BASTA! ON TOUR

Rainer Stropek | software architects gmbh

Entwicklung modularer Anwendungen mit C# und dem Managed Extensibility Framework (MEF)

Abstract (German)

Größere Softwareprojekte werden heute üblicherweise in Teams entwickelt. Dazu kommt, dass sich die Anforderungen, die sie zu erfüllen haben, häufig ändern. Aus diesem Grund sind monolithische Programmriesen nicht mehr zeitgemäß. Die Herausforderung lautet, größere Lösungen in kleinere Einheiten zu zerlegen, an denen unterschiedliche Teammitglieder arbeiten können. Es gilt Komponenten zu entwickeln, die mit klaren Schnittstellen versehen getrennt voneinander versionisiert und verteilt werden. Das Managed Extensibility Framework (MEF) ist ein Werkzeug von Microsoft, mit dem solche Prinzipien leicht umgesetzt werden können. War es früher ein Codeplex-Projekt ist MEF mittlerweile sowohl in .NET 4 als auch in Silverlight 4 von Haus aus enthalten. In dem Workshop zeigt Rainer Stropek die Grundlagen von MEF und demonstriert anhand von Beispielen, wie die Bibliothek in der Praxis nutzbringend eingesetzt werden kann.

Introduction

- software architects gmbh
- Rainer Stropek
- Developer, Speaker, Trainer
- MVP for Windows Azure
- rainer@timecockpit.com
- twitter FOLLOW ME @rstropek

http://www.timecockpit.com

http://www.software-architects.com

Why does the world need MEF?

THE PROBLEM

Original Goals

- Before MEF
  - Multiple extensibility mechanism for different Microsoft tools (e.g. Visual Studio, Trace Listeners, etc.)
  - Developers outside of MS had the same problem
- MEF: Provide standard mechanisms for hooks for 3rd party extensions
- Goal: *Open and Dynamic Applications*
  - make it easier and cheaper to build extensible applications and extensions

MEF „Hello World“

```
[Export(typeof(Shape))]
public class Square : Shape
{
  // Implementation
}

[Export(typeof(Shape))]
public class Circle : Shape
{
  // Implementation
}

[Export]
public class Toolbox
{
  [ImportMany]
  public Shape[] Shapes { get; set; }
  // Additional implementation...
}

[…]

var catalog = new AssemblyCatalog(typeof(Square).Assembly);
var container = new CompositionContainer(catalog);
Toolbox toolbox = container.GetExportedValue<Toolbox>();
```

Export with name or type

Defaults to `typeof(Toobox)`

„Attributed Programming Model“

MEF „Hello World“ (continued)

- *Parts*
  - `Square`, `Circle` and `Toolbox`
- *Dependencies*
  - Imports (`Import`-Attribute)
  - E.g. `Toolbox.Shapes`
- *Capabilities*
  - Exports (`Export`-Attribute)
  - E.g. `Square`, `Circle`

MEF „Hello World“ (5 Minutes)

DEMO 1

Exports And Imports

- `Export` attribute
  - Class
  - Field
  - Property
  - Method
- `Import` attribute
  - Field
  - Property
  - Constructor parameter
- Export and import must have the same contract
  - Contract name and contract type
  - Contract name and type can be inferred from the decorated element

Inherited Exports

```
[Export]
public class NumOne
{
    [Import]
    public IMyData MyData
       { get; set; }
}

public class NumTwo : NumOne
{
}

[InheritedExport]
public class NumThree
{
    [Export]
    Public IMyData MyData { get; set; }
}

public class NumFour : NumThree
{
}
```

Import automatically inherited

Export NOT inherited
→ NumTwo has no exports

Member-level exports are never inherited

Inherits export with contract `NumThree` (including all metadata)

MEF Catalogs

- Catalogs provide components
- Derived from `System.ComponentModel.Composition.Primitives.ComposablePartCatalog`
  - `AssemblyCatalog`
    - Parse all the parts present in a specified assembly
  - `DirectoryCatalog`
    - Parses the contents of a directory
  - `TypeCatalog`
    - Accepts type array or a list of managed types
  - `AggregateCatalog`
    - Collection of `ComposablePartCatalog` objects

Directory catalog sample

HANDS-ON LAB 1 (15 MINUTES)

How to import using MEF

IMPORT TYPES

Lazy Imports

- Imported object is not instantiated immediately
  - Imported (only) when accessed
- Sample:

```
public class MyClass
{
    [Import]
    public Lazy<IMyAddin> MyAddin `
      { get; set; }
}
```

Prerequisite Imports

- Composition engine uses parameter-less constructor by default
- Use a different constructor with `ImportingConstructor` attribute
- Sample:

```
[ImportingConstructor]
public MyClass(
  [Import(typeof(IMySubAddin))]IMyAddin
    MyAddin)
{
  _theAddin = MyAddin;
}
```

Could be removed here; automatically imported

Optional Imports

- By default composition fails if an import could not be fulfilled
- Use `AllowDefault` property to specify optional imports
- Sample:

```
public class MyClass
{
    [Import(AllowDefault = true)]
    public Plugin thePlugin { get; set; }
}
```

Creation Policy

- `RequiredCreationPolicy` property
- `CreationPolicy.Any`
  - `Shared` if importer does not explicitly request `NonShared`
- `CreationPolicy.Shared`
  - Single shared instance of the part will be created for all requestors
- `CreationPolicy.NonShared`
  - New non-shared instance of the part will be created for every requestor

Part Lifecycle

HANDS-ON LAB 2 (5-10 MINUTES)

Advanced exports

METADATA AND METADATA VIEWS

Goal

- Export provides additional metadata so that importing part can decide which one to use
- Import can inspect metadata without creating exporting part
- Prerequisite: Lazy import

Metadata and metadata views (10 Minutes)

DEMO 2

Metadata

```
namespace MetadataSample
{
  public interface ITranslatorMetadata
  {
    string SourceLanguage { get; }

    [DefaultValue("en-US")]
    string TargetLanguage { get; }
  }
}
```

Export Metadata can be mapped to metadata view interface

```
namespace MetadataSample
{
  [Export(typeof(ITranslator))]
  [ExportMetadata("SourceLanguage", "de-DE")]
  [ExportMetadata("TargetLanguage", "en-US")]
  public class GermanEnglishTranslator : ITranslator
  {
    public string Translate(string source)
    {
      throw new NotImplementedException();
    }
  }
}
```

Metadata (continued)

```
namespace MetadataSample
{
   class Program
   {
      static void Main(string[] args)
      {
         var catalog = new AssemblyCatalog(
            typeof(ITranslator).Assembly);
         var container = new CompositionContainer(catalog);

         // We need a translator from hungarian to english
         Lazy<ITranslator, ITranslatorMetadata> translator =
            container
            .GetExports<ITranslator, ITranslatorMetadata>()
            .Where(t => t.Metadata.SourceLanguage == "hu-HU"
               && t.Metadata.TargetLanguage == "en-US")
            .FirstOrDefault();
      }
   }
}
```

Custom Export Attributes

```
[TranslatorExport("de-DE", "en-US")]


public class GermanEnglishTranslator
  : Itranslator
{
  public string Translate(
    string source)
  {
    throw new NotImplementedException();
  }
}
```

Custom export attributes makes code much cleaner.

```
[Export(typeof(ITranslator))]
[ExportMetadata("SourceLanguage", "de-DE")]
[ExportMetadata("TargetLanguage", "en-US")]
public class GermanEnglishTranslator
  : Itranslator
{
  public string Translate(
    string source)
  {
    throw new NotImplementedException();
  }
}
```

Custom Export Attributes (continued)

```
[MetadataAttribute]
[AttributeUsage(AttributeTargets.Class, AllowMultiple = false)]
public class TranslatorExportAttribute
    : ExportAttribute, ITranslatorMetadata
{
    public TranslatorExportAttribute(
        string sourceLanguage, string targetLanguage)
        : base(typeof(ITranslator))
        {
            this.SourceLanguage = sourceLanguage;
            this.TargetLanguage = targetLanguage;
        }
        public string SourceLanguage { get; private set; }
        public string TargetLanguage { get; private set; }
    }
}
```

Using MEF To Extend A WPF Application

HANDS-ON LAB 3 (30 MINUTES)

MEF AND SILVERLIGHT

MEF In Silverlight

- Additional catalog `DeploymentCatalog`
  - Load exported parts contained in XAP files
  - Provides methods for asynchronously downloading XAP files containing exported parts (`DeploymentCatalog.DownloadAsync`)
- Goal
  - Minimize initial load times
  - Application can be extended at run-time

MEF and Silverlight

HANDS-ON LAB 4 (15 MINUTES)

Read more about help, find the right tools

RESOURCES

Resources About MEF

- Managed Extensibility Framework on MSDN
- Managed Extensibility Framework for .NET 3.5 on Codeplex
- Visual Studio 2010 and .NET Framework 4 Training Kit