

Forum:[Hack Forums](#)

Thread: [\[TUTORIAL\] Make your application support Plugins \(FULL SOURCE\)](#)

Washi - February 1, 2012, 9:10 am

Plug-in System in VB.NET

Hello all,

Today i'm going to show a way to create a basic plugin system with interfaces. I hope i'm clear enough and don't go to fast. I've split up the tutorial into 6 steps. Designing the Main Application

The Connector Library

The Connector in the Main Application

Creating the Plug-in

Finishing the source

Test!

Step 1 - Designing the Main Application

To start, you have to create first a new project that will represent your main application.

Spoiler [\(Click to View\)](#)

[img]http://imageshack.us/a/img839/1107/pic1mr.png[/img]

Then design the form something like this:

[img]http://imageshack.us/a/img560/6015/pic2ez.png[/img]

The first button is to load the user created plugin file, and the second to run it. I've created a groupbox with two labels in it to show you what the output can be. In this example we will change those label's texts by using the plug-in assembly.

Step 2 - The Connector Library

After the designing of your main application, save it and open a new visual studio window and create another project. This will be a class library that will connect the plugin with the main application.

Spoiler ([Click to View](#))

[img]http://imageshack.us/a/img43/3347/pic3au.png[/img]

Delete the contents of the standard class file and create two interfaces.

[img]http://imageshack.us/a/img685/1561/pic4o.png[/img]

What is an interface?

(skip this if you know what it is) Spoiler ([Click to View](#))

An interface is a class with functions or subs that don't have any body. If you implement an interface to a class, the class must have those functions or subs in it. In fact you can say that an interface describes a class. You can compare it with a car and a truck, they are both vehicles. So the interface can be something like this:

```
Code Public Interface Vehicle
```

```
Sub Forwards()  
Sub Backwards()  
Sub TurnLeft()  
Sub TurnRight()  
End Interface
```

If you implement it into the class Car, then that class must have those four subs, forwards, backwards, turnleft and turnright. And for the Truck it's the same story.

For example:

```
Code Public Class Car
```

```
Implements Vehicle
```

```
Public Sub Backwards() Implements Vehicle.Backwards  
'Do stuff to move car backwards  
End Sub
```

```
Public Sub Forwards() Implements Vehicle.Forwards  
'Do stuff to move car Forwards  
End Sub
```

```
Public Sub TurnLeft() Implements Vehicle.TurnLeft  
'...  
End Sub
```

```
Public Sub TurnRight() Implements Vehicle.TurnRight  
'...  
End Sub
```

```
End Class
```

```
Public Class Truck
```

```
Implements Vehicle
```

```
Public Sub Backwards() Implements Vehicle.Backwards
```

```
'Do stuff to move truck backwards  
End Sub
```

```
(...)
```

```
End Class
```

But why is it useful?

Well now you can access both vehicles into one variable. For example, if the user has the choice to choose a vehicle, you can store the vehicle into a Vehicle variable and start one of the subs. If you don't use an interface, and you have a lot types of vehicles, you have to write a lot of code to process all the vehicles.

An example below to make it clear:

Code Module Example

```
Public Sub Main()  
  
    'ask first the user what kind of vehicle he wants  
    Console.WriteLine("Choose option 1 (car) or 2 (truck)")  
  
    'get the vehicle  
    Dim int As Integer = CInt(Console.ReadLine())  
  
    'declare a new vehicle  
    Dim vehicle As Vehicle  
  
    'check if input is 1 or 2.  
    If int = 1 Then  
        vehicle = New Car()  
    Else  
        vehicle = New Truck()  
    End If  
  
    'run some subs of the vehicle.  
    vehicle.Forwards()  
    vehicle.TurnLeft()  
  
End Sub  
  
End Module  
[/code]
```

The IPlugin and IConnector interfaces will be used in the main application and the plug-in assembly. The IPlugin will be like the Entrypoint (Start) of the class. That's why this interface contains the sub StartPlugIn.

The IConnector will be the connector that connects the Main App with the Plug-in. The first two functions will get the labels' texts, and the next two subs will set the labels' texts. The last sub will disconnect the plug-in from the main application.

Step 3 - The Connector in the Main Application

After creating the connector library, save and build it and go back to your main application. Add your Plugin connector library reference to the project and create a new class, called Connector. Then add the following code to implement the IConnector interface.

Code Imports PluginConnector

Public Class Connector
Implements IConnector

If you press enter, the functions/subs Disconnect, GetLabel1Text, GetLabel2Text, SetLabel1Text and SetLabel2Text will be automatically added to your class. Before we add the code of the bodies of the functions and subs, we need to make another sub called Public Sub New(frm as Form1) and another variable. This is to specify the Form you want to edit the labels' texts. You'll get something like this:

Code Dim frm As Form1

```
Public Sub New(ByVal frm As Form1)
Me.frm = frm
End Sub
```

Now we can write the code for the four label functions and subs. Just use Return frm.Label.Text for the get functions and frm.Label.Text = ... for the set subs. We'll write the disconnect sub later.

After this, your source will look something like this picture below.

[img]http://imageshack.us/a/img10/3696/pic8fm.png[/img]

Step 4 - Creating the Plug-In

Save the main application project and open another visual studio window, and create another class library. This will be the plug-in assembly itself.

Spoiler ([Click to View](#))

[img]http://imageshack.us/a/img853/6502/pic9ld.png[/img]

First of all, add the plugin connector library reference to your project . Then add a form and design it to something like this:

[img]http://imageshack.us/a/img837/4160/pic11x.png[/img]

The first button will run the Get functions to get the labels' texts and put them into the textboxes. The Set button will run the Set Functions to set the labels' texts to the textboxes texts.

Back to your first class. Rename it to StartOfPlugin and implement the IPlugin interface. The sub

StartPlugIn will be automatically added. In this sub we will create a new form1 and show it.

```
Code Dim frm As New Form1
    frm.Connector = connector
    frm.Show()
```

As you maybe see, the frm.Connector doesn't exist. That's why we have to add the following line at the top of the form's class.

```
Code Public Connector As IConnector
```

Now double click onto the buttons and fill the subs with code. One to get and one to set. Then add a formclosing event to disconnect from the main application when the form closes.

After this, you have something like this:

[img]<http://imageshack.us/a/img198/3456/pic12aj.png>[/img]

Save the application and go back to the main application.

Step 5 -Finishing the source

At the very top of the main application form's source add the following line.

```
Code Imports System.Reflection
```

Then add a new variable inside the class. This will represents the current running plug-in.

```
Code Public CurrentPlugIn As IPlugin
```

Now double click onto button1 and use the following code to let the user open a file and set the CurrentPlugIn with the selected file.

```
Code Dim ofd As New OpenFileDialog
```

```
    If ofd.ShowDialog = Windows.Forms.DialogResult.OK Then
```

```
        'load the assembly
```

```
        Dim pluginassembly As Assembly = Assembly.LoadFile(ofd.FileName)
```

```
        'get the StartOfPlugin class by using GetType("<typename>")
```

```
        Dim pluginclass As IPlugin =
```

```
CType(Activator.CreateInstance(pluginassembly.GetType(pluginassembly.GetName.Name &
        ".StartOfPlugin")), IPlugin)
```

```
        'sets the loaded plugin to the current plugin
```

```
        CurrentPlugIn = pluginclass
```

```
    End If
```

Then double click onto button2 and add the following line to start the plugin.

```
Code CurrentPlugIn.StartPlugIn(Connector)
```

Your form code will now look something like this:

[img]<http://imageshack.us/a/img713/3707/pic13c.png>[/img]

Go back to the Connector class and write the following line into the Disconnect sub to clean the CurrentPlugIn Variable.

Code frm.CurrentPlugIn = Nothing
Now you're done with the source.

Step 6 - Test!

If you did everything right, you'll get something like this gif animation:

I hope you enjoyed this tutorial. Feel free to give comments. Please if you use this tutorial, say post a thanks ?

If you want to download the project, click [here](#)

For the ones who don't trust the file. I don't really understand why there are 2 false positives since it's a rar archive with sources.

[VirusTotal scan](#)

That's all ?

Ethernal Five - February 1, 2012, 9:18 am

Looks like a very good tutorial, very detailed and explained a lot. Thanks ;)

MoBaT - February 1, 2012, 10:34 am

This is such a well built tutorial. I will be sure to use this.

ibloodqman - February 1, 2012, 10:36 am

Greatest tutorial i ever seen (honestly). Big Vouch, i'd rep you but i can't.. keep it up

Androcir - February 1, 2012, 10:39 am

Very nice tutorial mate, you made it very comprehensive (if that's how it is spelled... I don't know..)

Anyways.. ;)

Washi - February 1, 2012, 1:26 pm

I'm glad you all liked it ?

Thanks for the commentary.

nuk - February 1, 2012, 1:31 pm

Very nice indeed. Great share man! Def. a keeper on the tutorials :)

M0D3M - February 1, 2012, 3:46 pm

Excellent tutorial. This is the first tutorial I have seen on this subject. However I have not been on in awhile. One green apple for you.

nrd - February 1, 2012, 3:56 pm

this was a great tutorial, can come in handy for some programs in the future

The MammoTh - February 1, 2012, 4:00 pm

This one is better then I coded, do you mind if I use it?

ParkII - February 1, 2012, 4:04 pm

Yeah...I needed this a long time ago ...

Sixem - February 1, 2012, 7:57 pm

That's pretty damn cool, detailed and easy to understand, great job.

Washi - February 1, 2012, 11:05 pm

(Feb 1, 2012 05:00 PM)The MammoTh Wrote: [\[CITE\]](#) This one is better then I coded, do you mind if I use it?

Of course you 're allowed, this is a tutorial, and you may use the source. ?

(Feb 1, 2012 05:04 PM)ParkII Wrote: [\[CITE\]](#) Yeah...I needed this a long time ago ...

(Feb 1, 2012 08:57 PM)Shaqways Wrote: [\[CITE\]](#) That's pretty damn cool, detailed and easy to understand, great job.

I'm glad you two liked it ?

log dot zip - November 12, 2012, 2:14 am

Yip yippity kiyay! (Didn't get the reference? Oh well) Anyways: AWESOME JOB.

Kotei - November 12, 2012, 2:18 am

Needs more publicity, truly a grand guide.

gungunfebrianza - November 12, 2012, 3:11 am

Great Tutorial sir Unknownprogrammer ?

now i am trying to implementing your tutorial for learning purpose . ?

thanks for sharing :)

Washi - November 12, 2012, 1:09 pm

(Nov 12, 2012 04:11 AM)gungunfebrianza Wrote: [\[CITE\]](#) Great Tutorial sir Unknownprogrammer ?

now i am trying to implementing your tutorial for learning purpose . ?

thanks for sharing :)

I'm glad to see that somebody is actually using it :D

SIM-62706 - November 12, 2012, 1:22 pm

can you please check the file url because it doesn't work?

thanks! :)

N4xus - November 12, 2012, 1:54 pm

fix the last pic

thx for ur info<

bro

nuk - November 12, 2012, 1:56 pm

I know I already posted on here about this, but I really gotta say that this is prob. one of the best tutorials on this I've seen yet. Easy to follow and a sweet share if I do say so. Kick ass man!

Washi - November 12, 2012, 10:23 pm

Fixed the last picture and download link.

Drug ballad - November 12, 2012, 10:36 pm

very nice tutorial :D
very explained 2

log dot zip - November 13, 2012, 2:04 am

Also; the 2 false/positives are default ones that comes with VB.NET
If you have a setup.exe or anything else in there.

setup.exe: WS.Reputation.1
I don't know the other one, but it seems like a F/P :-)

OneTXxL - November 13, 2012, 4:11 am

Hi this tutorial is perfect, thank you very much for sharing a TheUnknownProgrammer quality good way to proceed!

decoder787 - November 13, 2012, 8:32 am

Thanks For The Tutorial..>>.. Nice.

ProfitCoder - January 19, 2013, 2:23 pm

Great tutorial ;)
I'm trying to learn programing from internet and this is very nice how you explain,
Thanks

Washi - January 19, 2013, 2:27 pm

(Jan 19, 2013 02:50 PM)battlehaxplugins Wrote: [\[CITE\]](#) terrible way of making a plugin system, op cant code for shit.

Nice feedback ?, I respect the freedom of opinion, but at least tell me what I can improve instead of just saying it's bad. Btw, I never said this was the best way of doing it. This tutorial is only for learning purposes and it introduces people to work with reflection and how Plugin systems basically work.

(Jan 19, 2013 03:23 PM)profit22 Wrote: [\[CITE\]](#) Great tutorial ;)

I'm trying to learn programing from internet and this is very nice how you explain,
Thanks
Thanks for the nice reply :D

HostBoot - January 19, 2013, 2:29 pm

Really needed this :o such a coincidence i found it :)
thanks alot!!

HnDc - July 25, 2013, 8:31 am

Hi TUP, link dead, please reupload it!
Thanks.

Washi - July 25, 2013, 8:39 am

Updated the download url.

Spoiler [\(Click to View\)](#)

I must say looking back to this source code, this could be improved quite a lot lol

Felix.Frogg - July 25, 2013, 9:56 am

@TheUnknownProgrammer

Thank you, very nice tuto, can you advice me some details?

I'm try to use PluginConnector lib directly from project as new class but alway get stuck at executing instance

Quote:Unable to cast object type "Plugin.StartOfPlugin" to type "Plugin_System.IPlugin"

here:

Quote:Dim pluginclass As IPlugin =

CType(Activator.CreateInstance(pluginassembly.GetType(pluginassembly.GetName.Name & ".StartOfPlugin")), IPlugin)

?

?ICE? - July 25, 2013, 11:24 am

Nice tut that plugin looks nice as well.

SilverNET - July 25, 2013, 11:32 am

Great tutorial, really helpful. Thanks a lot :)

Washi - July 25, 2013, 12:36 pm

(Jul 25, 2013 09:56 AM)Felix.Frogg Wrote: [\[CITE\]](#) @TheUnknownProgrammer

Thank you, very nice tuto, can you advice me some details?

I'm try to use PluginConnector lib directly from project as new class but alway get stuck at executing instance

here:

?

Did you implement the interface in your class correctly? Can you show me the exact source of your plugin?

Leaving forever - July 25, 2013, 12:44 pm

Wow pretty good someone was grave-digging.

I was talking to my partner about that last night, I really need this!

Thanks :3

CyberTechnologyInc - July 25, 2013, 12:45 pm

Sometimes, grave-digging is useful! I will bookmark this tutorial for later ;D

SIM-1456245 - July 25, 2013, 4:58 pm

Nice, I was looking for this. Thanks.

BTW, NVM if it is grave-digged.

OneTXxL - July 25, 2013, 5:53 pm

Thank you TUp is very good code example plugin nice share

Felix.Frogg - July 25, 2013, 9:04 pm

(Jul 25, 2013 12:36 PM)TheUnknownProgrammer Wrote: [\[CITE\]](#) Did you implement the interface in your class correctly? Can you show me the exact source of your plugin?

PMed you my source. Thanks in advance

Best Regards, UL

?

stefsot - July 26, 2013, 4:29 am

When using interface you cannot add or remove anything in your class.

It could be used better like this

```
Code Public Class IPlugin
    Public Overridable Sub OnLoad()
    End Sub
    Public Overridable Sub Refresh()
    End Sub
Enc Class
Code Public Class Plugin_test
    Inherits IPlugin
    Overrides Sub OnLoad()
    Console.WriteLine("hi!!")
    End Sub
Enc Class
```

Washi - July 26, 2013, 4:56 am

(Jul 26, 2013 04:29 AM)stefsot Wrote: [\[CITE\]](#) When using interface you cannot add or remove anything in your class.

It could be used better like this

```
Code Public Class IPlugin
    Public Overridable Sub OnLoad()
    End Sub
    Public Overridable Sub Refresh()
    End Sub
Enc Class
Code Public Class Plugin_test
    Inherits IPlugin
    Overrides Sub OnLoad()
    Console.WriteLine("hi!!")
    End Sub
Enc Class
```

True, but this tutorial was more about using interfaces. Interfaces should contain only just a few declarations specific for the interface.

What I have used before is an interface called IPlugin, that has a definition like this:

```
PHP Code
<?php
Public Interface IPlugin
    Sub Execute(...)
End Interface
```

Using this in a class would require me to implement the Execute sub.

```
PHP Code
<?php
Public Class MyPlugin
    Implements IPlugin

    Sub Execute(...)
```

```
End Sub
End Class
```

If I wanted to extend the plugin with something like file handling operations, I need more methods indeed. I created another interface, called IFileHandler to solve this:

```
PHP Code
<?php
Public Interface IFileHandler
    Sub OpenFile(...)
    Sub SaveFile(...)
End Interface
```

Implementing these methods in the MyPlugin class will give me this:

```
PHP Code
<?php
Public Class MyPlugin
    Implements IPlugin, IFileHandler

    Sub Execute(...)
    End Sub

    Sub OpenFile(...)
    End Sub

    Sub SaveFile(...)
    End Sub
End Class
```

Now my plugin supports more methods to be called from the main application. In your case you would add an interface called IRefreshable or something among those lines.

The point of this is, that interfaces provides us a very good base for extending classes' functionality. I think interfaces are very well suited for these kind operations, as they give you the opportunity to "personalize" a type. This can be achieved partly with MustInherit classes, and I do believe your example is good and I sometimes use abstraction too instead of interfaces, but you should feel the power of interfaces because of their flexibility.