



Languages » MSIL » General **Advanced**
(GPL)

License: [The GNU General Public License](#)

C#, MSIL, VB 7.x, VB 8.0, VB 9.0,
Windows, .NET 1.0, .NET 1.1, .NET
2.0, .NET 3.0VS.NET2003, VS2005,
Architect, Dev

Assembly Manipulation and C# / VB.NET Code Injection

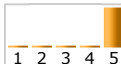
By [Sebastien LEBRETON](#)

Reflexil is an assembly editor and runs as a plug-in for Lutz Roeder's Reflector. Reflexil is able to manipulate IL code and save the modified assemblies to disk. Reflexil also supports "on-the-fly" C#/VB.NET code injection.

Version: **5 (See All)**
Posted: **19 Sep 2007**
Updated: **25 Oct 2009**
Views: **93,200**
Bookmarked: **228 times**

99 votes for this article.

Popularity: 9.77 Rating: **4.89** out of 5



[Download binaries - 713.48 KB](#)

[Download source - 1.49 MB](#)

Latest Releases

You can always get the latest [Reflexil](#) releases from [Sourceforge](#).

Introduction

[Reflector](#) is a great tool for doing an in-depth examination of various types of assemblies and also for disassembling IL code towards a supported .NET language. However, [Reflector](#) is unable to modify either the structure or the IL code of assemblies. [Reflexil](#) allows such modifications by using the powerful [Mono.Cecil](#) library written by [Jb EVAIN](#). [Reflexil](#) runs as a plug-in and is directed especially towards IL code handling. It accomplishes this by proposing a complete instruction editor and by allowing C#/VB.NET code injection, as we will see in the following two examples.

Demo Application

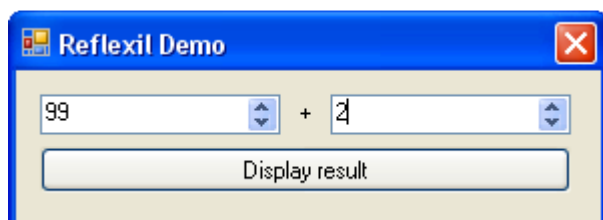
Let's use a very simple application that is able to add two numbers:

```
using System;
using System.Windows.Forms;

namespace ReflexilDemo
{
    public partial class DemoForm : Form
    {
        public DemoForm()
        {
            InitializeComponent();
        }

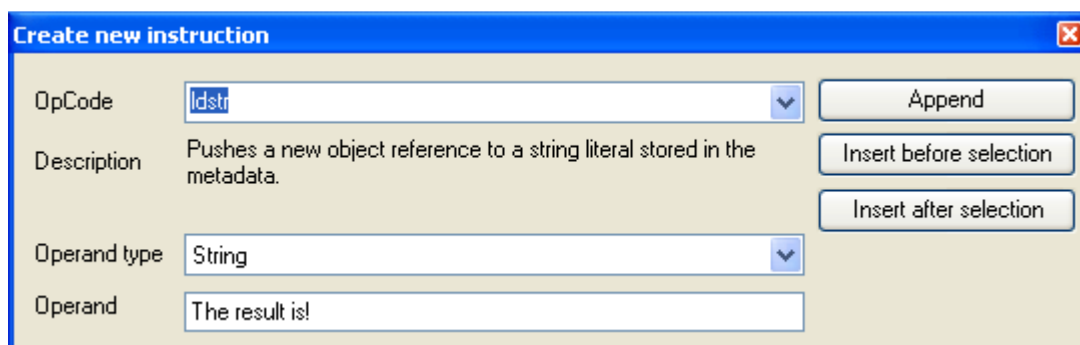
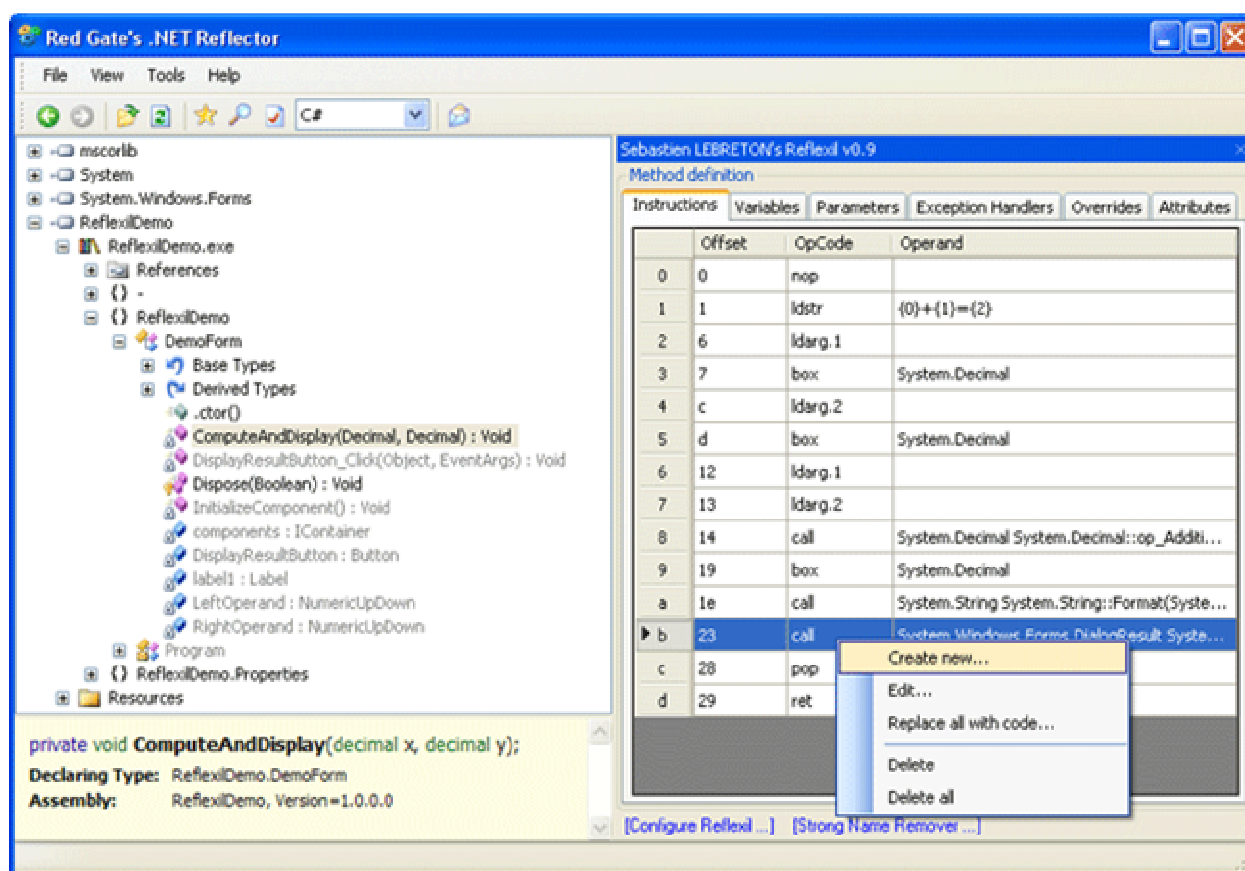
        private void ComputeAndDisplay(decimal x, decimal y)
        {
            MessageBox.Show(String.Format("{0}+{1}={2}", x, y, x + y));
        }

        private void DisplayResultButton_Click(object sender, EventArgs e)
        {
            ComputeAndDisplay(LeftOperand.Value, RightOperand.Value);
        }
    }
}
```

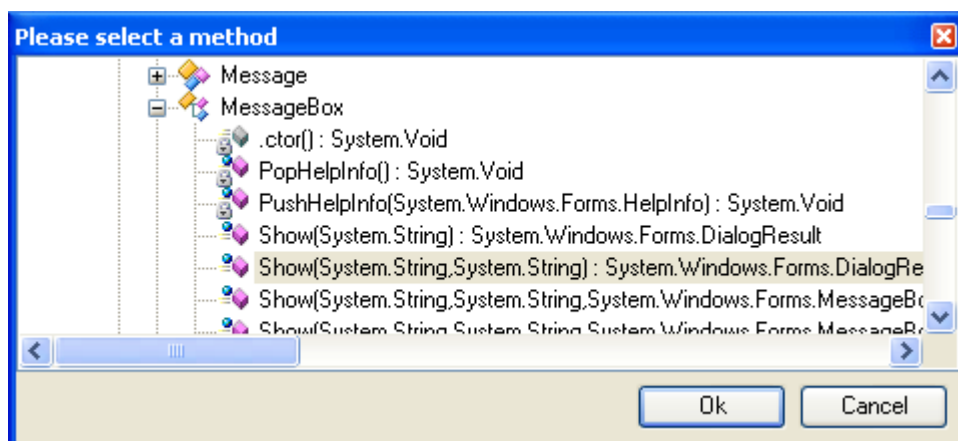
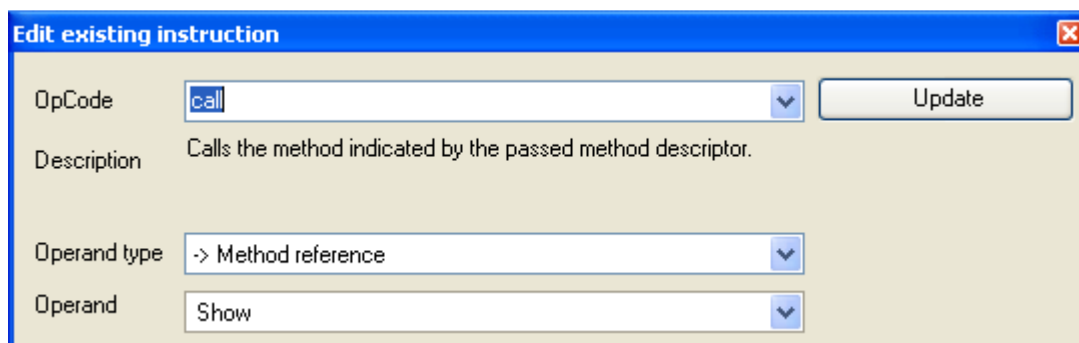
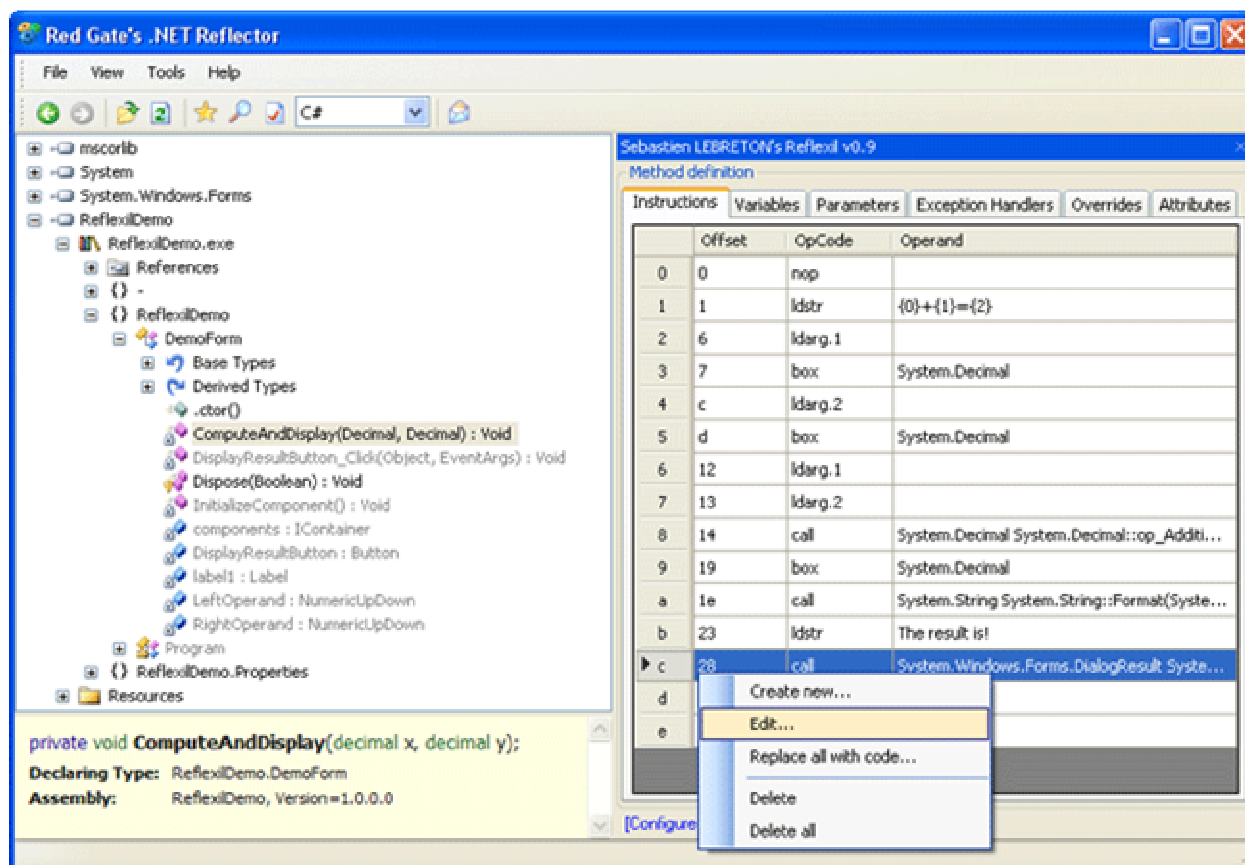


Using Instruction Editor

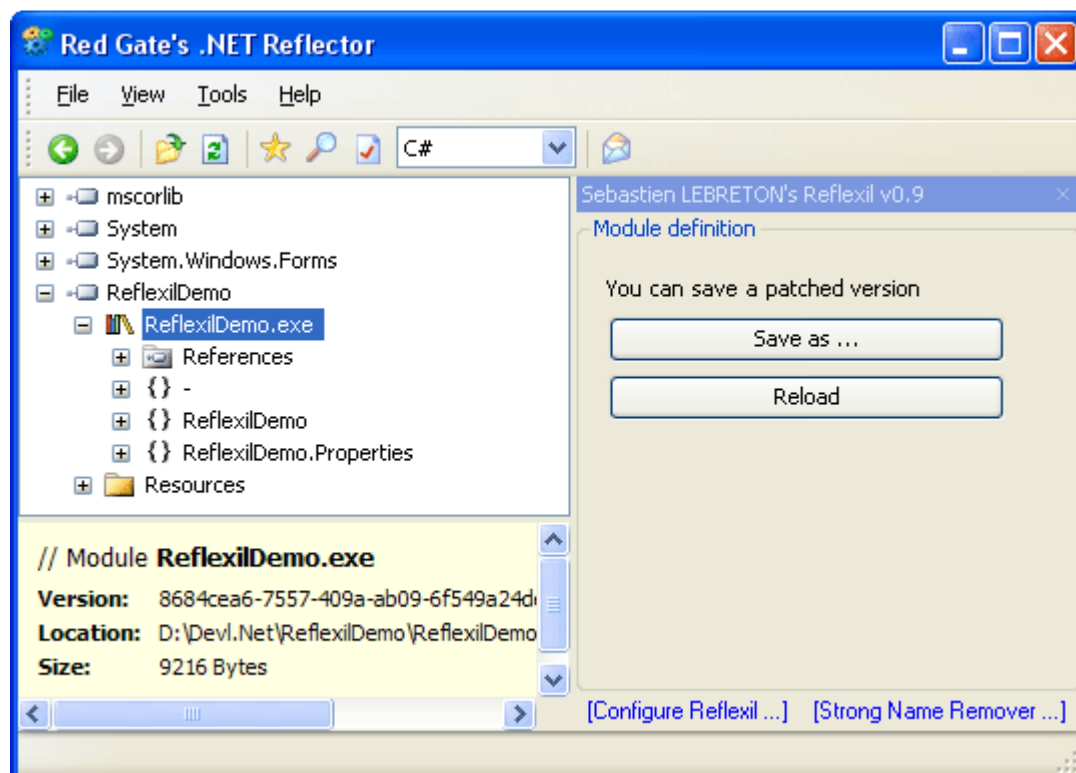
Using the instruction editor, let's update the `ComputeAndDisplay` method body by calling the overloaded method `MessageBox.Show`, which takes a title as a second parameter for the modal window displaying the result. In order to do so, we have to initially put a `string` parameter on the stack with `ldstr` opcode:



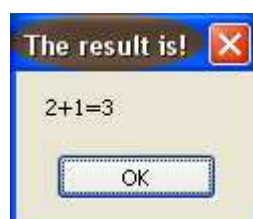
Then we have to update the "call" instruction to use the overloaded method `MessageBox.Show`, which uses the previously created parameter:



It is time to save our work and to test our patched assembly:



Our assembly is now using the overloaded method and the suitable parameter:



Instruction Editor Features

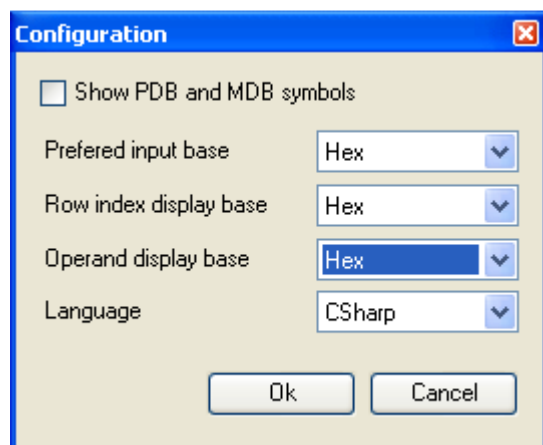
The instruction editor supports all opcodes defined in `Mono.Cecil`. The following operands are supported:

- Primitive types: byte, sbyte, int32, int64, single, double
- String
- Instruction reference
- Multiple instructions references (switch)
- Parameter or variable reference
- Internal generic type reference
- Type, field or method reference using a browser for selecting the suitable element. This browser is like the reflector's one (lazy loading, icons, etc.)

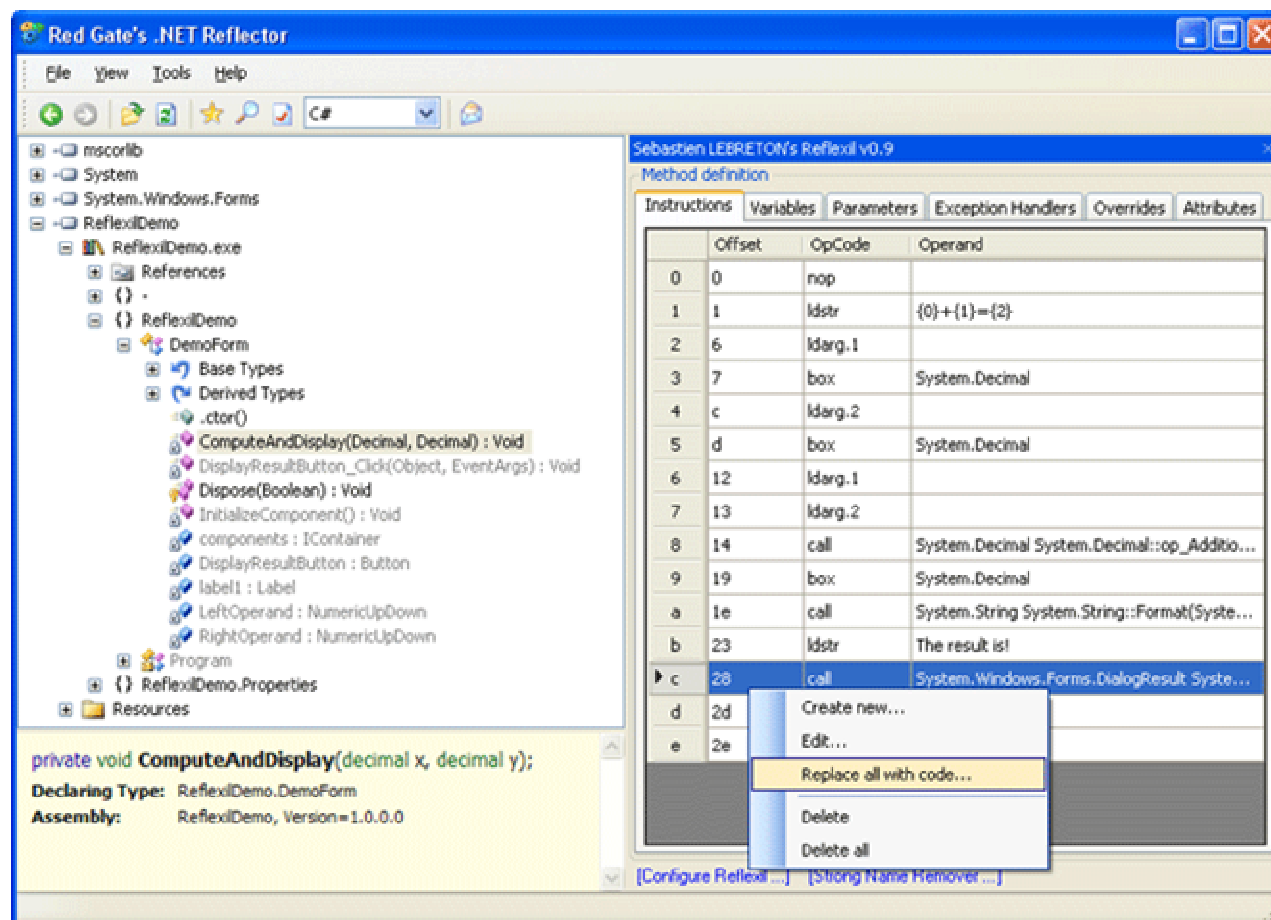
In a quite comprehensible way, the `Reflector` and `Reflexil` object models are not synchronous: updates made on the IL code will not impact the disassemble window of `Reflector`. `Reflexil` and `Mono.Cecil` do not perform any checks of the emitted code. The only constraint is about coherence between the operand type used for a given opcode. For those who find IL manipulation difficult, the following example shows how to update a method body with C# or VB.NET.

Using C# / VB.NET Code Injection

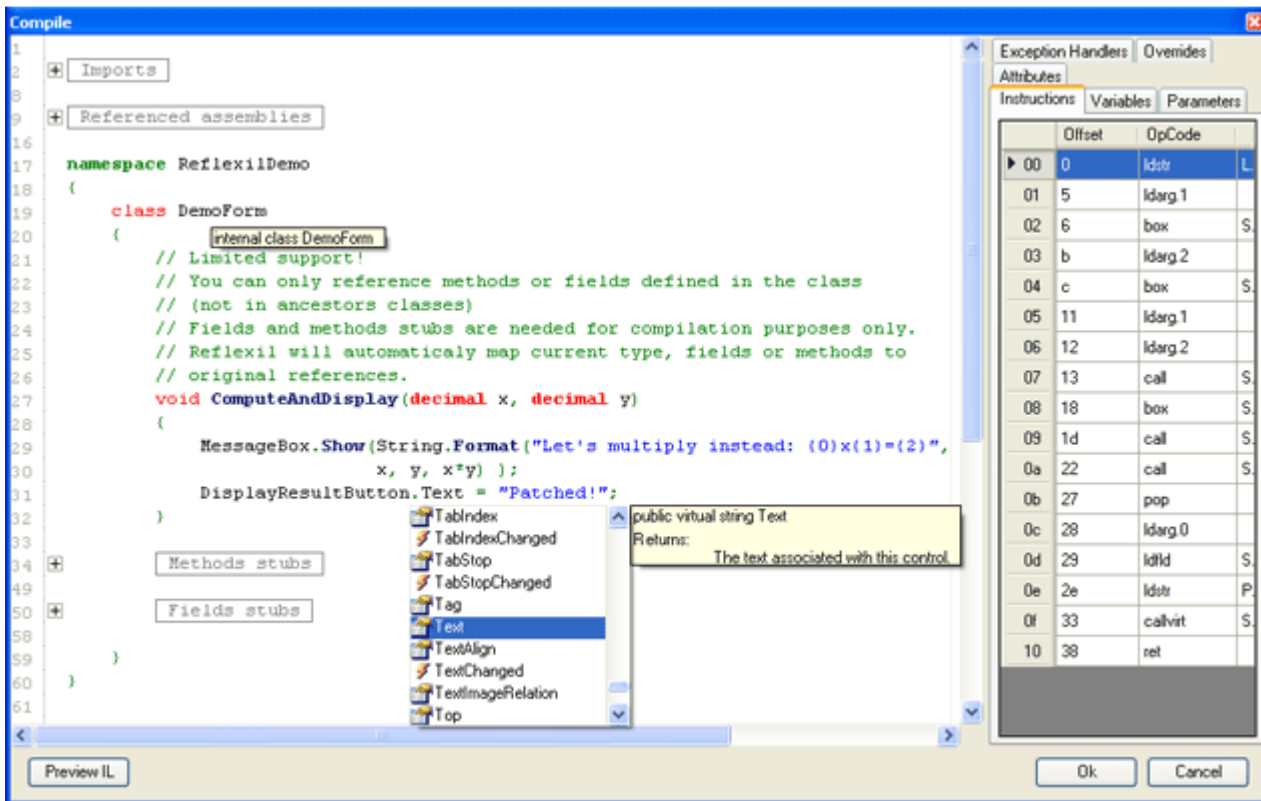
You can choose your preferred injection language and input/display bases (binary, octal, decimal, and hexadecimal) with the configuration form:



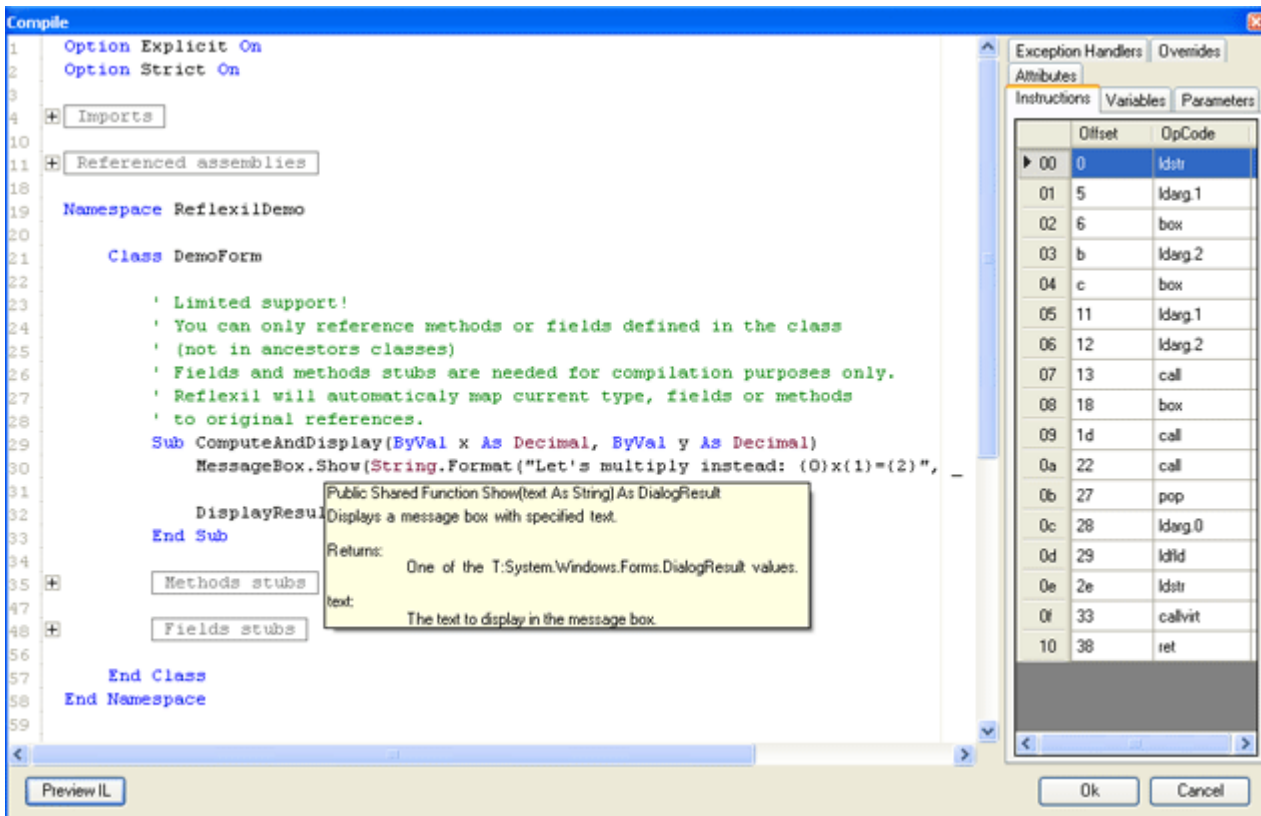
Let's use the "Replace all with code" feature on the body of the `ComputeAndDisplay` method:



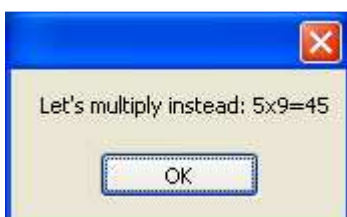
A compile window enables us to view the resulting IL code. Since v0.9 a basic support for intellisense/insight is provided:



We can do it again with VB.NET language. Note that in this simple case, we obtain an identical IL code (it is not always the case):



Let's save and test our patched assembly:

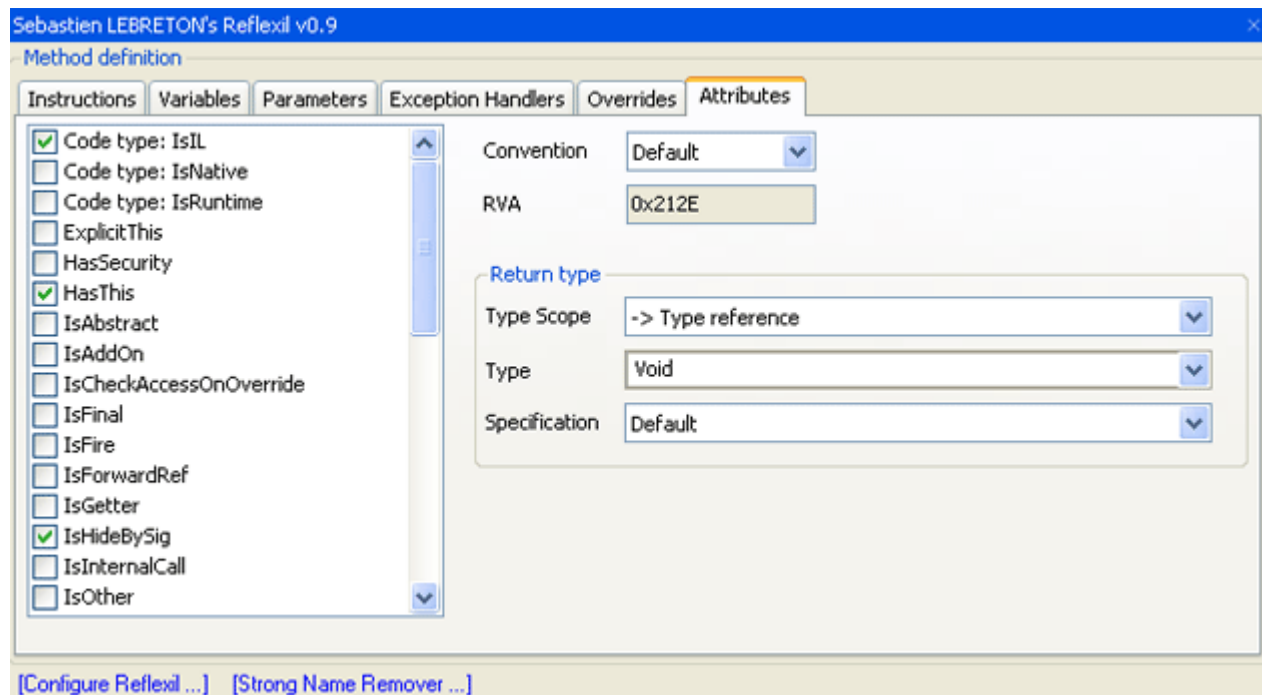


C# / VB.NET Code Injection Features

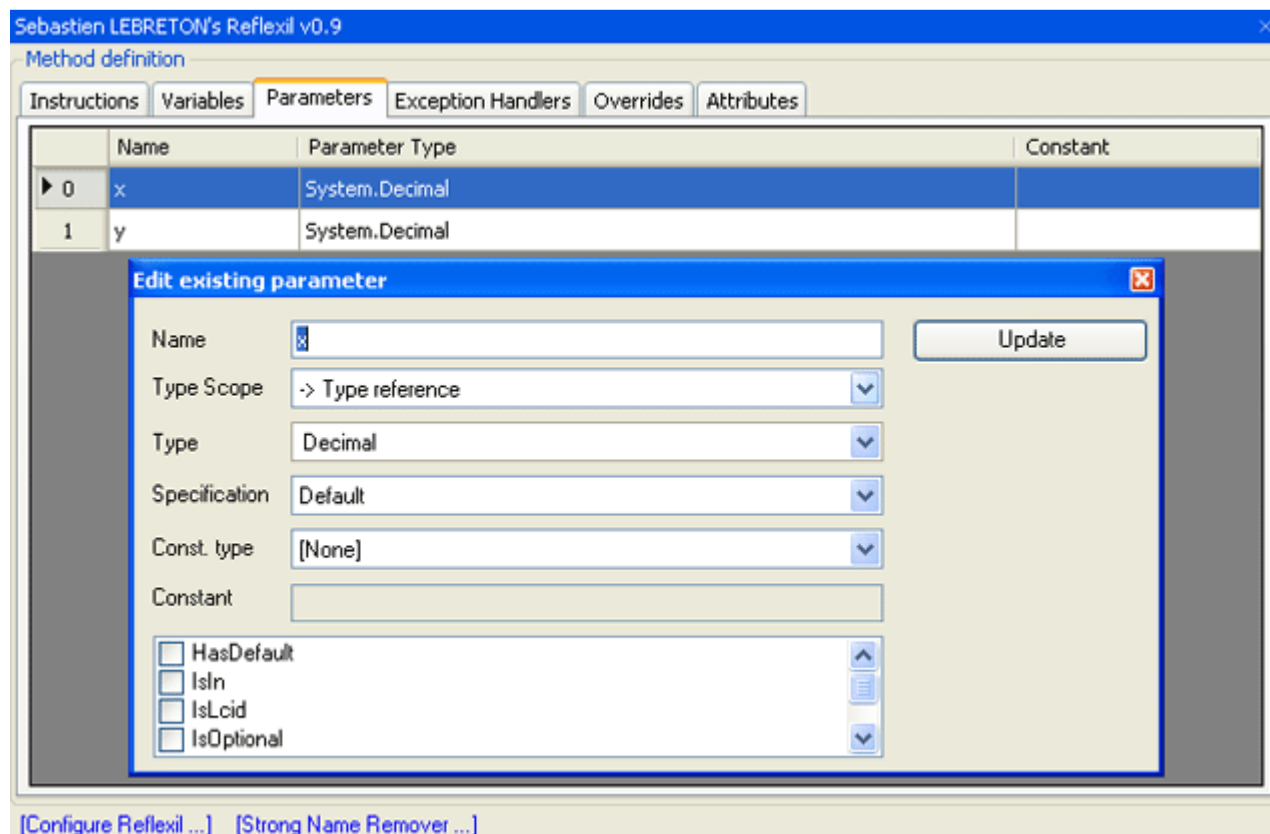
The code is compiled using `System.CodeDom` in a separate `AppDomain` for a correct resource release. Once compiled, instructions are extracted and then reinserted in the original method body. Parameters, variables, methods, fields and types references are adjusted to match the original assembly. Code injection is limited: it is not possible to refer either to the fields or to the methods defined in ancestral types of the owner type of the original method. Since v1.0, the compilation process is able to properly inject code with .NET 3.5 compiled assemblies.

Method Attributes Editor

You can easily update a method signature or change his visibility scope. You are also able to change the return type:

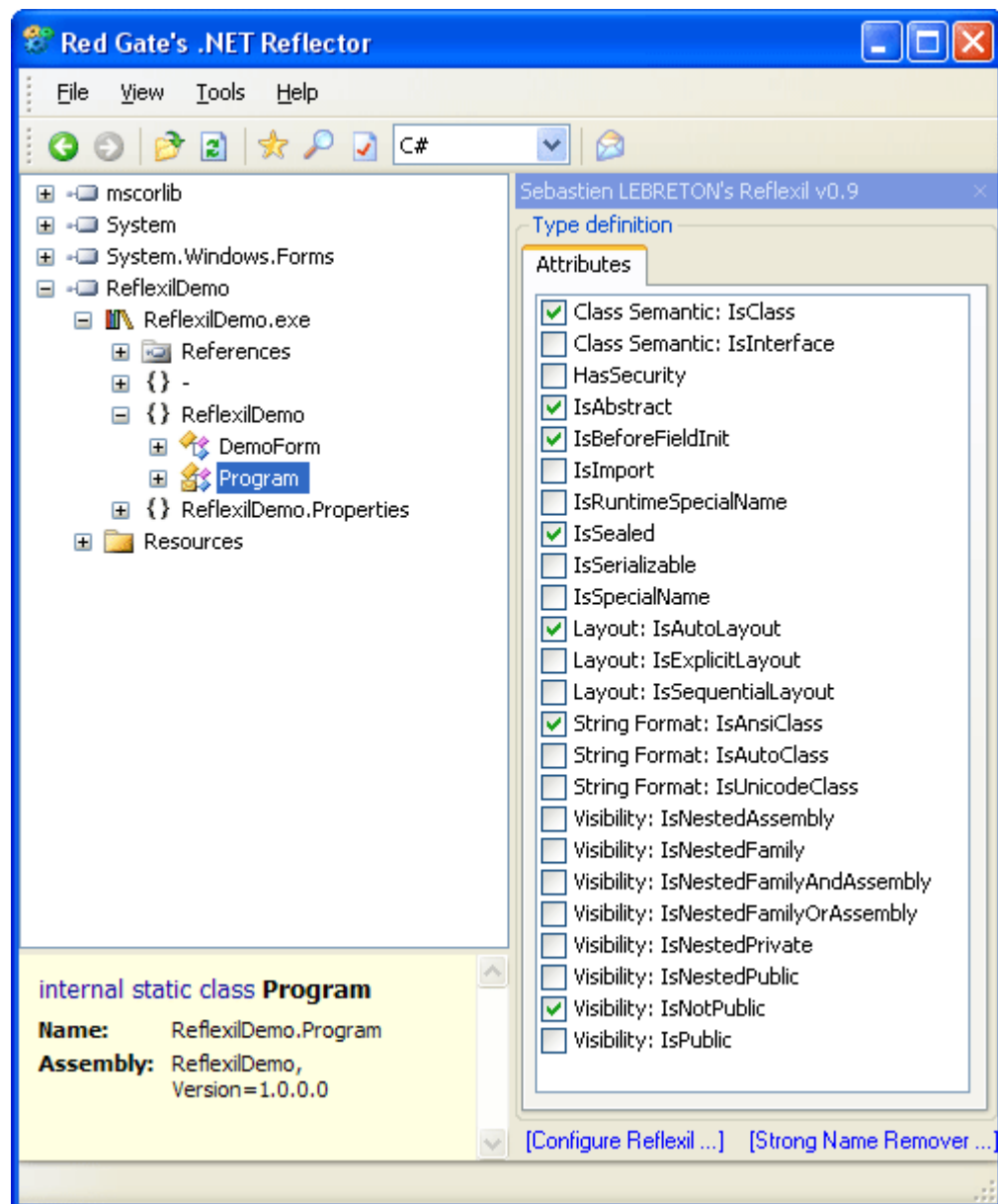


Method parameters (and variables) are also editable. Reflexil can load symbols (MDB and PDB files are supported) to display original variable names:



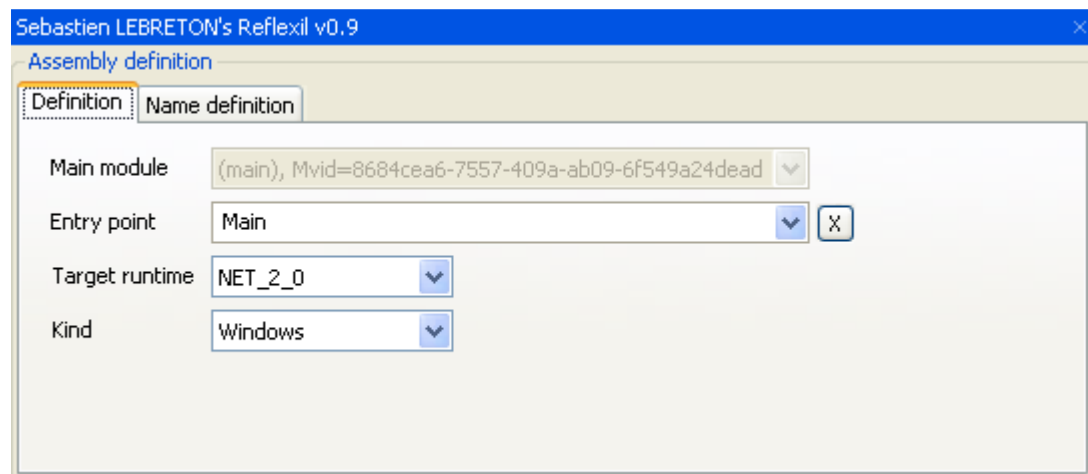
Type Attributes Editor

Like methods, you can change any type visibility scope. So you can expose a previously `private` type to the world:

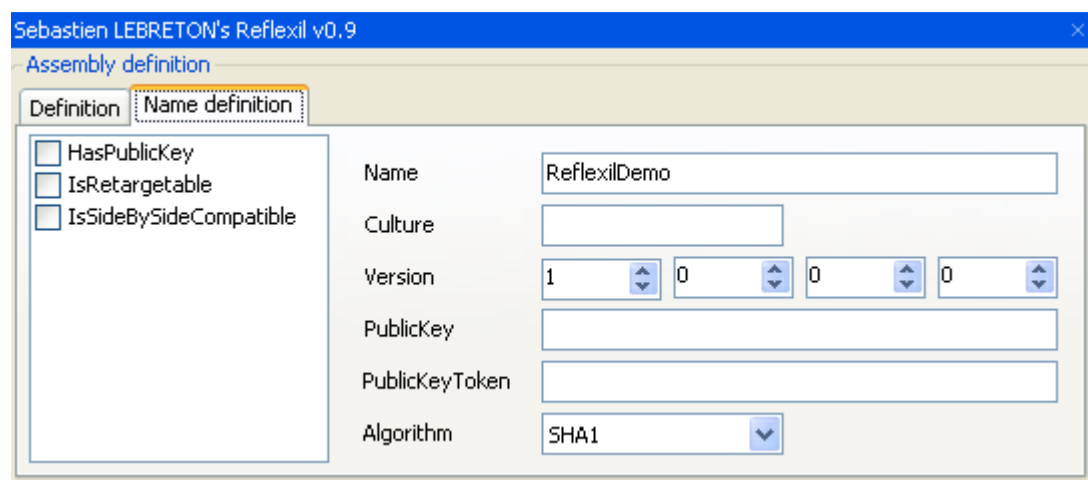


Assembly and Assembly Reference Editor

With the assembly editor, you can use a different entry point or simply change the kind of your application (transform an executable assembly to a DLL library for instance):



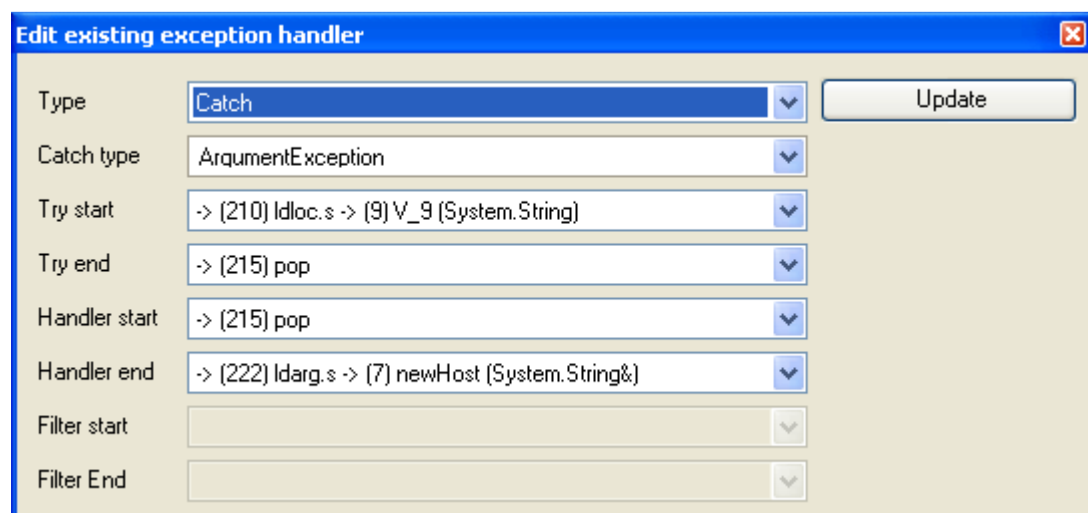
You can also update all information regarding identification: version, public key, name and culture. Note that you can also alter any referenced assembly so you can use a different version:



Exception Handlers

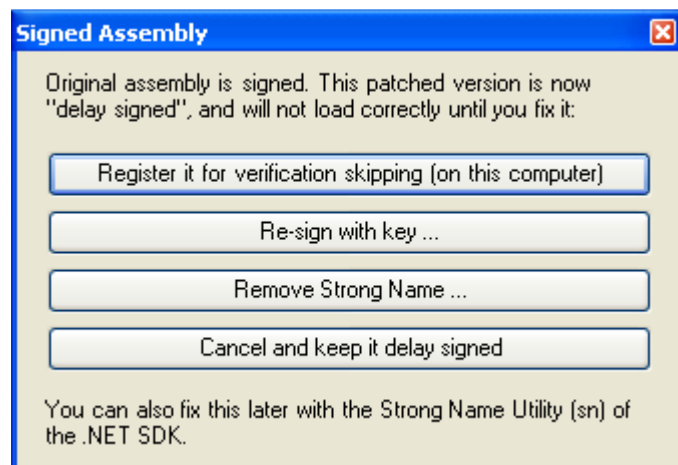
Reflexil allows to add/update/delete exception handlers associated with a method body. The following types are supported:

- **Catch**
- **Filter** (the VB.NET **Where** clause in **Try/Catch** blocks)
- **Finally**
- **Fault**

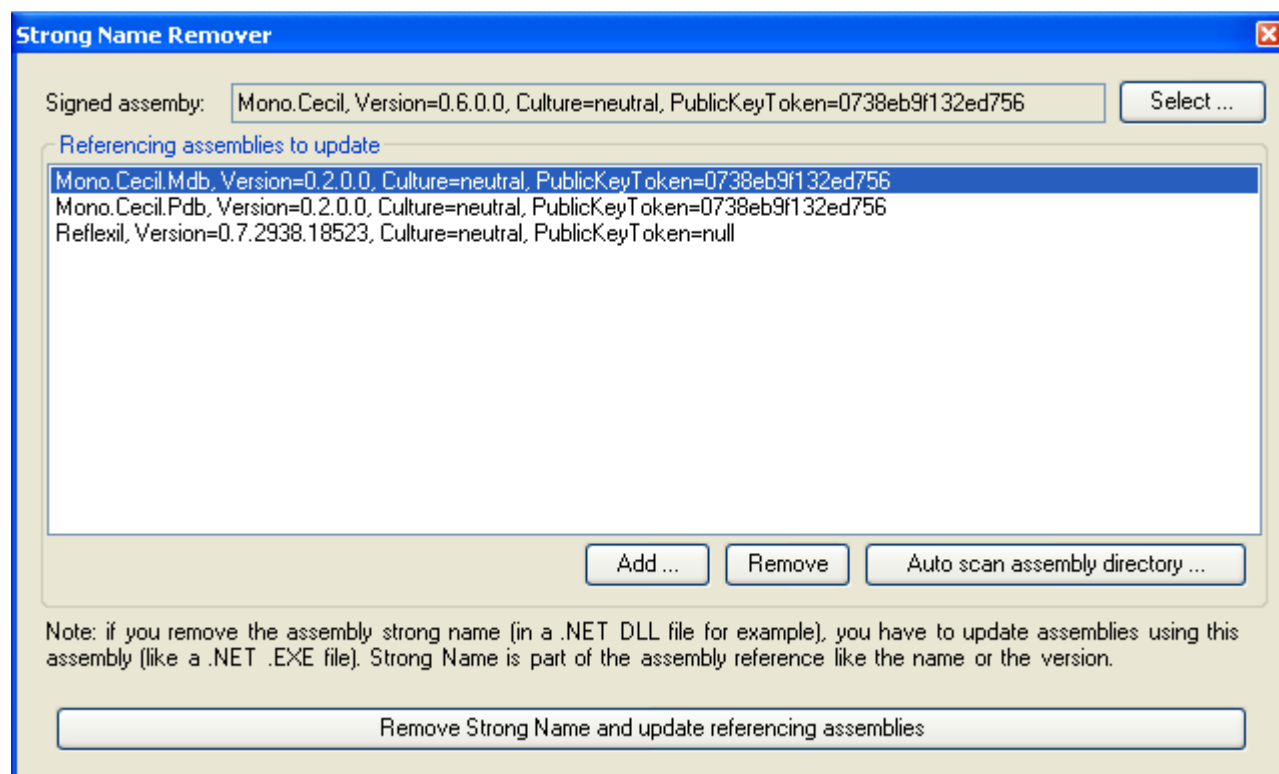


Signed Assemblies Support

When saving a signed assembly, the resulting assembly is placed in the "delay signed" state. **Reflexil** can use SDK tools to fix it.



Since v0.7, **Reflexil** is able to remove assembly strong name and update referencing assemblies. You can also do it by yourself with the assembly editor: remove the public key and set the **HasPublicKey** flag to **false**.



Conclusion

Reflexil is completely based on **Mono.Cecil**. The interesting thing is that **Mono.Cecil** can load assemblies without the help of the runtime, so there is no resource release constraint and **AppDomain** isolation for example. There is no relation between **System.Type** and **Mono.Cecil.TypeDefinition**, whereas they both materialize the .NET type concept. If we want to programmatically reproduce our first example (**Show** overload), we can write the following code, thanks to **Mono.Cecil**:

```
using System;
using Mono.Cecil;
using Mono.Cecil.Cil;

namespace ReflexilDemo
{
    static class Cecil
    {
        static MethodDefinition GetMethodDefinition(AssemblyDefinition adef,
            string owner, string name, int paramcount)
        {
            TypeDefinition tdef = adef.MainModule.Types[owner];
            foreach (MethodDefinition mdef in tdef.Methods)
            {
                // demo purpose only

                if ((mdef.Name == name) && (paramcount ==
                    mdef.Parameters.Count)) return mdef;
            }
            throw new ArgumentException("Unable to find this method!");
        }

        public static void DoPatch()
        {
            AssemblyDefinition targetasmdef =
                AssemblyFactory.GetAssembly("ReflexilDemo.exe");
            MethodDefinition computedefinition =
                GetMethodDefinition(targetasmdef, "ReflexilDemo.DemoForm",
                    "ComputeAndDisplay", 2);

            CilWorker worker = computedefinition.Body.CilWorker;
            Instruction insld =
                worker.Create(OpCodes.Ldstr, "The result is!");
        }
    }
}
```


- reflector item handler

bugfixes:

- field reference fix after code injection
- forms tabindexes

//
// v0.7 - 17/01/2008 //////////////////////////////////////
//

news:

- type attributes editor (sealed, semantic, layout, string format, visibility, ...)
- strong name remover

upgrades:

- sn.exe registry keys with framework 3.5
- C# / VB.NET code generator

bugfixes:

- static field code generation
- VB.NET 'Single' type alias code generation
- C# / VB.NET keywords used as field/method/parameter names
- main window flicker fix

//
// v0.6 - 30/10/2007 //////////////////////////////////////
//

news:

- symbol loading support (pdb and mdb)
- method attributes editor (member access, vtable layout, code type, management, calling convention, return type) -> so you can change a method visibility
- parameter editor -> so you can change a method signature
- variable editor

upgrades:

- multiple selection support in grids
- method RVA tooltip in grids
- assembly / method definition cache system

bugfixes:

- namespaces with type browser
- pointer type matching
- method matching
- generic type matching
- remoting timeout with compilation window
- Mono.Cecil import context update
- VB.NET arrays
- unsafe C# compilation setting
- prevent "insert after/insert before" when a list is empty

//
// v0.5 - 14/09/2007 //////////////////////////////////////
//

news:

- VB.NET code injection
- binary, octal, hexadecimal, decimal base support.
- configuration form

upgrades:

- code injection is no more 'context-free': type, fields and methods can be referenced, and are automatically mapped to original items.
- scroll positions are now saved when creating/updating/deleting instructions or exception handlers

bugfixes:

- injection code works even if the library is not in the same folder than Reflector.
- sn.exe (strong name utility) is correctly located even if PATH variable contains quotes.

//
// v0.4 - 29/08/2007 //////////////////////////////////////
//

```

news:
- exception handlers support.
- exception handler drag&drop.
- signed assembly support.

upgrades:
- Reflector bug report is sent to reflexil mailbox.

bugfixes:
- using non CLI images with Reflector.

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// v0.3 - 20/07/2007 //////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

news:
- instruction drag&drop.
- delete all instructions.
- C# code injection (preliminary support).

upgrades:
- opcodes autocomplete.

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// v0.2 - 08/07/2007 //////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

news:
- editors: type, method or field references.

upgrades:
- instruction edit form with opcodes descriptions (and grid tooltips).

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// v0.1 - 02/07/2007 //////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

History

- 19th September, 2007 -- Article posted
- 22nd October, 2007 -- Full source download added to article
- 13th June, 2008 -- v0.6, v0.7, v0.8 upgrades
- 29th January, 2009 -- v0.9 upgrades
- 23rd October, 2009 -- v1.0 upgrades

License

This article, along with any associated source code and files, is licensed under [The GNU General Public License \(GPL\)](#)

About the Author

Sebastien LEBRETON



Sebastien LEBRETON is a .NET Architect at ALDHERIS, FRANCE.

He is particularly interested in optimization, reverse engineering and distributed objects technologies.

Occupation: Architect

Location:  France

Member

Discussions and Feedback

 **42 messages** have been posted for this article. Visit <http://www.codeproject.com/KB/msil/reflexil.aspx> to post and view comments on this article, or click [here](#) to get a print view with messages.

[PermaLink](#) | [Privacy](#) | [Terms of Use](#)
Last Updated: 25 Oct 2009
Editor: [Deeksha Shenoy](#)

Copyright 2007 by Sebastien LEBRETON
Everything else Copyright © [CodeProject](#), 1999-2009
Web11 | [Advertise on the Code Project](#)