

Вступление

Я думаю вы часто встречали программы использующие плагины, например не безызвестный браузер **Mozila Firefox**. Так в чем же его секрет? Как программа узнает, что ей "подсунули" плагин, а не какую-то "левую" библиотеку? Об этом, и не только, вы узнаете далее.

Перед тем как начинать советую почитать про объектно ориентированное программирование, далее просто ООП, и полиморфизм с абстрактными классами/интерфейсами, а так же про рефлексии.

Сложность задания — выше среднего [Без знаний **ООП** и полиморфизма — **over9000**]

Начало. Построение задачи

Давайте поставим некую задачу, представим, что же должен делать наш плагин. Например я хочу, что бы мой плагин добавлял контрол на форму, пускай этот контрол будет кнопкой, а кнопка в свою очередь вызывала еще одну форму. Случай частный, но все же имеет право на существование.

Начало. Проект

Для начала нужно создать проект будущего приложения, назовем его **PlugIt**, это будет «основа» к которой будут подключаться плагины. После создания основы нужно добавить еще 2 проекта в наше решение(**Solution**), назовем их **Core** и **Plugin**, первый будет сердцем наших плагинов, он научит программу понимать и использовать их. Второй проект, как уже ясно из названия, и будет плагином. Обратите внимание, оба проекта являются библиотеками классов.

Теперь нужно добавить ссылку на проект **Core** в проекты **PlugIt** и **Plugin**, для этого щелкаем ПКМ по проекту и выбираем в контекстном меню «Добавить ссылку...» и на вкладке «Проекты» выбираем проект **Core**. С рутиной вроде бы разобрались, теперь можно преступать к самому программированию.

Начало. Интерфейс

В проекте **Core** студия сама создала пустой класс, замените его содержимое следующим:

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

namespace Core
{
    public interface IPlugin
    {
        Version Version { get; }
        string Name { get; }
        void Execute();
        void Load(Form main);
    }
}
```

Это тот самый интерфейс который будут реализовывать наши плагины, обратите внимание, что это не класс, а именно интерфейс и названия интерфейсов принято начинать с заглавной латинской буквы «I».

Все приготовления завершены, теперь можно приступить к самому интересному, к реализации нашей задумки.

Реализация

Начнем мы с конца, так в дальнейшем будет понятнее, а именно с создания плагина.

Добавим в проект **Plugin** форму, я назвал ее **TestForm**, она нам понадобится в дальнейшем.

Теперь нужно создать класс нашего плагина, я назвал его так же как и проект — **Plugin**, а вы можете назвать его как вашей душе угодно, и реализовать в нем интерфейс **IPlugin** который мы описали выше.

```
using System;
using System.Drawing;
using System.Windows.Forms;
using Core;

namespace Plugin
{
    public class Plugin : IPlugin
    {
        public Version Version
        {
            get { return new Version(0, 1, 1, 4); }
        }

        public string Name
        {
            get { return "Execute me!"; }
        }

        public void Execute()
        {
            MessageBox.Show("Автор Yukikaze\nSkype: N\\A\nICQ: N\\A", "Execute Me!");
        }

        public void Load(Form main)
        {
        }
    }
}
```

При реализации интерфейса класс обязан иметь такие же поля/свойства/методы/etc что и сам интерфейс. Сейчас плагин не несет никакого функционала, кроме имени и версии. Ну что же, будем добавлять.

Создадим кнопку, которая будет добавляться на форму и подпишем ее на событие **Click**

```
private Button executeButton;
private void Initialize()
{
    executeButton = new Button();
    executeButton.Text = "Execute me!";
    executeButton.Name = "button";
    executeButton.Location = new Point(5, 30);
    executeButton.AutoSize = true;
    executeButton.Click += button_Click;
}

private void button_Click(object sender, EventArgs e)
{
    TestForm form = new TestForm();
    form.ShowDialog();
}
```

Для большего удобства обернем инициализацию компонента в метод Initialize(), теперь нужно добавить контрол на форму, для этого допишем в метод Load следующие строки:

```
public void Load(Form main)
{
    Initialize();
    main.Controls.Add(executeButton);
}
```

Этот метод будет вызываться сразу при загрузке программы, а в качестве параметра он будет принимать нашу главную форму.

На этом мы заканчиваем написание плагина, теперь можно приступить к основе программы и ее «обучению».

Реализация. Рефлексия

Для динамической загрузки сборок используется рефлексия, дальше вы увидите пример двух методов для загрузки плагинов и преобразования их в объекты класса.

```
private IPlugin LoadFromFile(string file)
{
    Assembly asm;
    try
    {
        asm = Assembly.LoadFile(Path.GetFullPath(file)); //Загружаем сборку
    }
    catch { return null; }

    try
    {
        Type[] types = asm.GetTypes();
        Assembly core = AppDomain.
            CurrentDomain.GetAssemblies().
            Single(x =>
                x.GetName().Name.Equals("IPlugin")); //Получаем загруженные сборки
        Type type =
            core.GetType("Core.IPlugin"); // Получаем тип IPlugin
            //из сборки Core

        Type pluginInfo =
            types.Where(type.IsAssignableFrom).
            First(); // Проверка на наличие одинаковых
                    // типов,если такой имеется
                    // заносим его в переменную
        if (pluginInfo == null) return null; // А если не имеется(равно null)
            //то просто прерываем выполнения метода

        Object o =
            Activator.CreateInstance(pluginInfo); // Создаем экземпляр класса
        IPlugin plugin = (IPlugin)o; // Явно приводим его к к типу IPlugin
        return plugin;
    }
    catch { }
    return null;
}
```

Прочитав название метода сразу становится понятно, этот метод загружает плагин из .dll библиотеки и получает экземпляр класса IPlugin, если покрутить этот методот в цикле можно получить следующее:

```

private List<IPlugin> LoadFromFolder(string path)
{
    string[] files =
        Directory.GetFiles(path, "*.dll"); // Получаем все файлы с
                                           // расширением .dll

    return files.
        Select(LoadFromFile).
        Where(plugin => plugin != null).ToList(); // возвращаем список
                                                  // экземпляров класса
                                                  // полученных из dll'ок в папке
}

```

Этот метод загружает все плагины в указанной папке и преобразует их в список. Дальше работаем с полученным списком как хотим, например можно добавить на форму компонент MainMenuStrip, и разместить в нем все плагины.

```

private void InitializeMenuStrip()
{
    foreach (IPlugin plugin in pluginContainer) // Для каждого загруженного плагина
    {
        ToolStripItem item = new ToolStripMenuItem(); // создаем пункт в меню,
        item.Name = plugin.Name + plugin.Version; // даем ему уникальное имя
        item.Text = string.Format("{0} [ver. {1}]",
            plugin.Name,
            plugin.Version); // и собственно само название пункта
        IPlugin temp = plugin;
        item.Click += delegate { temp.Execute(); }; // Подписываем пункт меню на
                                                    // событие

        if (!((ToolStripMenuItem)MainMenu.
            Items["pluginToolStripMenuItem"]).
            DropDownItems.ContainsKey(item.Name)) // Если меню не содержит
                                                    // такого пункта
            ((ToolStripMenuItem)MainMenu.Items["pluginToolStripMenuItem"]).
            DropDownItems.Add(item); // то добавляем его
    }
}

```

А в конструктор формы добавим следующее

```

public List<IPlugin> pluginContainer;
public Main()
{
    pluginContainer = LoadFromFolder(@"Plugins");
    InitializeComponent();
    InitializeMenuStrip();

    foreach (IPlugin plugin in pluginContainer)
    {
        plugin.Load(this);
    }
}

```

Таким образом мы добавим в пункт меню "pluginToolStripMenuItem" подпункты с названиями плагинов и подпишем их на событие Click, при котором будет вызываться метод Execute. А при инициализации формы мы выполним метод Load для каждого плагина.