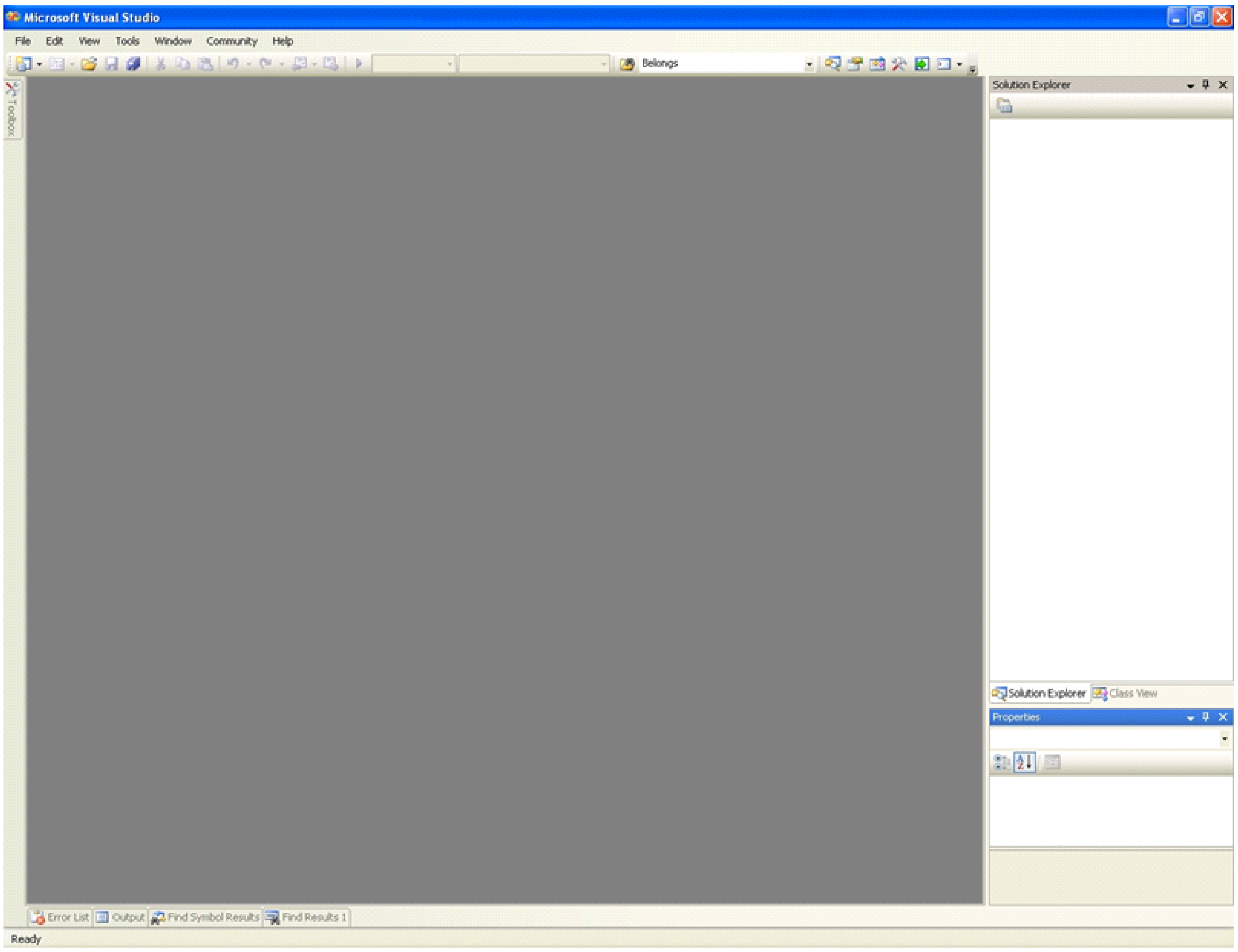
How to create a new convention

In this documents it is shown the step-by-step procedure for the implementation of a new Convention. In this example is used Microsoft Visual Studio 2005 environment, but it is possible to realize the same operations through other environment such as SharpDevelop.

A convention can be written in any .NET language, but in order to avoid convertion or compatibility problems it is better to use C# language.

Steps:

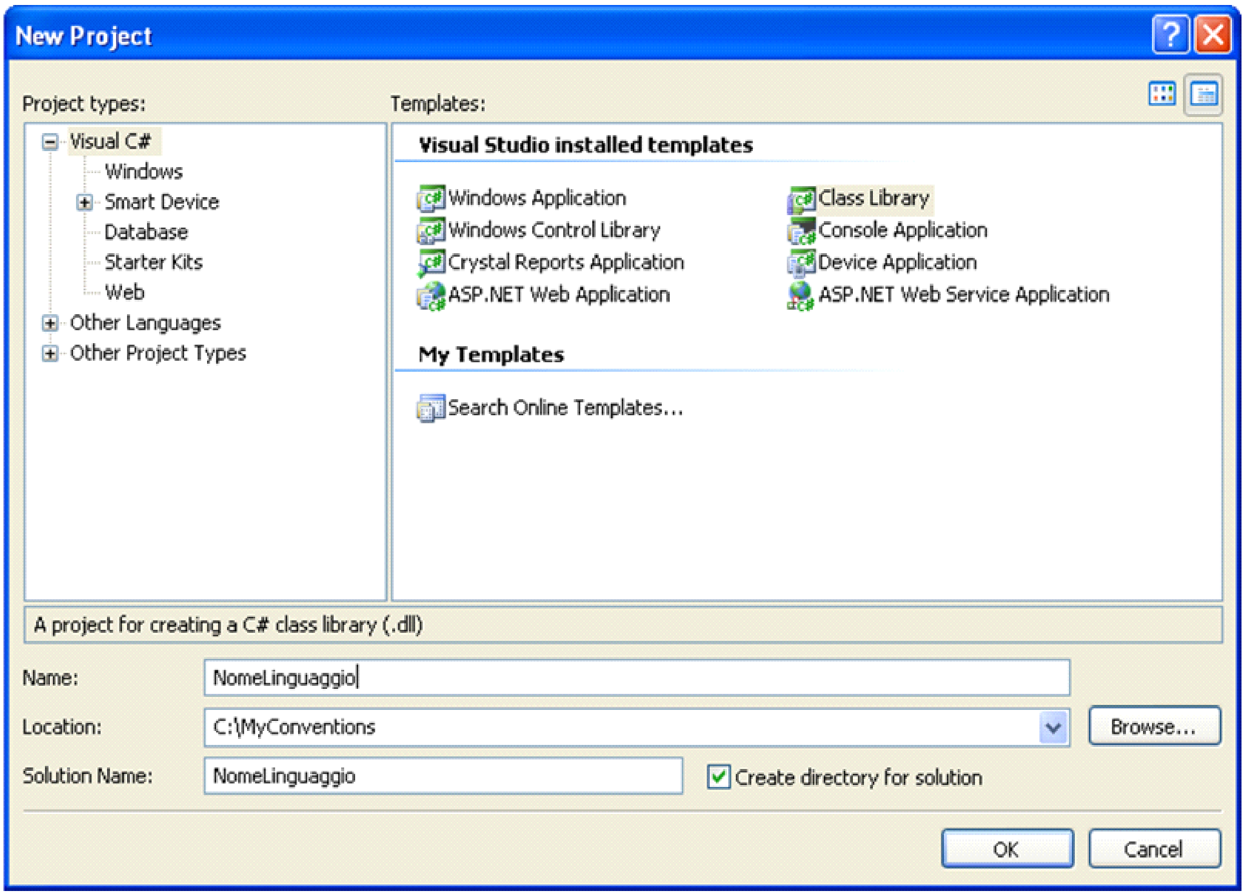
1. Open the development environment.



C.1 – Microsoft Visual Studio 2005 environment

2. Select `File -> New -> Project` (Visual Studio shortcut: Ctrl+Shift+N).
3. Select the `Class Library` template from Visual C# menu, in the same window write the name of the new Convention (it can be more simple to call the Convention with the name of the language, in this example: `NomeLinguaggio`) and select the desired project folder. Leave other options

unchanged and click OK. (see C.2).


New Project

Project types:
- Visual C#
  - Windows
  - Smart Device
  - Database
  - Starter Kits
  - Web
- Other Languages
- Other Project Types

Templates:

Visual Studio installed templates

Windows Application	Class Library
Windows Control Library	Console Application
Crystal Reports Application	Device Application
ASP.NET Web Application	ASP.NET Web Service Application

My Templates

Search Online Templates...

A project for creating a C# class library (.dll)

Name: NomeLinguaggio

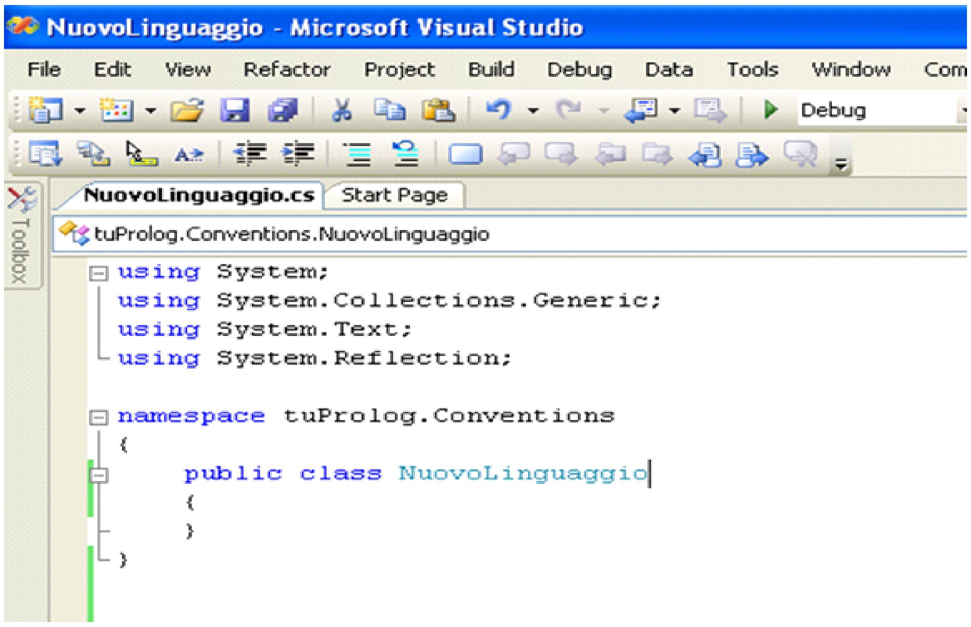
Location: C:\MyConventions

Solution Name: NomeLinguaggio

[✓] Create directory for solution

C.2 – New project

4. Rename the `Class1.cs` file with the name previously selected for the project (it can be done in Solution Explorer panel), in this case the name will be `NomeLinguaggio`. When Visual Studio ask if every reference and occurrence of the class name must be renamed too, click `Yes`.
5. Now we should have the renamed empty class. We need to change the namespace of the class in: `tuProlog.Conventions` (see C.3)

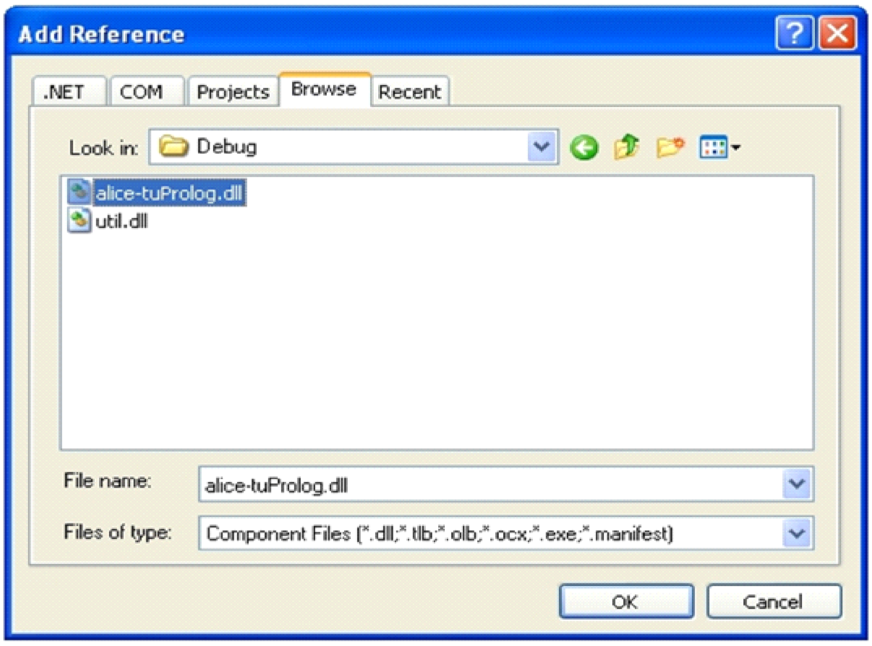


```
using System;
using System.Collections.Generic;
using System.Text;
using System.Reflection;

namespace tuProlog.Conventions
{
    public class NuovoLinguaggio
    {
    }
}
```

C.3 – Namespace changed

6. The new Convention must inherit from the abstract class `Convention` contained in `alice-tuProlog.dll` assembly. So we need to add this reference: right-click on `References` on Solution Explorer panel, then click on `Add Reference`...



Look in: Debug

alice-tuProlog.dll
util.dll

File name: alice-tuProlog.dll

Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

C.4 – Namespace changed

7. Now click on `Browse` section and search for the folder that contains the `alice-tuProlog.dll` file (see C.4). Once found select this file and click `Ok`.
8. It can be useful to add the `System.Reflection` namespace (`using System.Reflection`) in order to avoid the specification of the entire

namespace when using reflection classes. Now we can specify that our new class inherits by `Convention`:

```
public class NomeLinguaggio : Convention
```

9. With Visual Studio refactoring is possible to automatically define every abstract member of `Convention` class: right-click on `Convention` and click on `Implement Abstract Class`. However we are forced to implement just the property `Name` in readonly mode (`get`). In this case we have to define `Name` as follows:

```
public override string Name
{
    get { return "nomelinguaggio"; }
}
```

10. Now we can override all methods that we desire. For example `GetClassName`:

```
protected override string GetClassName(string oldClassName)
{
    <new implementation of this method>
}
```

11. If it is necessary calling a method of the base class `Convention` we use the keyword `base`. For example we can do it in `GetMethod` overriding:

```
public override MethodInfo GetMethod(Type type, string
  methodName, Type[] argTypes, object[] argValues)
{
   <codice…>

  MethodInfo m = base.GetMethod(type, methodName, argTypes,
  argValues);

  <codice…>
}
```

12. Once finished overriding methods, we can cimpile the solution and so the system creates the DLL file of our new Convention: click on `Build -> Build NomeLinguaggio` (or `Build solution`).

13. To use this new Convention in a .NET project we have to retrieve the DLL file containing the new Convention; in this example the file is called `NomeLinguaggio.dll` and is contained in this folder:

```
C:\MyConventions\NomeLinguaggio\NomeLinguaggio\bin\Debug
```

14. Copy this file into the compilation folder of our main .NET project (or into tuProlog IDE or CUI Console folder if we want to use those tools).

15. Then use the predicate or directive `load_convention/3` in order to load

our new Convention. In our example we will use the predicate but here is an example for each of them:

Predicate `load_convention/3`:

```
load_convention('NomeLinguaggio.dll','NomeLinguaggio', MyConvention)
```

Directive `load_convention/3`:

```
:-load_convention('NomeLinguaggio.dll','NomeLinguaggio',
                   myconvention).
```

16. The Convention will be automatically used in object management, but we can also use `MyConvention` variable (or `MyConvention` atom) to declare which object must be managed with our new Convention. Example:

```
cli_object(MyConvention, 'MyLibrary.dll', 'Namespace.Class', Object)
```

17. If we want to unload the new Convention we can use the `unload_convention/1` predicate:

```
mainclause :-
load_convention('NomeLinguaggio.dll','NomeLinguaggio',MyConvention),
cli_object(MyConvention, 'MyLibrary.dll', 'Namespace.Class', Object),
unload_convention(MyConvention),
. . .
```

18. From this point `MyConvention` becomes an uninstantiated variable and `NomeLinguaggio` Convention is no more used.