ハンズオン ラボ

Managed Extensibility Framework の概要

ラボ バージョン:　　1.0.0

最終更新日:　2010 年 2 月 9 日

目次

概要

Managed Extensibility Framework (MEF) を使用すると、開発者は、製品の開発企業やサードパーティ企業が提供する拡張機能へのフックを、.NET アプリケーションに用意することができます。MEF は汎用アプリケーション拡張機能と言えます。

開発者は MEF を使って、アプリケーション自体を拡張したり、他の専門知識を要する拡張を必要としないで、動的に拡張機能を作成できます。その結果、コンパイル時にアプリケーションと拡張機能に結び付きがなくても、再コンパイルすることなく、実行時にアプリケーションが拡張されます。MEF では、アプリケーションに拡張機能を読み込む前に、拡張機能アセンブリのメタデータを調べることも可能です。これは従来よりはるかに迅速な手法です。

このラボで説明する拡張機能関連の主要概念は以下のとおりです。

- **コンポジション**: 独自の機能を備えたオブジェクトを複数組み合わせて、1 つ以上のの複雑なオブジェクトに組み立てる手法です。コンポジションでは、親クラスから機能を継承するのではなく、複数の異なるオブジェクトを 1 つに組み立てます。たとえば、**Wing** オブジェクト、**Propeller** オブジェクト、**Fuselage** オブジェクト、**VerticalStablizer** オブジェクトが組み合わされて **Aircraft** オブジェクトの一部になります。
- **ComposableParts**: MEF の主要ビルド ブロックです。**ComposableParts** により、アプリケーションは "インポート" と "エクスポート" を使って、コンポーネント拡張機能の公開や利用が可能になります。
- **コントラクト**: インポートおよびエクスポートするコンポーネントとの通信手段です。コントラクトは、通常、**Interface** クラスを使って実装されます。MEF **ComposableParts** ではこのコントラクトを使って、他のコンポーネントと依存関係が生じることや、他のコンポーネントと緊密に結び付くことを回避しています。
- **条件付きバインド**: 特定のメタデータ条件に合ったコンポーネントだけを読み込めるようにします。上記の Aircraft オブジェクトを例にとると、グラファイト複合材だけで作られた **VerticalStabilizer** コンポーネントを選択して読み込みことなどが考えられます。

拡張の主な作業は、アプリケーションの拡張ポイントに "インポート" 属性を追加し、対応する "エクスポート" 属性を拡張機能に追加することです。インポートとエクスポートは、サプライヤーとコンシューマーの関係と言えます。エクスポート側のコンポーネントがなんらかの値を供給し、インポート側のコンポーネントがその値を利用します。まったく独自の拡張手法も含め、開発者には選択できる多くの拡張オプションがありますが、このラボでは今述べた基本的な手法に重点を置いて説明します。

目的

このハンズオン ラボでは、次のことを行う方法について学習します。

- コンポーネントに対する拡張ポイントを定義する
- 条件付きバインドとコンポーネントの作成を行う
- アプリケーション実行中に拡張アセンブリをインポートする

システム要件

このラボには、次のものが必要です。

- Microsoft Visual Studio 2010
- .NET Framework 4

セットアップ

> **メモ:** 日本語環境でこのラボを実行する場合は下記の Read Me を参考にして、セットアップを実行してください。
>
> http://msdn.microsoft.com/ja-jp/netframework/ff384798.aspx

構成ウィザード (Configuration Wizard) を使用すると、このラボの要件がすべて確認されます。すべての要件が正しく構成されていることを確認するには、次の手順を実行します。

> **メモ:** セットアップ手順を実行するには、管理者特権を使ってコマンド ウィンドウからスクリプトを実行する必要があります。

1. トレーニング キットの構成ウィザードを以前に実行していなければ、実行します。これには、**%TrainingKitInstallationFolder%\Labs\IntroToMEF\Source\Setup** フォルダーの **CheckDependencies.cmd** スクリプトを実行します。前提条件を満たしていなければ、必要な項目をすべてインストールし (必要に応じて再スキャンし)、ウィザードを完了します。

   > **メモ:** 便宜上、このラボで管理するコードの大半は、Visual Studio のコード スニペットとして使用できるようにしています。**CheckDependencies.cmd** ファイルによって Visual Studio インストーラー ファイルが起動し、コード スニペットがインストールされます。
   >
   > 複数のバージョンの Visual Studio がインストールされている場合、対象のコード スニペットをすべて選択した上で、インストール先に Visual Studio のバージョンを選択してください。

演習

このハンズオン ラボは以下の演習から構成されています。

1. MEF を使用したアプリケーションへのモジュールの動的追加
2. フォームの動的拡張

演習の教材

このハンズオン ラボには次の教材が含まれています。

- **Visual Studio ソリューション**: 演習の出発点として使用するため、Visual Studio ソリューションを演習ごとに用意しています。

行き詰まったら

このハンズオン ラボに付属するソース コードには end フォルダーがあり、各演習を修了すると完成する最終的な Visual Studio ソリューションが含まれています。演習中に支援が必要になった場合は、このソリューションをガイドとして利用できます。

ラボの推定所要時間: **30 分**

次の手順

演習 1: MEF を使用したアプリケーションへのモジュールの動的追加

演習 1: MEF を使用したアプリケーションへのモジュールの動的追加

Managed Extensibility Framework (MEF) の実用的な使用法の 1 つは、アプリケーションの実行時にモジュールを追加することです。この方法は、最初に選択して購入した特定のモジュールに、または本来インストールされていたモジュールに、後から新たなモジュールを追加するようなシナリオに有効です。MEF を使用すると、既知のディレクトリを監視し、そのディレクトリ内で見つかった任意のモジュール アセンブリを追加するようにアプリケーションを構成できます。モジュール アセンブリをそのディレクトリにドロップすれば、明示的な参照設定がなくても、アプリケーションにそのモジュール アセンブリが読み込まれます。

タスク 1 – メイン フォームを更新してサポート対象のモジュールを読み込む

このタスクでは既存のフォームにコードを追加して拡張機能のフックを作成し、後続の演習で作成するクラスを動的にインポートします。その際、構成済みのメタデータと一致するクラスだけがインポートされるようにします。

MainForm.cs 内で定義するメイン フォームには、ファイル システムからモジュールを読み取る手段が必要です。この演習では例を簡単にするため、フォームの初期読み込み時にモジュールをインポートします。

1. Microsoft Visual Studio 2010 を起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010]、[Microsoft Visual Studio 2010] の順にクリックします。
2. **MefLab.sln** ソリューション ファイルを開きます。既定では、**%TrainingKitInstallFolder%\Labs\IntroToMEF\Source\Ex01-DynamicallyAddModules\begin** 以下の **C#** フォルダもしくは **VB** フォルダにあります。お好きな言語を選択して使用してください。
3. ソリューション エクスプローラーで **MainForm** を右クリックし、[View Code] (コードの表示) を選択し、コード ビューで **MainForm** を開きます。
4. このフォームを MEF ライブラリを使用するように更新します。そのためには、**MainForm** のクラス定義の先頭に次のステートメントを追加します。

C#

```
using System.ComponentModel.Composition;
using System.ComponentModel.Composition.Hosting;
```

Visual Basic

```
Imports System.ComponentModel.Composition
Imports System.ComponentModel.Composition.Hosting
```

> **メモ**: MEF では、コントラクトを使用してインポートとエクスポートを組み立てます。これらのコントラクトはインターフェイスによって指定され、どの拡張ポイント (インポート側) がどのエクスポート側コンポーネントと関連付けられるかを定義します。**MainForm** では、**MefCommon** プロジェクト内で定義された **IMainFormContract** を実装しているモジュールをインポートします。これらのモジュールをインポートするには、プロパティ経由でモジュールの保持と公開を行うコレクションを定義します。MEF のコンポジション段階で、適用可能なインポートとエクスポートが解決され、このコレクションが読み込まれます。

5. MainForm.cs (C#) もしくは MainForm.vb (VB) ファイルのコンストラクターの直前に次のコードを追加します。

(コード スニペット – Intro to MEF Lab - Ex1 ImportedMainFormContracts - CSharp)

C#

```
[ImportMany]
public Lazy<IMainFormContract, IDictionary<string, object>>[] ImportedMainFormContracts { get;
set; }
```

(コード スニペット – Intro to MEF Lab - Ex1 ImportedMainFormContracts VB)

Visual Basic

```
<ImportMany()>
Public Property ImportedMainFormContracts() As Lazy(Of IMainFormContract, IDictionary(Of
String, Object))()
```

6. インポートとエクスポートの値と一致するコンポーネントを検索するために MEF の **CompositionContainer** が使用されます。**CompositionContainer** によって、コンポーネントとその値の読み込み、バインド、取得が可能になります。**MainForm** クラスの冒頭に次の変数宣言を追加して、グローバルな **CompositionContainer** を作成します。

C#

```
private CompositionContainer _container;
```

Visual Basic

```
Private _container As CompositionContainer
```

7. ここで、フォルダーからオブジェクトをインポートするヘルパー関数を作成する必要があります。このフォルダーは、**MainForm.cs** ファイル内で既に **_extensionDir** 変数を使って指定してあります。**MainForm.cs** ファイル内の **MainForm** クラスの末尾に次のコードを追加します。

(コード スニペット – Intro to MEF Lab - Ex1 GetContainerFromDirectory - CSharp)

C#

```
private CompositionContainer GetContainerFromDirectory()
{
    var catalog = new AggregateCatalog();
    var thisAssembly =
       new AssemblyCatalog(
          System.Reflection.Assembly.GetExecutingAssembly());
    catalog.Catalogs.Add(thisAssembly);
```

```
    catalog.Catalogs.Add(
      new DirectoryCatalog(_extensionDir));

    var container = new CompositionContainer(catalog);
    return container;
}
```

(コード スニペット – Intro to MEF Lab - Ex1 GetContainerFromDirectory VB)

Visual Basic

```
Private Function GetContainerFromDirectory() As CompositionContainer
   Dim catalog As AggregateCatalog = New AggregateCatalog()
   Dim thisAssembly As AssemblyCatalog = New
AssemblyCatalog(System.Reflection.Assembly.GetExecutingAssembly())
   catalog.Catalogs.Add(thisAssembly)

   catalog.Catalogs.Add(New DirectoryCatalog(_extensionDir))

   Dim container As CompositionContainer = New CompositionContainer(catalog)
   Return container
End Function
```

8. 次に、読み込まれたコンテナーを使用する必要があります。共通の MEF パターンに従って、**Compose** メソッドを使用して、すべての適用可能なインポート操作とエクスポート操作を MEF によって解決します。**GetContainerFromDirectory** メソッドの下に次のメソッドを追加します。

   (コード スニペット – Intro to MEF Lab - Ex1 Compose - CSharp)

   C#

```
private bool Compose()
{
   _container = GetContainerFromDirectory();

   try
   {
      _container.ComposeParts(this);
   }
   catch (CompositionException compException)
   {
      MessageBox.Show(compException.ToString());
      return false;
   }

   return true;
}
```

(コード スニペット – Intro to MEF Lab - Ex1 Compose VB)

```
Visual Basic
Private Function Compose() As Boolean
   _container = GetContainerFromDirectory()

   Try
      _container.ComposeParts(Me)
   Catch compException As CompositionException
      MessageBox.Show(compException.ToString())
      Return False
   End Try

   Return True
End Function
```

9. フォームのコンストラクターから **Compose** メソッドを呼び出します。次のようにコンストラクターのコードを追加します。

(コード スニペット – Intro to MEF Lab - Ex1 Compose call Charp)

```
C#
public MainForm()
{
   InitializeComponent();
   bool successfulCompose = Compose();
   if (!successfulCompose)
   {
      this.Close();
   }
}
```

(コード スニペット – Intro to MEF Lab - Ex1 Compose call VB)

```
Visual Basic
Public Sub New()
   InitializeComponent()

   Dim successfulCompose As Boolean = Me.Compose()
   If (Not successfulCompose) Then
      Me.Close()
   End If
End Sub
```

タスク 2 – 拡張モジュールを作成する

このタスクでは、フォームで従業員を管理する、既存の Windows フォーム プロジェクトで作業します。MEF を使用して、このフォームを別のアプリケーションにエクスポート可能と

してマークします。ここでは時間を節約するために、事前に作成済みのクラスをいくつか含む **MefEmployeeModule** プロジェクトを用意しています。

1. 先に進む前に、MEF ライブラリを参照する必要があります。**MefEmployeeModule** プロジェクトに MEF ライブラリへの参照を追加します。

    a. ソリューション エクスプローラーで MefEmployeeModule プロジェクトを選択し、[Project] (プロジェクト)、[Add Reference...] (参照の追加...) の順にクリックして [Add References] (参照の追加) ダイアログ ボックスを表示します。

    b. [.NET] タブをクリックします。

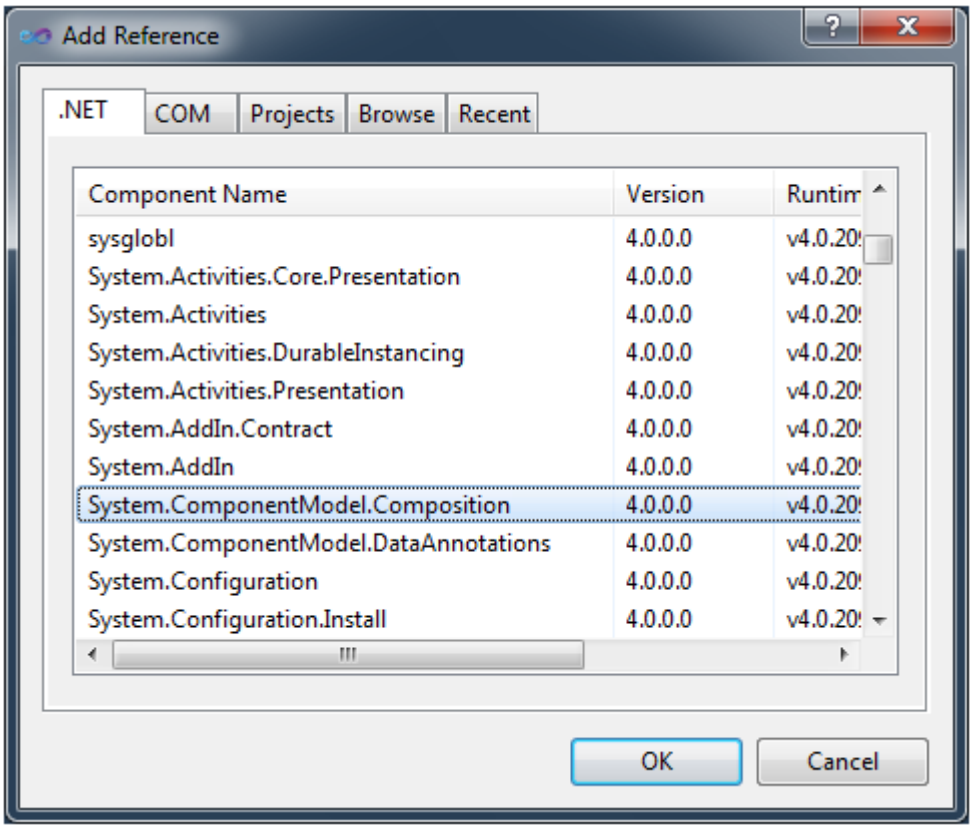

図 1

MEF の参照

    c. **System.ComponentModel.Composition** コンポーネントを選択します。[OK] をクリックして、このライブラリへの参照を追加します。

> **メモ: System.ComponentModel.Composition** コンポーネントが見つからない場合は、[Browse] (参照) タブをクリック

し、**%SystemRoot%\Microsoft.net\Framework\v4.0.{ビルド番号}** フォルダー、または これに対応する .NET Framework 4.0 フォルダーでアセンブリを検索してください。

2. また、対応する using ステートメントを使用するようクラス ファイルも更新します。コード ビューで **EmployeeMaintenance** フォームを開き、ファイル冒頭の using コード ブロックに次の行を追加します。

C#

```
using System.ComponentModel.Composition;
```

Visual Basic

```
Imports System.ComponentModel.Composition
```

3. この **EmployeeMaintenance** フォームが、メイン フォームに追加することになるモジュールです。**IMainFormContract** コントラクトを実装するコンポーネントだけを取得するために、**MainForm** で **ImportMany** 属性を使用したことを思い出してください。そのコントラクトに従い、**EmployeeMaintenance** フォームを指定します。クラス宣言を次のように変更します。

C#

```
[Export(typeof(IMainFormContract))]
public partial class EmployeeMaintenance : Form, IMainFormContract
```

Visual Basic

```
<Export(GetType(IMainFormContract))>
Public Class EmployeeMaintenance
```

4. **IMainFormContract** では、実装する必要がある 2 つのプロパティを指定します。**EmployeeMaintenance** コンストラクターの上に次のコードを追加します。

(コード スニペット – Intro to MEF Lab - Ex1 Implement IMainFormContract - CSharp)

C#

```
public string MenuItemText
{
   get { return "&Employees"; }
}

public string SubFormTitle
{
```

```
    get { return "Employee Pane"; }
}
```

(コード スニペット – Intro to MEF Lab - Ex1 Implement IMainFormContract VB)

Visual Basic

```
Public ReadOnly Property MenuItemText() As String Implements
MefCommon.IMainFormContract.MenuItemText
    Get
        Return "&Employees"
    End Get
End Property

Public ReadOnly Property SubFormTitle() As String Implements
MefCommon.IMainFormContract.SubFormTitle
    Get
        Return "Employee Pane"
    End Get
End Property
```

5. **MainForm** クラスをインポートする際に使用するメタデータでこのクラスを修飾します。MEF により、さまざまな箇所でこのメタデータの照会が可能になります。次のように **ExportMetaData** 属性を追加します。

(コード スニペット – Intro to MEF Lab - Ex1 ExportMetaData attributes - CSharp)

C#

```
[Export(typeof(IMainFormContract))]
[ExportMetadata("Name", "Employee Pane")]
[ExportMetadata("MenuText", "&Employees")]
public partial class EmployeeMaintenance : Form, IMainFormContract
```

(コード スニペット – Intro to MEF Lab - Ex1 ExportMetaData attributes VB)

Visual Basic

```
<Export(GetType(IMainFormContract))>
<ExportMetadata("Name", "Employee Pane")>
<ExportMetadata("MenuText", "&Employees")>
Partial Public Class EmployeeMaintenance
```

6. フォームを保存し、MefEmployeeModule プロジェクトをコンパイルします。[Build] (ビルド) メニューの [Build MefEmployeeModule] (MefEmployeeModule のビルド) をク

リックします。これで MefEmployeeModule アセンブリがコンパイルされ、MEF を使って別のアプリケーションにインポートできるようになります。

タスク 3 – EmployeeMaintenance フォームをインポートする

このタスクでは、アプリケーションのメイン フォームを更新して、アプリケーションの実行中にユーザーがメニュー項目をクリックしたときに、利用可能なモジュールをインポートします。

1. ソリューション エクスプローラーで、**MefLabMain** プロジェクトの **MainForm** をダブルクリックして、デザイナー ビューで **MainForm** を開きます。
2. ここで、**CompositionContainer** 内のコンポーネントを調べ、メニューのテキスト情報をエクスポートするコンポーネントを特定します。エクスポートされたメタデータに基づいてメニュー項目を読み込むようにフォームを変更します。**Compose** メソッドで **IMainFormContract** をサポートするモジュールの一覧が読み込まれるため、そのコレクションをループ処理してメニューに追加する項目を集めます。必要に応じてメニューを変更できるように、フォームの **Load** イベントのイベント ハンドラーを作成します。フォームの中央をダブルクリックします。新しく **MainForm_Load** メソッドを定義するコード ビュー ウィンドウが表示されます。次のコードを追加して、コレクションを反復処理して、必要に応じてメニュー項目を更新します。同時に、新しいメニュー項目を処理するイベント ハンドラーのフックも行います。

   (コード スニペット – Intro to MEF Lab - Ex1 Populate menu - CSharp)

C#

```
private void MainForm_Load(object sender, EventArgs e)
{
   foreach (var export in this.ImportedMainFormContracts)
   {
      var exportedMenuText = export.Metadata["MenuText"] as string;
      if (String.IsNullOrEmpty(exportedMenuText))
      {
         return;
      }
      ToolStripItem menuItem =
         modulesToolStripMenuItem.DropDownItems.Add(exportedMenuText);
      menuItem.Click += new System.EventHandler(this.LaunchModule_Click);
   }
}
```

(コード スニペット – Intro to MEF Lab - Ex1 Populate menu VB)

Visual Basic

```
Private Sub MainForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
   For Each export As Lazy(Of IMainFormContract, IDictionary(Of String, Object)) In
Me.ImportedMainFormContracts
      Dim exportedMenuText = TryCast(export.Metadata("MenuText"), String)

      If String.IsNullOrEmpty(exportedMenuText) Then
         Return
      End If

      Dim menuItem As ToolStripItem =
modulesToolStripMenuItem.DropDownItems.Add(exportedMenuText)
      AddHandler menuItem.Click, AddressOf LaunchModule_Click
   Next export
End Sub
```

> **メモ:** 上記ようにアプリケーションのプレゼンテーション層にロジックを混在させることは、ソフトウェア エンジニアリング手法としては不適切です。テストが困難になり、複数の懸案事項が混ざり合い、信頼できる他の設計原則の多くに違反することになります。
>
> ここでは、演習の例を簡潔かつ明確にするために、こうした原則を破り、**MainForm_Load** メソッドのようなフォームのメソッドにロジックを混在させています。実際のアプリケーションでは、モデル ビュー プレゼンター (MVP) のような手法を使って、ロジックを別のクラスに分離してください。柔軟性や保守性が高く、テストを実行しやすいアプリケーションを作成するためにも、設計原則として MVP とその関連テクノロジについて学習することをお勧めします。

3. 次に、メニュー操作からコンポーネントを起動する機能を追加します。これには、先ほどと同じ手法を使います。インポートされるコンポーネントの一覧で読み込まれるコンポーネントを調べ、そのメタデータをチェックして適用可能な項目を探します。**Value** プロパティによって特定のコンポーネントを起動できます。**MainForm** クラス定義の最後に次のメソッドを追加します。このメソッドは **MainForm_Load** メソッドに結び付けられ、動的に追加されたメニューのクリック イベントを処理します。

(コード スニペット – Intro to MEF Lab - Ex1 Menu Click handler - CSharp)

C#

```
private void LaunchModule_Click(object sender, EventArgs e)
{
   ToolStripItem thisItem = sender as ToolStripItem;
   if (thisItem == null) { return; }

   string thisItemTitle = thisItem.Text;

   foreach (var export in this.ImportedMainFormContracts)
   {
      string menuTitle = export.Metadata["MenuText"] as string;
      if (String.IsNullOrEmpty(menuTitle))
      {
         return;
      }
      if (menuTitle == thisItemTitle)
      {
         Form frm = export.Value as Form;
         if (frm == null) { return; }
         frm.Show(this);
         return;
      }
   }
}
```

(コードスニペット – Intro to MEF Lab - Ex1 Menu Click handler VB)

Visual Basic

```
Private Sub LaunchModule_Click(ByVal sender As Object, ByVal e As EventArgs)
   Dim thisItem As ToolStripItem = TryCast(sender, ToolStripItem)
   If thisItem Is Nothing Then
      Return
   End If

   Dim thisItemTitle As String = thisItem.Text

   For Each export As Lazy(Of IMainFormContract, IDictionary(Of String, Object)) In
Me.ImportedMainFormContracts
      Dim menuTitle As String = TryCast(export.Metadata("MenuText"), String)

      If String.IsNullOrEmpty(menuTitle) Then
         Return
      End If

      If menuTitle = thisItemTitle Then
         Dim frm As Form = TryCast(export.Value, Form)
         If frm Is Nothing Then
            Return
         End If
         frm.Show(Me)
         Return
      End If

   Next export
End Sub
```

次の手順

演習 1: 確認

演習 1: 確認

この確認では、インポートしたモジュールを使用する場合と使用しない場合の両方でアプリケーションを実行します。

1. 次のビルド後のコマンドを追加して、アドイン出力用フォルダーを作成します。これを行うには、**MefEmployeeModule** プロジェクトを右クリックして、[Properties] (プロパティ) をクリックします。
2. [Build Events] (ビルド イベント) タブをクリックして、[Post-build event command line] (ビルド後に実行するコマンド ライン) フィールドに次のコードを貼り付けます。ホワイトスペースを含むディレクトリ名が適切に処理されるよう、必ず二重引用符も含めてください。

   ビルド後のコマンド

   ```
   if not exist "$(SolutionDir)MefLabMain\bin\Debug\ExtModules" mkdir
   "$(SolutionDir)MefLabMain\bin\Debug\ExtModules"
   ```

3. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをコンパイルします。
4. **MefLabMain** をスタートアップ プロジェクトに設定します。ソリューション エクスプローラーで **MefLabMain** を右クリックし、[Set as StartUp Project] (スタートアップ プロジェクトに設定) をクリックします。
5. **F5** キーを押してアプリケーションを実行します。上部にメニューのある親ウィンドウが表示されます。この時点では [Modules] メニューにはメニュー項目がありません。

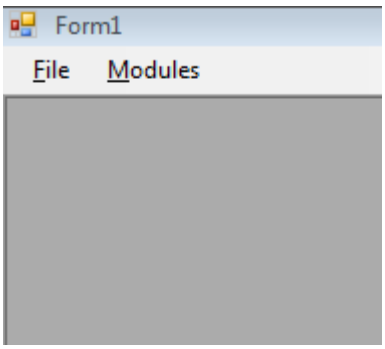

図 1

メニュー項目のない [Modules] メニュー

6. アプリケーションを終了します。[File]、[Exit] の順にクリックします。

7. **MefLabMain\bin\Debug\ExtModules** ディレクトリへの変更を監視するようにアプリケーションをビルドしています。プロジェクトのビルド後のコマンドで作成したフォルダーに、出力バイナリ ファイルを自動的に保存するように、**MefEmployeeModule** プロジェクトを更新します。次の太字コードをビルド後のコマンド ラインに追加します。

ビルド後のコマンド

```
if not exist "$(SolutionDir)MefLabMain\bin\Debug\ExtModules" mkdir
"$(SolutionDir)MefLabMain\bin\Debug\ExtModules"
copy "$(TargetPath)" "$(SolutionDir)MefLabMain\bin\Debug\ExtModules"
```

8. **F5** キーを押してアプリケーションを実行します。上部にメニューのある親ウィンドウが表示されます。今度は [Modules] メニューに [Employees] メニュー項目が表示されます。

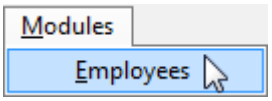

図 2

読み込んだモジュールを使用する [Modules] メニュー

9. メニュー項目の [Employees] をクリックします。EmployeeMaintenance フォームが表示されます。

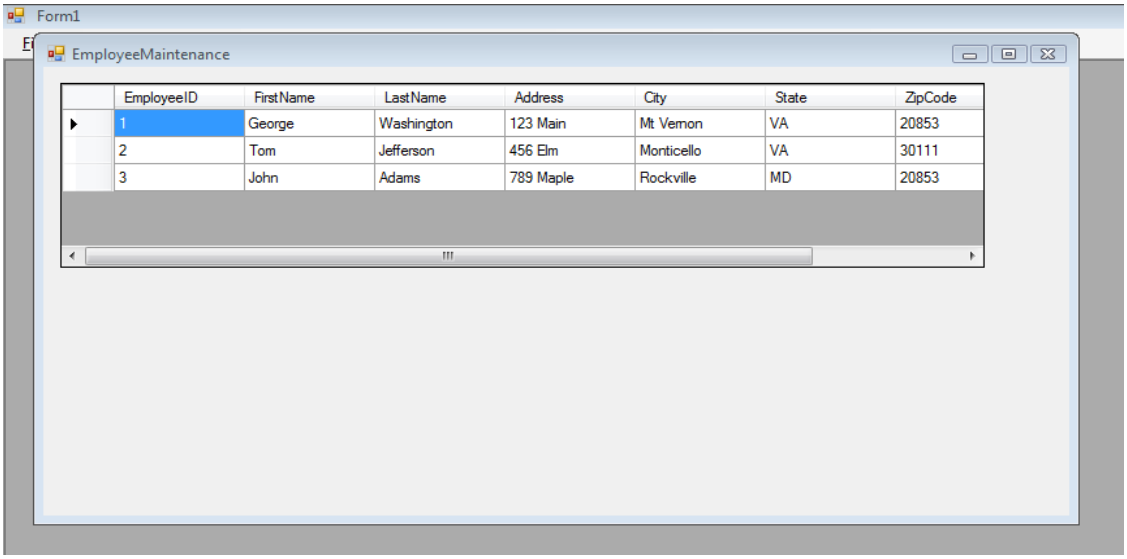

図 3

EmployeeMaintenance フォーム

> **メモ:** モジュール読み込み時に **CompositionExceptions** がスローされる場合は、通常、ExtModules フォルダーに必要なファイルがコピーされていません。

10. Employee モジュールは、明示的に参照を設定しなくても、親プロジェクトから呼び出されました。MEF がすべてのアセンブリの読み込みを処理し、まったく関係付けられていない一連のコンポーネントの作成を可能にしています。
11. フォーム右上の閉じるボタン ( ) をクリックして **EmployeeMaintenance** フォームを閉じます。
12. [File]、[Exit] の順にクリックして親フォームを閉じます。

次の手順

演習 2: フォームの動的拡張

演習 2: フォームの動的拡張

前の演習では、既知のディレクトリにアセンブリを単純にドロップすることで、アプリケーションに新しい従業員管理フォームを動的に追加しました。今度は MEF を使って、この新しく追加したフォームを拡張し、フォームに追加機能を提供するフォーム コントロールを動的に読み込みます。

アプリケーションは拡張機能の有無にかかわらず動作しなければなりません。そのため、EmployeeMaintenance フォームから拡張される機能は疎結合にする必要があります。しかし、拡張機能とアプリケーションは相互の内部動作を意識してはいけません。これを可能にするには、**Interface** 経由のコントラクトを使用します。

この演習では、このラボの主要目的に注目できるように、事前にビルドしたクラスをいくつか使用します。

この演習では、従業員のリストを操作するコマンド ボタンを動的に追加することで **EmployeeMaintenance** フォームを拡張します。**ICmdButtonInfo** という既存のインターフェイスを使用します。これにはコマンド ボタンと、エクスポートされたオブジェクトの継承元になる **CompanyInfo** オブジェクトが含まれています。

タスク 1 – AddEmployeeButton コントロールをエクスポート可能としてマークする

MEFEmployeeExtender プロジェクトには、実行時に EmployeeMaintenance フォーム上に動的に読み込まれるコントロールが含まれています。このラボでは、**ICmdButtonInfo** コントラクトを実装する、**AddEmployeeButton** コントロールと **DeleteEmployeeButton** コントロールを変更します。

このタスクでは、**AddEmployeeButton** を変更して、その情報を **ICmdButtonInfo** コントラクト経由でエクスポートします。

1. Microsoft Visual Studio 2010 を起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010]、[Microsoft Visual Studio 2010] の順にクリックします。
2. **MefLab.sln** ソリューション ファイルを開きます。既定では、**%TrainingKitInstallFolder%\Labs\IntroToMEF\Source\Ex02-**

DynamicallyExtendAForm\begin以下の **C#** フォルダもしくは **VB** フォルダにあります。お好きな言語を選択して使用してください。前の演習で作成したソリューションから作業を続行することもできます。

3. **EmployeeMaintenance** フォームで **AddEmployeeButto**n を使用するため、コントラクト経由でエクスポート可能としてマークする必要があります。AddEmployeeButton.cs クラスを変更して、エクスポート可能としてマークする **Export** 属性を設定します。

C#

```
[Export(typeof(ICmdButtonInfo))]
public class AddEmployeeButton : ICmdButtonInfo
```

Visual Basic

```
<Export(GetType(ICmdButtonInfo))>
Public Class AddEmployeeButton
```

4. ファイルを保存します。

タスク 2 – DeleteEmployeeButton コントロールをエクスポート可能としてマークする

1. **AddEmployeeButton** と同様、**DeleteEmployeeButton** コントロールのエクスポートも **ICmdButtonInfo** コントラクト経由で行うように MEF に指示する必要があります。DeleteEmployeeButton クラスを変更して、エクスポート可能としてマークする **Export** 属性を設定します。

C#

```
[Export(typeof(ICmdButtonInfo))]
public class DeleteEmployeeButton: ICmdButtonInfo
```

Visual Basic

```
<Export(GetType(ICmdButtonInfo))>
Public Class DeleteEmployeeButton
```

2. ファイルを保存します。

タスク 3 – EmployeeMaintenance フォームに Panel コントロールを追加する

このタスクでは、EmployeeMaintenance フォームに **Panel** コントロールを追加します。後でこのフォームに動的にボタンを追加します。ボタンを Panel コントロール上に配置することで、ボタンの整列が容易になります。

1. ソリューション エクスプローラーで、**MEFEmployeeModule** プロジェクトの EmployeeMaintenance ファイルをダブルクリックして、EmployeeMaintenance フォームをデザイナーで開きます。
2. ツールボックス パネルが表示されていなければ、[View] (ビュー) メニューの [Toolbox] (ツールボックス) をクリックします。
3. ツールボックスの [Containers] (コンテナー) タブから **Panel** コントロールをフォーム上にドラッグします。
    a. **DataGridView** の下に Panel コントロールを配置します。
    b. Panel コントロールの端をドラッグして、**DataGridView** の幅と同じになるまで広げます。
    c. Panel コントロールの名前を "ButtonsPanel" に変更します。

タスク 4 – EmployeeMaintenance フォームにコードを追加してエクスポートされたオブジェクトを読み込む

このタスクでは、EmployeeMaintenance フォームの **Load** イベント ハンドラーにコードを追加します。追加するコードでは、既存のディレクトリにエクスポートされるコントロールを含むアセンブリがあるかどうかチェックします。これらのコントロールが、実行時にフォームに動的に追加されます。

1. ソリューション エクスプローラーで EmployeeMaintenance クラスを右クリックして開き、[View Code] (コードの表示) をクリックします。
2. 拡張されたオブジェクトを保持するコレクションを追加します。拡張されたオブジェクトはこの特定のクラスにインポートされるため、このコレクションをインポートされるオブジェクトとして参照します。**ICmdButtonInfo** コントラクトに一致するオブジェクトのみをインポートします。**EmployeeMaintenance** クラスの先頭に、次のコードを追加します。

(コード スニペット – Intro to MEF Lab - Ex2 ImportedButtons - CSharp)

C#

```
[ImportMany]
public Lazy<ICmdButtonInfo>[] ImportedButtons
{
   get;
   set;
}
```

(コード スニペット – Intro to MEF Lab - Ex2 ImportedButtons - VB)

Visual Basic

```
<ImportMany()>
Public Property ImportedButtons() As Lazy(Of ICmdButtonInfo)()
```

3. コレクション全体を反復処理し、インポートされたオブジェクト (この場合はボタン) を読み込みます。**MainForm** のコードと同様、**Value** プロパティを使ってエクスポートされたオブジェクト自体にアクセスします。**EmployeeMaintenance** クラスの末尾に次のメソッドを追加します。

(コード スニペット – Intro to MEF Lab - Ex2 CreateButtons - CSharp)

C#

```
private void CreateButtons()
{
   int left = 10;

   foreach (var importedButton in this.ImportedButtons)
   {
      var btnInfo = importedButton.Value;

      btnInfo.CompanyInfo = this.CompanyInfo;
      btnInfo.CompanyInfo.EmployeeListChanged +=
         new EventHandler(CompanyInfo_EmployeeListChanged);
      Button btn = btnInfo.CommandButton;

      this.ButtonsPanel.Controls.Add(btn);
      btn.Left = left;
      left += btn.Width;
      left += 20;
   }
}
```

(コード スニペット – Intro to MEF Lab - Ex2 CreateButtons – VB)

Visual Basic

```
Private Sub CreateButtons()
   Dim left As Integer = 10

   For Each importedButton In Me.ImportedButtons
      Dim btnInfo = importedButton.Value

      btnInfo.CompanyInfo = Me.CompanyInfo
      AddHandler btnInfo.CompanyInfo.EmployeeListChanged, AddressOf
CompanyInfo_EmployeeListChanged
      Dim btn As Button = btnInfo.CommandButton

      Me.ButtonsPanel.Controls.Add(btn)
      btn.Left = left
      left += btn.Width
      left += 20
   Next importedButton
End Sub
```

4. フォームの Load メソッドを更新して、**CreateButton** メソッドを呼び出します。

C#

```
private void EmployeeMaintenance_Load(object sender, EventArgs e)
{
   GetAllEmployees();
   CreateButtons();
}
```

Visual Basic

```
Private Sub EmployeeMaintenance_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
MyBase.Load
   GetAllEmployees()
   CreateButtons()
End Sub
```

5. ユーザーが **DataGridView** の行を選択したときに、**CompanyInfo** オブジェクトの **SelectedEmployee** フィールドを更新します。これで **CompanyInfo** に疎結合されるボタンはどの従業員を操作するかを認識できます。**EmployeeMaintenance** クラスに次のメソッドを追加します。

(コード スニペット – Intro to MEF Lab - Ex2 empDataGridView_SelectionChanged - CSharp)

C#

```
private void empDataGridView_SelectionChanged(object sender, EventArgs e)
{
   if (empDataGridView.SelectedRows.Count > 0)
```

```
    {
        Employee selectedEmployee
            = (Employee)empDataGridView.SelectedRows[0].DataBoundItem;
        this.CompanyInfo.SelectedEmployee = selectedEmployee;
    }
    else
    {
        this.CompanyInfo.SelectedEmployee = null;
    }
}
```

(コード スニペット – Intro to MEF Lab - Ex2 empDataGridView_SelectionChanged - VB)

Visual Basic

```
Private Sub empDataGridView_SelectionChanged(ByVal sender As Object, ByVal e As EventArgs)
    If empDataGridView.SelectedRows.Count > 0 Then
        Dim selectedEmployee As Employee =
CType(empDataGridView.SelectedRows(0).DataBoundItem, Employee)
        Me.CompanyInfo.SelectedEmployee = selectedEmployee
    Else
        Me.CompanyInfo.SelectedEmployee = Nothing
    End If
End Sub
```

6. 上記のメソッドを **DataGridView** の **SelectionChanged** イベントに結び付けます。

   EmployeeMaintenance フォームをデザイナー モードで開きます。**DataGridView** コントロールを右クリックして、コンテキスト メニューの [Properties] (プロパティ) をクリックします。[Properties] (プロパティ) シートで、イベントを一覧する [Event] (イベント) アイコンをクリックします。一覧から **SelectionChanged** イベントを探し、プルダウン リストを使って **empDataGridView_SelectionChanged** メソッドを選択します。

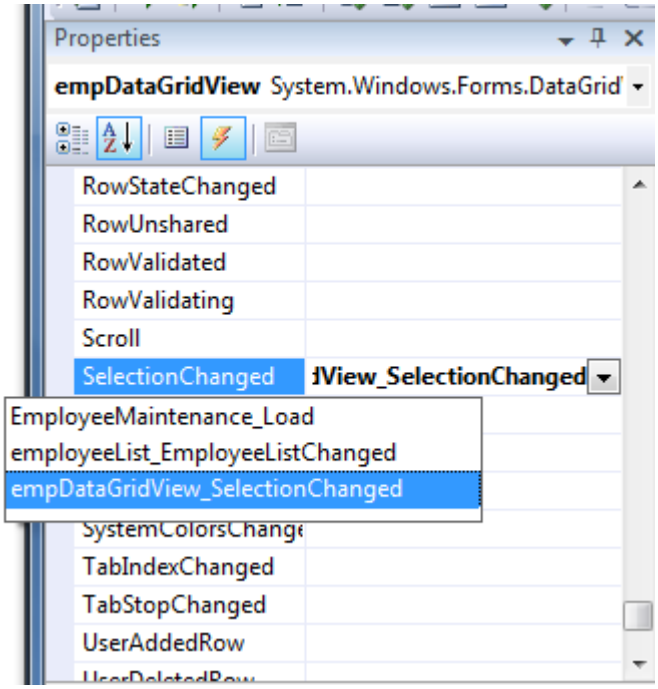

図 4

SelectionChanged イベントの接続

7. ソリューションをコンパイルします。[Build] (ビルド) メニューの [Build Solution] (ソリューションのビルド) をクリックします。

次の手順

演習 2: 確認

演習 2: 確認

この確認では、MEF による拡張機能を備えたソリューションを実行します。

1. [スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010]、[Microsoft Visual Studio 2010] を順にクリックして Microsoft Visual Studio 2010 を起動し、MefLab.sln ソリューションを開きます。
2. 新しい拡張機能を備えたアプリケーションをテストします。事前に構成済みのビルド後のイベントを使用して **MefEmployeeExtender** アセンブリが既に ExtModules フォルダーにコピーされています。

a. **F5** キーを押してアプリケーションを実行します。[Modules]、[Employees] の順にクリックします。

b. EmployeeMaintenance フォームが表示されます。今回はは新たに 2 つのボタンが追加されています。[Delete Employee] ボタンと [Add Employee] ボタンです。

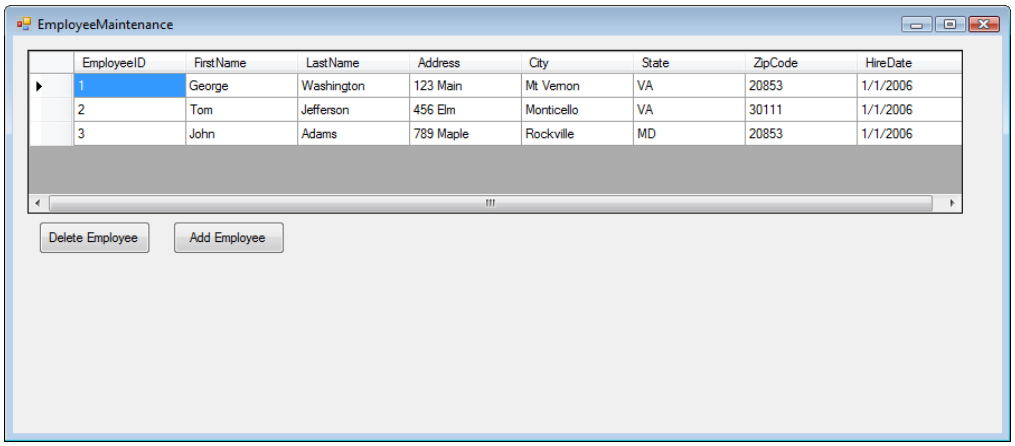

図 5

EmployeeMaintenance フォーム

c. [Add Employee] をクリックします。DataGridView に新しい行が表示されます。

d. **DataGridView** の行の左端にある灰色のボックスをクリックして、任意の行を強調表示します。[Delete Employee] をクリックします。選択した行が削除されます。

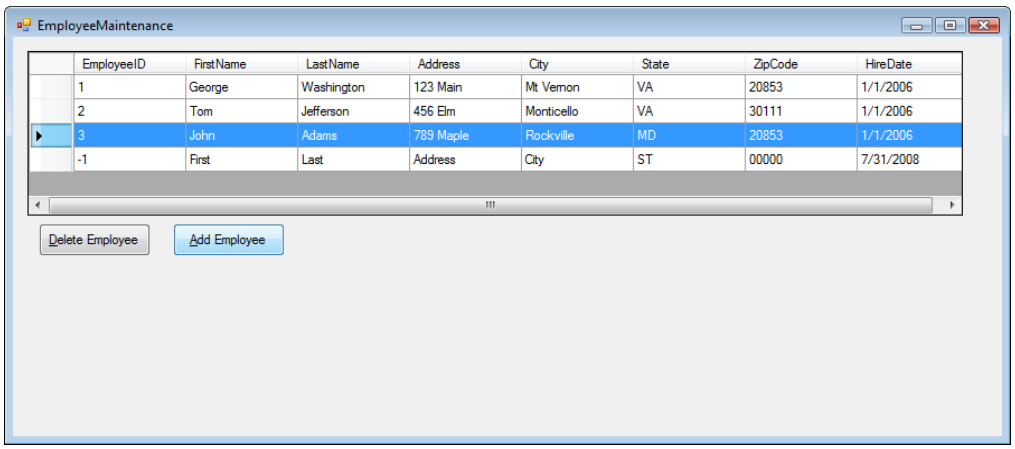

図 6

行の削除

e. EmployeeMaintenance の Load メソッドと各ボタンの Click メソッドにブレークポイントを設定して、コードをステップ実行し、何が起こるかを確認していきます。

f. EmployeeMaintenance フォームとメインフォームを閉じます。

次の手順

まとめ

まとめ

このハンズオン ラボでは Microsoft Managed Extensiblity Framework (MEF) を使用して、アプリケーションに拡張ポイントを追加し、それらの拡張ポイントにプラグインする拡張機能をビルドしました。

MEF を使用して、エクスポートするプロパティの設定、明示的参照を設定なしでの外部アセンブリの読み込み、実行時のアプリケーションの動的拡張を行いました。インポートとエクスポートの関連性や、Interfaces 経由でコントラクトを使用して、どのようなエクスポートコンポーネントを取り扱うかについても学習しました。