Introduction to building .NET Reflector add-ins

.NET Reflector has an extensive add-in framework, and there are plenty of add-ins already available to use as examples of what can be done.

A .NET Reflector add-in is fundamentally a dll/exe assembly file that contains packages. A package is a class that implements the `IPackage` interface, which defines a Load and Unload method. An `IServiceProvider` interface is passed during loading, and gives access to a set of services which are part of the .NET Reflector object model (the most common of which we'll see below).

Available services

The following table lists the most commonly-used services that can be accessed through the `GetService` method on`IServiceProvider`

Service	Description
IAssemblyBrowser	Maintains the currently selected Code Model object in the `ActiveItem` property. You can assign a Code Model object like IMethodDeclaration to the `ActiveItem` to programatically change the currently selected item in the browser window. ActiveItemChanged notifies that the selected item has changed.
IWindowManager	Manages the application window and pane windows. You can add your own pane windows to the Windows collection which will create an IWindow hosting frame. ShowMessage can be used to show notification messages to the user.
ICommandBarManager	Manages the Reflector menu bar, tool bar and context menus. You can lookup a context menu by its identifier and add items to it.
IConfigurationManager	Manages the sections from the Reflector configuration file as a set of IConfiguration objects. Lists of items are represented as properties named "0?, "1?, "2?, and so on.
IAssemblyManager	Maintains the list of currently loaded assemblies. LoadFile can be used to load an assembly file from disk. Unload allows you to unload an assembly from memory. The Assemblies collection holds all the currently loaded assemblies.
ILanguageManager	Manages formatting modules for different programming languages. The ActiveLanguage property exposes the ILanguage object currently used for rendering. You can add your own language rendering code by implementing the ILanguage interface. Use RegisterLanguage to add your add-in to ILanguageManager.

Although the .NET Reflector API exposes more interfaces than this, these are the most commonly used ones.

Building a HelloWorld add-in

A simple "HelloWorld" add-in can be created by implementing the `IPackage` interface.

The Load method is implemented to ask the `IServiceProvider` for the `IWindowManager` service, which allows you to communicate with .NET Reflector's windowing system. Finally, the `ShowMessage` method is used to show a message to the user:

```
using System;
using Reflector;
internal class HelloWorldPackage : IPackage
{
        private IWindowManager windowManager;
        public void Load(IServiceProvider serviceProvider)
        {
                this.windowManager = (IWindowManager) serviceProvider.GetService(typeof(IWindowManager));
                this.windowManager.ShowMessage("Loading HelloWorld!");
        }
        public void Unload()
        {
                this.windowManager.ShowMessage("Unloading HelloWorld!");
        }
}
```

The code can be compiled into an add-in dll, which is referencing Reflector.exe as a library:

```
csc.exe /target:library /out:HelloWorld.dll *.cs /r:Reflector.exe
```

The add-in can then be copied to your Reflector directory and loaded using the View Add-Ins menu. While this is a very basic add-in, the fundamentals of the construction and implementation don't change.

Adding items to command bars and context menus

The `ICommandBarManager` service allows you to add menu items to the .NET Reflector main menu and context menus. Each sub-menu and context menu is registered in the `CommandBars` collection with an identifier name, and the following table lists the most commonly used identifiers:

Identifier	Description
Tools	The tools menu shown as part of the main menu.
Browser.Assembly	The context menu for the currently selected assembly.
Browser.Namespace	The context menu for the currently selected namespace.
Browser.TypeDeclaration	The context menu for the currently selected type declaration.
Browser.MethodDeclaration	The context menu for the currently selected method declaration.

Learning more

Thoroughly documenting the .NET Reflector API is something we hope to improve in future.

We currently recommend this series of articles by Jason Haley:

- Getting Started with .NET Reflector add-ins
- Create your own add-in : The basics
- Create your own add-in : More details
- Wrapping .NET Reflector

For more examples, see:

- .NET Reflector Add-in Tutorial (Peli de Halleux)
- Building the .NET Reflector Add-in (Jamie Cansdale)