

Forum:[Hack Forums](#)

Thread: [Creating a builder for .NET \[3 WAYS\]](#)

LordCoder - June 20, 2014, 3:11 pm

Hello,

As I see some people are confused creating a builder for stub files I decided to create a complete tutorial.

If you don't know what does builder or stub mean let me show you an example.

You have an application that shows a message. That message can be anything. The user who creates the application needs to set a custom message as he's trying to do, for example, an application saying "Hello".

There are 3 ways (I think there aren't any other). The first one is using Mono.Cecil, an open source library to edit already compiled assemblies for .NET. It will edit us whatever we want inside the application using MSIL. The problem is to know how to use it (so this method could be more advanced if you are not into reversing).

The second method is using a part from the compiled .EXE and then get it. The options should be inside the application and you can't change the code. That's useful if you're going to make similar stubs and not very customized.

The last method is using a stub code and edit it as we want. Then compile it using some classes from .NET. This might be a good method if you're planning to do an easy application without many classes or references (as you will have it on a .txt file basically).

1st way: Mono.Cecil

Download [Mono.Cecil.dll](#) (already compiled for you).

I'm not going to show the basic of Mono.Cecil as there are a lot of tutorials about Mono.Cecil (or remember Google is your friend always).

Creating a basic "MsgBox Creator" application

-Create 2 projects: Builder and Stub.

-Add Mono.Cecil to your project.

On the Stub you should use, for example, a ConsoleApplication. On MainModule1 put:

Code Function GetMessage() as String

Return "Not Set - Error" ' Builder will do the job

End Function

Sub Main()

Dim msg as String = GetMessage

If msg = "Not Set - Error" Then ' Checks if Builder did it

MsgBox("No message was set. Closing now.")

Else

MsgBox(msg)

End If

End Sub

Now save it to "YourBuilderProject\bin\debug\stub_compiled.exe" and add this to the Builder:

Code Sub Compile(ByVal message As String)

Dim m As AssemblyDefinition = AssemblyDefinition.ReadAssembly(Application.StartupPath &

```

"\stub_compiled.exe") ' Opens our stub
    For Each i In m.MainModule.Types
        If i.Name = "MainModule1" Then ' Finds the MainModule of the ConsoleApplication
            For Each Me2 In i.Methods
                If Me2.Name = "GetMessage" Then ' Finds the GetMessage function
                    Me2.Body.Instructions(1).Operand = message ' I put 1 instead of 0 because compilers should
put always a 'Nop' instruction before the code
                End If
            Next
        End If
    Next
    m.Write(Application.StartupPath & "\stub_edited.exe") ' Saves the assembly
End Sub

```

Finally run "Builder" and call "Compile" sub with any message. It will save "stub_edited.exe" with your new message.

2nd way: Editing options from an EXE

This method doesn't require any extra library. I know my code is very old and I can use IO functions but I like it. I'm not going to make this thread more long to explain how does it work, it just works.

Creating a basic "MsgBox Creator" application

As we did with the Mono.Cecil example, create both projects "Stub" and "Builder". The stub can be a ConsoleApplication like before with this method:

```

Code Sub Main()
    Dim options as String()
    Dim text1 as String
    FileSystem.FileOpen(1, Application.ExecutablePath, OpenMode.Binary, OpenAccess.Read,
OpenShare.Shared, -1) ' Opens the file and saves to a local variable 1.
    Me.text1 = Strings.Space(CInt(FileSystem.LOF(1))) ' Gets the variable
    FileSystem.FileGet(1, Me.text1, -1, False)
    FileSystem.FileClose(1)
    Try
        Me.options = Strings.Split(Me.text1, "WHATEVERHEREWITHOUTSPACES", -1,
CompareMethod.Binary)
        Catch ex as Exception
            MsgBox("Error. No message was set")
        End
    End Try
    MsgBox(options(1))
End Sub

```

Compile it and do the same (copy the file to bin\debug as "stub_compiled.exe"). Then add this code:

```

Code Sub Compile(message as String)
    FileSystem.FileOpen(1, "stub_compiled.exe", OpenMode.Binary, OpenAccess.Read,
OpenShare.Default, -1)
    Dim str As String = Strings.Space(CInt(FileSystem.LOF(1)))
    FileSystem.FileGet(1, str, -1, False)
    FileSystem.FileClose(New Integer() { 1 })
    FileSystem.FileOpen(1, Me.SaveFileDialog1.FileName, OpenMode.Binary,
OpenAccess.ReadWrite, OpenShare.Default, -1)

```

```
FileSystem.FilePut(1, (str & "WHATEVERHEREWITHOUTSPACES" &
"YOURMESSAGEHEREIFUWANT"), -1, False)
FileSystem.FileClose(New Integer() { 1 })
End Sub
```

Remember that "WHATEVERHEREWITHOUTSPACES" should be the same for builder and stub and without spaces.

3rd way: Compiling a new assembly

You'll need System.Reflection namespaces so add them with Imports.

Creating a basic "MsgBox Creator" application

The trick to do this is editing the stub but as a text file. So let's say we have this:

```
Code Module MainModule1
```

```
Sub Main()
```

```
MsgBox("WHATTOSAY")
```

```
End Sub
```

```
End Module
```

We'll have to edit the stub to:

```
Code mystubsaved.Replace("WHATTOSAY", "Hey this is the new message")
```

And then compile it:

```
Code Public Sub CompileCode(ByVal is_vb As Boolean, sourceFile As String, _
exeFile As String)
```

```
Dim provider
```

```
If Not is_vb Then
```

```
provider = New CSharpCodeProvider()
```

```
Else
```

```
provider = New VBCodeProvider()
```

```
End If
```

```
Dim cp As New CompilerParameters()
```

```
cp.ReferencedAssemblies.Add("System.dll")
```

```
' NOTE: If your stub has more references you need to add them
```

```
cp.GenerateExecutable = True
```

```
cp.OutputAssembly = exeFile
```

```
cp.GenerateInMemory = False
```

```
Dim cr As CompilerResults = provider.CompileAssemblyFromFile(cp, sourceFile)
```

```
End Sub
```

That's all! I didn't explain everything step by step as I suppose you know about VB.NET.

Any comments are welcomed (I expect the bad ones too, please be bad with me because I can learn from those comments :D).

Nutrition - June 20, 2014, 3:11 pm

This is actually really helpful and informational!

bfizzy - June 20, 2014, 5:30 pm

I am getting this error with the first method.

anything wrong from my side?

My Code:

Error:

SIM-1456245 - June 20, 2014, 7:29 pm

This is one heck of an impressive tutorial. Well done.

LordCoder - June 20, 2014, 11:43 pm

(Jun 20, 2014 05:30 PM)bfizzy Wrote: [\[CITE\]](#) I am getting this error with the first method. anything wrong from my side?

My Code:

Error:

The function that returns the message should return a string, not an integer.

bfizzy - June 21, 2014, 4:51 am

(Jun 20, 2014 11:43 PM)? I???????? ? Wrote: [\[CITE\]](#) The function that returns the message should return a string, not an integer.
ohh, what a mistake
I will try it out as I get home.

TheBarret - June 21, 2014, 6:10 am

(Jun 21, 2014 04:51 AM)bfizzy Wrote: [\[CITE\]](#) ohh, what a mistake
I will try it out as I get home.
If you don't know that, why the heck are even trying to make this.

So much fail.

Proton! - September 21, 2014, 4:48 am

Good tutorial, is the second one compatible with crypters?

LordCoder - September 21, 2014, 4:57 am

(Sep 21, 2014 04:48 AM)Professor Proton Wrote: [\[CITE\]](#) Good tutorial, is the second one compatible with crypters?
Yes, but only if you use it properly.

BahNahNah - September 8, 2015, 4:49 am

There is also EOF (not recommended), and updating a native resource containing the settings on a compiled assembly.

LordCoder - September 8, 2015, 6:55 am

(Sep 8, 2015 04:49 AM)BahNahNah Wrote: [\[CITE\]](#) There is also EOF (not recommended), and updating a native resource containing the settings on a compiled assembly.
I wrote this a year ago ?

SIM-1676753 - September 8, 2015, 7:02 am

I use native methods such as 'LockResource' and 'LoadResource'. Simple and easy.

ShittyAdvice - September 9, 2015, 9:24 am

(Sep 8, 2015 04:49 AM)BahNahNah Wrote: [\[CITE\]](#) There is also EOF (not recommended), and updating a native resource containing the settings on a compiled assembly.

Any specific reasons why it's not recommended?

Also what would be the "best" way be and why?

Reverse_Engineer - September 10, 2015, 4:19 am

Good tutorial , Thanks for sharing this

Cecilia - September 10, 2015, 7:40 am

Your first method isn't too great, it can be improved.

GetTypes() gets all types in the module, including nested.

LordCoder - September 10, 2015, 7:49 am

(Sep 10, 2015 07:40 AM)LaPanthere Wrote: [\[CITE\]](#) Your first method isn't too great, it can be improved.

GetTypes() gets all types in the module, including nested.

Indeed, I made this thread a year ago as said and I was starting with this method. Thanks for feedback though.