



**CODE  
PROJECT®**  
For those who code

[articles](#)
[Q&A](#)
[forums](#)
[lounge](#)
 Search for articles, questions, tips


# Visual Studio Design Patterns add-in/extension



dmihailescu, 23 Sep 2015

[MS-PL](#)

Rate:



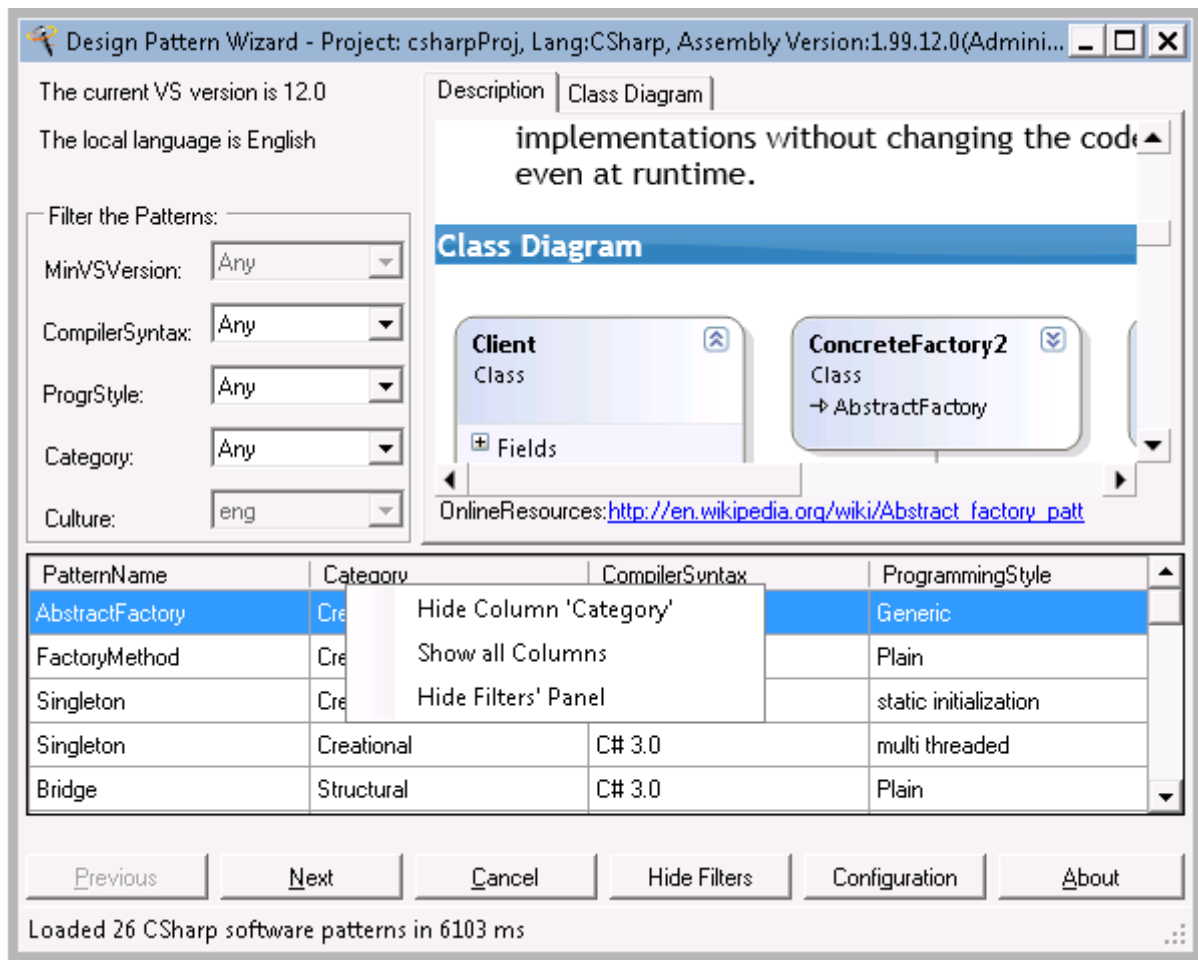
4.92 (159 votes)

A Visual Studio add-in that inserts some commonly known OO patterns into your working project and searches your highlights online.


[Download DesignPatternsAllSetups\\_2.10.zip - 6.27 MB](#)

[Download source\\_src.zip source\\* - 1.27 MB](#)

\*I recommend that you download the latest code and binaries from [codeplex TFS](#).



## Introduction

If you have been programming using a object oriented language, then you must have heard of [design patterns](#). There are a number of books and web sites on this topic, but while using Visual Studio, you would have to go out of the IDE, refresh your memory about

what a particular software pattern does and type or copy the code for it.

Wouldn't be easier to have an integrated Visual Studio tool to help you select the right pattern and insert the code for you with only a few clicks?

This is exactly what this Visual Studio add-in/extension does: it indexes some commonly known OO patterns like the *Gang's of four*, presents them in easy to search user interface and inserts the code into your working project.

It provides you with parameters to further customize the patterns and it works for languages supported by Visual Studio like C#, VB.net, C++ flavors and can be extended to other Visual studio supported languages like F#.

Additionally this add-in/extension also provides some unrelated functionality like the ability to easily search online the selected text or item from code, xml, html, error or references windows. The express versions of Visual Studio are unable to use this add-in, but all the premium or community editions could.

## Background

I've created this [Visual Studio add-in](#) for VS 2008, 2010, 2012 & 2013 and an extension for VS 2015 (no longer supports add-ins) to act as a convenience feature for people using the Microsoft IDE (the express versions do not support add-ins/extensions). If you download the binaries or the code from [There](#) is a lot of code to deal with [Visual Studio automation](#), [custom installer actions](#), and obviously the patterns loading and rendering. It should support Visual Studio 2008, 2010, 2012, 2013, 2015 on Windows XP or later. To allow for easy expansion and extensibility I used XML files and the file system's hierarchical to represent the object oriented pattern information as templates. Also the architecture is based on [Managed Extensibility Framework \(MEF\)](#) to provide for more customization in additional assemblies.

Since the code base is very large as you can see in the [codeplex repository](#), I'm only going to focus on usage and the addition of new patterns to the existing ones this time. If this article spurs enough interest I will come back with a sequel for creating custom assemblies for further customization, licensing, etc.

## Using the add-in

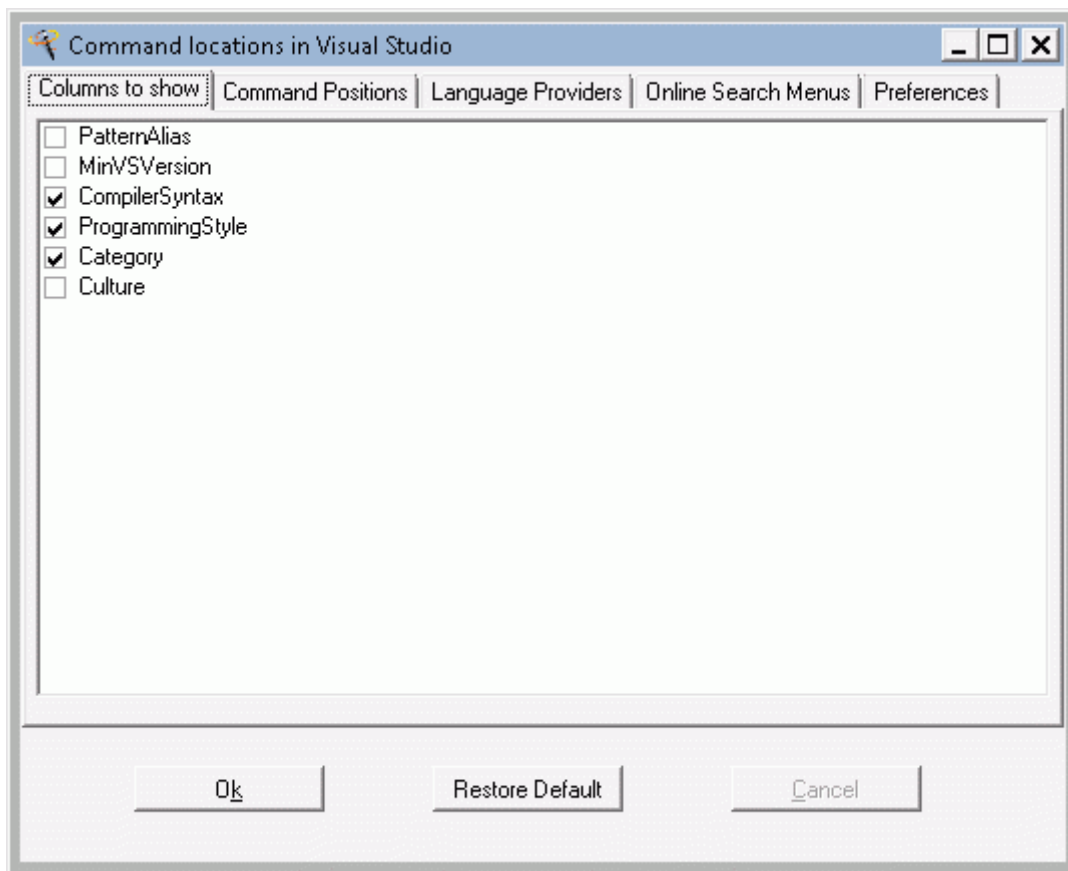
If you are in a rush to see a demo of what this code does you can watch this [youtube video](#). Below is a more detailed and up to date description of what is going on when installing and using it. The current source code and binaries do not have all the *Gang of four design patterns* implemented. If you want to experience all of them, they are available on a 90 day trial in a download from [codeplex](#) for C# and VB.net (so far).

## Installing the add-in/extension

Start by downloading the current installation archive, from this page or [codeplex](#) and unzip it. You should notice the presence of *TemplateLibrary.zip* file in the download and its corresponding expanded folder under the *Misc* directory. That is the folder holding the pattern templates and will be the target of our change in order to add your own templates to the installation script as you will see later.

It's best to close Visual Studio before the installation and uninstall any previous version of this add-in if you had one already. Also running as administrator seems to make the installation process go smoother. If you are still using Windows XP, look for setup.exe in the folder(s) corresponding to the same Visual Studio version(s) you are using, otherwise just start the install.bat and make your version selection. You will be presented with a dialog during the installation to select the menu positions and the programming languages you wish to use. You can keep the defaults and continue or choose the language providers, command positions, etc.

For VS2015 you don't have to do anything specific as you can select option 5 or just run the .vsix file. The rest of the setup will be triggered when you start the extension for the first time.

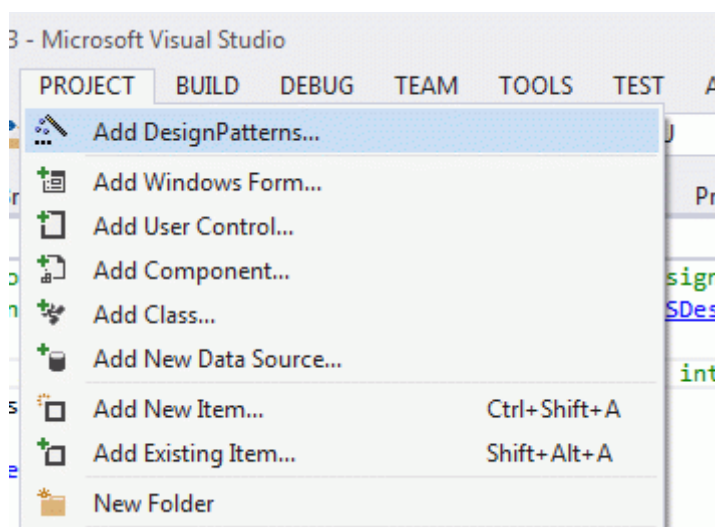


There is a specific add-in or extension for each version of Visual Studio starting with version 9 (2008) to 12 (VS2013). An uninstall link for its respective version should show up under windows startup menu. Also a 'Reset Add-in' menu in the *Start\DesignPatterns\Uninstall Design Patterns for 20XX* is provided in case for some reason this add-in does not show up in the menus after loading the projects or in the 'Manage Add-ins' Visual Studio menu. For VS 2015 you can uninstall this extension only from the "Tools" menu.

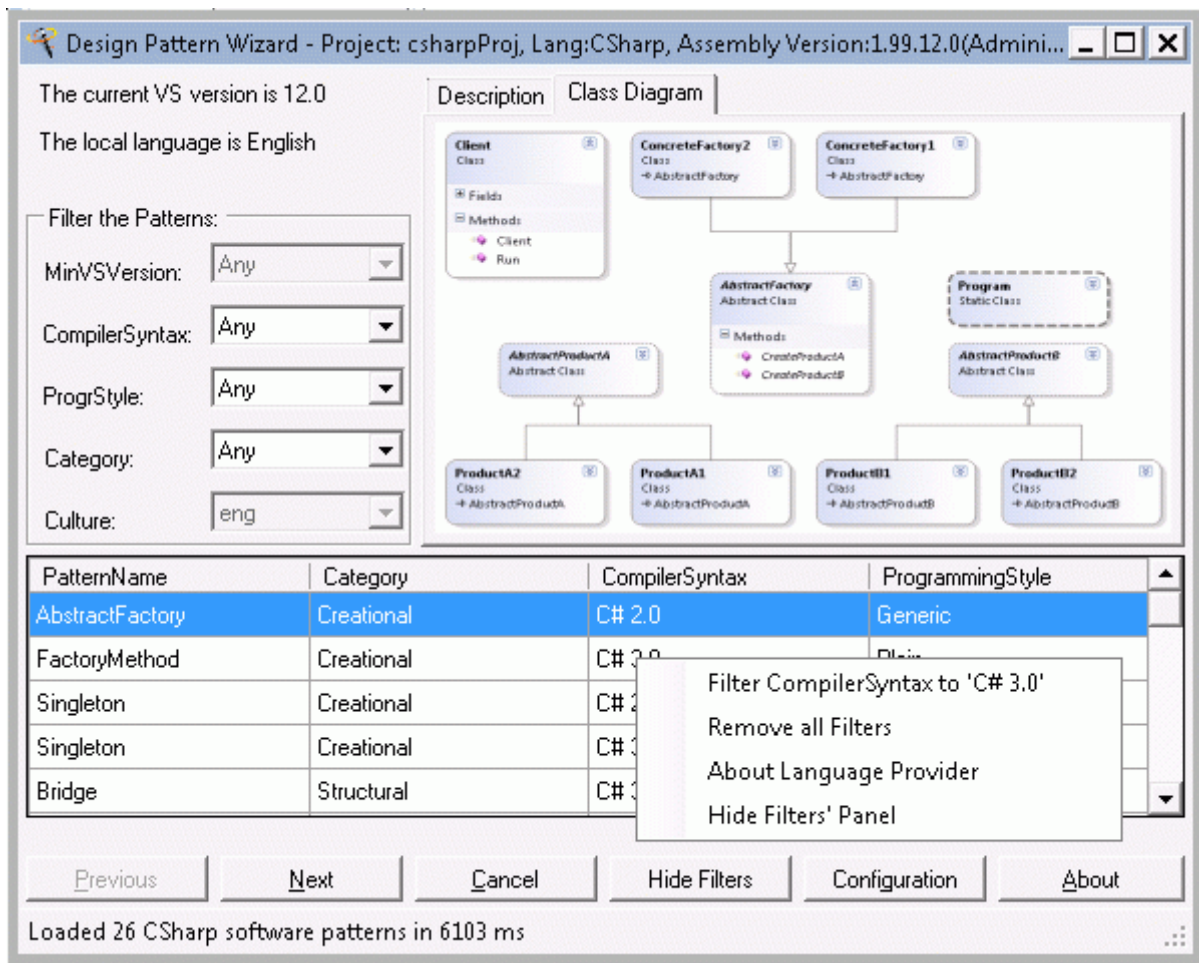
If some languages are of no interest to you, you can remove their respective provider by unchecking it from the configuration tab. You will have to restart Visual Studio to see the changes.

## Inserting a pattern into the current project

After installing it, open an existing project in Visual Studio. Supported project types are VB, C#, native C/C++ and CLI/C++. Right click on the working project and select *Add DesignPatterns...* or look for the same menu entry under the Tools or Project main menu.



A Window resembling the image below should open. In the title you should see the current project and its programming language.



You can narrow down to the pattern you need by filtering the patterns using the combo boxes on the top left of the form.

These filters can be hidden or shown by clicking on the Hide or Show Filters button.

The same filtering can be achieved by right clicking on the specific grid cell or on the grid's headers.

You can also reorder by clicking on the column headers of the grid. Typing the first letter of the pattern will focus on the row with the matching PatternName. If the bottom grid was too cluttered, you can also hide the columns you are not interested in by right clicking on their header. Select one pattern and click *Next*.

On this screen you can change the values of some parameters should you needed it (defaults are valid, you can always refactor them later in the IDE for languages like C# or VB.Net). Select *Next* again and you have a chance to see how the code will look like. You can go back by clicking on the *Previous* button, or select *Generate* to insert the code files into the current Visual Studio project. A *Configuration* button allows you can choose where to place the Pop-up menus or make other customizations by invoking the Configuration form as shown before.

## Add your own pattern template

### How the patterns templates are stored

Before adding new patterns you need to understand how the existing ones work.

The pattern templates and their XML descriptions are usually stored in C:\Program Files (x86)\Software Fantasies\DesignPatterns for VS 20XX\TemplateLibrary after the installation.

The metadata is stored in xml files like CSharpTemplates.xml, the name matching the target language:

Hide Copy Code

```
<?xml version="1.0" encoding="utf-8"?>
<PatternTemplates PatternsVersion="1.3">
.....
  <PatternTemplate CodeLang="CSharp" PatternName="Singleton">
    <PatternProperties>
      <PatternAlias>Singleton</PatternAlias>
      <MinVSVersion>9.0</MinVSVersion>
      <OnlineResources>http://msdn.microsoft.com/en-us/library/ff650316.aspx</OnlineResources>
      <CompilerSyntax>C# 2.0</CompilerSyntax>
    </PatternProperties>
  </PatternTemplate>
</PatternTemplates>
```

```

        <ProgrammingStyle>static initialization</ProgrammingStyle>
        <Category>Creational</Category>
    </PatternProperties>
    <FileTemplates>
        <FileTemplate FileName="Singleton.cs">SingletonTemplate.cs</FileTemplate>
    </FileTemplates>
    <TemplateArguments>
        <TemplateArgument ArgumentName="RootNamespace" IsRuntime="True" Description ="NameSpace"/>
        <TemplateArgument ArgumentName="Singleton" IsRuntime="false" Description ="Singleton class
Name">ConcreteSingleton</TemplateArgument>
    </TemplateArguments>
</PatternTemplate>
.....
</PatternTemplates>

```

In the xml snippet above you will recognize the columns that appear on the gridview in the main form. Some default values for the missing elements are based on the pattern name as described in the *PatternName* attribute. Most of the code responsible for loading a pattern template is shown below:

Hide Shrink ▲ Copy Code

```

public virtual void Init(ILanguageProvider languageProvider, XElement templateElem)
{
    //.....
    PatternName = templateElem.Attribute("PatternName").Value;
    try
    {
        try
        {
            this.CodeLanguage = (CodeLangEnum)Enum.Parse(typeof(CodeLangEnum),
templateElem.Attribute("CodeLang").Value, true);
        }
        catch
        {
            CodeLanguage = CodeLangEnum.Unsupported;
        }

        var properties = templateElem.Element("PatternProperties");
        PictureStoredAs = DefaultStorage;
        DescriptionStoredAs = DefaultStorage;
        Culture = "eng";
        MinVSVersion = _fCtrVSVersion;
        if (properties != null)
        {
            float minVSVer = _fCtrVSVersion;
            foreach (XElement e in properties.Elements())
            {
                switch (e.Name.ToString())
                {
                    case "PatternAlias":
                        PatternAlias = e.Value;
                        break;
                    case "MinVSVersion":
                        if (!float.TryParse(e.Value, out minVSVer))
                            minVSVer = _fCtrVSVersion;
                        MinVSVersion = minVSVer;
                        break;
                    case "Picture":
                        _iRepository.GetPicture(this, e);
                        break;
                    case "OnlineResources":
                        OnlineResources = e.Value;
                        break;
                    case "CompilerSyntax":
                        CompilerSyntax = e.Value;
                        break;
                    case "ProgrammingStyle":
                        ProgrammingStyle = e.Value;
                        break;
                }
            }
        }
    }
}

```

```

        case "Description":
            _iRepository.GetDescription(this, e);
            break;
        case "Category":
            Category = e.Value;
            break;
        case "Culture":
            Culture = e.Value; // "eng" if missing
            break;
        default:
            break;
    }
}
}
//fill in with defaults
IProjectInfo projectInfo = _languageProvider.GetAsService<IProjectInfo>();
if (OnlineResources == null)
{
    OnlineResources = string.Format(@"www.bing.com/search?q={0}+software+design+pattern+{1}",
PatternName, projectInfo.CodeLang);
}
if (Description == null)
{ //try adding default file if value not in xml file

    _iRepository.GetDescription(this, null);
}
if (Picture == null)
{ //try adding default file if value not in xml file
    _iRepository.GetPicture(this, null);
}
if (ProgrammingStyle == null)
{
    ProgrammingStyle = "Plain";
}
if (PatternAlias == null)
{
    PatternAlias = PatternName;
}
LoadCodeTemplates(templateElem);
}
catch (Exception ex)
{
    IWin32Window iwnd = _languageProvider.GetAsService<IWizardWindow>() as IWin32Window;
    iwnd.ShowMessageBox(ex.Message, "Can not initialize template " + PatternName);
}
}

```

You can notice the default values for the missing elements: *MinVSVersion*(9.0), *ProgrammingStyle* (Plain), *PatternAlias* ( the pattern name value), *Culture* (eng) and *OnlineResources* are automatically generated. The minimum you must populate are *CompilerSyntax*, *ProgrammingStyle* and *Category*.

For the *Description*, the default file name is generated by concatenating the *PatternName* with the .txt, .rtf or .html/.htm whichever is present. If a file with that name and one of those extensions can not be found, the description will become of *StoredAs* type as *Url*, with the value the same as the *OnlineResources*. This way even if you don't have a matching file, or a description with the *StoredAs* type as *Text*, you will still get something to display in the HTML control.

Hence it is a very good idea to add a valid *OnlineResources* element to navigate to the right page if you don't provide a local file with the description. Using a local htm or html file will give you the extra benefit of inserting the text from the *Intent* paragraph as a comment in the generated source code and also can offer collapsible headers if the same document structure was used.

For the *Picture*, the default file name is generated by concatenating the *PatternName* with the .png, .bmp or .jpg/.jpeg, whichever is present.

The content files are following the same rule, with the extension being of type .h, .cs or .vb based on the *CodeLang* attribute of the pattern. These files are in the subdirectories matching the pattern's name.

## Add the template pattern

Since the list of designed patterns implemented is not comprehensive, but rather just a starting point, I'm going to describe how to add your own new pattern templates consisting of description, code, picture and later, the optional arguments.



1. Start by adding a new *PatternTemplate* entry to the xml file corresponding to your target language. Then add another XML element called *PatternProperties*. If you fill at least the *PatternName*, *CodeLang*, *Category*, *ProgrammingStyle* and *CompilerSyntax*, you should be able to see them as well as the defaults in the gridview of the form. All the pattern specific files will go in the folder with the same name e.g. *C:\Program Files (x86)\Software Fantasies\DesignPatterns for VS 2013\TemplateLibrary\Singleton*
2. For the Description, if you don't like the fallback to the *OnlineResources* mentioned above, you can start by setting the *SavedAs* attribute to Text and specify the content in the value of the node. That's good as a start, but eventually you would end up creating an html, rtf or txt file with the name of the pattern. It is a good idea to use the existing htm files as a template for your own software patterns.
3. Add a picture with the class diagram and the file name same as the pattern. If you restart Visual Studio, the Wizard form should be populated correctly
4. Before adding the source code, I reiterate that you should change the default auto generated web link *Online Resources* to one of your liking.
5. In each pattern folder, you'll create a subfolder for each language you need to add your pattern like VB, CSharp and CPP. The latter holds both NativeCpp and ManagedCPP samples. The programming language is specified in the *CodeLang* attribute of the *PatternTemplate* element. Just add a file name of Pattern name concatenated with 'Template' and add the proper extension for the respective language e.g. .cs, .vb or .h.

At this point, following the wizard, you should be able to insert the file you've created into the selected project. If you get errors you should use the other Templates elements and language subfolders as examples to find the errors.

## Adding arguments and multiple files to the pattern

So far you saw that you could add a file to your selected project and that its content is up to you. That might be all right for C# or VB.net since you can later easily change class and member names using the built in refactoring of the IDE. For C++ and other languages this is not supported and usually you will want to insert multiple files at once not just one like a header. Hence the concept of file templates and template arguments is introduced. Here is how to add them:

1. Add an element call *FileTemplates* to the *PatternTemplate* element. Then add elements called *FileTemplate* to it. The *FileTemplateName* attribute is the name that will be saved on disk and in the project. The value of the element is the file name of the template with source code.
2. Add an element call *TemplateArguments* to the *PatternTemplate* element. Then add elements called *TemplateArgument* to it. Populate the *ArgumentName*, *IsRuntime*, *Description* and the value accordingly. These attributes will show up when you click Next on the Wizard form. Only the value can be changed by editing and will allow renaming of designated members. The only Runtime argument is RootNamespace and holds the default name space of the project. The other arguments are custom and they will replace text marked in the file templates.
3. In the source code file to be inserted replaced the strings for targeted members by adding a prefix and suffix made of the \$ character (e.g. Bridge becomes \$Bridge\$). The string without the \$ will become the *ArgumentName* and the value to replace it will be the value of the *TemplateArgument*. All arguments are at the *PatternTemplate* level not specific to each *FileTemplate*.

## Improving the pattern template creation

The drawback to the process described so far is that you will need to restart Visual Studio for each change made to the xml file or the source code file templates, because of the memory caching that is going on. If you forget that, you won't see your changes.

To avoid that, I have created program called VSEmulator.exe that will host the wizard form and do everything the Visual Studio does, except for the inserting of the files into the project. This executable takes a parameter as a project name to decide which language to work with. Here is the improved creation process:

1. Get the source code from [codeplex TFS](#) and copy the content from *C:\Program Files (x86)\Software Fantasies\DesignPatterns for VS 20XX\TemplateLibrary* into the Misc folder you've just created by importing the source code. As an alternative you can unzip the downloaded *TemplateLibrary.zip* into the *Misc* folder so the new directory path will look like *..\DesignPatterns\Misc\TemplateLibrary*. If you got it from TFS before, you can skip this step.
2. If you've installed it, uninstall the Design Patterns by selecting the Startup Menu by clicking the link *Start\DesignPattern\Uninstall Design Patterns for 20XX*. That is to avoid conflicts between different paths to the files.
3. Open the right version of the DesignPatterns\_vXX solution and build it. Set the VSEmulator\_vXX as the startup project, set the argument as a desired project file name (can be a fictitious one) and run it. Now you can easily see the results of the changes in the local *TemplateLibrary* folder, as described above immediately.
4. When you think that you've got the pattern in the final shape (you can compile the generated code), you can change the startup Project to DesignPatterns\_vXX and run it. You should be able to see the changes in the IDE (context or pull down menu), and then you could insert or view the new template source code you've created.
5. When done adding templates you could restart Visual Studio and build the installation kit by building *DesignPatternsSetup\_vXX*. That works for VS 2008 and 2010, but for the newer versions you will have to go in the *Misc* folder

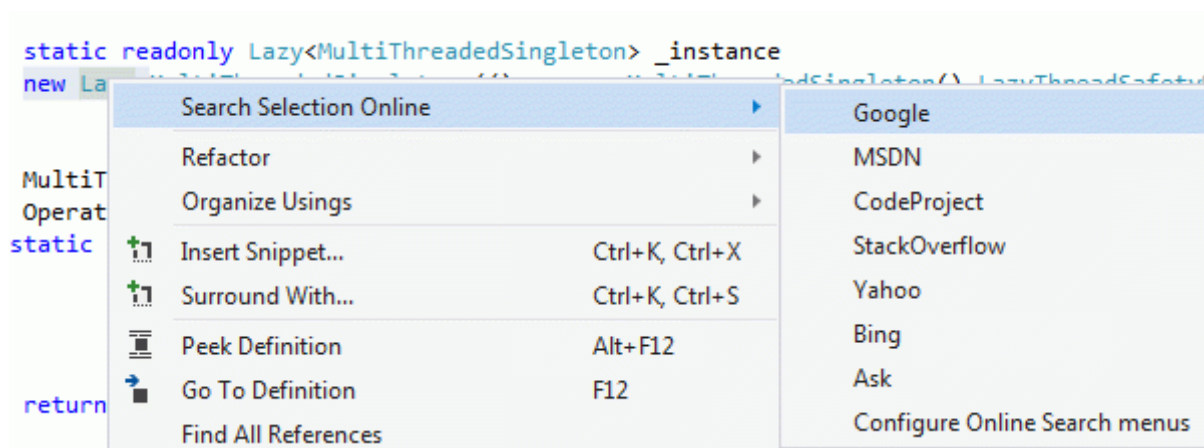
and run the *buildAll.bat* to build everything. This is the best way to build the installation scripts, but if you don't want to build everything, you can just zip the *..\DesignPatterns\Misc\TemplateLibrary* folder into the *TemplateLibrary.zip* archive and replace the old *TemplateLibrary.zip* with your new one in the installation folder. Now you can just run *install.bat* and get to see your new template patterns too.

The templates you add need not be only OO design patterns. It can be any content you wish: services, custom classes, etc. You just need to come up with new name categories if they don't fit the preexisting mold.

## Using the quick online search

### How to use it

The same add-in/extension allows you to select some text, error or reference, right click it and then search online without the extra step of copying and pasting in a search engine. Additionally you have the option to display the results directly in Visual Studio's internal browser window or the default Windows browser as you have chosen in the configuration form above and also to select which search sites you want in the configuration grid.



### Implementation

Here it is briefly how this functionality was implemented during add-in's OnConnection invocation.

Hide Shrink ▲ Copy Code

```
private void CreateSearchMenus(DTE2 _applicationObject)
{
    var menus = MEFController.Instance.SearchProviders.Select(p => new
    {
        SearchProviderName = p.Metadata.SearchProviderName,
        URL = p.Metadata.URL,
        SupportedWnds = p.SearchLocations,
        SP = p,
    });
    WindowPopupEnum[] allLocations = Utils.GetAllEnumValues<WindowPopupEnum>();

    var VSWindows = new[]
    {
        new {VSWndType = WindowPopupEnum.CodeWindow, Caption = "Search Selection Online"},
        new {VSWndType = WindowPopupEnum.XAMLEditor, Caption = "Search Selection Online"},
        new {VSWndType = WindowPopupEnum.ReferenceItem, Caption = "Search Reference Online"},
        new {VSWndType = WindowPopupEnum.ErrorList, Caption = "Search Error Online"},
        new {VSWndType = WindowPopupEnum.HTML, Caption = "Search Selection Online"},
    };
    try
    {
        for (int j = 0; j < VSWindows.Length; j++)
        {
            if (VSWindows[j].VSWndType == WindowPopupEnum.None)
                continue; //no popUp for this VS window
            CommandBar oCommandBar = ((CommandBars)_applicationObject.CommandBars)
                [VSWindows[j].VSWndType.GetDescription()];
```



```

//Add the main menu
CommandBarPopup oPopup = (CommandBarPopup)
oCommandBar.Controls.Add(MsoControlType.msoControlPopup,
System.Reflection.Missing.Value,
System.Reflection.Missing.Value, 1, true);
oPopup.Caption = VSWindows[j].Caption;

CommandBarButton self = oPopup.Control as CommandBarButton;
//Add a separator line
oPopup.BeginGroup = true;

//Add the search menus (sub-item)
int pos = 0;
foreach (SearchProvider item in MEFController.Instance.SearchProviders)
{
    SearchProvider searchProvider = item; //this assignment avoids Lambda iteration bug in
    VS2008, otherwise only the last element is yielded
    if ((VSWindows[j].VSWndType & searchProvider.SearchLocations) == WindowPopupEnum.None)
        continue;
    pos++;
    //Add the search menus (sub-item)
    CommandBarButton oControl =
        (CommandBarButton)oPopup.Controls.Add(MsoControlType.msoControlButton,
        System.Reflection.Missing.Value,
        System.Reflection.Missing.Value, pos, true);
    oControl.Caption = searchProvider.Metadata.SearchProviderName;
    oControl.Tag = searchProvider.Metadata.URL;
    oControl.ToolTipText = "Search " + searchProvider.Metadata.SearchProviderName + "
for...";
    oControl.Click += new _CommandBarButtonEvents_ClickEventHandler((CommandBarButton ctrl,
ref bool c) =>
    {
        DTE2 dte = (ctrl.Application as DTE2);
        if (dte == null)
            return;
        object oContext = WindowPopupEnum.None;
        SearchProvider.Context context =
dte.ActiveWindow.GetSelectedContext(searchProvider);
        if (!string.IsNullOrEmpty(context.SelectedText))
        {
            try
            {
                ItemOperations itemOperations = dte.ItemOperations;
                if (itemOperations != null)
                {
                    string navigationUrl = searchProvider.GetNavigationUrl(context);
                    switch (Utils.OnlineSearchDestination)
                    {
                        case 1:
                            System.Diagnostics.Process proc = new
System.Diagnostics.Process();
                            proc.StartInfo = new ProcessStartInfo(navigationUrl);
                            proc.Start();
                            break;
                        case 2:
                            Window window = itemOperations.Navigate(navigationUrl,
vsNavigateOptions.vsNavigateOptionsDefault);
                            break;
                        default:
                            break;
                    }
                }
            }
            catch
            {}
        }
    });

    if (!_commandBarButtons.Any(kv => kv.Key ==
searchProvider.Metadata.SearchProviderName))
        _commandBarButtons.Add(searchProvider.Metadata.SearchProviderName, oControl); //save
subitem from garbage collection in VS2008
}

```

```

pos++;

//Add the search menus (sub-item)
CommandBarButton cControl =
    (CommandBarButton)oPopup.Controls.Add(MsoControlType.msoControlButton,
        System.Reflection.Missing.Value,
        System.Reflection.Missing.Value, pos, true);
cControl.Caption = "Configure Online Search menus";
cControl.Tag = null;
cControl.ToolTipText = "Search Configuration";
cControl.Click += new _CommandBarButtonEvents_ClickEventHandler((CommandBarButton ctrl, ref
bool c) =>
{
    using (ConfigForm cfg = new ConfigForm(3))
    {
        cfg.ShowDialog(this);
    }

});
if (!_commandBarButtons.Any(kv => kv.Key == "SearchConfig " +
VSWindows[j].VSWndType.GetDescription()))
    _commandBarButtons.Add("SearchConfig " + VSWindows[j].VSWndType.GetDescription(),
cControl);//save subitem from garbage collection in VS2008
oPopup.BeginGroup = false;
}
}
catch (Exception ex)
{
    Logging.Instance.Log(1, "Exception thrown in CreateContextMenu :{0}", ex.Message);
}
}

```

For VS 2015 that uses an extension this code is a bit different. One of the search providers is described below. All of them are similar and are subclasses of *SearchProvider*.

Hide Shrink ▲ Copy Code

```

public class SearchProvider
{
    public class Context
    {
        public string SelectedText { get; set; }
        public string Caption { get; set; }
        public IProjectInfo ProjectInfo { get; set; }
        public string Suffix { get; set; }
        public string SearchPhrase
        {
            get
            {
                if (string.IsNullOrEmpty(Suffix))
                    return SelectedText.Trim();
                else
                    return string.Format("{0} {1}", SelectedText.Trim(), Suffix);//only for code and
error window, not xaml or references
            }
        }
    }

    public Context CreateContext(IProjectInfo prInf)
    {
        Context ctx = new Context() { Suffix = null, ProjectInfo = prInf,/*SP = this,*/ };
        return ctx;
    }

    protected virtual WindowPopupEnum DefaultSearchLocations
    {
        get
        {
            return Utils.AllPopUpWnds;
        }
    }

    public WindowPopupEnum SearchLocations
    {

```

```

        get
        {
            WindowPopupEnum loc = DefaultSearchLocations;
            RegistryKey reg = Registry.CurrentUser.OpenSubKey(string.Format("{0}\\{1}",
Common.Utils.AppHive, Metadata.SearchProviderName), false);
            if (reg != null)
            {
                object obj = reg.GetValue("SearchLocations", loc, RegistryValueOptions.None);
                loc = (WindowPopupEnum)System.Convert.ToInt32(obj);
                loc &= Metadata.SupportedWnds;
                reg.Close();
            }
            return loc;
        }
        set
        {
            RegistryKey reg = Registry.CurrentUser.CreateSubKey(string.Format("{0}\\{1}",
Common.Utils.AppHive, Metadata.SearchProviderName), RegistryKeyPermissionCheck.Default);
            if (reg != null)
            {
                reg.SetValue("SearchLocations", (uint)(value & Metadata.SupportedWnds),
RegistryValueKind.DWord);
                reg.Close();
            }
        }
    }

    public ISearchProviderMetaData Metadata { get; protected set; }

    public string GetNavigationUrl(Context context)
    {
        //add more from Context
        return Metadata.URL +
Uri.EscapeUriString(context.SearchPhrase).Replace("#", "%23").Replace("/", "%2F").Replace("+", "%2B"); ;
    }
    internal SearchProvider()
    {
        if (this.GetType() != typeof(SearchProvider))
        {
            string type = this.GetType().Name;
            Metadata = MEFController.Instance.GetProviderSearchMetadata(type);
        }
    }
}

[ExportSearchProvider(SearchProviderName = "Google", Description = "Google search provider", URL =
"http://www.google.com/search?q=",
SupportedWnds = Utils.AllPopUpWnds)]
public sealed class Google : SearchProvider
{
    protected override WindowPopupEnum DefaultSearchLocations
    {
        get
        {
            return WindowPopupEnum.CodeWindow | WindowPopupEnum.ErrorList | WindowPopupEnum.HTML |
WindowPopupEnum.XAMLEditor;
        }
    }
    private Google()
    { }
}
}

```

It should be easy to add your own search providers by just creating similar classes annotated appropriately . This way they will show in the configuration grid and the pop up menu.

## Points of Interest

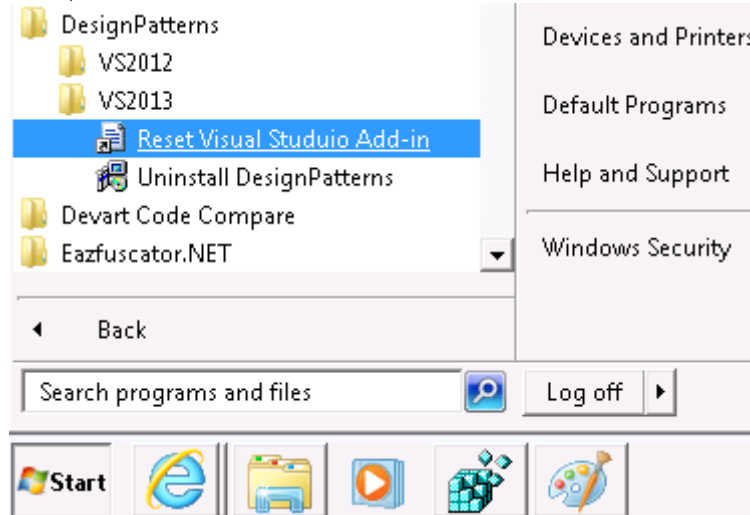
The source code is quite large and I recommend using the [codeplex TFS repository](#) to get the latest, rather than the zip provided.

Same is true for the released installation scripts. Currently only some OO patterns for C#, VB, native C++ and C++\CLI are implemented, but you can add them yourself based on the instructions given above, or just get the rest of the *Gang of four patterns* by downloading the [latest 90 day trial](#).

So far, I have tested it only on Windows 7 64 bit and Windows XP 32 bit.

If the first time you invoke the add-in won't load correctly, just restart Visual Studio.

If the error persists for VS 2008 - 2013 and this add-in does not show up in the menus or in the 'Manage Add-ins' Visual Studio menu, there is a 'Reset Visual Studio Add-in' menu in the Windows' Start button->DesignPatterns\VS20XX to fix this issue.



Should you still see errors when Visual Studio (2008 to 2013) starts, from the VS command prompt run "devenv /resetaddin \*" command. More documentation might be found [here](#). For VS 2015, the extension can be uninstalled from the VS menu "Tools"->"Extensions and Updates..." ->"Design Patterns Extension"->"Uninstall", or reinstalled by running the install.bat.

## History

1. May 7, 2014: Version 1.91 - first release.
2. August 22 2014: Version 1.93 added some UI improvements.
3. November 13 2014: Version 1.93 adds some UI improvements and version 1.94 adds a 60 day [trial](#) for C# and VB.net for the missing patterns of the *Gang of four*.
4. February 2015: Version 1.95 added quick online search menus.
5. May 2015: Version 1.97 improved project type detection and added pattern description to the code generation.
6. May 2015: Version 1.98 added argument validation, more configuration options, collapsible headers and pattern cross-reference.
7. September 2015: Versions 2.0 & 2.10 added an extension that can be used by VS 2015 and some minor fixes and features.

The last version might not be the latest since the most up to date code is on [codeplex](#). Also another link to this add-in is on [Visual Studio Gallery](#).

## License

This article, along with any associated source code and files, is licensed under [The Microsoft Public License \(Ms-PL\)](#)

## Share



## About the Author



## dmihailescu

Software Developer (Senior)

United States 

Decebal Mihailescu is a software engineer with interest in .Net, C# and C++.

## You may also be interested in...

[An Extensible Master-Page Framework for ASP.NET 1.1 Using Pattern Oriented Design](#)

[#COBOLrocks TechCasts: New tricks for COBOL devs](#)














[Extensions for Visual Studio](#)

[Visual COBOL New Release: Small point. Big deal](#)

[Code InfoBox Visual Studio Extension \(VSX\) 2010](#)






[Getting to Know the Arduino 101 Platform](#)

## Comments and Discussions

Add a Comment or Question 			Search Comments <input type="text"/>	<input type="button" value="Go"/>
First Prev Next				
<b>Porting to Sharp Develop ?</b>   <b>new</b>				2
stixoffire 24-Sep-15 19:52				
<b>A nice tool</b>   <b>new</b>				2
bsa1958 29-May-15 19:03				
<b>Thank you, Vote 5</b>   <b>new</b>				1
jewel321 29-May-15 4:36				
<b>Add support for hi dpi displays</b>   <b>new</b>				3
mr_squall 12-May-15 3:17				
<b>VS 2015</b>   <b>new</b>				3
Christian Schiffer 9-May-15 19:01				
<b>Another brick</b>   <b>new</b>				2
George Shimanovich 7-May-15 18:26				

<b>My vote of 5</b>   new	1
Robert J. Good 7-May-15 16:58	
<b>Good article but</b>   new	6
Leigh Pointer 10-Mar-15 4:15	
<b>Thank you, very useful! Vote 5</b>   new	2
Dmitry Tsarevich 27-Feb-15 15:03	
<b>My vote of 5</b>   new	2
BillWoodruff 26-Feb-15 14:13	
<b>Is free or has a cost after 60 days?</b>   new	4
Oscar R. Onorato 26-Feb-15 11:09	
<b>My vote of 5</b>   new	1
_Vitor Garcia_ 26-Feb-15 9:27	
<b>Can you add it to NuGet</b>   new	2
AE~1 18-Feb-15 5:37	
<b>Misleading tags?</b>   new	3
Mike Nordell 2-Feb-15 13:19	
<b>My vote of 5</b>   new	1
popara 18-Nov-14 21:17	
<b>My vote of 5</b>   new	1
Mahsa Hassankashi 15-Nov-14 4:27	
<b>My vote of 3</b>   new	2
Forogar 14-Nov-14 15:10	
<b>My vote of 5</b>   new	1
Humayun Kabir Mamun 14-Nov-14 0:17	
<b>That is realy great.</b>   new	1
musketier 25-Aug-14 14:57	
<b>Error when installing</b>   new	2
Member 2227802 1-Jul-14 16:15	
<b>My vote of 5</b>   new	1
Ahmed Bensaid 19-Jun-14 8:44	
<b>My vote of 5</b>   new	1
Shemeer NS 20-May-14 10:48	
<b>Win 8 64 bit support ?</b>   new	3
delphoiq 20-May-14 2:59	
<b>My vote of 5</b>   new	1
Raul Iloc 16-May-14 7:16	
<b>My vote of 5</b>   new	1
Prasad Khandekar 13-May-14 6:38	
<b>Looks very useful</b>   new	1
Forogar 8-May-14 8:38	



<b>My vote of 5</b>  <span>new</span> Humayun Kabir Mamun 8-May-14 5:57	1
<b>Great Pluggin</b>  <span>new</span> nilesh07 8-May-14 4:11	1
<b>Great article about the integrated Visual Studio tool (Design Patterns add-in)</b>  <span>new</span> Volynsky Alex 8-May-14 2:50	1
<b>My vote of 5</b>  <span>new</span> Klaus Luedenscheidt 8-May-14 0:06	3
<b>My vote of 4</b>  <span>new</span> Rajesh Varma Buddaraju 7-May-14 23:55	1
Refresh	1

 General
 News
 Suggestion
 Question
 Bug
 Answer
 Joke
 Praise
 Rant
 Admin

[Permalink](#) | [Advertise](#) | [Privacy](#) | [Terms of Use](#) | [Mobile](#)  
Web02 | 2.8.160621.1 | Last Updated 23 Sep 2015

Layout: [fixed](#) | [fluid](#)

Article Copyright 2014 by dmihailescu  
Everything else Copyright © [CodeProject](#), 1999-2016