عنوان: **آشنایی با M.A.F - قسمت اول**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۹/۳۰ ۲۱:۳۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MAF

در طی چند مقاله قصد بررسی نحوه‌ی تولید برنامه‌های توسعه پذیر (extensible) را با استفاده از plug-ins و یا add-ins داریم.

افزونه‌ها عموما در سه گروه قرار می‌گیرند:
الف) افزونه، سرویسی را به هاست ارائه می‌دهد. برای مثال یک میل سرور نیاز به افزونه‌هایی برای ویروس یابی یا فیلتر کردن هرزنامه‌ها دارد؛ یا یک برنامه پردازش متنی نیاز به افزونه‌ای جهت بررسی غلط‌های املایی می‌تواند داشته باشد و یا یک مرورگر وب می‌تواند با کمک افزونه‌ها قابلیت‌های پیش فرض خود را به شدت توسعه و افزایش دهد (نمونه‌ی بارز آن فایرکس است که عمده‌ترین دلیل اقبال عمومی به آن سهولت توسعه پذیری آن می‌باشد).
ب) در گروه دوم، هاست، رفتار مشخصی را ارائه داده و سپس افزونه بر اساس آن، نحوه‌ی عملکرد هاست را مشخص می‌کند. در این حالت هاست است که سرویسی را به افزونه ارائه می‌دهد. نمونه‌ی بازر آن افزونه‌های آفیس هستند که امکان اتوماسیون فرآیندهای مختلف آن‌را میسر می‌سازند. به این صورت امکان توسعه‌ی یک برنامه به شکلی که در طراحی اولیه آن اصلا انتظار آن نمی‌رفته وجود خواهد داشت. همچنین در اینجا نیازی به داشتن سورس کد برنامه‌ی اصلی نیز نمی‌باشد.
ج) گروه سوم افزونه‌ها تنها از هاست جهت نمایش خود استفاده کرده و عملا استفاده‌ی خاصی از هاست ندارد. برای مثال نوار ابزاری که خود را به windows explorer متصل می‌کند و تنها از آن جهت نمایش خود بهره می‌جوید.

در حال حاضر حداقل دو فریم ورک عمده جهت انجام این‌کار و تولید افزونه‌ها برای دات نت فریم ورک مهیا است:
الف) managed addin framework یا MAF
ب) managed extensibility framework یا MEF

فضای نام جدیدی به دات نت فریم ورک سه و نیم به نام System.AddIn اضافه شده است که به آن Managed AddIn Framework یا MAF نیز اطلاق می‌شود. از این فریم ورک در VSTO (تولید افزونه برای مجموعه‌ی آفیس) توسط خود مایکروسافت استفاده شده است.

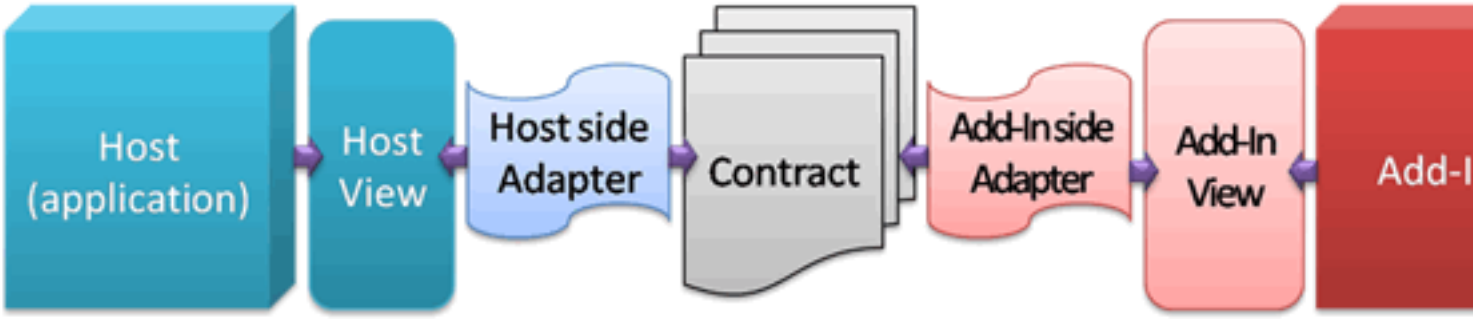
فریم ورک توسعه‌ی افزونه‌های مدیریت شده در دات نت فریم ورک سه و نیم، مزایای زیر را در اختیار ما خواهد گذاشت:
- امکانات load و unload افزونه‌های تولید شده
- امکان تغییر افزونه‌ها در زمان اجرای برنامه اصلی بدون نیاز به بستن آن
- ارائه‌ی محیطی ایزوله با ترسیم مرزی بین افزونه و برنامه اصلی
- مدیریت طول عمر افزونه
- مدیریت سازگاری با نگارش‌های قبلی و یا بعدی یک افزونه
- امکانات به اشتراک گذاری افزونه‌ها با برنامه‌های دیگر
- تنظیمات امنیتی و مشخص سازی سطح دسترسی افزونه‌ها
و ...

یک راه حل مبتنی بر MAF می‌تواند شامل 7 پروژه باشد (که به روابط تعریف شده در آن pipeline هم گفته می‌شود):

Host : همان برنامه‌ی اصلی است که توسط یک سری افزونه، توسعه یافته است.
Host View : بیانگر انتظارات هاست از افزونه‌ها است. به عبارت دیگر افزونه‌ها باید موارد لیست شده در این پروژه را پیاده سازی کنند.
Host Side Adapter : پل ارتباطی Host View و پروژه‌ی Contract است.
Contract: اینترفیسی است که کار برقراری ارتباط بین Host و افزونه‌ها را برعهده دارد.

Add-In Side Adapter : پل ارتباطی بین Add-In View و Contract است.
Add-In View : حاوی متدها و اشیایی است که جهت برقراری ارتباط با هاست از آن‌ها استفاده می‌شود.
Add-In : اسمبلی است که توسط هاست جهت توسعه‌ی قابلیت‌های خود بارگذاری می‌شود (به آن Plug-In ، Extension ، Add-On و Snap-In هم گفته می‌شود).

هدف از این جدا سازی‌ها ارائه‌ی راه حل loosely-coupledایی است که امکان ایزوله سازی، اعمال شرایط امنیتی ویژه و همچنین کنترل نگارش‌های مختلف را تسهیل می‌بخشد و این امر با استفاده از interface های معرفی شده میسر گردیده است. این pipeline از قسمت‌های ذیل تشکیل می‌شود:




قرار داد یا Contract
برای تولید یک افزونه نیاز است تا بین هاست و افزونه قراردادی بسته شود. با توجه به استفاده از MAF ، روش تعریف این قرار داد برای مثال در یک افزونه‌ی مترجم به صورت زیر باید باشد:

```
[AddInContract]
public interface ITranslator : IContract
{
   string Translate(string input);
}
```

استفاده از ویژگی AddInContract و پیاده سازی اینترفیس IContract جزو مراحل کاری استفاده از MAF است. MAF هنگام تولید پویای pipeline ذکر شده به دنبال ویژگی AddInContract می‌گردد. این موارد در فضای نام System.AddIn.Pipeline تعریف شده‌اند.

دیدگاه‌ها یا Views
دیدگاه‌ها کدهایی هستند که کار تعامل مستقیم بین افزونه و هاست را بر عهده دارند. هاست یا افزونه هر کدام می‌توانند دیدگاه خود را نسبت به قرار داد بسته شده داشته باشند. این موارد نیز همانند قرار داد در اسمبلی‌های مجزایی نگهداری می‌شوند.

دیدگاه هاست نسبت به قرار داد:

```
public abstract class TranslatorHostView
{
   public abstract string Translate(string input);
}
```

دیدگاه افزونه نسبت به قرار داد:

```
[AddInBase]
public abstract class TranslatorHostView
{
   public abstract string Translate(string input);
}
```

هر دو کلاس فوق بر اساس قرار موجود بنا می‌شوند اما وابسته به آن نیستند. به همین جهت به صورت کلاس‌هایی abstract تعریف شده‌اند. در سمت افزونه، کلاس تعریف شده دیدگاه آن با کلاس دیدگاه سمت هاست تقریبا یکسان می‌باشد؛ اما با ویژگی AddInBase تعریف شده در فضای نام System.AddIn.Pipeline مزین گردیده است.

وفق دهنده‌ها یا Adapters

آخرین قسمت pipeline ، وفق دهنده‌ها هستند که کار آن‌ها اتصال قرار داد به دیدگاه‌ها است و توسط آن مدیریت طول عمر افزونه و همچنین تبدیل اطلاعات بین قسمت‌های مختلف انجام می‌شود. شاید در نگاه اول وجود آن‌ها زائد به نظر برسد اما این جدا سازی کدها سبب تولید افزونه‌هایی خواهد شد که به نگارش هاست و برنامه اصلی وابسته نبوده و بر عکس (version tolerance). به دو کلاس زیر دقت نمائید:

کلاس زیر با ویژگی [HostAdapter] تعریف شده در فضای نام System.AddIn.Pipeline، مزین شده است و کار آن اتصال HostView به Contract می‌باشد. برای این منظور TranslatorHostView ایی را که پیشتر معرفی کردیم باید پیاده سازی نماید. علاوه بر این با ایجاد وهله‌ای از کلاس ContractHandle ، کار مدیریت طول عمر افزونه را نیز می‌توان انجام داد.

```
[HostAdapter]
public class TranslatorHostViewToContract : TranslatorHostView
{
   ITranslator _contract;
   ContractHandle _lifetime;

   public TranslatorHostViewToContract(ITranslator contract)
   {
       _contract = contract;
       _lifetime = new ContractHandle(contract);
   }

   public override string Translate (string inp)
   {
       return _contract.Translate(inp);
   }
}
```

کلاس سمت افزونه نیز بسیار شبیه قسمت قبل است و کار آن اتصال AddInView به Contract می‌باشد که با پیاده سازی ContractBase و Itranslator صورت خواهد گرفت. همچنین این کلاس به ویژگی AddInAdapter مزین گردیده است.

```
[AddInAdapter]
public class TranslatorAddInViewToContract : ContractBase, ITranslator
{
   TranslatorAddInView _view;

   public TranslatorAddInViewToContract(TranslatorView view)
   {
       _view = view;
   }

   public string Translate(string inp)
   {
       return _view.Translate(inp);
   }
}
```

قسمت عمده‌ای از این کدها تکراری است. جهت سهولت تولید این کلاس‌ها و پروژه‌های مرتبط، تیم مربوطه برنامه‌ای را به نام

pipeline builder ارائه داده است که از آدرس زیر قابل دریافت است:

http://www.codeplex.com/clraddins

این برنامه با دریافت اسمبلی مربوط بهcontract ، کار ساخت خودکار کلاس‌های adapters و views را انجام خواهد داد.

ایجاد افزونه

پس از ساخت قسمت‌های مختلف pipeline ، اکنون می‌توان افزونه را ایجاد نمود. هر افزونه باید add-in view را پیاده سازی کرده و با ویژگی AddIn مزین شود. برای مثال:

```
[AddIn("GoogleTranslator", Description="Universal translator",
Version="1.0.0.0", Publisher="YourName")]
public class GoogleAddIn : TranslatorAddInView
{
   public string Translate(string input)
   {
       ...
   }
}
```

ادامه دارد ....

عنوان: **آشنایی با M.A.F - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۱۰/۰۲ ۱۶:۱۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MAF

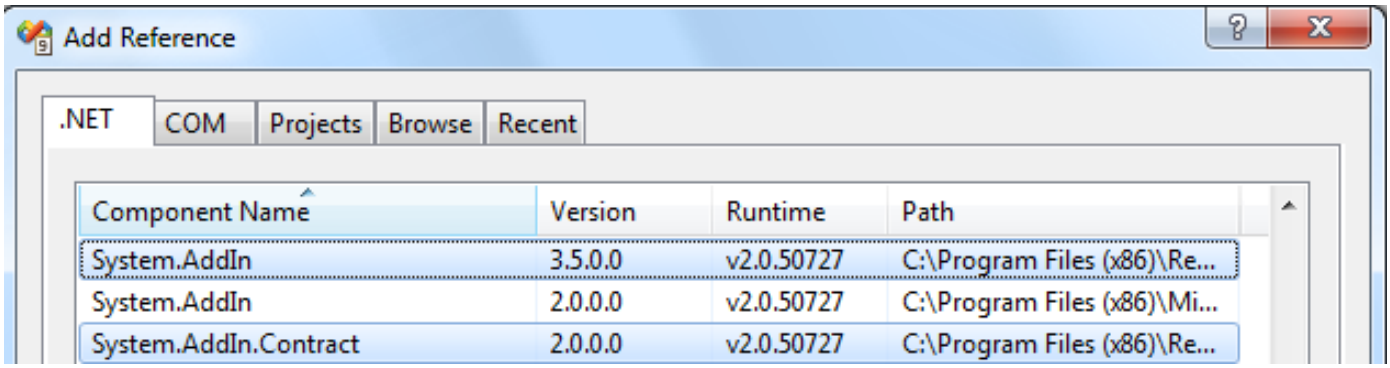
قسمت قبل

بیشتر آشنایی با یک سری از اصطلاحات مرتبط با فریم ورک MAF بود و همچنین نحوه‌ی کلی استفاده از آن. در این قسمت یک مثال ساده را با آن پیاده سازی خواهیم کرد و فرض قسمت دوم بر این است که افزونه‌ی Visual Studio Pipeline Builder را نیز نصب کرده‌اید.

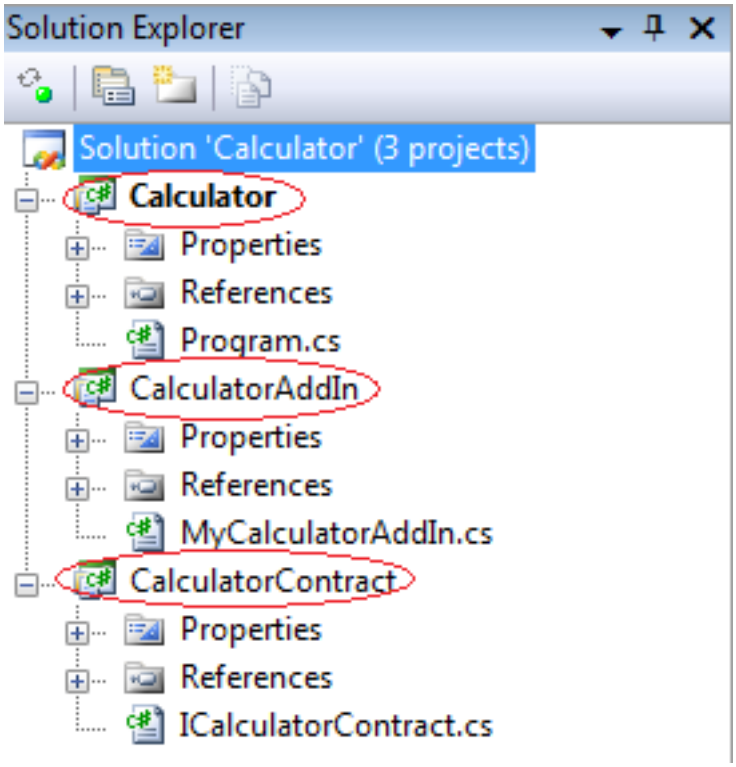
یک نکته پیش از شروع:
- اگر افزونه‌ی Visual Studio Pipeline Builder پس از نصب به منوی Tools اضافه نشده است، یک پوشه‌ی جدید را به نام Addins در مسیر Documents\Visual Studio 2008 ایجاد کرده و سپس فایل‌های آن‌را در این مسیر کپی کنید.

ساختار اولیه یک پروژه MAF

- پروژه‌هایی که از MAF استفاده می‌کنند، نیاز به ارجاعاتی به دو اسمبلی استاندارد System.AddIn.dll و System.AddIn.Contract.dll دارند (مطابق شکل زیر):

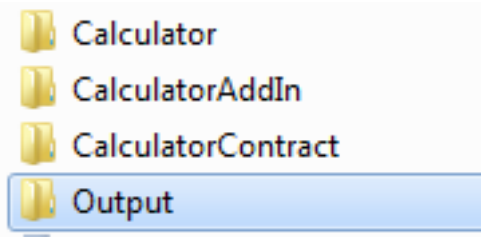



- ساختار آغازین یک پروژه MAF از سه پروژه تشکیل می‌شود که توسط افزونه‌ی Visual Studio Pipeline Builder به 7 پروژه بسط خواهد یافت.
این سه پروژه استاندارد آغازین شامل موارد زیر هستند:

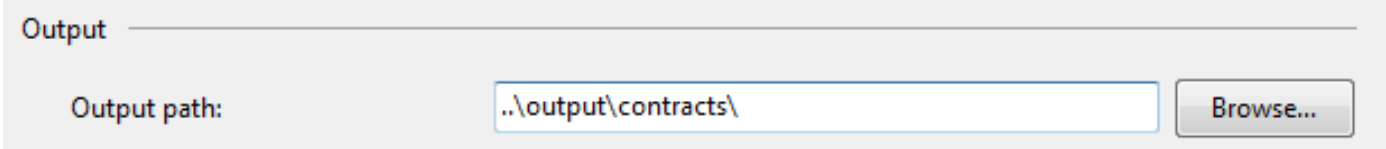



- هاست: همان برنامه‌ی اصلی که قرار است از افزونه استفاده کند.
- قرار داد: نحوه‌ی تعامل هاست و افزونه در این پروژه تعریف می‌شود. (یک پروژه از نوع class library)
- افزونه: کار پیاده سازی قرار داد را عهده دار خواهد شد. (یک پروژه از نوع class library)

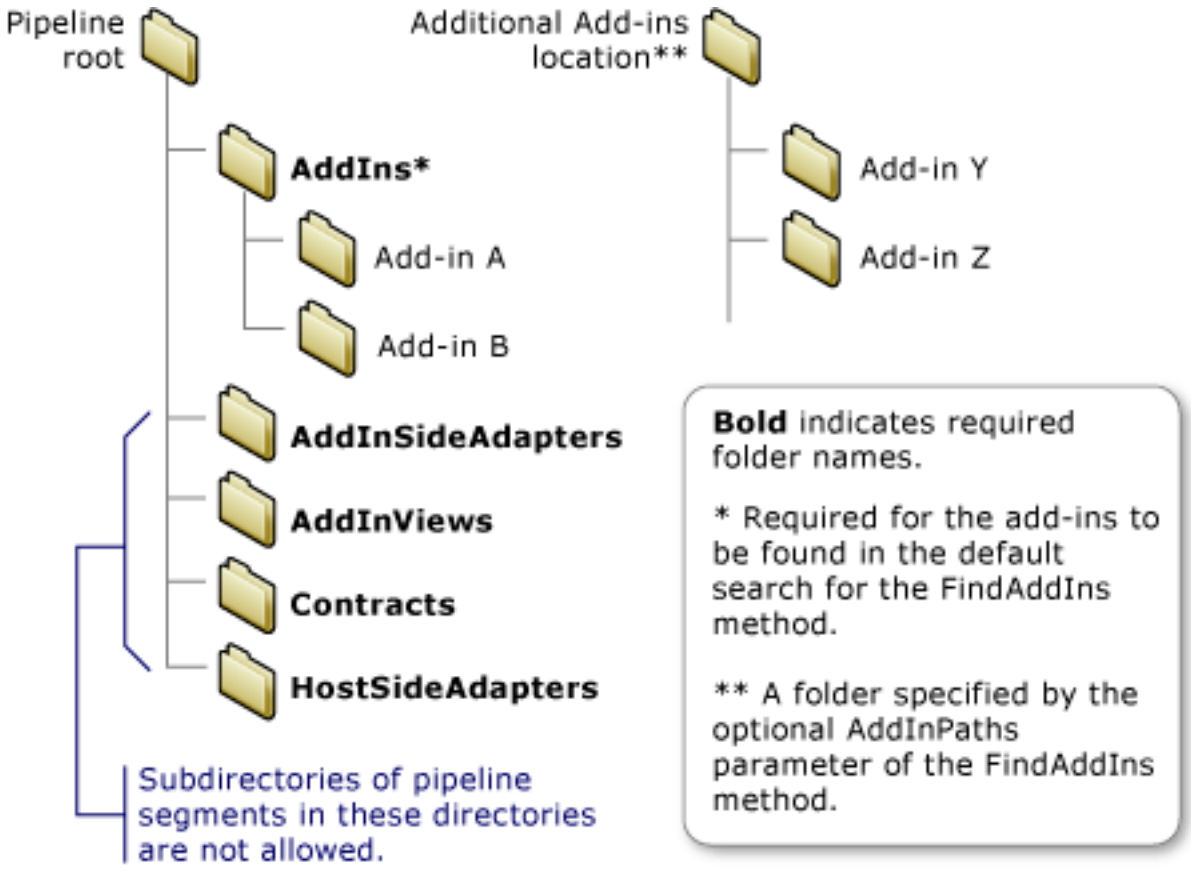
- همچنین مرسوم است جهت مدیریت بهتر خروجی‌های حاصل شده یک پوشه Output را نیز به این solution اضافه کنند:




اکنون با توجه به این محل خروجی، به خواص Build سه پروژه موجود مراجعه کرده و مسیر Build را اندکی اصلاح خواهیم کرد (هر سه مورد بهتر است اصلاح شوند)، برای مثال:



نکته‌ی مهم هم اینجا است که خروجی host باید به ریشه این پوشه تنظیم شود و سایر پروژه‌ها هر کدام خروجی خاص خود را در پوشه‌ای داخل این ریشه باید ایجاد کنند.




تا اینجا قالب اصلی کار آماده شده است. قرارداد ما هم به شکل زیر است (ویژگی AddInContract آن نیز نباید فراموش شود):

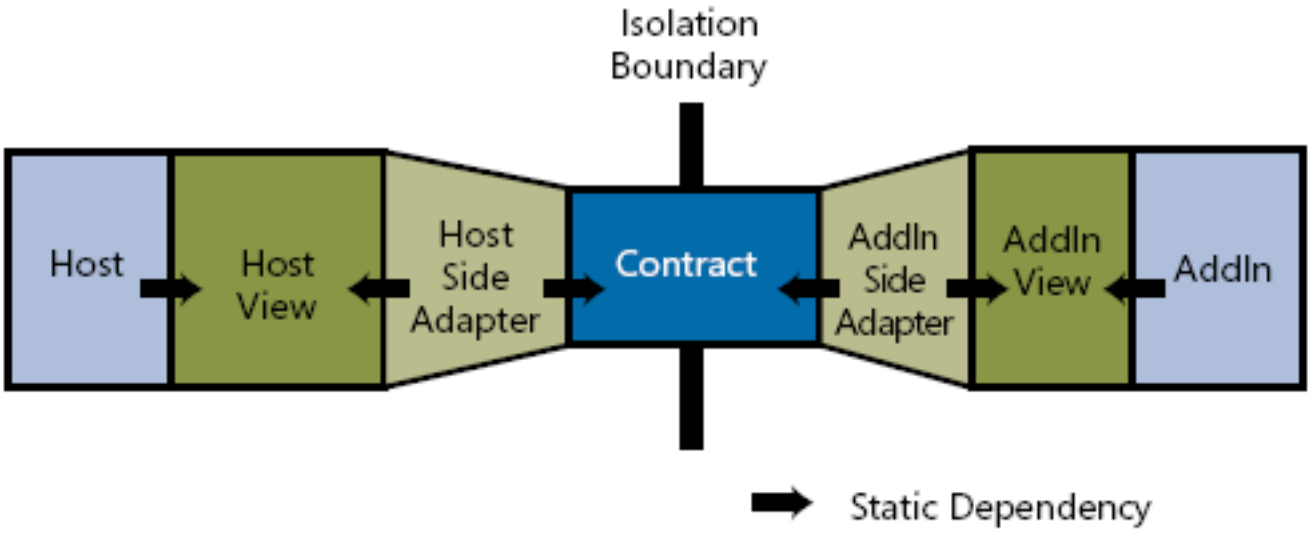
```
using System.AddIn.Pipeline;
using System.AddIn.Contract;

namespace CalculatorConract
{
  [AddInContract]
  public interface ICalculatorContract : IContract
  {
      double Operate(string operation, double a, double b);
  }
}
```

به عبارت دیگر برنامه‌ای محاسباتی داریم (هاست) که دو عدد double را در اختیار افزونه‌های خودش قرار می‌دهد و سپس این افزونه‌ها یک عملیات ریاضی را بر روی آن‌ها انجام داده و خروجی را بر می‌گردانند. نوع عملیات توسط آرگومان operation مشخص می‌شود. این آرگومان به کلیه افزونه‌های موجود ارسال خواهد شد و احتمالا یکی از آن‌ها این مورد را پیاده سازی کرده است. در غیر اینصورت یک استثنای عملیات پیاده سازی نشده صادر می‌شود.

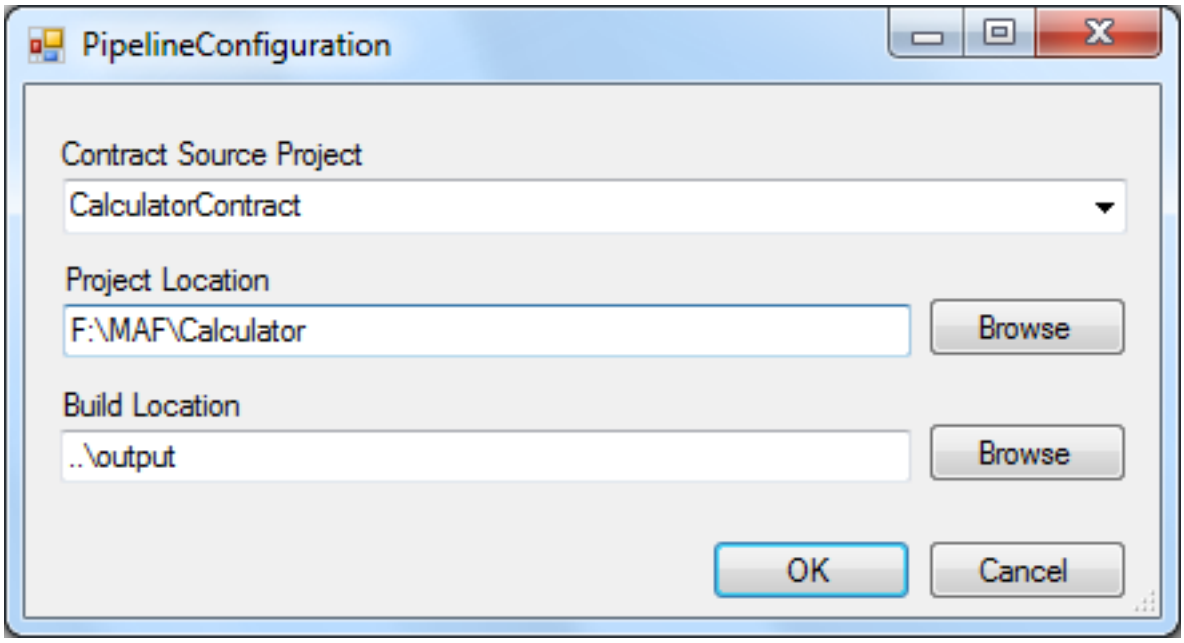
البته روش بهتر طراحی این افزونه، اضافه کردن متد یا خاصیتی جهت مشخص کردن نوع و یا انواع عملیات پشتیبانی شده توسط افزونه است که جهت سادگی این مثال، به این طراحی ساده اکتفا می‌شود.

ایجاد pipeline

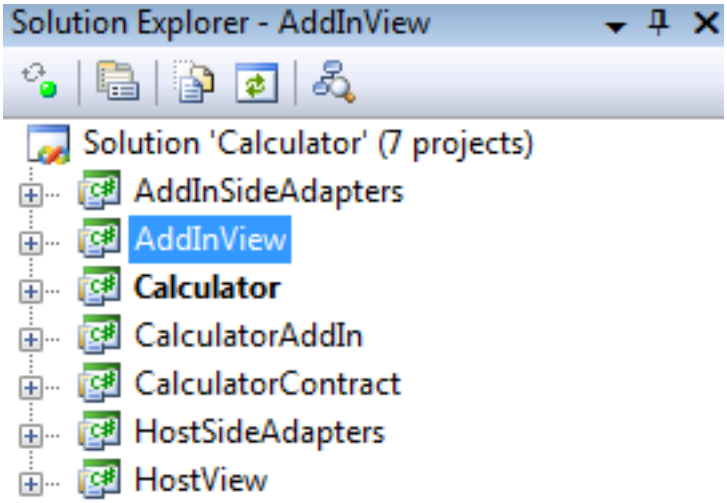
اگر قسمت قبل را مطالعه کرده باشید، یک راه حل مبتنی بر MAF از 7 پروژه تشکیل می‌شود که عمده‌ترین خاصیت آن‌ها مقاوم کردن سیستم در مقابل تغییرات نگارش قرارداد است. در این حالت اگر قرار داد تغییر کند، نه هاست و نه افزونه‌ی قدیمی، نیازی به تغییر در کدهای خود نخواهند داشت و این پروژه‌های میانی هستند که کار وفق دادن (adapters) نهایی را برعهده می‌گیرند.



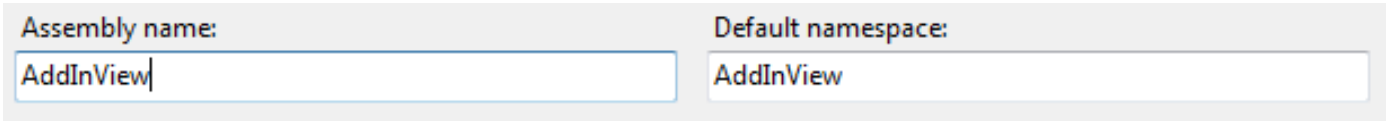

برای ایجاد خودکار View ها و همچنین Adapters ، از افزونه‌ی Visual Studio Pipeline Builder که پیشتر معرفی شد استفاده خواهیم کرد.



سه گزینه‌ی آن هم مشخص هستند. نام پروژه‌ی قرارداد، مسیر پروژه‌ی هاست و مسیر خروجی نهایی معرفی شده. پیش از استفاده از این افزونه نیاز است تا یکبار solution مورد نظر کامپایل شود. پس از کلیک بر روی دکمه‌ی OK، پروژه‌های ذکر شده ایجاد خواهند شد:




پس از ایجاد این پروژه‌ها، نیاز به اصلاحات مختصری در مورد نام اسمبلی و فضای نام هر کدام می‌باشد؛ زیرا به صورت پیش فرض هر کدام به نام template نامگذاری شده‌اند:




پیاده سازی افزونه

قالب کاری استفاده از این فریم ورک آماده است. اکنون نوبت به پیاده سازی یک افزونه می‌باشد. به پروژه AddIn مراجعه کرده و ارجاعی را به اسمبلی AddInView خواهیم افزود. به این صورت افزونه‌ی ما به صورت مستقیم با قرارداد سروکار نداشته و ارتباطات، در راستای همان pipeline تعریف شده، جهت مقاوم شدن در برابر تغییرات صورت می‌گیرد:

```
using System;
using CalculatorConract.AddInViews;
using System.AddIn;

namespace CalculatorAddIn
{
   [AddIn]
  public class MyCalculatorAddIn : ICalculator
  {
      public double Operate(string operation, double a, double b)
      {
          throw new NotImplementedException();
```

```
      }
  }
}
```

در اینجا افزونه‌ی ما باید اینترفیس ICalculator مربوط به AddInView را پیاده سازی نماید که برای مثال خواهیم داشت:

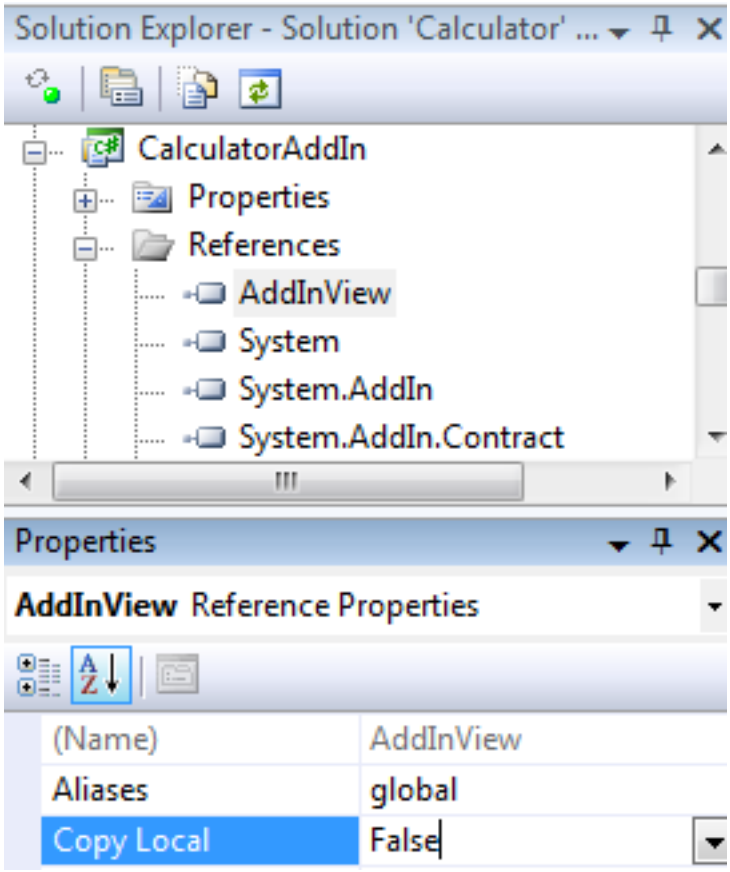
```
using System;
using CalculatorConract.AddInViews;
using System.AddIn;

namespace CalculatorAddIn
{
  [AddIn("افزونه یک", Description = "توضیحات", Publisher = "نویسنده", Version = "نگارش یک")]
  public class MyCalculatorAddIn : ICalculator
  {
      public double Operate(string operation, double a, double b)
      {
          switch (operation)
          {
              case "+":
                  return a + b;
              case "-":
                  return a - b;
              case "*":
                  return a * b;
              default:
                  throw new NotSupportedException("عملیات مورد نظر توسط این افزونه پشتیبانی نمی‌شود");
          }
      }
  }
}
```

همانطور که در قسمت قبل نیز ذکر شد، این کلاس باید با ویژگی AddIn مزین شود که توسط آن می‌توان توضیحاتی در مورد نام ، نویسنده و نگارش افزونه ارائه داد.

استفاده از افزونه‌ی تولید شده

هاست برای استفاده از افزونه‌هایی با قرارداد ذکر شده، مطابق pipeline پروژه، نیاز به ارجاعی به اسمبلی HostView دارد و در اینجا نیز هاست به صورت مستقیم با قرارداد کاری نخواهد داشت. همچنین هاست هیچ ارجاع مستقیمی به افزونه‌ها نداشته و بارگذاری و مدیریت آن‌ها به صورت پویا انجام خواهد شد.

نکته‌ی مهم!
در هر دو ارجاع به HostView و یا AddInView باید خاصیت Copy to local به false تنظیم شود، در غیر اینصورت افزونه‌ی شما بارگذاری نخواهد شد.




پس از افزودن ارجاعی به HostView، نمونه‌ای از استفاده از افزونه‌ی تولید شده به صورت زیر می‌تواند باشد که توضیحات مربوطه به صورت کامنت آورده شده است:

```
using System;
using System.AddIn.Hosting;
using CalculatorConract.HostViews;

namespace Calculator
{
  class Program
  {
      private static ICalculator _calculator;

      static void doOperation()
      {
          Console.WriteLine("1+2: {0}", _calculator.Operate("+", 1, 2));
      }

      static void Main(string[] args)
      {
          //مسیر پوشه ریشه مربوطه به خط لوله افزونه‌ها
          string path = Environment.CurrentDirectory;

          //مشخص سازی مسیر خواندن و کش کردن افزونه‌ها
          AddInStore.Update(path);

          //یافتن افزونه‌هایی سازگار با شرایط قرارداد پروژه
          //در اینجا هیچ افزونه‌ای بارگذاری نمی‌شود
          var addIns = AddInStore.FindAddIns(typeof(ICalculator), path);

          //اگر افزونه‌ای یافت شد
          if (addIns.Count > 0)
          {
              var addIn = addIns[0]; //استفاده از اولین افزونه
              Console.WriteLine("1st addIn: {0}", addIn.Name);

              //فعال سازی افزونه و همچنین مشخص سازی سطح دسترسی آن
```

```
            _calculator = addIn.Activate<ICalculator>(AddInSecurityLevel.Intranet);

            //یک نمونه از استفاده آن
            doOperation();
        }

        Console.WriteLine("Press a key...");
        Console.ReadKey();
    }
  }
}
```

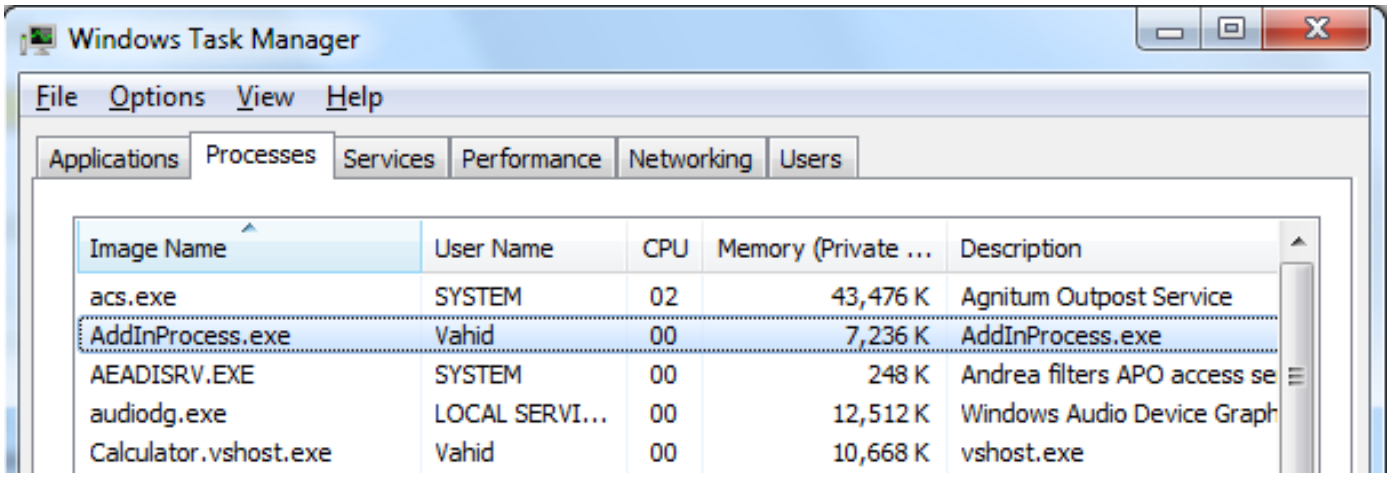
چند نکته جالب توجه در مورد قابلیت‌های ارائه شده:
- مدیریت load و unload پویا
- امکان تعریف سطح دسترسی و ویژگی‌های امنیتی اجرای یک افزونه
- امکان ایزوله سازی پروسه اجرای افزونه از هاست (در ادامه توضیح داده خواهد شد)
- مقاوم بودن پروژه به نگارش‌های مختلف قرارداد

اجرای افزونه در یک پروسه مجزا

حتما با امکانات مرورگر کروم و یا IE8 در مورد اجرای هر tab آن‌ها در یک پروسه‌ی مجزا از پروسه اصلی هاست مطلع هستید. به این صورت پروسه‌ی هاست از رفتار tab ها محافظت می‌شود، همچنین پروسه‌ی هر tab نیز از tab دیگر ایزوله خواهد بود. یک چنین قابلیتی در این فریم ورک نیز پیش بینی شده است.

```
//فعال سازی افزونه و همچنین مشخص سازی سطح دسترسی آن
//همچنین جدا سازی پروسه اجرایی افزونه از هاست
   _calculator = addIn.Activate<ICalculator>(
                 new AddInProcess(),
                 AddInSecurityLevel.Intranet);
```

در این حالت اگر پس از فعال شدن افزونه، یک break point قرار دهیم و به task manager ویندوز مراجعه نمائیم، پروسه‌ی مجزای افزونه قابل مشاهده است.




برای مطالعه بیشتر + ، + ، + و +