Articles » General Programming » Macros and Add-ins » Add-ins/Plug-ins

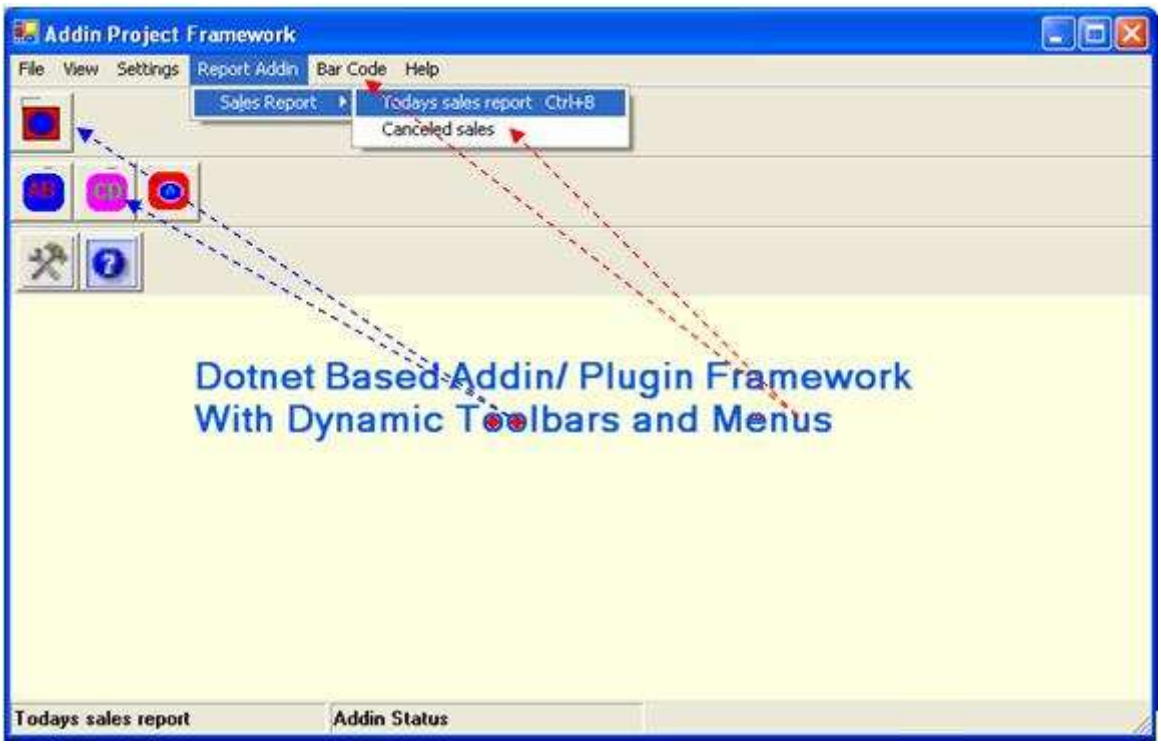
.NET Based Add-in/Plug-in Framework with Dynamic Toolbars and Menus

thomas_tom99, 2 May 2006

An article on a .NET based add-in/plug-in framework, with dynamic toolbars and menus.

Download demo project - 27.4 Kb

Download source - 88.5 Kb



Introduction

Building a software product which can meet the future needs of customers is always a challenging task. This is especially the case when the software enters into the enhancement / maintenance stage. At this stage, the team members who had originally created the software might have left the company, or they may be in other project teams. You will be lucky if you can find a team member who was originally in the development team to help you. If a programmer who is new to the product is involved in the enhancement of the product, then there is a greater chance that he will do some mistake and will break the existing

functionality, which will be **unidentified** till it happens in the customer's site. Worst is the case if the software is not designed well and proper documentation is not available. Also, the developmental cost of adding a new functionality to an existing product is also very high just because the entire system needs to be tested again and shipped as another version. This is where the Addin Project Framework wins. In my previous article, I had shown you how to write a plug-in framework with dynamic toolbars and menus in VC++, MFC, ATL, and COM. In this article, I will show you how to do the same thing using the .NET framework libraries.

In spite of the fact that Microsoft is pushing their customers to .NET technology, their product VS.NET 2005 Product Framework is still written in old Win32, MFC, and COM technologies. This forced me to think that there may be some problems with the current .NET framework libraries which forces Microsoft not to use it in their main products such as VS.NET 2005 (or is there any other reason why they didn't use the .NET technology in their products?). This article is the **outcome of this thinking**. This article is just written out of curiosity to know how difficult it is to write a plug-in framework with dynamic toolbars and menus, in .NET. To my surprise, writing a .NET plug-in framework similar to the one which I wrote in VC++, MFC, and ATL is very easy, and it will take only half the time compared to VC++, MFC, and ATL. Apart from some small limitations like keyboard shortcut key customization in .NET (the .NET library provides only predefined short cut keys), I didn't see any difference, and it is more efficient as far as the implementation is concerned.

In this article, I will explain this .NET based add-in project framework architecture, how the plug-in loads and its interaction with the framework, how to invoke the add-in functions from menus and toolbars etc. I will also discuss the pros and cons of both the approaches (the .NET based approach, and VC++, MFC, and ATL/ COM based approach) which I think will be useful in your future projects. There are other plug-in frameworks based on the .NET framework which are available in CodeProject. The advantage with this framework is that it is very simple and easily understandable compared to other frameworks. You can extend this framework to suit your needs, once you get an idea about how it works.

Background

This article assumes that you have some background in C# programming and are familiar with some common namespaces and libraries which are available, that you have some idea about building .NET components, XML, interfaces etc. The core of this framework depends on the `System.Reflection` namespace and a good idea about this namespace is assumed.

Brief Architecture

The brief architecture of the .NET plug-in framework model is shown below:

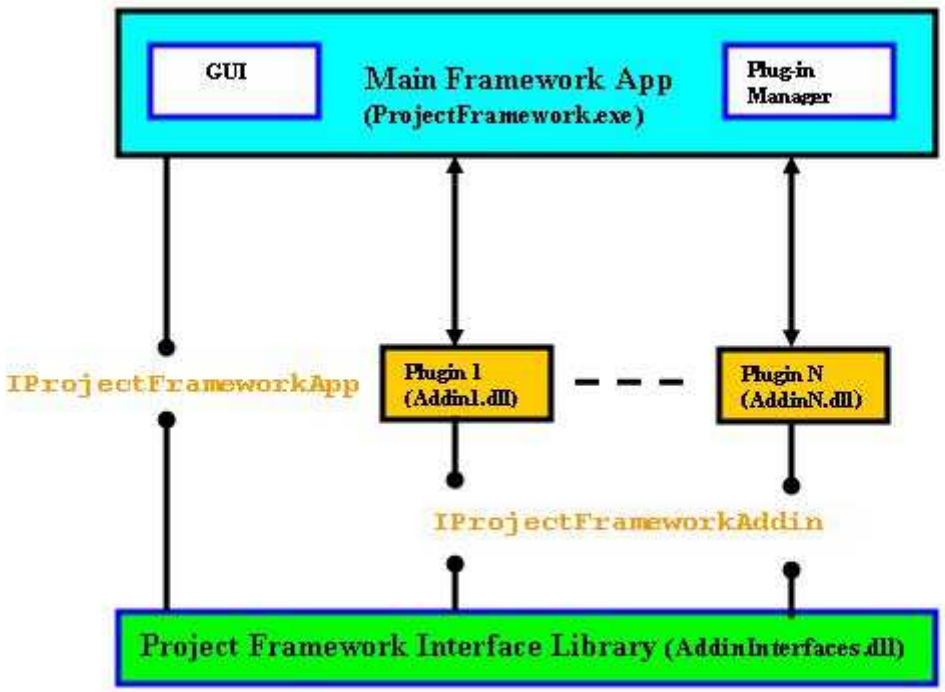


Fig. 1 .NET Plug-in Framework Architecture.

The architecture consists of an interface DLL called *AddinInterfaces.dll* which contains the necessary interfaces which every framework (`IProjectFrameworkApp`) and plug-in (`IProjectFrameworkAddin`) should implement. The basic interface definition of each interface is given below:

```
public interface IProjectFrameworkAddin
{
  void InitializeAddin(ref IProjectFrameworkApp ref
       ProjectFrameworkApp, long lSession,bool bFirstLoad);
  void UnInitializeAddin(long lSession);
}
```

This is the interface which every plug-in should implement so as to establish a connection between the plug-in and the framework. The functions in this interface is called by the framework at appropriate times.

```
public interface IProjectFrameworkApp
{
  void AddCommandsInfo(string strXMLMenuInfo, long lSession,
       object lInstanceHandle, object lToolbarInfo);
  void SendMessage(string strMessage);
}
```

This interface is implemented by the framework which is used by the plug-ins so as to send information and messages to the framework.

You can add additional interfaces in this DLL which will be implemented by the framework or the add-in, depending on the needs.

XML Plug-in Menu Format

The framework uses the following **XML** structure to exchange information about the menu, toolbars, and other information in the add-ins.

```
<ProjectFrameworkAddin>
 <AppVer>1</AppVer>
 <AddinName>Report Addin</AddinName>
 <ToobarButtonCount>1</ToobarButtonCount>
 <MainMenu>
  <Name>Bar Code</Name>
  <ShortcutKeyIndex>1</ShortcutKeyIndex>
  <SubMenu>
      <Name>Bar Code</Name>
      <ShortcutKeyIndex>1</ShortcutKeyIndex>
    <LeafMenu>
      <Name>Test Menu</Name>
      <FunctionName>AddinFunctionName</FunctionName>
      <HelpString>Some Status bar text</HelpString>
      <ToolTip>Some tool tip text</ToolTip>
      <ToolBarIndex>Addin2Settings.ico</ToolBarIndex>
      <ShortCutKey>Ctrl + H</ShortCutKey>
    </LeafMenu>
    <LeafMenu>
      ........
      ........
    </LeafMenu>
  </SubMenu>
  <SubMenu>
    ..............
    ..............
  </SubMenu>
 </MainMenu>
 <MainMenu>
  ......
```

```
  ......
 </MainMenu>
</ProjectFrameworkAddin>
```

Program Flow

The basic program flow of this framework is in shown in fig 2 as a high level sequence diagram.

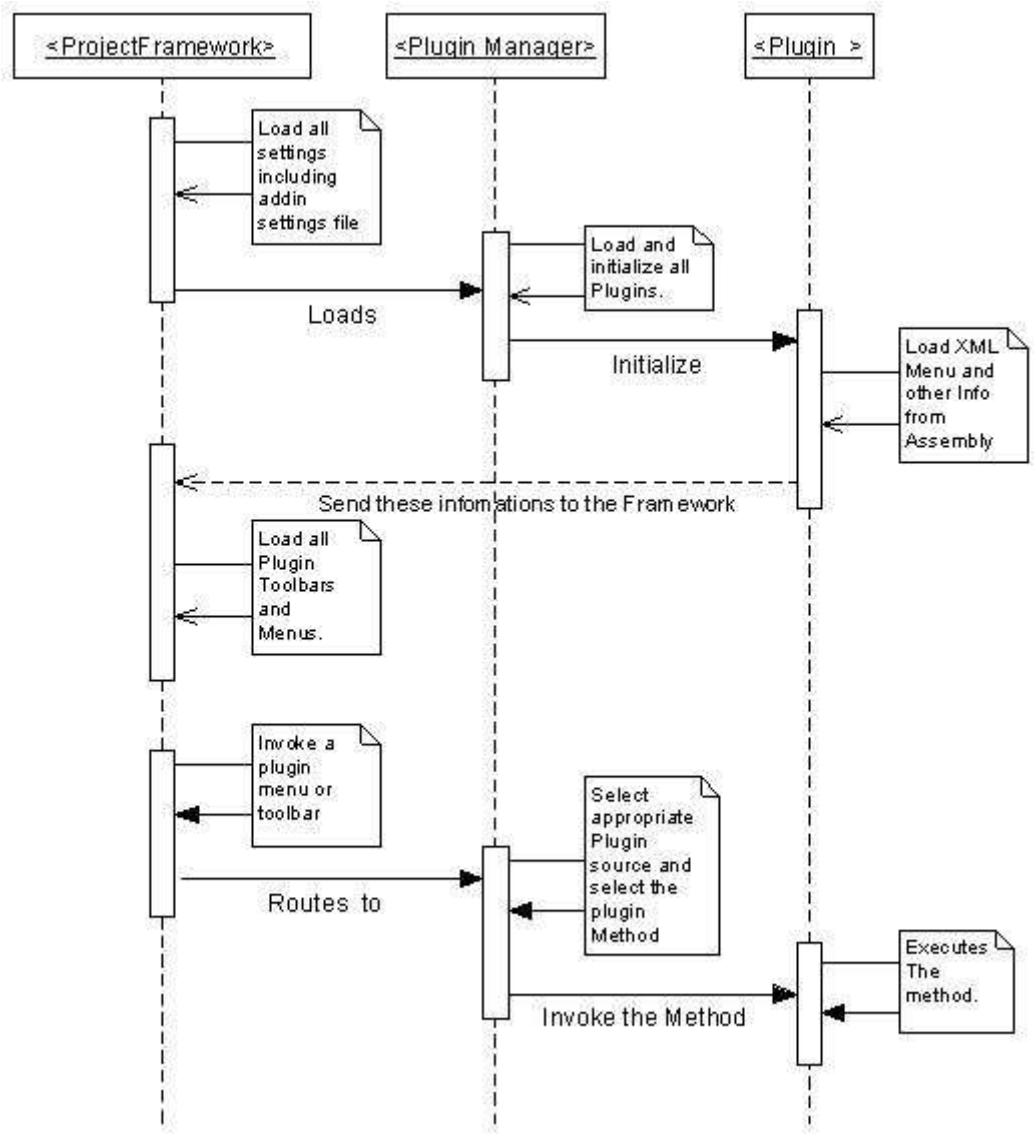


Fig. 2 Program Flow

Using the code

The sample source code contains four projects, vis. *ProjectFramework*, *AddinInterfaces*, *Addin1*, and *Addin2*. The ProjectFramework is the core app which loads all the add-ins which are found in the addin directory. AddinInterfaces is the DLL which is **referenced** by the framework and the add-ins, which contains all the interface definitions. Addin1 and Addin2 are two sample add-ins. Please see the source code for more details. The important blocks of code on which the application is built up are given below.

In the `AddinProjectFramework` constructor, please note the code in bold and see the high level program flow diagram (Fig. 2).

```
public AddinProjectFramework()
{
```

```
    // // Required for Windows Form Designer support //
    m_PluginManager= new PluginManager();
    m_FrameworkApp = new ProjectFrameworkApp();
    m_AddinToolbarArray= new ArrayList();

    System.Configuration.AppSettingsReader
         configurationAppSettings = new
         System.Configuration.AppSettingsReader();
    m_strPluginFile=((string)
         (configurationAppSettings.GetValue(
         "PluginFile", typeof(string))));

    InitializeComponent();
    m_PluginManager.ProjectFramework=this;
    m_FrameworkApp.ProjectFramework=this;
    //Set the IProjectFrameworkApp reference
    m_PluginManager.m_FrameworkApp =
      (IProjectFrameworkApp)m_FrameworkApp;

    //Load all addins
    if(CheckAddinSettings())
    {
        LoadAllAddins(m_bLoadAllAddins);
        m_PluginManager.LoadAddinMenus();
        m_PluginManager.LoadAddinToolbars();
        m_PluginManager.LoadAddinToolbarMenus();
        //Remove the about box from
        //the Middle and add it towards the end.
        //Help menu is here
        mainProjectMenu.MenuItems.RemoveAt(3);
        mainProjectMenu.MenuItems.Add(menuItemHelp);
    }
}
```

The core of the framework depends on a class called `PluginManager`. The skeleton of the plug-in manager is given below:

```
namespace ProjectFramework
{

/// This class handles all the plugin
/// related activities their
///loading, saving etc
public struct AddinCommadInfo
{
    //All Menus before coming to the leaf node
    public ArrayList MenuStringsArray;
    public int iCommandID;
    public string strMenuString;
    public string strHelpString;
    public string strToolTip;
    public string strFunction;
    public string strShortCutKey;
    public string strToolbarIndexName;
    public int iSeparator;
    public MenuItem Menu;
    void Initialize()...
};
public struct AddinInfo
{
    public string strAddinName;
    public string strAddinDllName;
    public string strAddinVersion;
    public object lInstanceHandle;
    public object lToobarRes;
    public bool bLoadAddin;
    public int lToolbarButtonCount;
    public Assembly AddinAssembly;
    public Type AddinType;
```

```
    public IProjectFrameworkAddin FrameworkAddin;
    public string strAddinInterfaceName;
    public ArrayList AddinCommadInfoArray;
    void Initialize() ....
};
public class PluginManager
{
    public AddinInfo[] AddinInfoArray;
    public AddinProjectFramework ProjectFramework;
    public IProjectFrameworkApp m_FrameworkApp;
    public bool m_bLoadAddinsOnStartup;
    private string m_strXMLFileName;
    public PluginManager()....
    public bool LoadPluginDetailsFromXML(string
                strXMLFileName)..
    public bool SavePluginDetailsToXML()...
    public bool LoadAddAddinAssemblyInfo(string
                strAddinFolderName)...
    public bool UnloadAllAddins()...
    public bool InvokeAddinMember(int iAddinIndex,
                string strFunctionName)...
    public string GetMainInterfaceName(string strDllName)...
    public int GetAddinIndex(string strDllName)...
    public bool InvokeMenu(ref object AddinMenuItem)...
    public bool UpdateAddinMenuStatus(int iAddinIndex,
                bool bEnable)...
    public bool LoadAddinMenus()...
    public bool LoadAddinToolbars()...
    public bool LoadAddinToolbarMenus()...
    public void GetMainMenuItem(string strMenuName,
                out MenuItem Item)....
    public void GetMenuItem(string strMenuName,
                ref MenuItem ItemParent,
                out MenuItem Item)...
}
```

Comparison Between .NET based and MFC, ATL, COM-Based Plug-in Frameworks

Feature	.NET Based Plug-in Framework	MFC, ATL, COM-Based Plug-in Framework
Ease of use	Very easy to use.	Difficult to use. Needs to know a lot about the intricacies of COM.
Flexibility	For **database applications**, this plug-in framework is good. **Not** flexible for high-end applications like spread sheets and graphics packages.	Good for all sorts of frameworks where a great deal of flexibility is needed.
Modularity	.NET libraries are loosely coupled. So easy to design and develop pattern oriented applications.	MFC libraries are tightly coupled. Difficult to design and develop pattern oriented applications.
Speed	Once the IL is converted into machine code, performance is comparable to native code.	Performance is the best. A lot of optimization techniques are available.
Memory	Memory consumption is high compared to native code. Also, the CLR determines the lifetime of unused objects. Not deterministic as far as memory optimization is concerned.	Has more control over memory. Can make very good memory optimizations.
Support	.NET is a new technology. Can expect somewhat good support in the future.	Very good support available till now. Time will only tell whether Microsoft will support MFC in the future.

Cost of development	Low	High.
Data type support	Any type which .NET framework is supporting.	Supports only COM specific data types, interfaces, and custom interfaces.

Points of Interest

In this version, the features implemented are:

- Dynamic menus.
- Dynamic toolbars.
- Help string and tool tip support.
- Invoking of methods in plug-ins from add-in menu and toolbar.
- Automation support explained using a dialog box invoked from one of the plug-ins (Report Add-in -> Sales report -> Today's Sales Report, Ctrl + B menu).
- Keyboard accelerator support for plug-in.
- Loading / unloading of add-ins.
- Support for getting notification events from add-ins.

Links

- Len Holgate's 'Writing extensible applications'.
- ATL COM Based Addin / Plugin Framework With Dynamic Toolbars and Menus.
- The Razor Framework :: Part 1 :: Plugins/Extensibility.

History

- Initial release: May 2, 2006.

License

Share

About the Author

thomas_tom99

Chief Technology Officer KTS INFOTECH PVT LTD

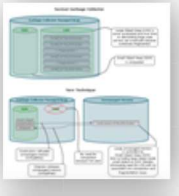
India

You may also be interested in...

SAPrefs - Netscape-like Preferences Dialog

OLE DB - First steps

Window Tabs (WndTabs) Add-In for DevStudio



To Heap or not to Heap; That's the Large Object Question?

Introduction to D3DImage

Creating alternate GUI using Vector Art

Comments and Discussions

33 messages have been posted for this article Visit **https://www.codeproject.com/Articles/13985/NET-Based-Add-in-Plug-in-Framework-with-Dynamic-T** to post and view comments on this article, or click **here** to get a print view with messages.