will specify the HTML code for the extension.

Copy/paste the following code to the **Code between <BODY> tag** field:

```
$files$
```

Note that we've added only a single variable with the name `$files$`.

This will be used as a place holder for the actual HTML code which will be generated based on a XML template. See step 3.

### Step 3

Click the Properties tab.

- Click 'Add Category', enter 'General' and click OK.
- To add properties to this category, make sure the text 'General' is selected before you click the 'Add Property' button.
- Insert 1 new property and give it the following values:

Name	Description	Type	Variable	Default value
Files	Specifies the files	dataset	\$files\$	

Because we have selected the 'dataset' type we must also specify a xml template. This template will tell WYSIWYG Web Builder what HTML/XML must be generated for the data. In step 4 we will create this template file.

Next, add these elements (click 'Add' to a new element):

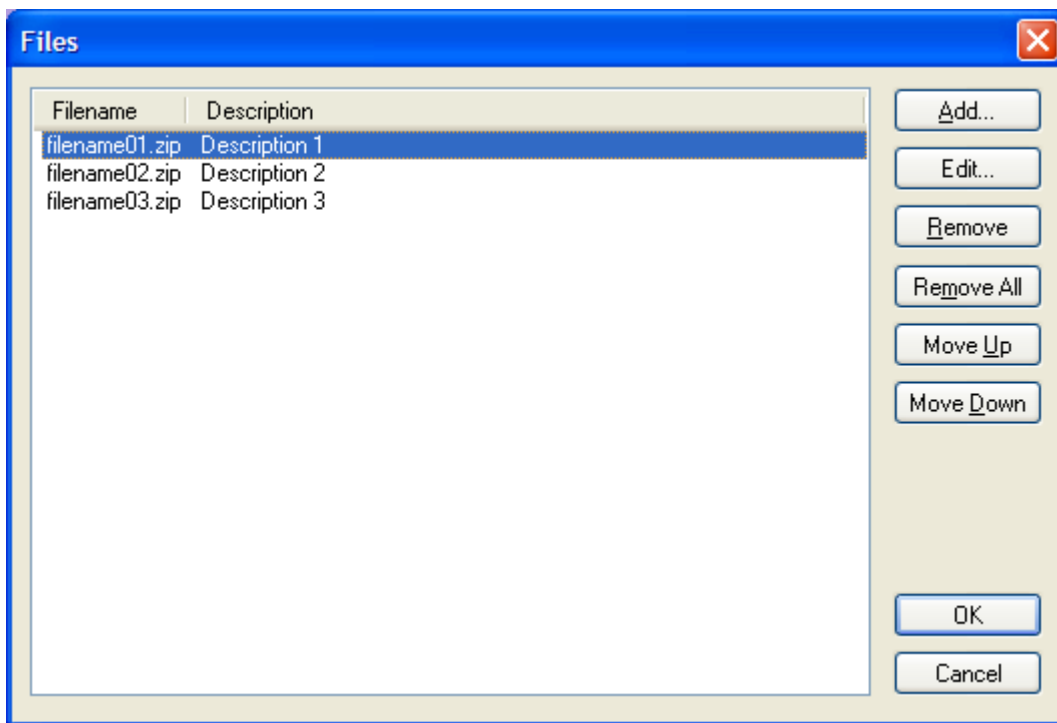
Display name	Description	Type	Element name	Default value
File	Specifies the filename	file	FILENAME	
Description	Description of the file	edit	DESCRIPTION	

The configuration above will result in a dataset that looks something like this:

```
<DATASET>
  <ITEM>
    <FILENAME>filename01.mp3</FILENAME>
    <DESCRIPTION>Description 1</DESCRIPTION>
  </ITEM>
  <ITEM>
    <FILENAME>filename02.mp3</FILENAME>
    <DESCRIPTION>Description 2</DESCRIPTION>
  </ITEM>
  <ITEM>
    <FILENAME>filename03.mp3</FILENAME>
    <DESCRIPTION>Description 3</DESCRIPTION>
  </ITEM>
</DATASET>
```

Note the tags DATASET and ITEM will be created by Web Builder. For every item the user adds to the list in the extension's properties, a new ITEM will be added to the dataset. The XML template determines how the data will be transformed to HTML (see step 4).

Values entered in Web Builder by the user:



**Tip:**

Holding 'SHIFT' while clicking 'Add' in dataset properties will copy/clone the selected item.

#### Step 4

Now we will create the template for code generation. Open notepad (or another text editor). Create a new file and copy/paste the code below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match='/'>
<table border="1">
<xsl:for-each select="DATASET/ITEM">
<tr>
<td><a href="{FILENAME}"><xsl:value-of select="FILENAME"/></a>
</td>
<td>
<xsl:value-of select="DESCRIPTION"/>
</td>
</tr>
</xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>
```

Save the file as 'filelist.xml'.

Go back to the properties and configure this file for the 'Files' property.

When the page is published, the resulting HTML may look like this:

```
<table border="1">
<tr>
<td><a href="filename01.mp3">filename01.mp3</a></td>
<td>Description 1</td>
</tr>
<tr>
<td><a href="filename02.mp3">filename02.mp3</a></td>
<td>Description 2</td>
</tr>
<tr>
<td><a href="filename03.mp3">filename03.mp3</a></td>
<td>Description 3</td>
</tr>
</table>
```

<a href="#">filename01.zip</a>	Description 1
<a href="#">filename02.zip</a>	Description 2
<a href="#">filename03.zip</a>	Description 3

More information about xml templates is available online:

[http://www.w3schools.com/xsl/xsl\\_templates.asp](http://www.w3schools.com/xsl/xsl_templates.asp)

**Step 5**

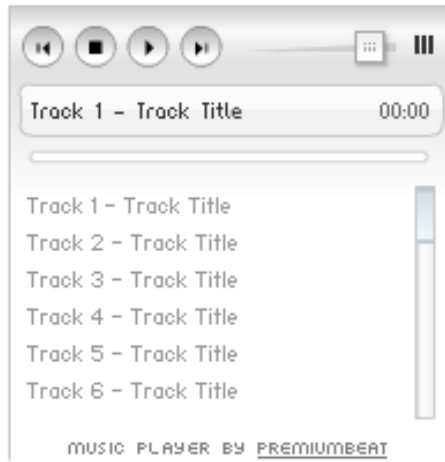
Select Menu->File->Save As and save the project as **filelist.xwb**

**Step 6**

The final step is to build the extension so it can be used in WYSIWYG Web Builder.  
Select Menu-> Build->Build extension. A file called filelist.wbx will be created in the same folder as the project file.

## Example 7

Here's an example of how to create a (multi) MP3 player extension.



### Step 1

Download **Multiple Tracks Flash mp3 Player With Menu** from this website:

[http://www.premiumbeat.com/flash\\_resources/free\\_flash\\_music\\_player/](http://www.premiumbeat.com/flash_resources/free_flash_music_player/)

Unzip the files to a folder on your computer.

### Step 2

Select Menu->File->New to create a new project.

Click the **General** tab to enter the general settings of the new extension.

- Set the Extension title to 'Multi MP3 Player'.
- Select a bitmap for the toolbar icon. If you leave this field empty the default icon will be used.
- Set the minimum width and height to 50.
- Disable 'Use DIV' (uncheck!).
- Set the Render method to 'Display text only'.

Set the display text to 'Placeholder for MP3 Player Extension'.

### Step 3

Click the HTML tab.

Copy/paste the following code to the **Code between <HEAD> tag** field:

```
<script type="text/javascript" src="swfobject.js"></script>
```

#### Step 4

Copy/paste the following code to the **Code between <BODY> tag** field:

```
<div id="flashPlayer" style="position:absolute; left:$LEFT$px; top:$TOP$px; width:$WIDTH$ ;height:$HEIGHT$; z-index:$Z_INDEX$">
This text will be replaced by the flash music player.
</div>

<script type="text/javascript">
var so = new SWFObject("playerMultipleList.swf", "mymovie", "$WIDTH$", "$HEIGHT$", "7", "#FFFFFF");
so.addVariable("playerSkin", "$playerskin$");
so.addVariable("autoPlay", "$autoplay$");
so.addVariable("playlistPath", "playlist.xml");
so.write("flashPlayer");
</script>
```

Note:

The variables \$ID\$, \$LEFT\$, \$TOP\$, \$WIDTH\$, \$HEIGHT\$ and \$Z\_INDEX\$ are predefined variables. They will be replaced by the actual position and size of the object.

#### Step 5

Click the Files tab. Insert swfobject.js and playerMultipleList.swf.

#### Step 6

Create a new file in notepad. Copy/paste the code below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
    $tracks$
</xml>
```

The **\$tracks\$** variable is a reference to a XML template that we will create in one of the next steps.

Save the file as 'playlist.xml' and insert this file to the list in extension builder. Select 'playlist.xml' in the list and click 'Properties'. Enable 'This file requires processing', so that the variables inside the XML file will be replaced with the user's selections.

#### Step 7

Click the Properties tab.

- Click 'Add Category', enter 'General' and click OK.
- To add properties to this category, make sure the text 'General' is selected before you click the 'Add Property' button.
- Insert the following properties:

Name	Description	Type	Variable	Default	Options
MP3 Files	Select the files to be played	dataset	\$tracks\$		
PlayerSkin	Skin of the player	Options	\$playerskin\$	4	1, 2, 3, 4, 5
AutoPlay	Enable auto play	Options	\$autoplay\$	no	no, yes

Because we have selected the 'dataset' type we must also specify a xml template. This template will tell WYSIWYG Web Builder what HTML/XML must be generated for the gallery. In step 8 we will create this template file.

Next, add these elements to the dataset (MP3 Files):

Display name	Description	Type	Element name	Default value
Path	Specifies the path of the MP3 file	file	PATH	
Title	Specifies the title	edit	TITLE	

### Step 8

Now we will create the template for code generation. Open notepad (or another text editor). Create a new file and copy/paste the code below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
<xsl:for-each select="DATASET/ITEM">
<track>
  <path><xsl:value-of select="PATH"/></path>
  <title><xsl:value-of select="TITLE"/></title>
</track>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Save the file as 'multimp3.xsl'.

Go back to the properties and configure this file for the 'MP3 Files' property.

### Step 8

Select Menu->File->Save As and save the project as **multimp3.xwb**

### Step 9

The final step is to build the extension so it can be used in WYSIWYG Web Builder. Select Menu-> Build->Build extension. A file called multimp3.wbx will be created in the same folder as the project file.

## Example 8

In this example we use the 'navigation' property which allows us to generate menu scripts. This can be either a simple (one level) menu or complex tree (multi level) menu. Just like the 'gallery' and 'dataset' properties, the 'navigation' property also uses XML templates to generate the output code. If you are not yet familiar with templates please read the information in previous examples first.

### Step 1

Select Menu->File->New to create a new project.

Click the **General** tab to enter the general settings of the new extension.

- Set the Extension title to 'Menu List'.
- Select a bitmap for the toolbar icon. If you leave this field empty the default icon will be used.
- Set the minimum width and height to 50.
- Enable 'Use DIV'

### Step 2

Click the HTML tab. Now we will specify the HTML code for the extension.

Copy/paste the following code to the **Code between <BODY> tag** field:

```
$menu_items$
```

Note that we've added only a single variable with the name `$menu_items$`.

This will be used as a place holder for the actual HTML code which will be generated based on a XML template. See step 3.

### Step 3

Click the Properties tab.

- Click 'Add Category', enter 'General' and click OK.
- To add properties to this category, make sure the text 'General' is selected before you click the 'Add Property' button.
- Insert 1 new property and give it the following values:

Name	Description	Type	Variable	Default value
Items	Specifies the menu items	navigation	\$menu_items\$	

Set the menu type to 'List'.



#### Step 4

Now we will create the template for code generation. Open notepad (or another text editor). Create a new file and copy/paste the code below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
<table border="1">
<xsl:for-each select="MENU/ITEM">
<tr>
<td><xsl:value-of select="TITLE"/></td>
<td>
<xsl:choose>
<xsl:when test="@current='true'">
<xsl:value-of select="URL"/>
</xsl:when>
<xsl:otherwise>
<a href="{URL}"><xsl:value-of select="URL"/></a>
</xsl:otherwise>
</xsl:choose>
</td>
<td><xsl:value-of select="TARGET"/></td>
<td><xsl:value-of select="ALT"/></td>
</tr>
</xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>
```

Save the file as 'menulist.xml'.

The following values are available for the select attribute:

**TITLE**

The name/title of the current item.

**URL**

The URL for the current item

**TARGET**

The target for the URL

**ALT**

The alternate text for the menu item.

Note that we also used a conditional statement to check for the current page (@current is an attribute of the ITEM tag). In this case the URL value will not be a link. In a real menu the @current attribute can be useful if you want to display the current page in a different style.

When the page is published the resulting HTML may look like this:

```
<table border="1">
<tr>
<td>Home</td>
<td>index.html</td>
<td>_blank</td>
<td>Home</td>
</tr>
<tr>
<td>Products</td>
<td><a href="products.html">products.html</a></td>
<td>_blank</td>
<td>Products</td>
</tr>
<tr>
<td>Downloads</td>
<td><a href="downloads.html">downloads.html</a></td>
<td>_self</td>
<td>Downloads</td>
</tr>
</table>
```

Home	<a href="#">./index.html</a>	_blank	Home
Products	<a href="#">./products.html</a>	_blank	Products
Downloads	<a href="#">./downloads.html</a>	_self	Downloads

### Step 5

Select Menu->File->Save As and save the project as **menulist.xwb**

### Step 6

Select Menu-> Build->Build extension. A file called menulist.wbx will be created in the same folder as the project file.

## Example 9

Here's another example for the 'navigation' property. This one uses the 'tree' option which can be used to create complex tree (multi level) menus. Please check out **Example 8** first for a basic menu example.

### Step 1

Select Menu->File->New to create a new project.

Click the **General** tab to enter the general settings of the new extension.

- Set the Extension title to 'Menu List'.
- Select a bitmap for the toolbar icon. If you leave this field empty the default icon will be used.
- Set the minimum width and height to 50.
- Enable 'Use DIV'

### Step 2

Click the HTML tab. Now we will specify the HTML code for the extension.

Copy/paste the following code to the **Code between <BODY> tag** field:

```
$menu_items$
```

Note that we've added only a single variable with the name `$menu_items$`.

This will be used as a place holder for the actual HTML code which will be generated based on a XML template. See step 3.

### Step 3

Click the Properties tab.

- Click 'Add Category', enter 'General' and click OK.
- To add properties to this category, make sure the text 'General' is selected before you click the 'Add Property' button.
- Insert 1 new property and give it the following values:

Name	Description	Type	Variable	Default value
Items	Specifies the menu items	navigation	\$menu_items\$	

Set the menu type to 'Tree'.

#### Step 4

Now we will create the template for code generation. Open notepad (or another text editor). Create a new file and copy/paste the code below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="/">
  <xsl:call-template name="menu" />
</xsl:template>

<xsl:template name="menu">
  <ul id="menutree">
    <xsl:apply-templates select="MENU/ITEM" />
  </ul>
</xsl:template>

<xsl:template match="ITEM">
  <li>
    <a href="{URL}" target="{TARGET}"><xsl:value-of select="TITLE"/></a>
    <xsl:if test="ITEM">
      <ul>
        <xsl:apply-templates select="ITEM"/>
      </ul>
    </xsl:if>
  </li>
</xsl:template>

</xsl:stylesheet>
```

Save the file as 'menutree.xsl'.

Note that this template recursively creates a (nested) list of all menu items. Many menu scripts use this structure as the basis for multi level menus.

When the page is published the resulting HTML may look like this:

```
<ul id="menutree">
  <li><a href="page1.html">Item 1</a>
  <ul>
    <li><a href="page2.html ">Sub Item 1</a> </li>
    <li><a href="page3.html ">Sub Item 2</a>
    <ul>
      <li><a href="page4.html ">Sub Item 1</a> </li>
      <li><a href="page5.html ">Sub Item 2</a> </li>
    </ul>
  </li>
</ul>
</li>
<li><a href="page6.html ">Item 2</a>
</li>
<li><a href="page7.html ">Item 3</a>
</li>
</ul>
```

## Step 5

Now let's turn this into a drop down menu by adding some styling to the items.



Copy/paste the following code in the '**Between <head> tag**' section:

```
<style type="text/css">
#menutree, #menutree ul
{
  padding: 0;
  margin: 0;
  list-style-type: none;
}
#menutree
{
  margin: 25px 0 100px 10px;
  position: relative;
  z-index: 2;
}
#menutree li
{
  float: left;
  position: relative;
}
#menutree a, #menutree a:visited
{
  background: #CCCCCC;
  border: 1px solid #BBBBBB;
  color: #444444;
```

```

        display: block;
        font-family: arial;
        font-size: 12px;
        height: 25px;
        line-height: 24px;
        text-decoration: none;
        text-indent: 5px;
        width: 120px;
    }
    #menutree li:hover > a
    {
        background: #444444;
        color: #FFFFFF
    }
    #menutree li ul
    {
        display: none;
    }
    #menutree li:hover > ul
    {
        display: block;
        left: 121px;
        position: absolute;
        top: 0;
    }
    #menutree > li:hover > ul
    {
        left: 0;
        top: 25px;
    }
</style>

```

### Step 6

Select Menu->File->Save As and save the project as **menutree.xwb**

### Step 7

Select Menu-> Build->Build extension. A file called menutree.wbx will be created in the same folder as the project file.

### **How to install extensions so they will be available in Web Builder?**

You can use the Extension Manager (Menu->Tools->Extension Manager) to install the extension.

Alternatively, you can manually copy all files from the zip file to the Web Builder extensions folder. Usually this folder is in this location:

My Documents\WYSIWYG Web Builder\system\extensions\

More information is available here:

<https://www.wysiwygwebbuilder.com/forum/viewtopic.php?t=7234>

### **Requirements**

- Windows Windows7/Window8/Windows10 or later
- Registered version of WYSIWYG Web Builder 16 or later
- Knowledge of HTML/XML/Scripting
- It's recommended to copy 'ExtBuilder.exe' to the same folder as the WYSIWYG Web Builder (C:\Program Files\WYSIWYG Web Builder 16\), so it can access the default themes and other assets.

### **License**

This application is free of charge for registered users of WYSIWYG Web Builder 15 or newer.

This application is provided as-is, with no warranty expressed or implied. Use this application at your own risk. The author assumes no liability for any loss associated with the use of this application. If you do not agree with the terms of this license, do not install this application.

This software is not officially supported by Pablo Software Solutions. However, bugs and other issues will usually be fixed a.s.a.p.

### **Feedback**

Comments/suggestions are welcome on the support forum:

<https://www.wysiwygwebbuilder.com/forum/>

This application was created by Pablo  
Copyright 2022 Pablo Software Solutions  
<https://www.wysiwygwebbuilder.com>