Thin Books

Licence Agreement
1) You are not allowed to distribute this book in any format.
2) You may not reproduce any part of this book
3) You can add a link to your site to http://www.RealSoft.co.uk.
4) RealSoft reserves the right to remove this book without notice.
5) RealSoft reserves the right to change the terms of this agreement without notice.
6) The Term "Thin Books"™ is a trademark of RealSoft Limited
7) The Term "Thin Book"™ is a trademark of RealSoft Limited

The idea behind "Thin Books" is to provide you with the basics in any given topic, they do not try to dig into the depths but simply provide you with a quick start and from there you can decide to, or not to, read more details by searching the rest of the sprawling web.

Note this book does assume basic coding / c# knowledge

This is a Draft

Download Source Code :

Thin Books

Table of Contents

Thin Books

1 What Is Reflection?

Reflection is the ability to discover the operations and properties of an artefact at runtime, it also provides the means to be able to create (instantiate) and carry out operations on libraries that are discovered at runtime.

You may also hear the term Introspection, read as reflection.

2 Using Assembly Class

You can you this classes static operations to create dynamically loaded assemblies.

2.1 LoadFrom Static Operation

This operation will load an assembly from disk

```
private static Assembly LoadAssemblyFromFile()
{
        string assemblyName = "RealSoft.Samples.ReflectionBasics.exe";
        string fullPathToAssembly = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, assemblyName);

        Assembly ass = System.Reflection.Assembly.LoadFrom(fullPathToAssembly);
        return ass;
}
```

2.2 Create An Instance Of A Class

```
private static void CreateInstanceOfClass(Assembly ass, out Type klass, out RealSoft.Samples.ReflectionBasics.Program pgm)
    {
        string classFullName = "RealSoft.Samples.ReflectionBasics.Program";
        klass = ass.GetType(classFullName);

        ConstructorInfo ctor = klass.GetConstructor(new Type[] { });
        pgm = (RealSoft.Samples.ReflectionBasics.Program)(ctor.Invoke(null));
    }
```

2.3 Executing The Methods

```
private static void ExecuteMethods(Type klass, RealSoft.Samples.ReflectionBasics.Program pgm)
    {
        BindingFlags bindingFlags = BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Static;
        MethodInfo[] mi = klass.GetMethods(bindingFlags);
        MethodInfo m = klass.GetMethod("ExecuteStaticMethod", bindingFlags);

        //Execute the static and instance operations
        m.Invoke(null, null); //Static Method
        pgm.ExecuteMethod(); //Instance Method
    }
```

2.4 The Whole Sample

You should be able to simply copy and paste this, make sure you change the name of the produced assembly to ***RealSoft.Sample.RelectiomBasics.exe***

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Reflection;
namespace RealSoft.Samples.ReflectionBasics{
   internal class Program
   {
      static void Main(string[] args)
      {
         LoadUsingAssemblyClass();

         Console.ReadKey();
      }

      private static void LoadUsingAssemblyClass()
      {
         Assembly ass = LoadAssemblyFromFile();

         Type klass;
         RealSoft.Samples.ReflectionBasics.Program pgm;
         CreateInstanceOfClass(ass, out klass, out pgm);

         ExecuteMethods(klass, pgm);
      }

       private static Assembly LoadAssemblyFromFile()
      {
         string assemblyName = "RealSoft.Samples.ReflectionBasics.exe";
         string fullPathToAssembly = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, assemblyName);

         Assembly ass = System.Reflection.Assembly.LoadFrom(fullPathToAssembly);
         return ass;
      }

      private static void CreateInstanceOfClass(Assembly ass, out Type klass, out RealSoft.Samples.ReflectionBasics.Program pgm)
      {
         string classFullName = "RealSoft.Samples.ReflectionBasics.Program";
         klass = ass.GetType(classFullName);
         ConstructorInfo ctor = klass.GetConstructor(new Type[] { });
         pgm = (RealSoft.Samples.ReflectionBasics.Program)(ctor.Invoke(null));
      }

      private static void ExecuteMethods(Type klass, RealSoft.Samples.ReflectionBasics.Program pgm)
      {
         BindingFlags bindingFlags = BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.Static;
```

```
        MethodInfo[] mi = klass.GetMethods(bindingFlags);
        MethodInfo m = klass.GetMethod("ExecuteStaticMethod", bindingFlags);

        //Execute the static and instance operations
        m.Invoke(null, null); //Static Method
        pgm.ExecuteMethod(); //Instance Method
    }

    private static void ExecuteStaticMethod()
    {
        Console.WriteLine("ExecuteStaticMethod");
    }

    private void ExecuteMethod()
    {
        Console.WriteLine("ExecuteMethod");
    }
  }
}
```

2.5 How to Get .ctors That Have Signatures

In the example above the default constructor was used. If you had a constructor that took, for example, a string and an int you would make the following changes

1) `ConstructorInfo ctor = klass.GetConstructor(new Type[] { });` would become
   `ConstructorInfo ctor = klass.GetConstructor(new Type[] { typeOf(string), typeOf(int) });`

2) `pgm = (RealSoft.Samples.ReflectionBasics.Program)(ctor.Invoke(null));` would become
   `pgm = (RealSoft.Samples.ReflectionBasics.Program)(ctor.Invoke("SomeString", 10));`

Thin Books

3 Using The Activator Class