

## Lab 4 – イベント

May, 2011 by A. Nagy  
Updated by DevTech AEC WG  
Last modified: 8/4/2015

<C#>C#バージョン</C#>

**目的:**この実習では、イベントを監視する方法を学習します。学習する項目は次のとおりです。

- 特定のイベントを監視する
- ダイナミック モデル アップデータ を使用する

この実習の実装と確認の手順は、下記のとおりです:

1. 新しい外部アプリケーションを作成する
2. イベントを監視する
3. ダイナミック モデル アップデータを実装する
4. サマリ

### 1. 新しい外部アプリケーションを作成する

現在のプロジェクトに新しい外部アプリケーションを追加します。

1.1 新しいファイルを追加して、プロジェクトに新しい外部アプリケーションを定義します。ファイル名とクラス名は、下記のようにしてください:

- ファイル名: **4\_Event.cs**
- アプリケーション クラス名: **UIEventApp**

(繰り返しになりますが、ここで希望する名前を使用しても構いません。ただし、その場合、プロジェクト名など、このドキュメント内では記述されている名称は、自分でつけた名称で代替して参照してください)

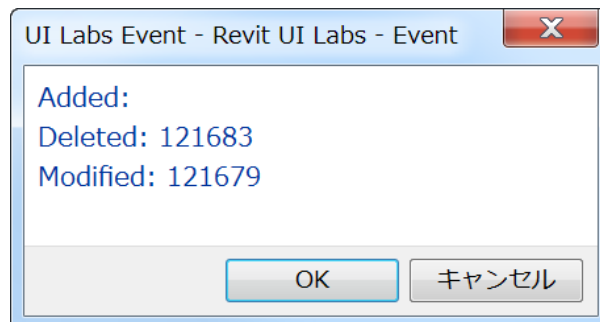
**要求される名前空間:**

この実習で必要とする名前空間は次のとおりです:

- Autodesk.Revit.DB
- Autodesk.Revit.UI
- Autodesk.Revit.ApplicationServices
- Autodesk.Revit.Attributes
- Autodesk.Revit.UI.Selection(選択用)
- Autodesk.Revit.DB.Events(イベント処理用)
- 

注意: (VB.NET のみ):VB.NET でコードを記述している場合、プロジェクト・レベルで名前空間をインポートします(つまり、プロジェクトプロパティでは、各ファイル内で明示的にインポートする必要はありません)。

## 2. イベントを監視する



Revit 内部の様々なイベントを監視することができます。Visual Studio で、Application、UIApplication、Document などの様々なクラスに属するイベントの一覧を参照してみてください。

例えば、DocumentChanged イベントを監視することで、Revit の内部でのオブジェクトの追加、削除、修正が発生するたびに、それらの通知を受け取ることができます。

イベントを監視するには、対象とするイベント名を含む関数を定義する必要があります。

```
<C#>
public void UILabs_DocumentChanged(
    object sender, DocumentChangedEventArgs args )
{
}
</C#>
```

OnStartup() 関数内で、+=(cs)、または、AddHandler(vb) を使用して、上記の関数にイベントを割り当てることができます。

```
<C#>
app.ControlledApplication.DocumentChanged += UILabs_DocumentChanged;
</C#>
```

DocumentChanged イベントハンドラの内部では、ドキュメント内で何が変更されたか、args によって取得して利用可能な情報、TaskDialog を使ってリストできます。

Revit ドキュメントが変更された際に、ダイアログを表示するか否かをコントロールする bool 変数を UIEventClass に加えてください。

また、3 つの異なる外部コマンドを実装してみてください。1 つは、この bool 変数を True に設定し、もう 1 つは False に設定して、最後の 1 つはそれをトグルにするものとします。

### 3. ダイナミック モデル アップデータを実装する

前述のイベント監視の方法は、変更を追跡する場合には非常に便利です。例えば、ドキュメントが変更される度に、外部データベースの情報を更新する必要がある場合などです。

しかしながら、たった今起こった変更に基づいて、ドキュメントの一部を修正したい場合には、ダイナミック モデル アップ データと呼ばれる異なるメカニズムが必要になります。

ここでは、一例として、壁を編集した際に、その壁をホストとしている窓やドアが、常に壁の中央の位置を維持するように、ダイナミック モデル アップデータを作成します。

まず最初に、IUpdater と、その 5 つの関数を実装するクラスを作成する必要があります。クラスの内部では、アドインの GUID とは別の GUID に基づいた UpdaterId のインスタンスを作成する必要があります。— この GUID は、Visual Studio を使用して作成することができます。

```
<C#>
public class WindowDoorUpdater : IUpdater
{
    // Unique id for this updater = addin GUID + GUID for this specific
    // updater.

    UpdaterId m_updaterId = null;

    // Flag to indicate if we want to perform an update

    public static bool m_updateActive = false;

    /// <summary>
    /// Constructor
    /// </summary>

    public WindowDoorUpdater(AddInId id)
    {
        m_updaterId = new UpdaterId( id,
            new Guid( "EF43510F-38CB-4980-844C-72174A674D56" ) );
    }

    /// <summary>
    /// This is the main function to do the actual job.
    /// For this exercise, we assume that we want to keep
    /// the door and window always at the center.
    /// </summary>

    public void Execute(UpdaterData data)
    {
        if( !m_updateActive ) return;
    }

    /// <summary>
    /// This will be shown when the updater is not loaded.
    /// </summary>

    public string GetAdditionalInformation()
    {
        return "Door/Window updater: keeps doors and windows at the center of
walls.";
    }

    /// <summary>
    /// Specify the order of executing updaters.
    /// </summary>
}
```

```

public ChangePriority GetChangePriority()
{
    return ChangePriority.DoorsOpeningsWindows;
}

/// <summary>
/// Return updater id.
/// </summary>

public UpdaterId GetUpdaterId()
{
    return m_updaterId;
}

/// <summary>
/// User friendly name of the updater
/// </summary>

public string GetUpdaterName()
{
    return "Window/Door Updater";
}
}
</C#>

```

次に、OnStartup() 関数にて、クラスのインスタンスを作成し、ElementClassFilter クラスを使用して、監視する要素を壁に制限してください。

```

</C#>
WindowDoorUpdater winDoorUpdater =
    new WindowDoorUpdater( app.ActiveAddInId );

// ActiveAddInId is from addin manifest.
// Register it

UpdaterRegistry.RegisterUpdater( winDoorUpdater );

// Tell which elements we are interested in being notified about.
// We want to know when wall changes its length.

ElementClassFilter wallFilter =
    new ElementClassFilter( typeof( Wall ) );
UpdaterRegistry.AddTrigger( winDoorUpdater.GetUpdaterId(),
    wallFilter, Element.ChangeTypeGeometry() );

</C#>

```

残りをご自身で実装してみてください。与えられた壁のドア、または窓を取得する関数を作成してください。

そのためには、データベースからすべてのドアと窓を取得する `FilteredElementCollector` を使用する必要があるでしょう。また、それらのオーナーが、与えられた壁かどうかをチェックする必要もあります。

次に、壁の `LocationCurve` に基づいたドアか窓を、中央に配置する別の関数も作成してください。

## 4. サマリ

この実習では、イベントを監視する方法を学習しました。学習した項目は次のとおりです。

- 特定のイベントを監視する
- ダイナミック モデル アップデータを使用する