BASTA Workshop

Re-Imagine C#

Inhalt

Im Zentrum dieses erfolgreichen Klassikers unter den BASTA!-Workshops stehen diesmal die Themen **Anwendungsmodularisierung** und **asynchrones Programmieren**.

Am **Vormittag** beschäftigen wir uns mit der Frage, wie man **vom Monolithen zur modularen Anwendung** kommt, die leichter zu warten, weiterzuentwickeln und zu verteilen ist. Rainer Stropek wird Ihnen **MEF, System.AddIn und NuGet** anhand durchgängiger Beispiele erklären. Sie lernen, wie die verschiedenen Modularisierungstechniken funktionieren und wie sie sich voneinander abgrenzen.

Im Laufe des **Nachmittags** gehen wir auf die **Neuerungen in C# 5** hinsichtlich **asynchroner Programmierung** ein. Rainer Stropek zeigt Ihnen, was wirklich hinter async/await steckt und wie Sie die Sprachneuerungen in der Praxis erfolgreich einsetzen.

Modularisierung

Warum?

Grundregeln für Anwendungs-modularisierung

Bildquelle: http://www.flickr.com/photos/zooboing/4580408068/

Design

Was bei Framework Design zu beachten ist

Bildquelle: http://www.flickr.com/photos/designandtechnologydepartment/3968172841/

Tools

NuGet
MEF
MAF

Bildquelle: http://www.flickr.com/photos/46636235@N04/7115529993/

Case Study

MEF in Prism

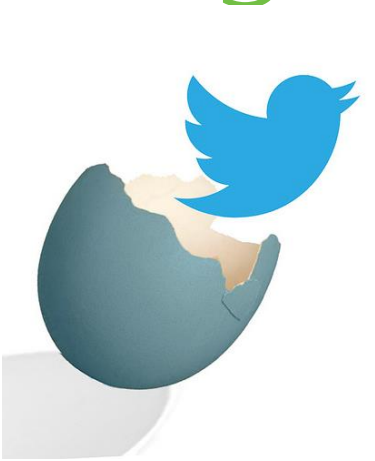
Bildquelle: http://www.flickr.com/photos/waagsociety/9182955544/

Async Programming

Grundlagen

Wie funktioniert async/await?

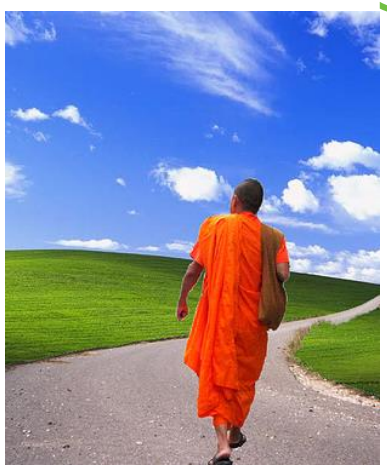
Bildquelle: http://www.flickr.com/photos/jimmiehome schoolmom/3423923394/



Server/Web

Beispiel: Async Web mit WebAPI

Bildquelle: http://www.flickr.com/photos/ mkhmarketing/8476983849/



Client

Beispiel: Async im Full Client UI

Bildquelle: http://www.flickr.com/photos/ oddsock/60344273/

Modularisierung

Grundlagen

Warum modulare Programmierung?

The benefits expected of modular programming are:

- Managerial - development time should be shortened because separate groups would work on each module with little need for communication
- Product flexibility - it should be possible to make drastic changes to one module without a need to change others
- Comprehensibility - it should be possible to study the system one module at a time. The whole system can therefore be better designed because it is better understood.

Quelle: Parnas, On the Criteria To Be Used in Decomposing Systems into Modules, Carnegie-Mellon University, 1972

Was ist eine Softwarefabrik?

A software factory is an organizational structure that specializes in producing computer software applications or software components [...] **through an assembly process**.

The software is created by **assembling predefined components**. Traditional coding, is left only for creating new components or services.

A **composite application** is the end result of manufacturing in a software factory.

Quelle: "Software Factory" in Wikipedia

Vorteile einer Softwarefabrik (1/2)

- Kernkompetenzen der Mitarbeiter werden hervorgehoben
  Fachlich orientierte Berater konfigurieren (Vokabular näher bei der jeweiligen Domäne des Kunden)
  Softwareentwickler erstellen Basiskomponenten
- Steigerung der Effizienz
  Das Rad wird weniger oft neu erfunden
- Steigerung der Qualität
  Anspruchsvolle QS-Maßnahmen für Basiskomponenten

Vorteile einer Softwarefabrik (2/2)

- **Reduktion der Projektrisiken**
  Fachberater mit besserem Kundenverständnis
  Höhere Qualität der Basiskomponenten
- **Steigerung des Firmenwerts**
  Systematisches Festhalten von Wissen über die Herstellung einer Familie von Softwarelösungen
  Design Patterns, Frameworks, Modelle, DSLs, Tools
- **Vereinfachung des Vertriebs- und Spezifikationsprozesses**
  Konzentration auf projektspezifische Lösungsteile
  Quick Wins durch Standardkomponenten

Was eine Softwarefabrik nicht will…

- Reduktion des Entwicklungsprozesses auf standardisierte, mechanische Prozesse
  Im Gegenteil, mechanische Prozesse sollen Tool überlassen werden
- Verringerung der Bedeutung von Menschen im Entwicklungsprozess
- „Handwerk“ der Softwareentwicklung wird nicht gering geschätzt sondern gezielt eingesetzt
- Entwicklung von Frameworks statt Lösungen für Kunden

Software Factories → Economy of Scope

Economy of Scale

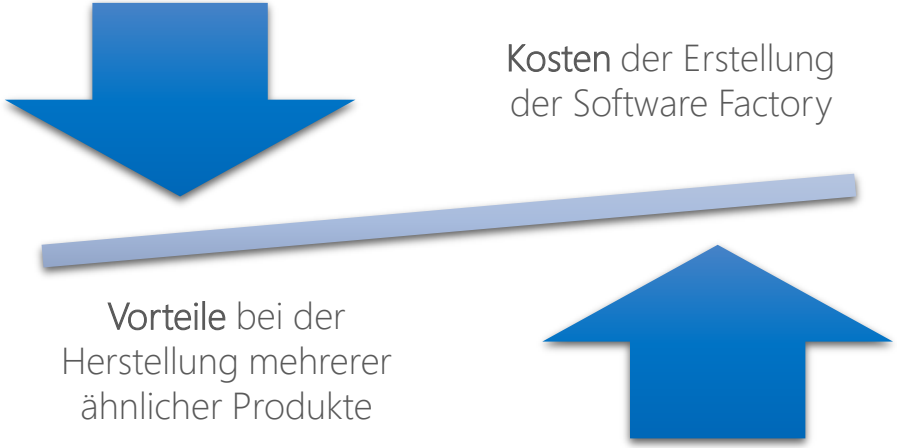
- Multiple implementations (=Copies) of the same design
- Mehr oder weniger mechanische Vervielfältigung von Prototypen
  - Massengüter
  - Software (z.B. Auslieferung auf Datenträger)

Economy of Scope

- Production of multiple designs and their initial implementations
- Ähnliche Designs (=Familien von Anwendungen) auf Grundlage gemeinsamer Techniken und Technologien
  - Individuelle physische Güter (z.B. Brücken, Hochhäuser)
  - Individualsoftware, Softwareplattformen (vgl. PaaS)

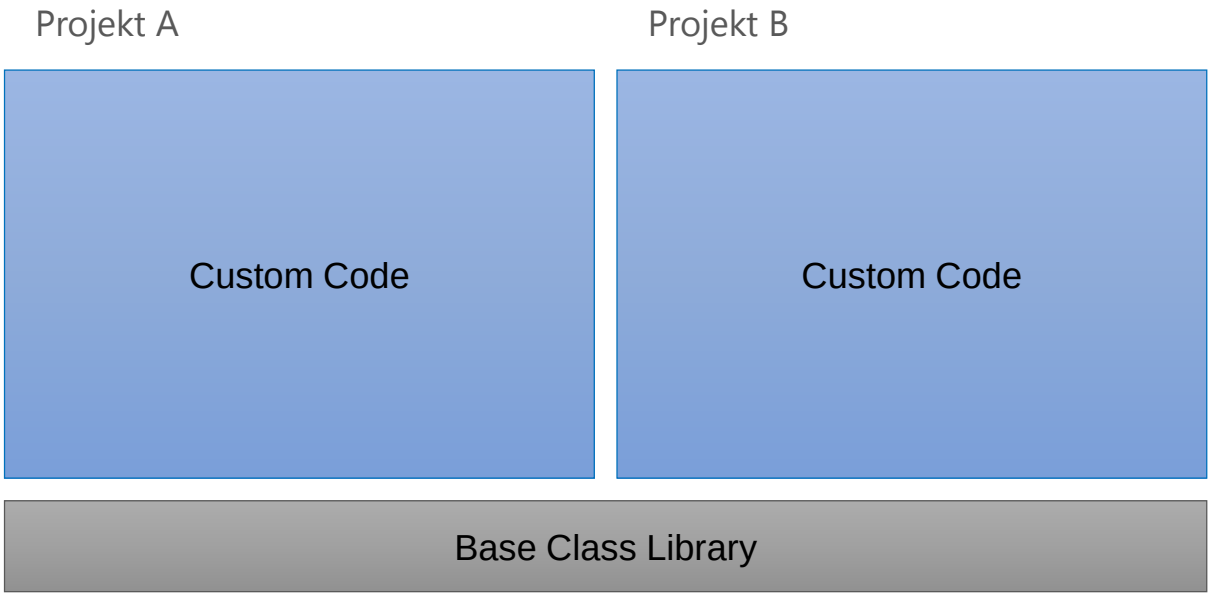
Quelle: Greenfield J., Short K.: Software Factories

Kosten/Nutzen Abwägung

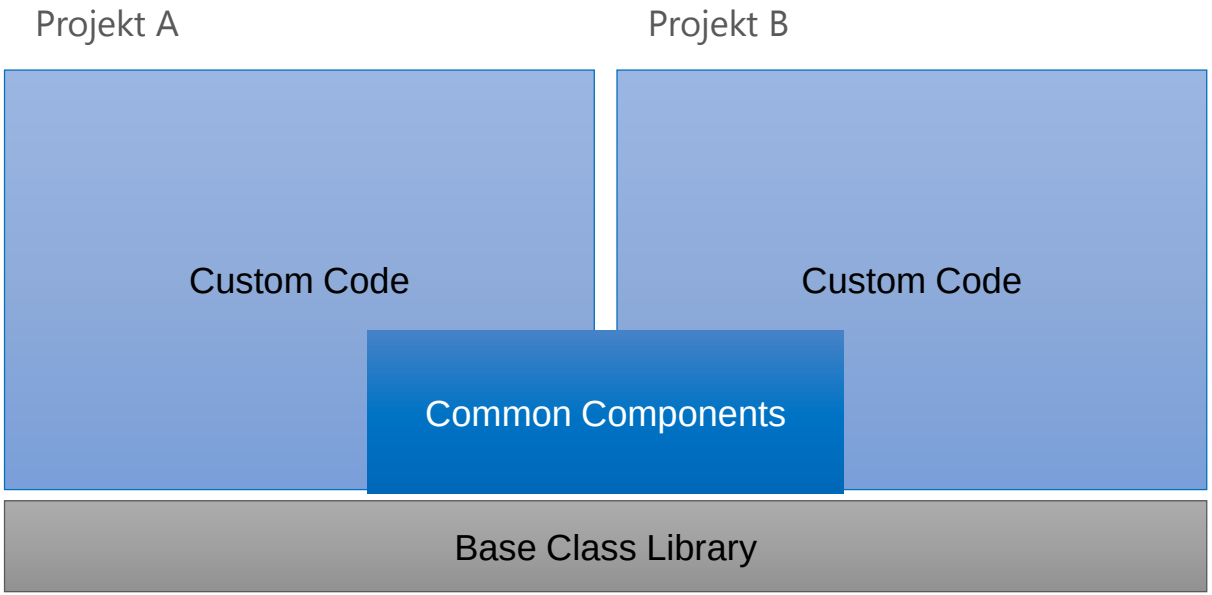


Spezialisierung als zentraler Erfolgsfaktor beim Einsatz der Software Factory Prinzipien

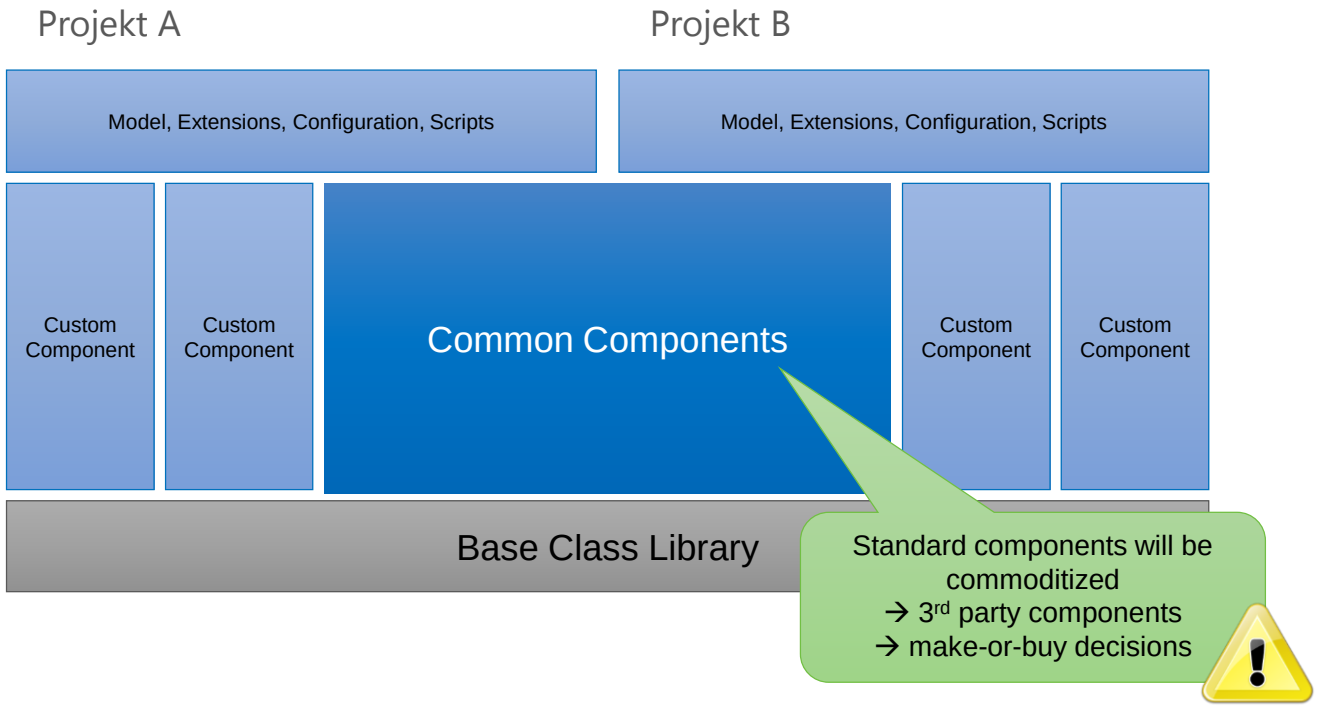
Entwicklung einer Software Factory



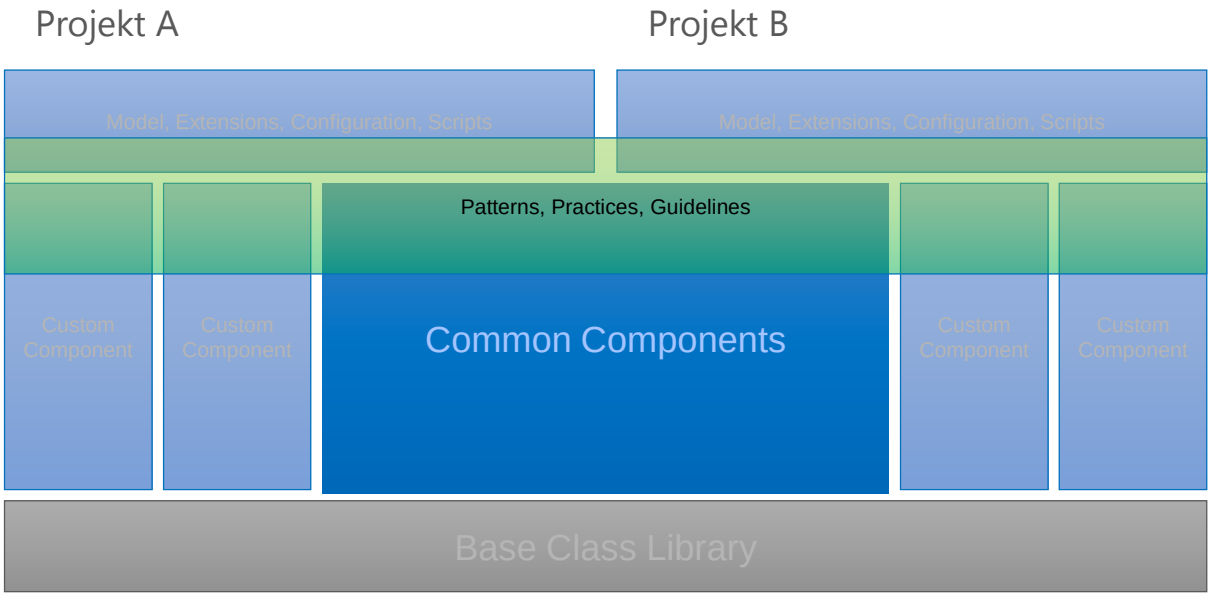
Entwicklung einer Software Factory



Entwicklung einer Software Factory



Entwicklung einer Software Factory



Nette Theorie, aber in der Praxis??

- Abstraktionsgrad
  Je abstrakter desto mächtiger ☺
  Je abstrakter desto spezifischer ☹
- Abhängigkeiten
  Vertrauen in Werkzeuge
  Vertrauen in Lieferanten
  Vertrauen in Mitarbeiter
- Kleinster gemeinsamer Nenner
  Ausnutzung aller Möglichkeiten der zugrunde liegenden Plattform
  Performanceprobleme (Beispiel: OR-Mapper vs. T-SQL)

Werkzeuge

- Klassenbibliotheken
  - Dokumentation
  - Statische Codeanalyse
  - Deployment
- Codegeneratoren
  - Vorlagen
  - Patterns in Form von Assistenten
- Domänenspezifische Sprachen
  - XML-basierend oder individuell (Compiler-Compiler)
  - Compiler (Codegenerator) vs. interpretiert
- Scriptsprachen
- Anwendungsmodularisierung
- Prozessautomatisierung
  - Qualitätssicherung
  - Build

- MS Framework Design Guidelines
  - Sandcastle
  - StyleCop, Code Analysis, 3rd party tools
  - NuGet
- Codegeneratoren
  - T4, ANTLR StringTemplates
  - Visual Studio Templates
- Domänenspezifische Sprachen
  - XAML (UI und Workflows), EF ANTLR (Compiler-Compiler)
- DLR, Project „Roslin“
- MEF, MAF
- Prozessautomatisierung
  - Visual Studio Unit Tests
  - TFS Buildautomatisierung

Framework Design

Guidelines

Die wichtigsten Gebote für Klassenbibliotheken

- Je häufiger wiederverwendet desto höher muss die Qualität sein
  - An der zentralen Klassenbibliothek arbeiten Ihre besten Leute
  - Design, Code und Security Reviews
- Folgen Sie den Microsoft Framework Design Guidelines
  - Nutzen Sie StyleCop und Code Analysis (siehe folgende Slides)
- Schreiben Sie Unit Tests
  - Gleiche Qualitätskriterien wie beim Framework selbst
  - Monitoring der Code Coverage
- Verwenden Sie Scenario Driven Design

Tipps für Frameworkdesign

- Beste Erfahrungen mit Scenario-Driven Design
  Client-Code-First (hilft auch für TDD ☺)
  Welche Programmiersprachen sind dabei für Sie interessant? Dynamische Sprachen nicht vergessen!
- „Simple things should be simple and complex things should be possible“ (Alan Kay, Turing-Preisträger)
- Einfache Szenarien sollten ohne Hilfe umsetzbar sein
  Typische Szenarien von komplexen Szenarien mit Namespaces trennen
  Einfache Methodensignaturen bieten (Defaultwerte!)
  Einfache Szenarien sollten das Erstellen von wenigen Typen brauchen
  Keine langen Initialisierungen vor typischen Szenarien notwendig machen
  Sprechende Exceptions

Bibliographie
Framework Design Guidelines

- Cwalina, Abrams: Framework Design Guidelines
  Sporadische Aktualisierungen im Blog von Cwalina
  Abrams ist nicht mehr bei MS (früherer Blog)

- Auszug aus dem Buch kostenlos in der MSDN verfügbar
  Design Guidelines for Developing Class Libraries

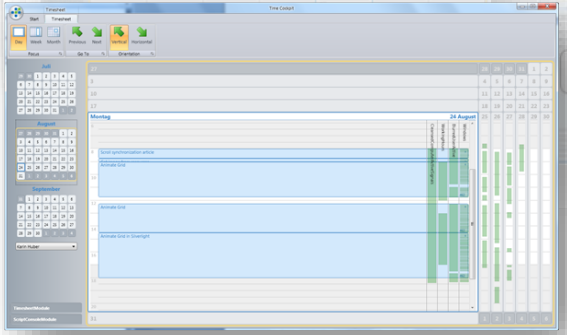
- Scenario Driven Design
  Blogartikel Cwalina

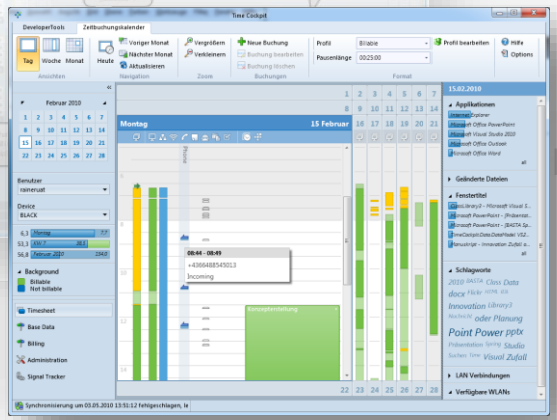
- Tipp: Portable Library Tools
  Interessant für portable Libraries (.NET Framework, Silverlight, Windows Phone, XNA)
  Noch nichts dergleichen für WinRT verfügbar

Früher Prototyp
(Funktional)

UI Skizzen

Programmierte Prototypen





This solution would replace `Install.stg` and `CodeBatchCollection`. There would be a Xaml file compiled into time cockpit's resources. There has to be a function to apply all update batches in the Xaml file similar to today's `InstallBatchManager.Install`. Additionally there will be functions to

1. find out all update batches that are missing on a certain database.
2. find out if the application can work with a certain database.

The following code snippets show how the API to install update batches would work:

Get update batch from XAML file stored in the assembly's resources.

```
UpdateBatch updateBatch = this.ReadUpdateBatchFromResources();
```

Note that DbClient.Create *will* ***not*** *automatically install update batches in the future any more.*

```
using (var dbClient = new DbClient.Create(...))
{
```

Find out which update batches are not installed in the database that dbClient *is pointing to.*

```
        IEnumerable<UpdateBatch> missingBatches =
              dbClient.GetMissingUpdateBatches(updateBatch);
        foreach (var missingBatch in missingBatches)
        {
              Console.WriteLine("{0} is missing", missingBatch.Guid);
        }
```

Find out if app can run without executing any batches (i.e. if mandatory batches are missing).

```
        switch (dbClient.GetUpdateBatchStatus(updateBatch))
        {
              case BatchStatus.Complete:
                    Console.WriteLine("Update batch is completely installed.");
                    break;
              case BatchStatus.Acceptable:
                    Console.WriteLine("Some non-mandatory batches are missing.");
                    break;
              case BatchStatus.Incomplete:
                    Console.WriteLine("Mandatory batches are missing.");
                    break;
        }
```

Install all missing update batches.

```
        dbClient.InstallMissingUpdateBatches(updateBatch);
}
```

Feature Files

On model level time cockpit will be extended by "features". A feature is a part of the logical data

2009-08-25 Planning Sprint 7 - TCQL Extensions.pptx - Microsoft PowerPoint

Must-have Requirement
Subqueries in the Select Clause

SUBQUERIES

```
From P In Project
Where P.Customer.CustomerName = "ABC"
Select New With
{
  .Project = P,
  .Hours = ( From T In P.Timesheets
             Where :Year(T.StartDate) = 2009
                   And P.Type = "XYZ"
             Select new With {
               .Hours = Sum(T.DurationInHours) } )
}

From P In Project
Where P.Customer.CustomerName = "ABC"
Select New With
{
  .Project = P,
  .Hours = Sum(P.Timesheets.DurationInHours)
}
```

Option – should we do that?

Folie 15 von 32 | "FFG Zwischenpräsentation 2009-08-04" | Deutsch (Österreich) | 63 %

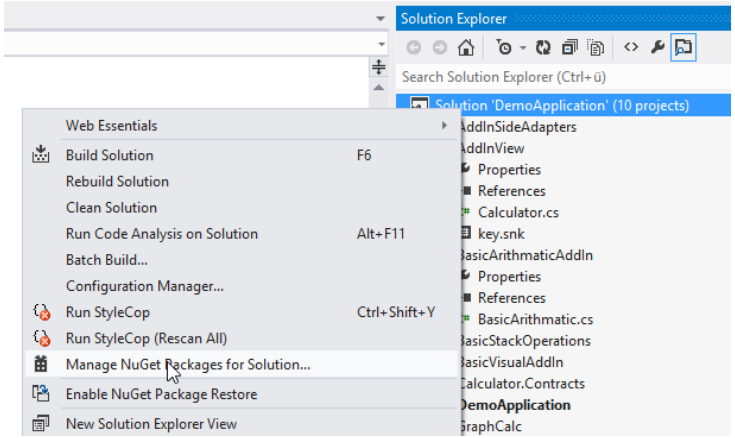
NuGet

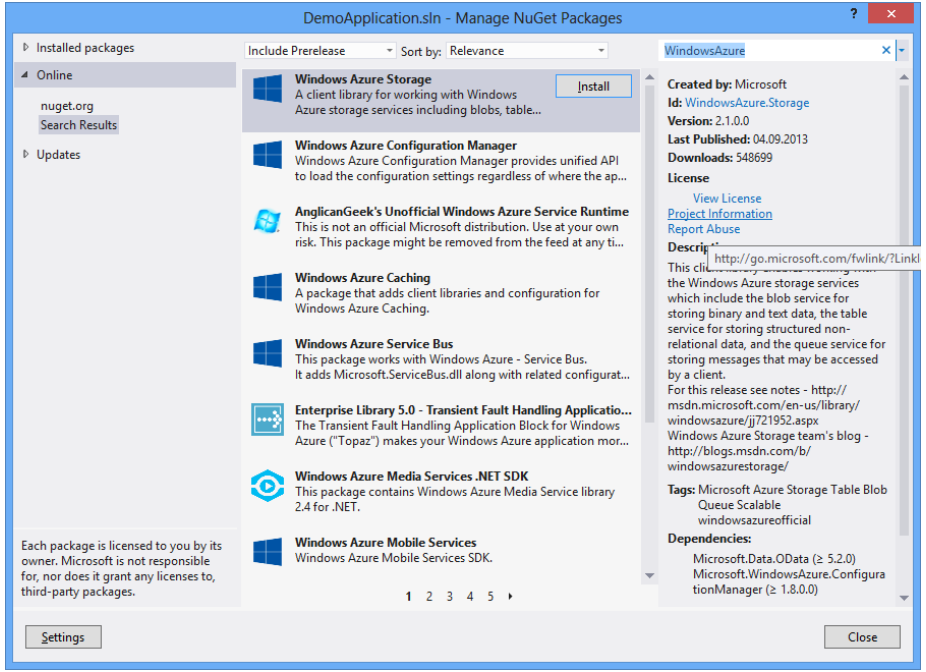
Package Manager für die Microsoft-Entwicklungsplattform

Was ist NuGet?

- Werkzeug zur einfachen Verteilung von Paketen (=Libraries und Tools)
- Alles Notwendige in einem Paket
  Binaries für verschiedene Plattformen
  Optional Symbole und Sourcen
  Anpassungen am Projekt (z.B. Referenzen, Änderungen am app/web.config)
- UI-Integration in Visual Studio
  Ab VS2010, eingeschränkt auch in Mono
  Express-Editionen werden unterstützt
- http://www.nuget.org
  http://nuget.codeplex.com/ (Sourcecode)

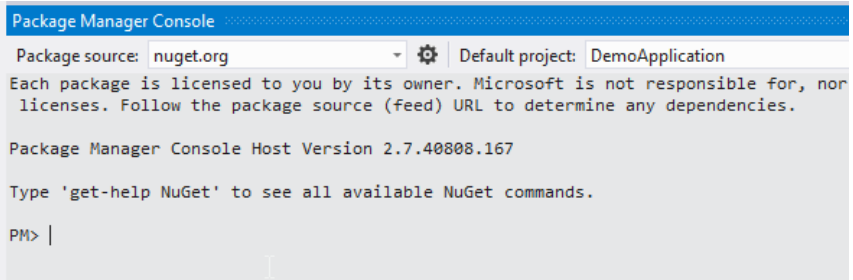
NuGet in Visual Studio

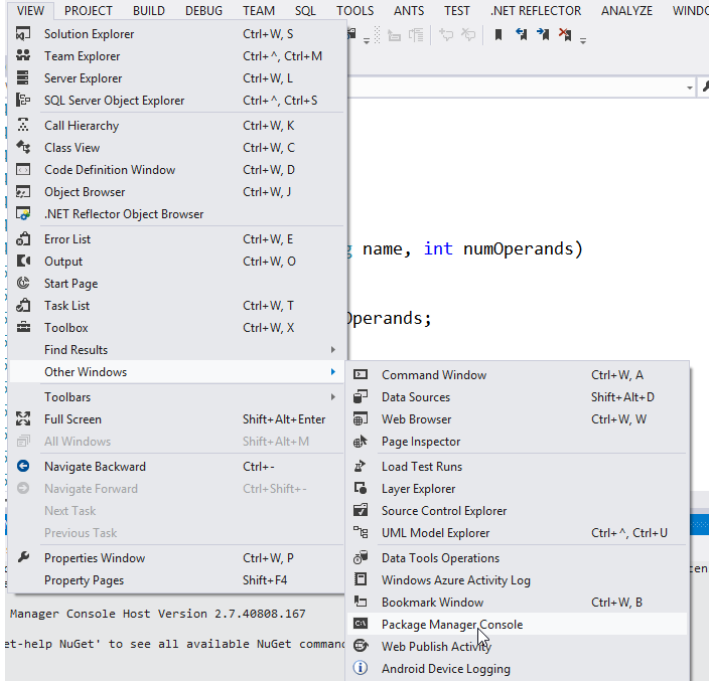




NuGet in Visual Studio

- Package Manager Console
- PowerShell console in Visual Studio
- Automate Visual Studio and NuGet
- NuGet PowerShell Reference





NuGet Pakete erstellen

- Kommandozeilentool *nuget.exe*
  Pakete erstellen (*Pack* Command)
  Pakete veröffentlichen (*Push*, *Delete* Command)
  Paket installieren (*Install*, *Restore*, *Update* Command)
  Generieren eines *nuspec*-Files (*Spec* Command)
  Wichtig für Buildautomatisierung
  Kommandozeilenreferenz

- NuGet Package Explorer
  Grafisches UI zur Erstellung/Bearbeitung von NuGet Paketen und *nuspec* Files
  http://npe.codeplex.com/

Example

nuspec File

```
<?xml version="1.0" encoding="utf-16"?>
<package xmlns="http://schemas.microsoft.com/packaging/2012/06/nuspec.xsd">
  <metadata>
    <id>CockpitFramework.Data</id>
    <version>$version$</version>
    <title>Cockpit Framework Data Layer</title>
    <authors>software architects gmbh</authors>
    <owners>software architects gmbh</owners>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>...</description>
    <releaseNotes></releaseNotes>
    <dependencies>
      <group targetFramework=".NETFramework4.0">
        <dependency id="CockpitFramework.Dependencies"
          version="[$version$]" />
      </group>
      <group targetFramework="sl5">
        <dependency id="CockpitFramework.Dependencies"
          version="[$version$]" />
      </group>
    </dependencies>
  </metadata>
```

Example

nuspec File

```
<dependencies>
  <group targetFramework=".NETFramework4.0">
    <dependency id="log4net" version="[1.2.11]" />
    <dependency id="Microsoft.SqlServer.Compact.Private"
      version="[4.0.8482.1]" />
    <dependency id="AvalonEdit" version="[4.2.0.8783]" />
    <dependency id="ClosedXML" version="[0.68.1]" />
    <dependency id="DocumentFormat.OpenXml" version="[1.0]" />
    <dependency id="IronPython" version="[2.7.3]" />
    <dependency id="LumenWorks.Framework.IO" version="[1.0.0]" />
    <dependency id="Newtonsoft.Json" version="[5.0.6]" />
    <dependency id="WindowsAzure.Storage" version="[2.0.5.1]" />
    <dependency id="Microsoft.Bcl.Async" version="[1.0.16]" />
  </group>

  <group targetFramework="sl5">
    …
  </group>
</dependencies>
```

Version range syntax

```
1.0   = 1.0 ≤ x
(,1.0]  = x ≤ 1.0
(,1.0)  = x < 1.0
[1.0] = x == 1.0
(1.0) = invalid
(1.0,) = 1.0 < x
(1.0,2.0) = 1.0 < x < 2.0
[1.0,2.0] = 1.0 ≤ x ≤ 2.0
empty = latest version.
```

Example

nuspec File

```
  <files>
    <!-- net4 -->
    <file src=".\$configuration$\TimeCockpit.Common.dll"
      target="lib\net4" />
    <file src=".\$configuration$\TimeCockpit.Data.dll"
      target="lib\net4"/>
    …

    <!-- sl5 -->
    <file src=".\SL\$configuration$\TimeCockpit.Common.dll"
      target="lib\sl5" />
    <file src=".\SL\$configuration$\TimeCockpit.Data.dll"
      target="lib\sl5" />
    …

    <!-- include source code  for symbols -->
    <file src=".\...\*.cs" target="src\TimeCockpit.Common" />
    <file src=".\...\*.cs" target="src\TimeCockpit.Data" />
</package>
```

Folder Structure

```
\lib
    \net11
        \MyAssembly.dll
    \net20
        \MyAssembly.dll
    \net40
        \MyAssembly.dll
    \sl40
        \MyAssembly.dll
```

```
\content
    \net11
        \MyContent.txt
    \net20
        \MyContent20.txt
    \net40
    \sl40
        \MySilverlightContent.html

\tools
    init.ps1
    \net40
        install.ps1
        uninstall.ps1
    \sl40
        install.ps1
        uninstall.ps1
```

For Details see [NuGet Docs]

Versioning Notes

- Things to remember
  NuGet never installs assemblies machine-wide (i.e. not in GAC)
  You cannot have multiple versions of the same DLL in one AppDomain

- DLL Hell
  Policy too loose: Problems with breaking changes
  Policy too tight: Problems with library having dependencies on other libraries
  (e.g. ANTLR and ASP.NET MVC, everyone depending on Newtonsoft JSON)

- For Library Publishers: SemVer
  X.Y.Z (Major.Minor.Patch)
  Rethink your strong naming policies

```
<runtime>
   <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
         <assemblyIdentity name="mscorcfg"
           publicKeyToken="b03f5f7f11d50a3a" culture=""/>
         <bindingRedirect
           oldVersion="0.0.0.0-65535.65535.65535.65535"
           newVersion="1.0.3300.0"/>
      </dependentAssembly>
   </assemblyBinding>
</runtime>
```

Binding Redirects

Note: NuGet can generate this for you

Add-BindingRedirect Command

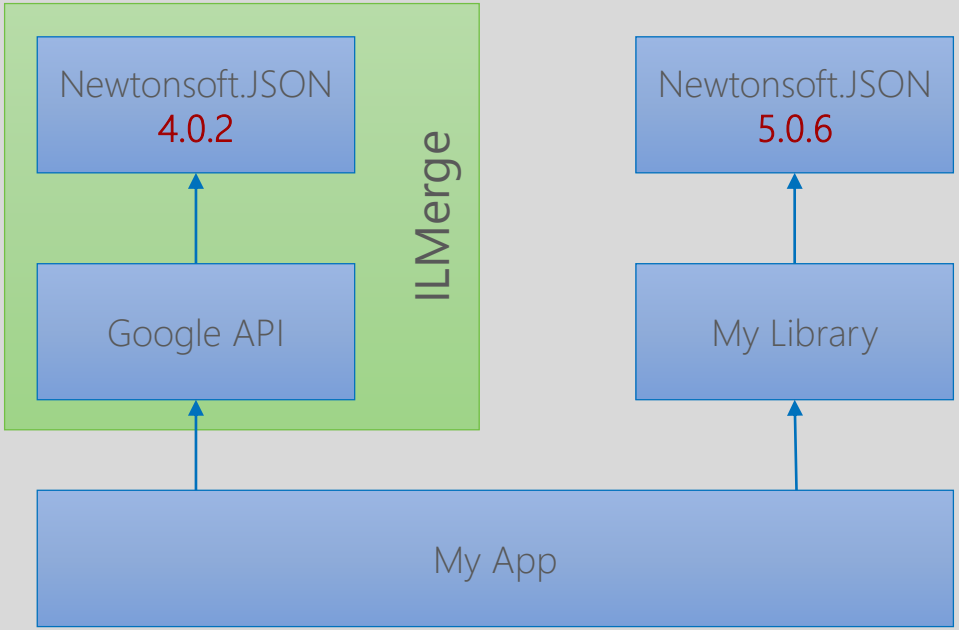
See online reference for details

```xml
<?xml version="1.0" encoding="utf-8"?>
<packages>
    <package id="SomePackage"
      version="2.1.0"
      allowedVersions="[2,3)" />
</packages>
```

Versioning

Constraings in packges.config

Manual editing necessary



ILMerge

Soliving version conflicts

[Microsoft Download]

ILMerge

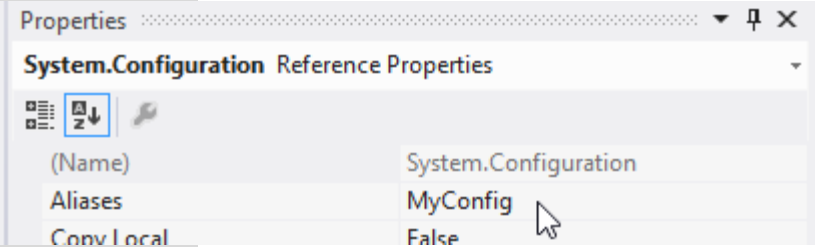
Soliving version conflicts

C# *extern alias*

```
ILMerge
  "Assemblies\Google.Apis.Authentication.OAuth2.dll"
  "Assemblies\Google.Apis.dll"
  "Assemblies\Google.Apis.Latitude.v1.dll"
  "Assemblies\DotNetOpenAuth.dll" "Assemblies\log4net.dll"
  "Assemblies\Newtonsoft.Json.Net35.dll"
  /out:"c:\temp\Google.Apis.All.dll" /lib:"Lib„

extern alias MyConfig;
using Conf = MyConfig::System.Configuration;

namespace MyTinyMvvmToolkit
{
  public class NotificationObject
  {
    public void ReadConfiguration()
    {
      var setting =
        Conf.ConfigurationManager.AppSettings["MyDB"];
    }
  }
}
```



Content Files

```
<?xml version="1.0" encoding="utf-16"?>
<package xmlns="http://schemas.microsoft.com/packaging/2012/06/nuspec.xsd">
  <metadata>…</metadata>

  <files>
    <file src="content\app.config.transform"
      target="content\" />
    <file src="content\TimeCockpitInitialization.cs.pp"
      target="content\" />
  </files>
</package>
```

```
<configuration>
    <startup>
      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
   </startup>
   <system.data>
      <DbProviderFactories>
         <remove invariant="System.Data.SqlServerCe.4.0"/>
         <add name="Microsoft SQL Server Compact Data Provider 4.0"
               invariant="System.Data.SqlServerCe.4.0"
               description=".NET Framework Data Provider for Microsoft SQL Server Compact"
               type="System.Data.SqlServerCe.SqlCeProviderFactory, System.Data.SqlServerCe,
               Version=4.0.0.1, Culture=neutral, PublicKeyToken=89845dcd8080cc91"/>
      </DbProviderFactories>
   </system.data>
</configuration>
```

New in NuGet 2.6: XDT

```
…

namespace $rootnamespace$
{
  using System;

  /// <summary>
  /// Class taking care of cockpit framework initialization
  /// </summary>
  public class TimeCockpitInitialization
  {
    …
  }
}
```

Content Files

Sourcecode Transformations in .cs.pp File

Available properties see MSDN

User PowerShell scripts to modify project properties

NuGet Docs

Demo

Nuget at software architects

Nuget in Practice

Nuspec files

- Files
- Dependencies
- Build

Packages in NuGet Explorer

Packages in VS

Publishing NuGet Packages

- http://www.nuget.org
  Public NuGet Feed
- Create your private feed
  File system
  Private NuGet Server
  For details see NuGet Help
- Use a NuGet SaaS like *MyGet*
  http://www.myget.org/

MEF

Managed Extensibility Framework

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 vs. MAF

- Managed AddIn Framework
  System.AddIn
- MAF has higher-level goals
  Isolate extension
  Load and unload extensions
  API Compatibility
- Adding MAF leads to higher effort than adding MEF
  A single application can use both

MEF „Hello World“

Anatomy of a program with MEF

Attributed Programming Model

```
[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(Toolbox)*

MEF „Hello World“

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

MEF „Hello World“

MEF Basics

- Basic Exports
- Basic Imports
- Catalogs
- Composition

Demo

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

Catalogs

Loading of modules using *DirectoryCatalog*

Demo

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

Lazy Imports

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

Prerequisite Imports

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

Could be removed here; automatically imported

Composition engine uses parameter-less constructor by default

Use a different constructor with *ImportingConstructor* attribute

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

Optional Imports

By default composition fails if an import could not be fulfilled

Use *AllowDefault* property to specify optional imports

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

MEF Object Lifetime

- Container holds references to all disposable parts
  Only container can call *Dispose* on these objects
- Manage lifetime of disposable objects
  Dispose the container → it will dispose all managed objects
  Call *ReleaseExport* on a non-shared object to dispose just this object
  Use *ExportFactory<T>* to control lifetime
- *IPartImportsSatisfiedNotification*
  Implement if you need to get informed when composition has been completed

Part Lifecycle

Demo

Metadata and Metadata views

Advanced exports

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

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

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

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();
      }
   }
}
```

Export Metadata can be mapped to metadata view interface

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

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

[TranslatorExport("de-DE", "en-US")]
public class GermanEnglishTranslator
  : ITranslator
{
  public string Translate(
    string source)
  {
    throw new NotImplementedException();
  }
}
```

Custom export attributes makes code much cleaner.

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; }
  }
}
```

Convention-Based Programming

New in .NET 4.5

Goals

- Reduce the need for attributes
  Note that attributes in the source code override conventions
- Convention-based coupling
  Infer MEF attributes for objects' types
  Example: Export all classes derived from *Controller*
- *System.ComponentModel.Composition.Registration.RegistrationBuilder*
  ForType – creates a rule for a single type
  ForTypesDerivedFrom – creates a rule for all types derived from a certain type
  ForTypesMatching – creates a custom rule based on a predicate
  Returns a *PartBuilder* object that is used to configure imports and exports

Example

```
public interface IShapeMetadata { bool Is2D { get; } };

public class Shape { }
public class Circle : Shape { }
public class Rectangle : Shape { }

[…]

[ImportMany(typeof(Shape))]
private Shape[] shapes;

[…]

// Export all descendants of Shape and add some metadata
var rb = new RegistrationBuilder();
var pb = rb.ForTypesDerivedFrom<Shape>();
pb.Export<Shape>(eb => eb.AddMetadata("Is2D", true))
   .SetCreationPolicy(CreationPolicy.NonShared);

// Use registration builder with catalog
var me = new Program();
var container = new CompositionContainer(
   new AssemblyCatalog(me.GetType().Assembly, rb));
container.ComposeParts(me);
```

Export all descendants of a given class

Add some metadata

Set a creation policy

Resources

Read more about help, find the right tools

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

MAF

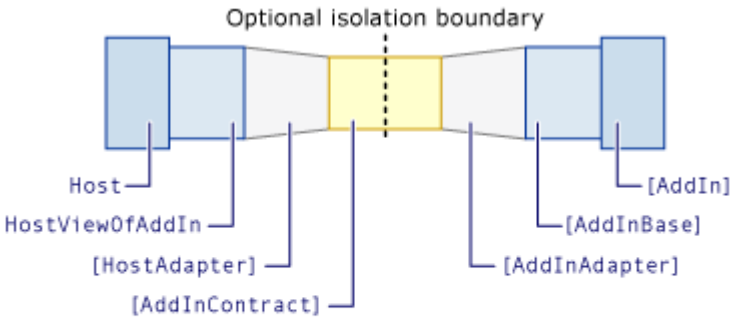
Manged Add-In Framework (System.AddIn)

MEF or MAF?

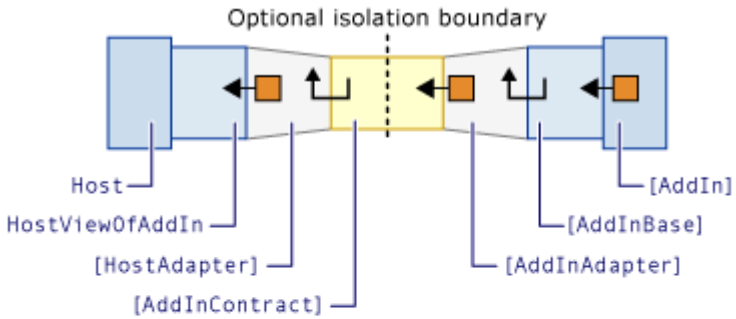
- MEF
  More up-to-date
  Also supported for Windows Store apps
  Under active development (e.g. many extensions in .NET 4.5)

- MAF
  Few resources, little documentation
  Provides addin isolation (AppDomain or Process)
  Solves versioning issues (upwards/downward compatibility of add in's)

Pipelines

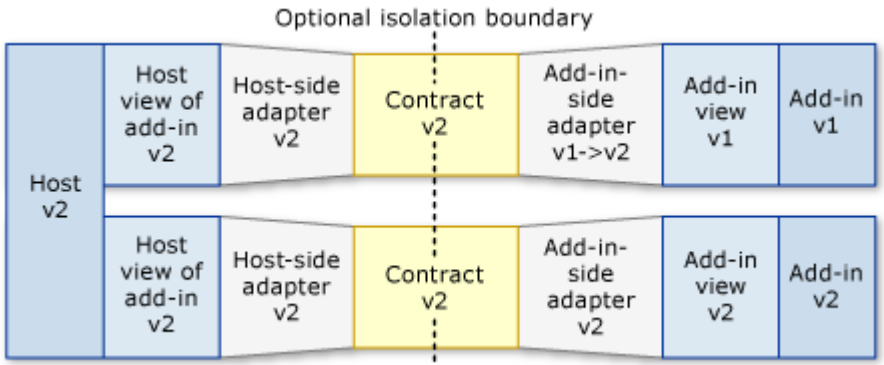


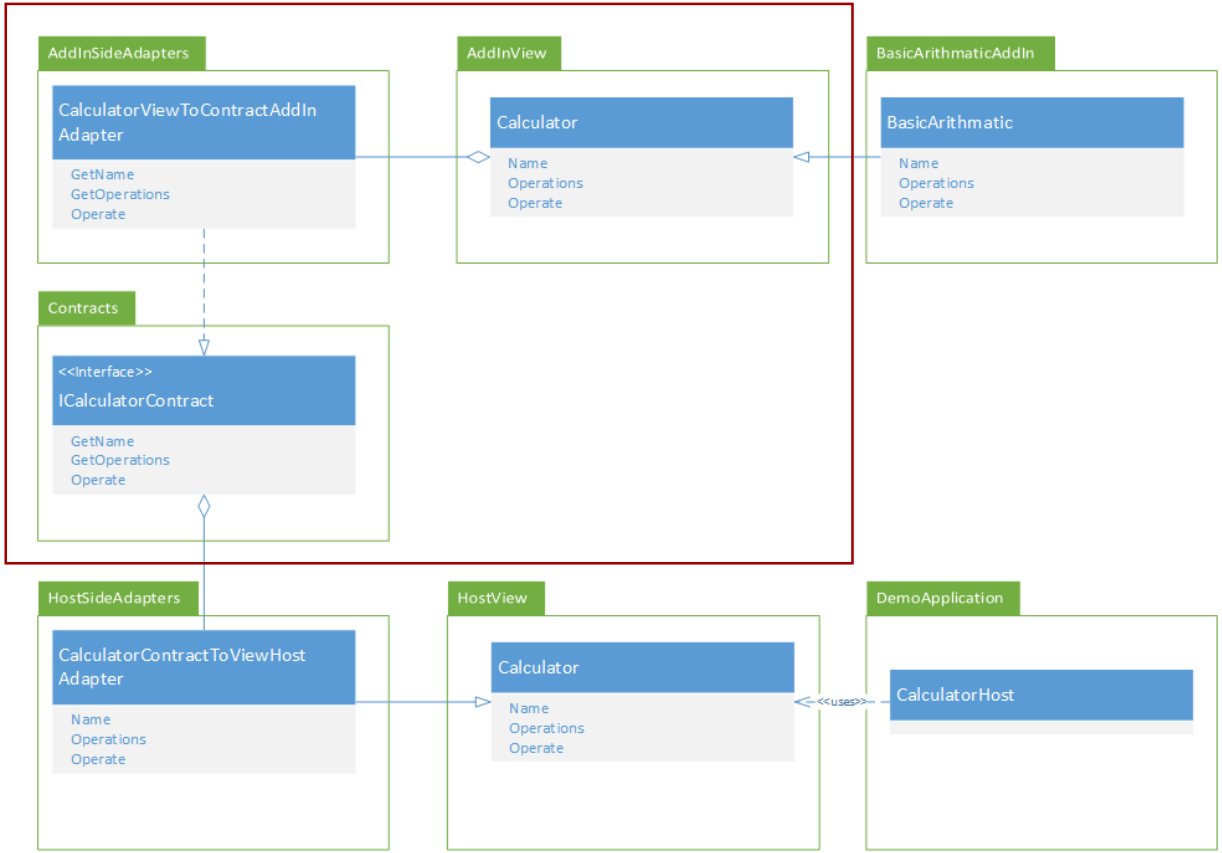
[]	Indicates required attributes on types for discoverability.
	The host and HostViewOfAddIn types do not require attributes.



	The target type takes the type at the base of the arrow as its constructor.
	The target type is inherited by the type at the base of the arrow.
[]	Brackets indicate types that require attributes.

Pipelines


Optional isolation boundary
Host v2
Host view of add-in v2
Host-side adapter v2
Contract v2
Add-in-side adapter v1->v2
Add-in view v1
Add-in v1
Host view of add-in v2
Host-side adapter v2
Contract v2
Add-in-side adapter v2
Add-in view v2
Add-in v2


AddInSideAdapters
CalculatorViewToContractAddIn Adapter
GetName
GetOperations
Operate
AddInView
Calculator
Name
Operations
Operate
BasicArithmaticAddIn
BasicArithmatic
Name
Operations
Operate
Contracts
<<Interface>>
ICalculatorContract
GetName
GetOperations
Operate
HostSideAdapters
CalculatorContractToViewHost Adapter
Name
Operations
Operate
HostView
Calculator
Name
Operations
Operate
<<uses>>
DemoApplication
CalculatorHost

```
namespace Calculator.Contracts
{
    [AddInContract]
    public interface ICalculatorContract : IContract
    {
        IListContract<IOperationContract> GetOperations();
        [SecurityCritical]
        double Operate(IOperationContract op, double[] operands);
        string GetName();
    }

    [AddInContract]
    public interface IVisualCalculatorContract : IContract
    {
        IListContract<IOperationContract> GetOperations();
        INativeHandleContract Operate(IOperationContract op,
          double[] operands);
        string GetName();
    }

    public interface IOperationContract : IContract
    {
        string GetName();
        int GetNumOperands();
    }
}
```

Contract

Implement *IContract*

Add *AddInContract* attribute for add ins

INativeHandleContract for adding UI features

Add In View

Apply *AddInBase* attribute

```
[AddInBase]
public abstract class Calculator
{
   public abstract String Name { get; }
   public abstract IList<Operation> Operations { get; }
   public abstract double Operate(Operation op, double[] operands);
}

[AddInBase]
public abstract class VisualCalculator
{
   public abstract String Name { get; }
   public abstract IList<Operation> Operations { get; }
   public abstract FrameworkElement Operate(Operation op,
      double[] operands);
}

public class Operation
{
   private string _name;
   private int _numOperands;

   public Operation(string name, int numOperands) {…}
   public String Name { get { return _name; } }
   public int NumOperands { get { return _numOperands; } }
}
```

Add In Adapter

Apply *AddInAdapter* attribute

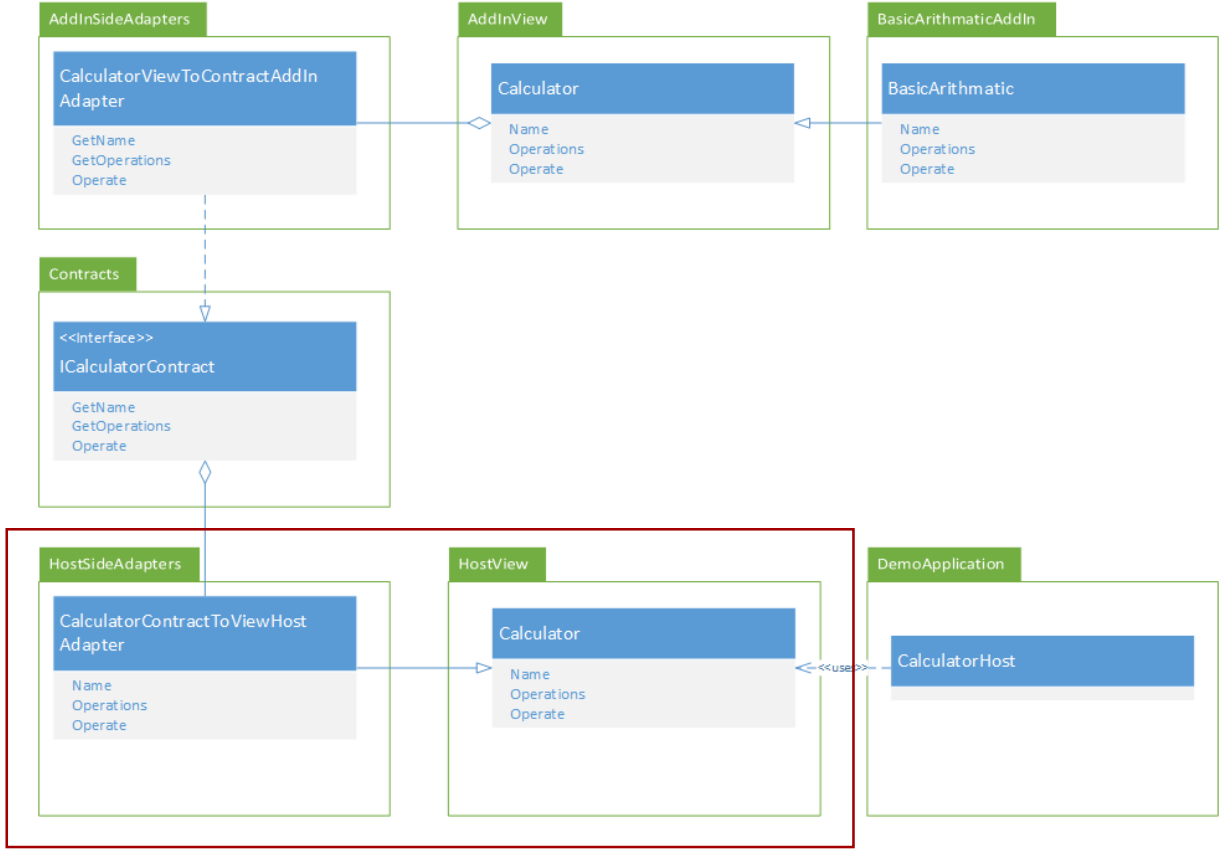
```
[AddInAdapter]
public class CalculatorViewToContractAddInAdapter
   : ContractBase, Calculator.Contracts.ICalculatorContract
{
   private AddInView.Calculator _view;

   public CalculatorViewToContractAddInAdapter(AddInView.Calculator view)
   {
      _view = view;
   }

   public IListContract<Calculator.Contracts.IOperationContract> GetOperations()
   {
      return CollectionAdapters.ToIListContract(
         _view.Operations,
         OperationViewToContractAddInAdapter.ViewToContractAdapter,
         OperationViewToContractAddInAdapter.ContractToViewAdapter);
   }

   public double Operate(Calculator.Contracts.IOperationContract op,
      double[] operands)
    {
        return _view.Operate(
           OperationViewToContractAddInAdapter.ContractToViewAdapter(op),
           operands);
    }

    public string GetName() { return _view.Name; }
}
```


AddInSideAdapters
CalculatorViewToContractAddIn Adapter
GetName
GetOperations
Operate
AddInView
Calculator
Name
Operations
Operate
BasicArithmaticAddIn
BasicArithmatic
Name
Operations
Operate
Contracts
<<Interface>>
ICalculatorContract
GetName
GetOperations
Operate
HostSideAdapters
CalculatorContractToViewHost Adapter
Name
Operations
Operate
HostView
Calculator
Name
Operations
Operate
<<use>>
DemoApplication
CalculatorHost

Host View

```
public abstract class CalculatorBase
{
   public abstract String Name { get; }
   public abstract IList<Operation> Operations { get; }
}

public abstract class Calculator : CalculatorBase
{
   public abstract double Operate(Operation op, double[] operands);
}

public abstract class VisualCalculator : CalculatorBase
{
   public abstract FrameworkElement Operate(Operation op,
      double[] operands);
}

public abstract class Operation
{
   public abstract string Name { get; }
   public abstract int NumOperands { get; }
}
```

Host View

```
[HostAdapter]
public class CalculatorContractToViewHostAdapter : HostView.Calculator {
    private Calculator.Contracts.ICalculatorContract _contract;
    private ContractHandle _handle;

    public CalculatorContractToViewHostAdapter(
      Calculator.Contracts.ICalculatorContract contract) {
        _contract = contract;

        // The ContractHandle is critical to lifetime management. If you fail to
        // keep a reference to the ContractHandle object, garbage collection will
        // reclaim it, and the pipeline will shut down when
        // your program does not expect it.
        _handle = new ContractHandle(_contract);
    }

    public override string Name { get { return _contract.GetName(); } }

    public override IList<HostView.Operation> Operations {
        get {
            return CollectionAdapters.ToIList(
                _contract.GetOperations(),
                OperationHostAdapters.ContractToViewAdapter,
                OperationHostAdapters.ViewToContractAdapter); }
    }

    public override double Operate(HostView.Operation op, double[] operands) {
        return _contract.Operate(
            OperationHostAdapters.ViewToContractAdapter(op), operands);
    }
}
```

Adapter with UI

```
// ADD IN ADAPTER ====================================================
public INativeHandleContract Operate(
   Calculator.Contracts.IOperationContract op,
   double[] operands)
{
    return FrameworkElementAdapters.ViewToContractAdapter(
     _view.Operate(
       OperationViewToContractAddInAdapter.ContractToViewAdapter(op),
       operands));
}
```

```
// HOST ADAPTER ===================================================
public override FrameworkElement Operate(
   HostView.Operation op,
   double[] operands)
{
    return FrameworkElementAdapters.ContractToViewAdapter(
     _contract.Operate(
       OperationHostAdapters.ViewToContractAdapter(op),
       operands));
}
```

```
AddInStore.Rebuild(path);

var tokens =
  AddInStore.FindAddIns(typeof(Calculator), path);
Var visualTokens =
  AddInStore.FindAddIns(typeof(VisualCalculator), path);

foreach (var token in tokens)
{
  _calcs.Add(token.Activate<CalculatorBase>(
    AddInSecurityLevel.FullTrust));
}

foreach (var token in visualTokens)
{
  _calcs.Add(token.Activate<CalculatorBase>(
    AddInSecurityLevel.FullTrust));
}
```

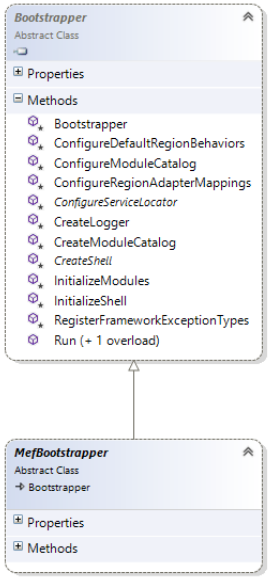
Host

Use overloads of *Activate* to controll add in isolation

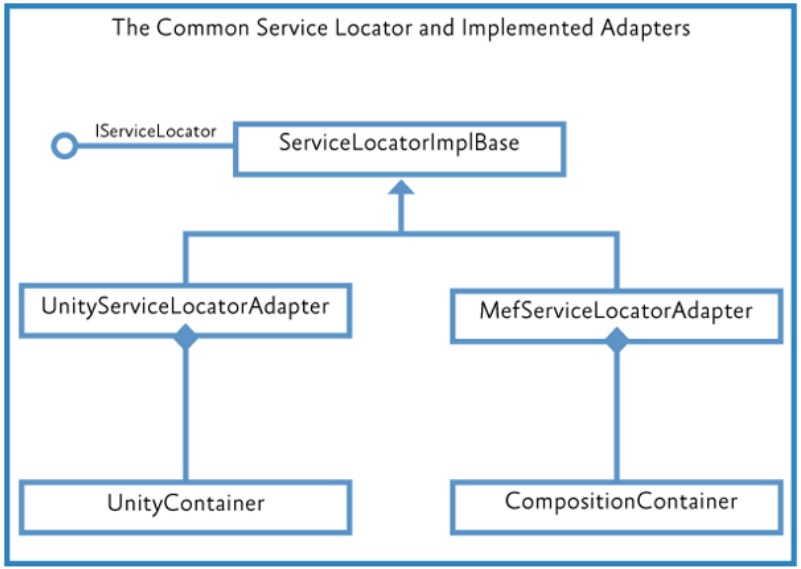
For details see MSDN

Bootstrapping Process

- Call bootstrapper in WPF application startup routine
  Create a class derived indirectly from Prism's *Bootstrapper* classes *MefBootstrapper* or *UnityBootstrapper*
- Setup module catalog
- Setup dependency injection container
  Here: MEF, Option: Unity, you have to decide
- Create the shell



Service Locator




```
...public interface IServiceLocator : IServiceProvider
{
    ...IEnumerable<TService> GetAllInstances<TService>();
    ...IEnumerable<object> GetAllInstances(Type serviceType);
    ...TService GetInstance<TService>();
    ...TService GetInstance<TService>(string key);
    ...object GetInstance(Type serviceType);
    ...object GetInstance(Type serviceType, string key);
}
```

Use *IServiceLocator* where you need container-agnostic code

Prism uses *IServiceLocator* internally

It is <u>not</u> generally recommended to prefer *IServiceLocator* over direct use of your specific container

```
using System.Windows;

namespace PrismDemoApp
{
  public partial class App : Application
  {
    protected override void OnStartup(StartupEventArgs e)
    {
      base.OnStartup(e);

      var bootstrapper = new AppBootstrapper();
      bootstrapper.Run();
    }
  }
}
```

Bootstrapping

Call bootstrapper in WPF application startup routine

Bootstrapping

Prism Code Walkthrough

```
public override void Run(bool runWithDefaultConfiguration){
    …
    this.ModuleCatalog = this.CreateModuleCatalog();
    …
    this.ConfigureModuleCatalog();
    …
    this.AggregateCatalog = this.CreateAggregateCatalog();
    …
    this.ConfigureAggregateCatalog();

    this.RegisterDefaultTypesIfMissing();
    …
    this.Container = this.CreateContainer();
    …
    this.ConfigureContainer();
    …
    this.ConfigureServiceLocator();
    …
    this.ConfigureRegionAdapterMappings();
    …
    this.ConfigureDefaultRegionBehaviors();
    …
    this.RegisterFrameworkExceptionTypes();
    …
    this.Shell = this.CreateShell();
    if (this.Shell != null) {
        …
        RegionManager.SetRegionManager(this.Shell, this.Container.GetExportedValue<IRegionManager>());
        …
        RegionManager.UpdateRegions();
        …
        this.InitializeShell();
    }

    IEnumerable<Lazy<object, object>> exports = this.Container.GetExports(typeof(IModuleManager), null, null);
    if ((exports != null) && (exports.Count() > 0)) {
        …
        this.InitializeModules();
    }
    …
}
```

Setup Module Catalog

- *Bootstrapper.CreateModuleCatalog*
  Default: Create empty *ModuleCatalog*
  Override it to create your custom instance of *IModuleCatalog*

- Create module catalog
  In Code
  ModuleCatalog.AddModule
  From XAML
  ModuleCatalog.CreateFromXaml
  From app.config
  ConfigurationModuleCatalog
  From directory
  DirectoryModuleCatalog

Module Definition

In app.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="modules"
      type="Microsoft.Practices.Prism.Modularity.ModulesConfigurationSection,
            Microsoft.Practices.Prism"/>
  </configSections>
  <modules>
    <module assemblyFile="ModularityWithMef.Desktop.ModuleE.dll"
      moduleType="ModularityWithMef.Desktop.ModuleE, ModularityWithMef.Desktop.ModuleE,
         Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" moduleName="ModuleE„
         startupLoaded="false" />
    <module assemblyFile="ModularityWithMef.Desktop.ModuleF.dll"
      moduleType="ModularityWithMef.Desktop.ModuleF, ModularityWithMef.Desktop.ModuleF,
         Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" moduleName="ModuleF"
         startupLoaded="false">
      <dependencies>
        <dependency moduleName="ModuleE"/>
      </dependencies>
    </module>
  </modules>
</configuration>
```

```
protected override IModuleCatalog CreateModuleCatalog()
{
    return new DirectoryModuleCatalog() {
      ModulePath = @".\Modules"};
}
```

```
protected override void ConfigureAggregateCatalog()
{
    base.ConfigureAggregateCatalog();

    DirectoryCatalog catalog =
      new DirectoryCatalog("DirectoryModules");
    this.AggregateCatalog.Catalogs.Add(catalog);
}
```

Module Definition

Load modules from a directory

Load modules from directory with *DirectoryModuleCatalog*

Load modules from directory with MEF

Module Definition

In Code

```
protected override void ConfigureModuleCatalog()
{
    Type moduleCType = typeof(ModuleC);
    ModuleCatalog.AddModule(
      new ModuleInfo()
      {
          ModuleName = moduleCType.Name,
          ModuleType = moduleCType.AssemblyQualifiedName,
      });
}
```

Setup Dependency Injection

- Prism standard services:

Service interface	Description
IModuleManager	Defines the interface for the service that will retrieve and initialize the application's modules.
IModuleCatalog	Contains the metadata about the modules in the application. The Prism Library provides several different catalogs.
IModuleInitializer	Initializes the modules.
IRegionManager	Registers and retrieves regions, which are visual containers for layout.
IEventAggregator	A collection of events that is loosely coupled between the publisher and the subscriber.
ILoggerFacade	A wrapper for a logging mechanism, so you can choose your own logging mechanism. The Stock Trader Reference Implementation (Stock Trader RI) uses the Enterprise Library Logging Application Block, via the **EnterpriseLibraryLoggerAdapter** class, as an example of how you can use your own logger. The logging service is registered with the container by the bootstrapper's **Run** method, using the value returned by the **CreateLogger** method. Registering another logger with the container will not work; instead override the **CreateLogger** method on the bootstrapper.
IServiceLocator	Allows the Prism Library to access the container. If you want to customize or extend the library, this may be useful.

- Add application-specific services if needed

```
protected override void ConfigureContainer()
{
  base.ConfigureContainer();

  // Publish container using MEF
  this.Container.ComposeExportedValue<CompositionContainer>(
    this.Container);
}

protected override void ConfigureAggregateCatalog()
{
  base.ConfigureAggregateCatalog();
  this.AggregateCatalog.Catalogs.Add(
    new AssemblyCatalog(typeof(Shell).Assembly));
}
```

Dependency Injection

Here: MEF

Optional

- Override *CreateContainer* and *ConfigureContainer*
- Make sure to call base class' implementation to get standard services

Override

ConfigureAggregateCatalog

Create Shell (MEF)

```
protected override DependencyObject CreateShell()
{
  return this.Container.GetExportedValue<Shell>();
}

protected override void InitializeShell()
{
  Application.Current.MainWindow = this.Shell as Window;
  Application.Current.MainWindow.Show();
}
```

Create and initialize the shell

```
[ModuleExport(typeof(DataManagementModule),
   InitializationMode = InitializationMode.WhenAvailable)]
public class DataManagementModule : IModule
{
   public void Initialize()
   {
      […]
   }
   […]
}

// MEF
[ModuleExport(typeof(ModuleA), DependsOnModuleNames =
   new string[] { "ModuleD" })]
public class ModuleA : IModule
{
   […]
}

// Unity
[Module(ModuleName = "ModuleA")]
[ModuleDependency("ModuleD")]
public class ModuleA: IModule
{
    ...
}
```

Modules

Module Creation

Implement *IModule*

Register named modules in XAML, app.config, or code (see above)

Declarative metadata attributes for modules

Dependency management (incl. cycle detection)

Tip: Use *IModuleManager.LoadModuleCompleted* to receive information about loaded modules

IModule.Initialize

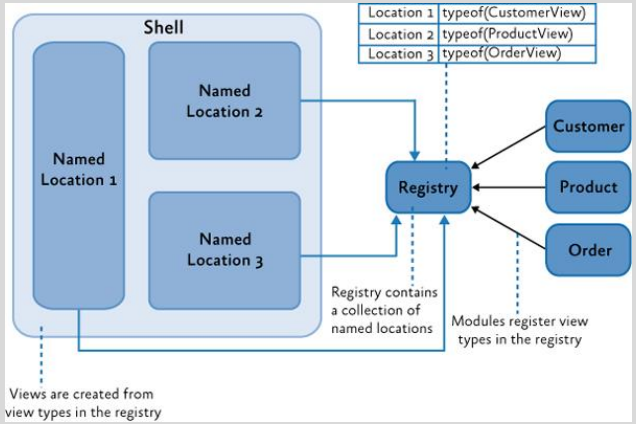
- Add the module's views to the application's navigation structure
- Subscribe to application level events or services
- Register shared services with the application's dependency injection container

Module Communication

- Loosely coupled events
  Event aggregation
- Shared services
  Registered in DI container
- Shared resources
  Database, files, web services

```
// View discovery using composition container
this.regionManager.RegisterViewWithRegion(
   "MainRegion", typeof(EmployeeView));
// …or delegate
this.regionManager.RegisterViewWithRegion(
   "MainRegion", () => this.container.Resolve<EmployeeView>());


// Add view in code
IRegion region = regionManager.Regions["MainRegion"];
var ordersView = container.Resolve<OrdersView>();
region.Add(ordersView, "OrdersView");
```



Loading Content Into Regions

View Discovery

RegisterViewWithRegion

Create view and display it when region becomes visible

View Injection

Obtain a reference to a region, and then programmatically adds a view into it

Navigation (see later)

Basic Navigation

```
IRegion mainRegion = ...;
mainRegion.RequestNavigate(
  new Uri("InboxView", UriKind.Relative));

// or
IRegionManager regionManager = ...;
regionManager.RequestNavigate(
  "MainRegion",
  new Uri("InboxView", UriKind.Relative));
```

```
[Export("InboxView")]
public partial class InboxView : UserControl
```

If you want ViewModel-first navigation, use Data Templates

Specify a callback that will be called when navigation is completed

Parameters

```
Employee employee = Employees.CurrentItem as Employee;
if (employee != null)
{
    UriQuery query = new UriQuery();
    query.Add("ID", employee.Id);
    _regionManager.RequestNavigate(RegionNames.TabRegion,
         new Uri("EmployeeDetailsView"
           + query.ToString(), UriKind.Relative));
}
```