

### Lab4 – 要素の修正

Created by M. Harada, July 2010

Updated by Ryuji Ogasawara

Last modified: 8/8/2025

<C#>C#バージョン</C#>

**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:

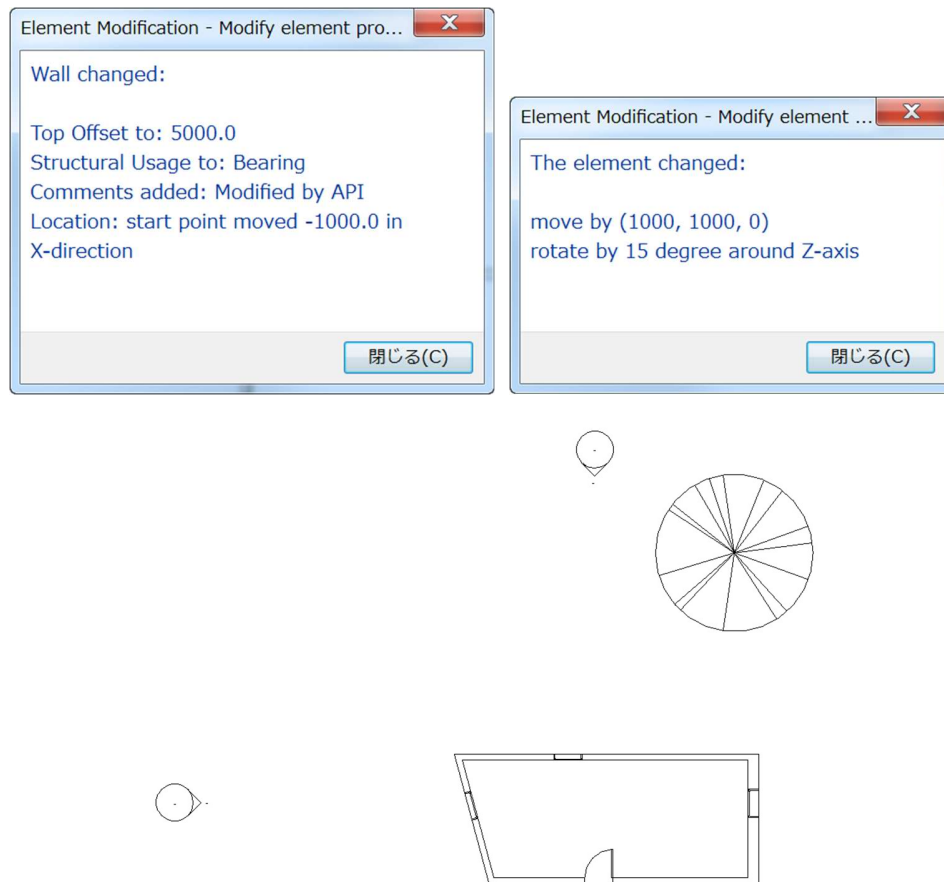


図 1.プロパティの変更と変換ユーティリティ メソッドで要素を修正

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.cs**
- 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:

```
<C#>
// Element Modification - learn how to modify elements

[Transaction(TransactionMode.Manual)]
public class ElementModification : IExternalCommand
{
    // member variables
    Application m_rvtApp;
    Document m_rvtDoc;

    public Result Execute(
        ExternalCommandData commandData,
        ref string message,
        ElementSet elements)
    {
        // Get the access to the top most objects.
        UIApplication rvtUIApp = commandData.Application;
        UIDocument rvtUIDoc = rvtUIApp.ActiveUIDocument;
        m_rvtApp = rvtUIApp.Application;
    }
}
```

```

        m_rvtDoc = rvtUIDoc.Document;

        // ...

        return Result.Succeeded;
    }
}
</C#>

```

## 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:

```

<C#>
    // (1) pick an object on a screen.
    Reference refPick = rvtUIDoc.Selection.PickObject(
        ObjectType.Element, "Pick an element");
    Element elem = m_rvtDoc.GetElement(refPick);
</C#>

```

## 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
- Location Curve

### 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:

```

<C#>
    // e.g., we assume we can only modify a wall
    Wall aWall = (Wall)elem;

    // find a wall family type with the given name.
    Element newWallType = ElementFiltering.FindFamilyType(m_rvtDoc,
        typeof(WallType), "標準壁", "外壁 - メタル スタッドのレンガ", null);

    // assign a new family type.
    aWall.WallType = (WallType)newWallType;
</C#>

```

また、ドアを使った例は下記のようになります:

```

<C#>
    // e.g., an element we are given is a door.
    FamilyInstance aDoor = (FamilyInstance)elem;

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

    // assign a new family type
    aDoor.Symbol = (FamilySymbol)newDoorType;
</C#>

```

ChangeTypeId メソッドを使用するのが一般的です。

```

<C#>
    // or use a general way: ChangeTypeId:
    aDoor.ChangeTypeId(newDoorType.Id);
</C#>

```

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。

```

<C#>
public void ModifyElementPropertiesWall(Element elem)
{
    // 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 string wallFamilyName = "標準壁";
    const string wallTypeName = "一般 - 225 mm 組積造";
    const string wallFamilyAndTypeName =
        wallFamilyName + ": " + wallTypeName;

    // for simplicity, we assume we can only modify a wall

```

```

    if (!elem is Wall)
    {
        TaskDialog.Show("Revit Intro Lab",
            "Sorry, I only know how to modify a wall. Please select a wall.");
        return;
    }
    Wall aWall = (Wall)elem;

    // keep the message to the user.
    string msg = "Wall changed: " + "\n" + "\n";

    // (1) change its family type to a different one.
    // (You can enhance this to import symbol if you want.)

    Element newWallType = ElementFiltering.FindFamilyType(m_rvtDoc,
        typeof(WallType), wallFamilyName, wallTypeName, null);

    if (newWallType != null)
    {
        aWall.WallType = (WallType)newWallType;
        msg = msg + "Wall type to: " + wallFamilyAndTypeName + "\n";
    }

    //...
}
</C#>

```

### 3.2 パラメータの変更

パラメータの値を変更するには、まず変更の対象とす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「Set」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：

- Set(ElementId)
- Set(Double)
- Set(Int32)
- Set(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：

```

<C#>
aWall.get_Parameter(BuiltInParameter.WALL_TOP_OFFSET).Set(14.0);
aWall.get_Parameter(BuiltInParameter.ALL_MODEL_INSTANCE_COMMENTS).Set(
    "API で変更");
</C#>

```

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)

```
<C#>
// (2) change its parameters.
// as a way of exercise, let's constrain the top of the wall
// to the level1 and set an offset.

// find the level 1 using the helper function we defined in the lab3.
Level level1 = (Level)ElementFiltering.FindElement(
    m_rvtDoc, typeof(Level), "レベル 1", null);
if (level1 != null)
{
    aWall.get_Parameter(BuiltInParameter.WALL_HEIGHT_TYPE).
        Set(level1.Id);
    // Top Constraint
    msg += "Top Constraint to: Level 1" + "\n";
}

// Top Offset Double. hard coding for simplisity here.
double topOffset = mmToFeet(5000.0);
aWall.get_Parameter(BuiltInParameter.WALL_TOP_OFFSET).Set(topOffset);

// Structural Usage = Bearing(1)
aWall.get_Parameter(BuiltInParameter.WALL_STRUCTURAL_USAGE_PARAM).
    Set(1);

// Comments - String
aWall.get_Parameter(BuiltInParameter.ALL_MODEL_INSTANCE_COMMENTS).
    Set("API で変更");

msg += "Top Offset to: 5000.0" + "\n";
msg += "Structural Usage to: Bearing" + "\n";
msg += "Comments added: Modified by API" + "\n";
</C#>
```

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。)

```
<C#>
const double _mmToFeet = 0.0032808399;

public static double mmToFeet(double mmValue)
{
    return mmValue * _mmToFeet;
}
</C#>
```

### 3.2 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:

```
<C#>
    LocationCurve wallLocation = (LocationCurve)aWall.Location;

    // create a new line bound.
    XYZ newPt1 = new XYZ(0.0, 0.0, 0.0);
    XYZ newPt2 = new XYZ(20.0, 0.0, 0.0);
    Line newWallLine = Line.CreateBound(newPt1, newPt2);

    // finally change the curve.
    wallLocation.Curve = newWallLine;
</C#>
```

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):

```
<C#>
    // (3) Optional: change its location, using location curve
    // LocationCurve also has move and rotation methods.
    // Note: constraints affect the result.
    // Effect will be more visible with disjointed wall.
    // To test this, you may want to draw a single standing wall,
    // and run this command.

    LocationCurve wallLocation = (LocationCurve)aWall.Location;

    XYZ pt1 = wallLocation.Curve.GetEndPoint(0);
    XYZ pt2 = wallLocation.Curve.GetEndPoint(1);

    // hard coding the displacement value for similitude here.
    double dt = mmToFeet(1000.0);
    XYZ newPt1 = new XYZ(pt1.X - dt, pt1.Y - dt, pt1.Z);
    XYZ newPt2 = new XYZ(pt2.X - dt, pt2.Y - dt, pt2.Z);

    // create a new line bound.
    Line newWallLine = 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" + "\n";

    TaskDialog.Show("Revit Intro Lab", msg);
</C#>
```

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とする、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:

```
<C#>
// move by displacement
XYZ v = new XYZ(10.0, 10.0, 0.0);
ElementTransformUtils.MoveElement(m_rvtDoc, elem.Id, v);
</C#>
```

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:

```
<C#>
// try rotate: 15 degree around z-axis.
XYZ pt1 = XYZ.Zero;
XYZ pt2 = XYZ.BasisZ;
Line axis = Line.CreateBound(pt1, pt2);
ElementTransformUtils.RotateElement(m_rvtDoc, elem.Id, axis, Math.PI / 12.0);
</C#>
```

次の例は、与えられた要素を移動して回転させる例です:

<C#>

```
// A sampler function that demonstrates how to modify an element
// through document methods.

public void ModifyElementByTransformUtilsMethods(Element elem)
{
    // keep the message to the user.
    string msg = "The element changed: " + "\n\n";

    // try move
    double dt = mmToFeet(1000.0);
    // hard cording for simplicity.
    XYZ v = new XYZ(dt, dt, 0.0);

    ElementTransformUtils.MoveElement(m_rvtDoc, elem.Id, v);

    msg = msg + "move by (1000, 1000, 0)" + "\n";

    // try rotate: 15 degree around z-axis.
    XYZ pt1 = XYZ.Zero;
    XYZ pt2 = XYZ.BasisZ;
    Line axis = Line.CreateBound(pt1, pt2);

    ElementTransformUtils.RotateElement(m_rvtDoc, elem.Id, axis, Math.PI/12.0);

    msg = msg + "rotate by 15 degree around Z-axis" + "\n";

    // message to the user.
    TaskDialog.Show("Modify element by utils methods", msg);
}
```

</C#>

## 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。