Установка самописных компонентов

Прочитать на сайте

После того, как вы сделали собственные визуальные компоненты, я думаю, вы захотите их использовать в различных проектах, а впоследствии и поделится с миром наработками. Как их установить на панель инструментов в студии, как добавить эту библиотеку в “Add reference…” диалог и как избежать подводных камней в этом деле – все это будет описано в этой статье.

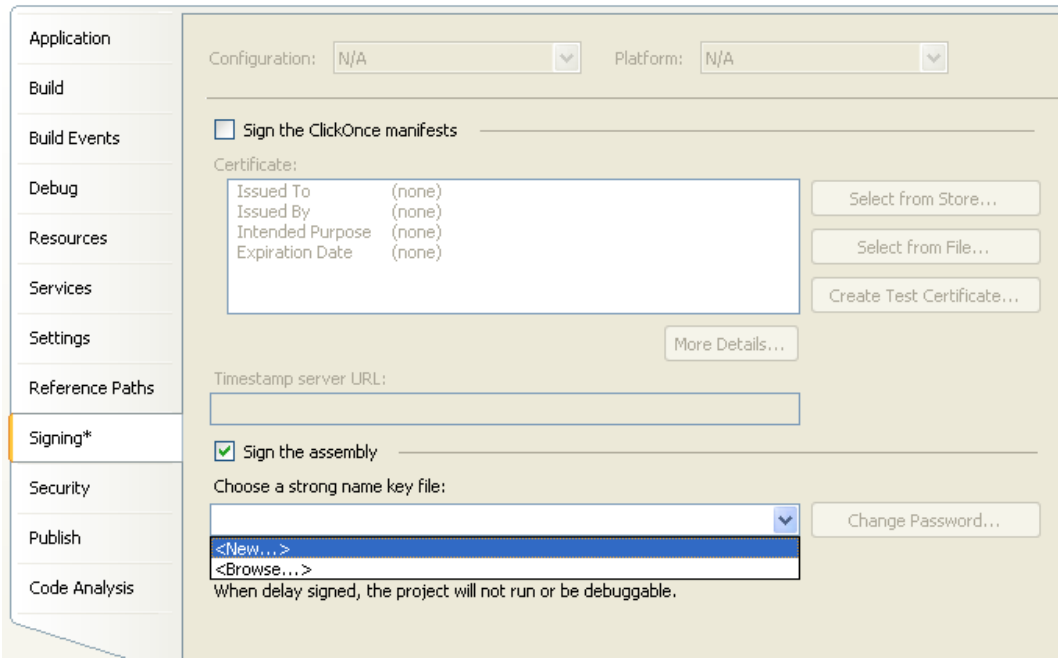
Подготовка сборки (assembly)

Когда у вас есть проект с настроенными компонентами и вы чувствуете что они выглядят и работают точно так как вам хочется – вы, возможно, захотите автоматизировать процесс установки. Для этого потребуется:

1. Подписать сборку;
2. Установить сборку в GAC;
3. Добавить в “Add reference…” диалог;
4. Добавить в панель инструментов студии.

Далее я расскажу как автоматизировать почти все пункты из этого списка, в небольших примерах. Для дальнейшего повествования буду исходить их предположения, что библиотека с компонентами зовется **myCustomControl.dll.**

Вообще, подписать сборку более чем легко. Вызываете контекстное меню для проекта, выбираете пункт Properties. Далее выбираем закладку Signing. Ставим галочку напротив “Sign the assembly” и выпадающий список будет доступен для редактирования.



Выбираем пункт “<New…>”, заполняем поля в появившемся окне и жмем OK. Пересобираем приложение. Вот и все. Только что создали подпись (strong key) для сборки.

Следующим шагом будет…

Добавление в GAC

Думаю, что более подходящим способом будет использование утилиты gacutil.exe. Ее можно вызвать из Visual Studio Command Prompt следующим образом:

gacutil /i “c:\my controls\myCustomControl.dll”

Незабываем, что путь надо писать в кавычках, если он содержит пробелы. Эта операция добавит сборку в GAC. В нашем случае, в программе я использую ключ /if (install force) для того, чтобы переписать сборку, если она уже есть. Код ниже, автоматизирует вызов утилиты:

Gacutil распространяется с .NET SDK. Для версии 3.5 путь `C:\Program Files\Microsoft SDKs\Windows\v6.0A\Bin\gacutil.exe;` 4.0 – `C:\Program Files\Microsoft SDKs\Windows\v7.0A\Bin\gacutil.exe.` Принимая во внимание только что сказанное:

```
public class InstallToGac {
        public void Install(string fullAssemblyLocation) {
            var net35 = @"C:\Program Files\Microsoft
SDKs\Windows\v6.0A\Bin\gacutil.exe";
            var net40 = @"C:\Program Files\Microsoft
SDKs\Windows\v7.0A\Bin\gacutil.exe";

            var value = File.Exists(net35)
                            ? net35
                            : File.Exists(net40) ? net40 : "";

            if (value == "") return;

            var process = new Process();

            process.StartInfo.FileName = value;
            process.StartInfo.Arguments = string.Format("/if \"{0}\"",
fullAssemblyLocation);
            process.StartInfo.CreateNoWindow = true;

            process.Start();
            process.WaitForExit();
        }
    }
```

В процессе выполнения может мелькнуть окно консоли и все. Сборка в GAC. Для того, чтобы удалить вашу библиотеку из GAC, надо выполнить команду с ключом “/u” и добавить “f” если надо эту операцию сделать принудительно.

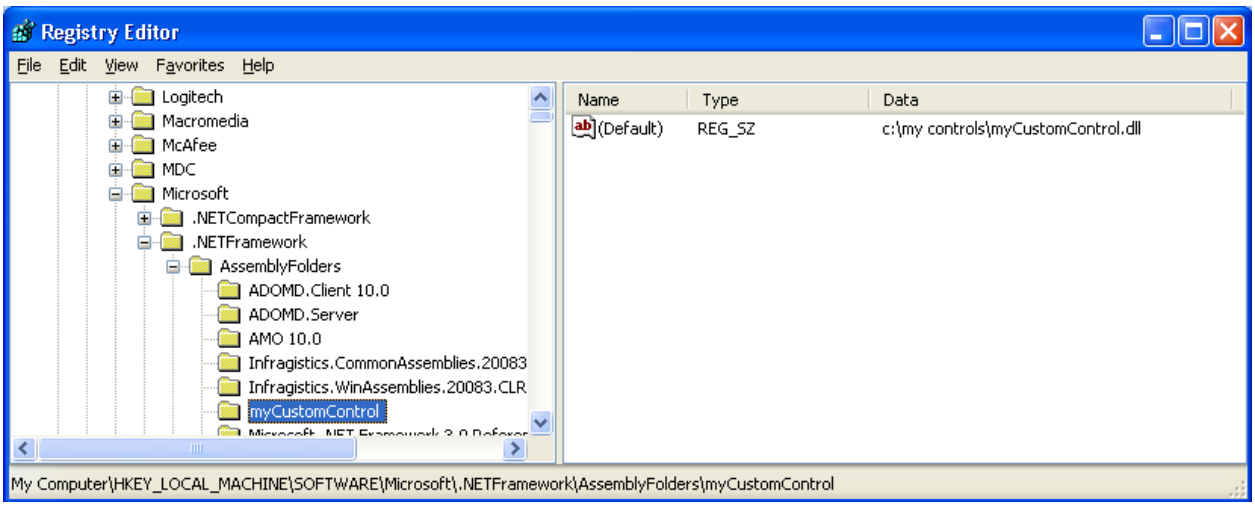
Добавление в диалог “Add reference…” в Visual Studio

Большинство моих коллег думают, что достаточно добавить библиотеку в GAC, как она тут же появится в вышеозначенном окне. К сожалению это не так, добавление в GAC это лишь один из шагов. Необходимо добавить еще ключ в реестре.

Цель:

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\AssemblyFolders.`

По этому пути надо создать ключ с точно таким же именем, как и библиотека, но без dll. После всех действий должно получится как на картинке:



Теперь, как это провернуть с помощью кода:

```
public class InstallToRefLibrary {
        public void Install(string assemblyLocation, string dllName) {
            var p =
@"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\AssemblyFolders";
            var key = string.Format("{0}\\{1}", p, dllName);

            var path = Registry.LocalMachine.OpenSubKey(p, true);

            if (path == null) {
                return;
            }

            path.SetValue(key, assemblyLocation);
            path.Close();
        }
    }
```

Для удаления – просто сотрите ключ из реестра.

Добавление в панель инструментов Visual Studio

Самые странные и уму непостижимые вещи начинаются здесь. Большинство из них из-за COM прошлого студии, некоторые появляются без вообще какого бы то ни было логического объяснения. Ну ладно, начнем с дурацких ограничений, которые могут всплыть.

- Если у вас стоит AnkhSVN, то это может привести к COM ошибкам. Я пробовал создавать закладки для панели инструментов на Win7 (7100) и получал ошибку, пока не снес этот плагин для студии. Под XP все работает отлично даже с AnkhSVN. Описанную проблему можно найти на форуме Infragistics – у них та же проблема с AnkhSVN; и на собственном форуме этого плагина, но без описания почему это происходит и есть ли какое-нибудь лекарство. Так что помните об этом.
- Новая закладка с компонентами может быть установлена только тогда, когда WinForm проект существует в текущем открытом солюшене. Иначе только пустая закладка появится. К счастью, это легко обходится программным путем и я покажу как это сделать.

Перед началом написания кода необходимо добавить проектные ссылки на:

- EnvDTE;
- EnvDTE80;

- EnvDTE90.

От винта!

Установка при запущенной студии

В целом, при запущенной студии, достаточно легко работать. Для этого случая я буду предполагать, что солюшен с WinForm проектом все же существует и загружен. Ну или зачем тогда устанавливать визуальные компоненты? ;)

Первым делом надо определиться с тем, что мы будем устанавливать в новую панель, и новую панель тоже надо как-то идентифицировать. Для этой задачи создадим класс-помощник VSControl. Этот класс будет содержать информацию об имени новой панели, пути к сборке, типе компонента. Так же нам потребуется 2 конструктора: один для панели,

```
public VSControl(string tabGroupName) {
            Control = null;
            AssemblyPath = "";
            TabName = tabGroupName;
            IsToolBoxTab = true;
        }
```

и второй для элементов на нем. Выглядит сложновато, но конечная цель – более легкий код в дальнейшем. Я думаю что тут можно избавиться от `assemblyPath` и использовать взамен assembly.Location.

```
        public VSControl(Assembly assembly, string assemblyPath, string
typeName, string tabName) {
            var fullname = assembly.FullName.Split(new[] { ',' })[0];
            Control = assembly.GetType(fullname + "." + typeName);
            AssemblyPath = assemblyPath;
            TabName = tabName;
            IsToolBoxTab = false;
        }
```

Следующие действия:

1. Получить ссылку на VS;
2. Получить ссылку на панель инструментов;
3. Создать новую панель;
4. Создать набор компонентов.

Все эти действия, на мой взгляд, лучше всего собрать под крышей одного класса DevEnvironment. Этот класс будет иметь только один публичный метод, к примеру, RegisterControls, который будет принимать в качестве параметров массив компонентов и номер версии студии.

```
public class DevEnvironment {
    public static bool RegisterControls(string dteVersion, params VSControl[]
controls) { … }
```

В деталях это все будет:

Получить ссылку на VS

Я предлагаю реализовать это как метод, который будет принимать DTE (главный объект в модели автоматизации студии) (http://msdn.microsoft.com/en-us/library/envdte.dte.aspx) версию и флаг, который будет показывать запущена студия или нет.

```
(DTE2)Marshal.GetActiveObject("VisualStudio.DTE.9.0")
```

Этот код применим для VS2008. Если вы захотите установить свои компоненты в несколько студий, я советую создать нумератор, где будут перечислены все возможные варианты.

```
[Flags]
    public enum DTEVersion {
        None = 0x0000,
        [Description("VisualStudio.DTE.9.0")]
        VS2008 = 0x0001,
        [Description("VisualStudio.DTE.8.0")]
        VS2005 = 0x0002,
        [Description("VisualStudio.DTE.7.0")]
        VS2003 = 0x0004,
        [Description("VisualStudio.DTE.10.0")]
        VS2010 = 0x0008,
    }
```

Использование нумераторов делает код более гибким. Можно не делать проверку на соответствие строк, к тому же `RegisterControls` так же может принимать DTEVersion. Внутри метода `RegisterControls` можно будет написать тогда так:

```
public static void RegisterControls(DTEVersion dteVersion, params VSControl[]
controls) {
            DTE2 dte = null;
            var alreadyCreated = false;

            if ((dteVersion & DTEVersion.VS2008) > DTEVersion.None) {
dte = (DTE2)Marshal.GetActiveObject("VisualStudio.DTE.9.0");

// other code

 …  }
          if ((dteVersion & DTEVersion.VS2010) > DTEVersion.None) {
dte = (DTE2)Marshal.GetActiveObject("VisualStudio.DTE.10.0");

// other code
…  }

        }
```

Появление в коде строковых переменных весьма странно, когда уже есть нумератор с описаниями. И вот способ как это можно использовать:

```
internal static string GetEnumDescription(Enum value) {
            var fieldInfo = value.GetType().GetField(value.ToString());

            var attributes =
(DescriptionAttribute[])fieldInfo.GetCustomAttributes(typeof(DescriptionAttri
bute), false);
            return (attributes.Length > 0) ? attributes[0].Description :
value.ToString();
        }
```

И тогда получение ссылки на студию

```
dte = (DTE2)Marshal.GetActiveObject("VisualStudio.DTE.9.0");
```

можно переписать следующим образом:

```
dte = GetDesignTimeEnvironment(DTEVersion.VS2008, ref alreadyCreated);
```

в новом методе мы проверим еще кое-что. Когда вы получаете DTE2, это занимает некоторое время для того, чтобы COM-метод вызвался и вернул результат. После этого, было бы неплохо проверить, что ссылка на студию действительно получена, и попробовать переключится на какой-нибудь базовый экран, к примеру, на окно свойств.

```
public static DTE2 GetDesignTimeEnvironment(DTEVersion dteVersion, ref bool
alreadyCreated) {
            alreadyCreated = false;
            var progID = GetEnumDescription(dteVersion);
            DTE2 result;
            try {
                result = (DTE2)Marshal.GetActiveObject(progID);

                Thread.Sleep(5000);
                try {
                    result.ExecuteCommand("View.PropertiesWindow", "");
                    alreadyCreated = true;
                } catch //You have to open a new VS.Net
                {
                    result = null;
                }
            } catch //There is no open VS.Net
            {
                result = null;
            }
            return result;
        }
```

Осталось проверить результат на null и если не null, то создаем новую панель.

Получение указателя на панель инструментов

Вы будете удивлены, но это очень легко! Всего 2 строчки кода.

```
var toolbox = dte.Windows.Item(Constants.vsWindowKindToolbox);
var tabs = ((ToolBox)toolbox.Object).ToolBoxTabs;
```

Создание новой панели

Так как компоненты и новая панель уже должны быть обернуты в VSControl – новый метод должен принимать массив этого класса. Ищем в массиве объявления панелей и создаем их.

```
internal static void RegisterControls(DTE2 dte, bool alreadyCreatedDTE,
VSControl[] controls) {
            var toolbox = dte.Windows.Item(Constants.vsWindowKindToolbox);
            var tabs = ((ToolBox)toolbox.Object).ToolBoxTabs;

            controls
                .ToList()
                .FindAll(i => i.IsToolBoxTab)
                .ForEach(i => {
                            if (GetToolBoxTab(tabs, i.TabName) == null)
                                tabs.Add(i.TabName);
                        });
}
```

```
private static ToolBoxTab3 GetToolBoxTab(ToolBoxTabs tabs, string tabName) {
   foreach (ToolBoxTab3 tab in tabs) {
     if (smenglish.CompareInfo.Compare(tab.Name, tabName,
CompareOptions.IgnoreCase) == 0)
                    return tab;
            }

            return null;
        }
```

И поле класса:

```
internal static CultureInfo smenglish =
CultureInfo.CreateSpecificCulture("en");
```

Создание новых компонентов

Теперь, когда созданы панели для новых компонентов, время создать реальные объекты. Для этого надо активировать соответствующую панель и создать элемент. Тоже не тяжело. Добавляем этот код после создания панелей.

```
foreach (var control in controls) {
                var tab = GetToolBoxTab(tabs, control.TabName);
                if (tab != null && !control.IsToolBoxTab) {
                    tab.Activate();
                    tab.ToolBoxItems.Add("anyName", control.Control,
vsToolBoxItemFormat.vsToolBoxItemFormatDotNETComponent);
                }
            }
```

Здесь есть небольшие нюансы: когда добавляете новый элемент с помощью кода `tab.ToolBoxItems.Add`, второй параметр – объект. Если передать строку, то она будет распознана как путь до сборки и все публичные компоненты оттуда будут добавлены. Если передать тип, то только этот объект будет добавлен.

Когда программа закончит добавлять компоненты к панели(ям), надо будет закрыть соединение с DTE2.

```
dte.Quit();
```

Вот и всё, для случая запущенной студии.

Установка при незапущенной студии

Думаю, что вам не захочется просить пользователя запустить Visual Studio и создать проект WinForm, перед тем как запустить установщик. И далее будет описано, как сделать это все автоматически.

Для начала нам надо создать новый экземпляр студии, для того чтобы продолжить работу. Думаю, что лучшим местом для этого куска кода будет метод GetDesignTimeEnvironment.

```
if (result == null) {
                //Open a new VS.Net
                var type = Type.GetTypeFromProgID(progID);
                result = (DTE2) Activator.CreateInstance(type, true);
    }
```

Как я уже ранее говорил, необходимо создать WinForm проект для того, чтобы все нормально установить. И это возможно сделать кодом тоже. Открываем метод RegisterControls и добавляем в начало.

```
if (!alreadyCreatedDTE) {
    var tmpFile = Path.GetFileNameWithoutExtension(Path.GetTempFileName());
    var tmpDir = string.Format("{0}{1}", Path.GetTempPath(), tmpFile);
    var solution = dte.Solution as Solution2;
    var templatePath = solution.GetProjectTemplate("WindowsApplication.zip",
"CSharp");
    solution.AddFromTemplate(templatePath, tmpDir, "dummyproj", false);
            }
```

На мой взгляд код тут самодокументирующийся и в комментариях не нуждается. Но не забываем закрыть солюшен перед концом программы.

```
dte.Solution.Close(false);
```

Параметр является ответом на вопрос, сохранять проект или нет.

Теперь новые компоненты будут установлены независимо от того, запущена студия или нет. К сожалению COM ошибки могут нарушить всю идиллию. Во время разработки я получал тонны ошибок, которые сообщали, что COM-объект все еще занят и не может быть вызван. Поиски в интернете дали ответ на то, как это избежать.

Как избежать COM ошибок

Основная идея в том, чтобы отфильтровывать сообщения от COM-объектов и получать только те, на которые мы подпишемся.

Необходимо объявить интерфейс `IOleMessageFilter`

```
[ComImport, Guid("00000016-0000-0000-C000-000000000046"),
     InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
    internal interface IOleMessageFilter {
        [PreserveSig]
        int HandleInComingCall(
            int dwCallType,
            IntPtr hTaskCaller,
            int dwTickCount,
            IntPtr lpInterfaceInfo);

        [PreserveSig]
        int RetryRejectedCall(
            IntPtr hTaskCallee,
            int dwTickCount,
            int dwRejectType);

        [PreserveSig]
        int MessagePending(
            IntPtr hTaskCallee,
            int dwTickCount,
            int dwPendingType);
    }
```

И сделать реализацию к нему.

```
public class MessageFilter : IOleMessageFilter {
        //
        // Class containing the IOleMessageFilter
```

```
        // thread error-handling functions.

        // Start the filter.
        public static void Register() {
            IOleMessageFilter newFilter = new MessageFilter();
            IOleMessageFilter oldFilter = null;
            CoRegisterMessageFilter(newFilter, out oldFilter);
        }

        // Done with the filter, close it.
        public static void Revoke() {
            IOleMessageFilter oldFilter = null;
            CoRegisterMessageFilter(null, out oldFilter);
        }

        //
        // IOleMessageFilter functions.
        // Handle incoming thread requests.
        int IOleMessageFilter.HandleInComingCall(int dwCallType, IntPtr
hTaskCaller, int dwTickCount, IntPtr lpInterfaceInfo) {
            //Return the flag SERVERCALL_ISHANDLED.
            return 0;
        }

        // Thread call was rejected, so try again.
        int IOleMessageFilter.RetryRejectedCall(IntPtr hTaskCallee, int
dwTickCount, int dwRejectType) {
            if (dwRejectType == 2)
                // flag = SERVERCALL_RETRYLATER.
            {
                // Retry the thread call immediately if return >=0 &
                // <100.
                return 99;
            }
            // Too busy; cancel call.
            return -1;
        }

        int IOleMessageFilter.MessagePending(IntPtr hTaskCallee, int
dwTickCount, int dwPendingType) {
            //Return the flag PENDINGMSG_WAITDEFPROCESS.
            return 2;
        }

        // Implement the IOleMessageFilter interface.
        [DllImport("Ole32.dll")]
        private static extern int
            CoRegisterMessageFilter(IOleMessageFilter newFilter, out
IOleMessageFilter oldFilter);
    }
```

В итоге должно получиться что-то в духе:

```
MessageFilter.Register();
RegisterControls(dte, alreadyCreated, controls);
MessageFilter.Revoke();
```

С этим кодом все у вас будет в ажуре! =)

Я тестировал на семерке, XP и все работало замечательно. Я надеюсь получить ваши комментарии, вопросы, предложения. Кстати да, в процессе написания я заметил места, где можно улучшить код, но оставил его до лучших времен.

Source code

Hard’n’heavy!