

Lab4 – 要素の修正

Created by M. Harada, July 2010

Updated by DevTech AEC WG

Last modified: 6/9/2017

<VB.NET>VB.NET バージョン</VB.NET>

目的:この実習では、要素を修正する方法を学習します。学習する項目は次のとおりです。

- 要素のプロパティ、パラメータ、位置を変更して要素を修正する
- 移動や回転のようなトランスフォーム ユーティリティ メソッドを使用して要素を修正する

タスク:要素の選択をユーザに促して、そのプロパティを修正するコマンドを記述します。その後、再度、要素を選択して、ユーティリティ メソッドで選択した要素を回転させます。

1. 要素を選択する
2. 選択要素のファミリ タイプを修正する
3. 選択要素のパラメータを修正する
4. 選択要素の位置(オプション)を修正する
5. 要素を選択する
6. ElementTransformUtils.MoveElement() メソッドで選択要素を移動させる
7. ElementTransformUtils.RotateElement() メソッドで選択要素を回転させる

図 1 は、この実習で定義するコマンドを実行した後に出力されるサンプル画像を示します:

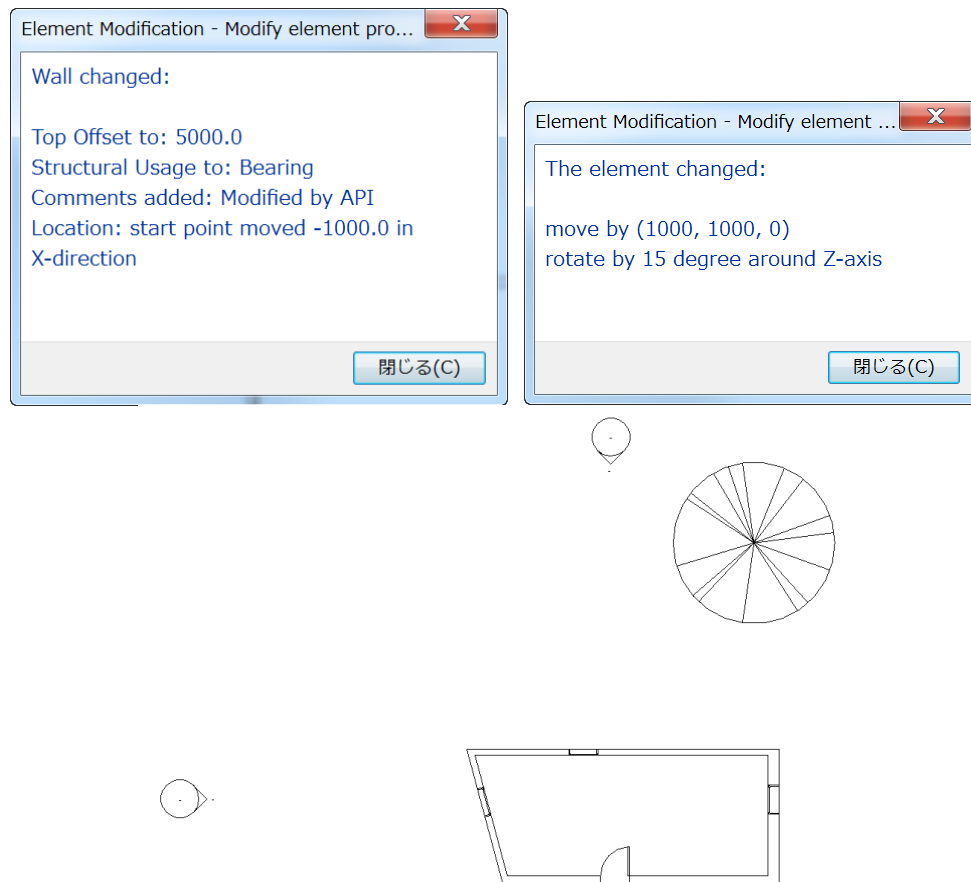


図 1.プロパティの変更と変換ユーティリティ メソッドで要素を修正

この実習の実装と確認の手順は、下記のとおりです:

1. 新しい外部コマンドを定義する
2. 要素を選択する
3. 要素のプロパティを修正する
4. トランスフォーム ユーティリティ メソッドを使用して要素を移動して回転させる
5. サマリ

1. 新しい外部コマンドを定義する

現在のプロジェクトに新しい外部コマンドを追加します。

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コマンドを定義します。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_ElementModification.vb**
- コマンド クラス名: **ElementModification**

追加の留意事項:

前の実習で記述した ElementFiltering クラスから次の関数を使用します:

- ElementFiltering.FindFamilyType()
- ElementFiltering.FindElement()

1.2 前回の実習で行ったように、メンバ変数を定義します。DB レベルのアプリケーションとドキュメントを保持する m_rvtApp と m_rvtDoc を定義します。下記はその例です:

```
<VB.NET>
'' Element Modification - learn how to modify elements
<Transaction(TransactionMode.Automatic)> _
Public Class ElementModification
    Implements IExternalCommand

    '' member variables
    Dim m_rvtApp As Application
    Dim m_rvtDoc As Document

    Public Function Execute(
        ByVal commandData As ExternalCommandData, _
        ByRef message As String, _
        ByVal elements As ElementSet) _
        As Result _
        Implements IExternalCommand.Execute

        '' Get the access to the top most objects.
        Dim rvtUIApp As UIApplication = commandData.Application
        Dim rvtUIDoc As UIDocument = rvtUIApp.ActiveUIDocument
        m_rvtApp = rvtUIApp.Application
```

```

        m_rvtDoc = rvtUIDoc.Document

        '' ...

        Return Result.Succeeded

    End Function

End Class
</VB.NET>

```

2. 要素を選択する

Lab2 で要素を選択する方法を既に学習しました。ここでは、再度、スクリーン上のオブジェクトを選択するために、オーバーロードされた PickObject() メソッドの1つを使用します:

- UIDocument.Selection.PickObject(ObjectType.Element, promptString)

下記はそのサンプルコードです:

```

<VB.NET>
    '' (1) pick an object on a screen.
    Dim ref As Reference = rvtUIDoc.Selection.PickObject( _
        ObjectType.Element, "Pick an element")
    Dim elem As Element = m_rvtDoc.GetElement(ref)
</VB.NET>

```

3. 要素のプロパティを修正する

Lab2では、要素を構成するものと、どの種類の情報がAPIによってアクセス可能かを学習しました。このセクションでは、それら情報のいくつかを修正する方法を見ていきます。与えられたモデル要素のインスタンスについて、次の修正を加えていきます:

- 与えられたクラスのプロパティ
- パラメータ
- 位置カーブ

3.1 ファミリタイプのインスタンスを修正する

Wall.WallTypeやFamilyInstance.Symbolのように、クラスのプロパティとして直接アクセス可能な公開された情報については、それを直接変更することができます。次の例は、壁に再度新しいファミリタイプを割り当てています。ここで、前の実習で定義した FindFamilyType() メソッドを使用しています:

```

<VB.NET>
    '' e.g., an element we are given is a wall.
    Dim aWall As Wall = elem

```

```

    ' find a wall family type with the given name.
    Dim newWallType As Element = ElementFiltering.FindFamilyType( m_rvtDoc, _
        GetType(WallType), "標準壁", "外壁 - メタル スタッドのレンガ")

    ' assign a new family type.
    aWall.WallType = newWallType

```

</VB.NET>

また、ドアを使った例は下記のようになります:

<VB.NET>

```

    ' e.g., an element we are given is a door.
    Dim aDoor As FamilyInstance = elem

    ' find a door family type with the given name.
    Dim newDoorType As Element = ElementFiltering.FindFamilyType( m_rvtDoc, _
        GetType(FamilySymbol), "片側フラッシュ", "0762 x 2032mm", _
        BuiltInCategory.OST_Doors)

    ' assign a new family type.
    aDoor.Symbol = newDoorType

```

</VB.NET>

ChangeTypeId メソッドを使用するのが一般的です。

< VB.NET >

```

    // or use a general way: ChangeTypeId:
    aDoor.ChangeTypeId(newDoorType.Id);

```

</VB.NET >

下記は、前後のサポート情報を含むサンプル コードです。単純化するために、ここでは既に壁を持っていると考えます。他の種類のオブジェクトについても、同様のアプローチを適用することができます。

<VB.NET>

```

Sub ModifyElementPropertiesWall(ByVal elem As Element)

    ' Constant to this function.
    ' this is for wall. e.g., "Basic Wall: Exterior - Brick on CMU"
    ' you can modify this to fit your need
    '

    Const wallFamilyName As String = "標準壁"
    Const wallTypeName As String = "外壁 - メタル スタッドのレンガ"
    Const wallFamilyAndTypeName As String =
        wallFamilyName + ": " + wallTypeName

    ' for simplicity, we assume we can only modify a wall
    If Not (TypeOf elem Is Wall) Then
        TaskDialog.Show("Revit Intro Lab", _
            "Sorry, I only know how to modify a wall. Please select a wall.")
        Return
    End If

```

```

    'keep the message to the user.
    Dim aWall As Wall = elem
    Dim msg As String = "Wall changed: " + vbCrLf + vbCrLf

    ' (1) change its family type to a different one.
    ' To Do: change this to enhance import symbol later.
    ''

    Dim newWallType As Element = _
    ElementFiltering.FindFamilyType(m_rvtDoc, GetType(WallType), _
    wallFamilyName, wallTypeName)

    If newWallType IsNot Nothing Then
        aWall.WallType = newWallType
        msg = msg + "Wall type to: " + wallFamilyAndTypeName + vbCrLf
    End If

    '' ...

End Sub
</VB.NET>

```

3.2 パラメータの変更

パラメータの値を変更するために、最初に関心のあるパラメータを取得して、次に、新しい値でパラメータを修正する「セット」メソッドを使用します。4つのオーバーロードメソッドが存在します。次のデータタイプの値を変更することができます:

- セット(ElementId)
- セット(Double)
- セット(Int32)
- セット(String)

次のサンプルは、壁の「トップ オフセット」と「コメント」パラメータを変更するものです:

```

<VB.NET>
aWall.Parameter(BuiltInParameter.WALL_TOP_OFFSET).Set(14.0)
aWall.Parameter(BuiltInParameter.ALL_MODEL_INSTANCE_COMMENTS).Set( _
    "API で変更")
</VB.NET>

```

下記のコードは、コマンド内の前後のサポート情報を備えた例です。ここでは、壁インスタンスに対する次のパラメータの値を変更しています:

- 上部の拘束を“レベル 1”へ変更(ElementId)
- 上部のオフセットを5000.0mmまで変更 (Double)
- 構造使用法を軸受(1)へ変更 (Integer)
- コメントを変更(String)

```

<VB.NET>
    ' (2) change its parameters.
    ' constrain top of the wall to the level1 and set an offset.

    ' find the level 1 using the function we defined in lab3.

```

```

Dim level1 As Level = _
    ElementFiltering.FindElement(m_rvtDoc, GetType(Level), "レベル 1")
If level1 IsNot Nothing Then
    '' Top Constraint
    aWall.Parameter(BuiltInParameter.WALL_HEIGHT_TYPE).Set(level1.Id)
    msg = msg + "Top Constraint to: Level 1" + vbCrLf
End If

Dim topOffset As Double = mmToFeet(5000.0) '' hard coding
'' Top Offset Double
aWall.Parameter(BuiltInParameter.WALL_TOP_OFFSET).Set(topOffset)
'' Structural Usage = Bearing(1)
aWall.Parameter(BuiltInParameter.WALL_STRUCTURAL_USAGE_PARAM).Set(1)
'' Comments - String
aWall.Parameter(BuiltInParameter.ALL_MODEL_INSTANCE_COMMENTS).Set( _
    "API で変更")

msg += "Top Offset to: 5000.0" + vbCrLf
msg += "Structural Usage to: Bearing" + vbCrLf
msg += "Comments added: Modified by API" + vbCrLf

```

</VB.NET>

mmToFeet() は、単位をミリメートルからフィートに替える単純な関数です。

(注意:内部的にRevitはフィートでデータを保存します。)

<VB.NET>

```

Const _mmToFeet As Double = 0.0032808399

Public Shared Function mmToFeet(ByVal mmValue As Double) As Double
    Return mmValue * _mmToFeet
End Function

```

</VB.NET>

3.3 Location Curveの変更

位置情報の値を変更するために、まず、与えられたインスタンスから位置情報を取得して、インスタンスから LocationCurve にキャストします。これによって、カーブ情報にアクセスすることが可能になります。その後、線分境界を作成して、壁の位置に新しいカーブを割り当てます:

<VB.NET>

```

Dim wallLocation As LocationCurve = aWall.Location

'' create a new line bound.
Dim newPt1 = New XYZ(0.0, 0.0, 0.0)
Dim newPt2 = New XYZ(20.0, 0.0, 0.0)
Dim newWallLine As Line = Line.CreateBound(newPt1, newPt2)

'' change the curve.
wallLocation.Curve = newWallLine

```

</VB.NET>

下記のコードは、壁を(-1000.0,0.0,0.0) まで移動する例です(コマンド内の前後のサポート情報含む):

```
<VB.NET>
    ' (3) change its location, using location curve
    ' get the location curve from a wall
    Dim wallLocation As LocationCurve = aWall.Location
    ' define a new location based on the current location
    Dim pt1 As XYZ = wallLocation.Curve.GetEndPoint(0)
    Dim pt2 As XYZ = wallLocation.Curve.GetEndPoint(1)

    ' hard coding the displacement value for similitly here.
    Dim dt As Double = mmToFeet(1000.0)
    Dim newPt1 = New XYZ(pt1.X - dt, pt1.Y - dt, pt1.Z)
    Dim newPt2 = New XYZ(pt2.X - dt, pt2.Y - dt, pt2.Z)

    ' create a new line bound.
    Dim newWallLine As Line = _
        Line.CreateBound(newPt1, newPt2)
    ' finally change the curve.
    wallLocation.Curve = newWallLine
    ' message to the user
    msg += "Location: start point moved -1000.0 in X-direction" + vbCr

    TaskDialog.Show("Revit Intro Lab", msg)
</VB.NET>
```

LocationCurveには、さらに移動と回転のメソッドが用意されています。

実習:

- 要素のインスタンスを引数にとり、ファミリ タイプ、パラメータ値および位置情報の値を修正する関数を実装する(この実習では、与えられた要素が壁やドアのような特定タイプのオブジェクトであると考えてよい)
- メインのExecute() メソッドから、選択した要素でこの関数を呼び出す

注意:既存の拘束が、これらの修正の結果に影響するかもしれません。例えば、コーナーで接続された他の壁があり、拘束に違反する方法で壁を修正しようすると、Revitは修正を無効にします。テストの目的のために、単一の自立壁を作成し、コマンドを実行してみてください。

4. トランスフォーム ユーティリティ メソッドを使用して要素を移動して回転

要素を修正する他の方法には、トランスフォーム ユーティリティを使用する方法があります。このユーティリティ クラス(ElementTransformUtils)は、次のタイプの操作を提供します:

- 移動
- 回転
- 鏡像
- コピー

RevitAPI.chm や Revit API 開発者用ガイドの「[要素を編集する](#)」項目 は、これらのメソッドのいくつかの使用方法を示すサンプルコードを含んでいます。メソッドのバリエーションに関しては、それらを参照してください。

このトレーニング実習では、例として、移動と回転の方法を見ていきます。下記は、(10.0,10.0,0.0) まで与えられた要素を移動する例です:

```
<VB.NET>
    '' move by displacement
    Dim v As XYZ = New XYZ(10.0, 10.0, 0.0)
    ElementTransformUtils.MoveElement(m_rvtDoc, elem.Id, v)
</VB.NET>
```

また、下記は、Z軸のまわりで15度($=\pi/12$)ずつ与えられた要素を回転させる例です:

```
<VB.NET>
    '' rotate by 15 degree around z-axis.
    Dim pt1 = XYZ.Zero
    Dim pt2 = XYZ.BasisZ
    Dim axis As Line = Line.CreateBound(pt1, pt2)
    ElementTransformUtils.RotateElement(m_rvtDoc, elem.Id, axis, Math.PI/12.0);
</VB.NET>
```

次の例は、与えられた要素を移動して回転させる例です:

```
<VB.NET>
    '' modify an element through transform utils methods, Move and Rotate
    ''
    Sub ModifyElementByTransformUtilsMethods(ByVal elem As Element)

        '' keep the message to the user.
        Dim msg As String = "The element changed: " + vbCrLf + vbCrLf

        '' try move
        Dim dt As Double = mmToFeet(1000.0) '' hard cording for simplicity.
        Dim v As XYZ = New XYZ(dt, dt, 0.0)
        ElementTransformUtils.MoveElement(m_rvtDoc, elem.Id, v)

        msg = msg + "move by (1000, 1000, 0)" + vbCrLf

        '' rotate: 15 degree around z-axis.
        Dim pt1 = XYZ.Zero
        Dim pt2 = XYZ.BasisZ
        Dim axis As Line = Line.CreateBound(pt1, pt2)
        ElementTransformUtils.RotateElement(m_rvtDoc, elem.Id, axis, Math.PI/12.0)

        msg = msg + "rotate by 15 degree around Z-axis" + vbCrLf

        '' show the message to the user.
        TaskDialog.Show("Revit Intro Lab", msg)

    End Sub
</VB.NET>
```

グラフィックスの再作図

注意: 要素を修正し、その修正によりモデルのジオメトリが変更される場合、かつ、更新されたグラフィックスにアクセスする必要がある場合には、グラフィックスを再作図する必要があります。Document.Regenerate() メソッドの呼び出しで、これをコントロールすることができます。

```
m_rvtDoc.Regenerate()
```

実習:

- 要素を取得して、任意の値で要素を移動・回転させる関数を実装する

5. サマリ

この実習では、要素を修正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。

- 要素のプロパティ、パラメータ、位置を変更して要素を修正する
- 移動や回転のようなトランスフォーム ユーティリティ メソッドを使用して要素を修正する

次の実習では、Revit 内で要素を作成してモデルを構築する方法を学習します。