

## Lab 5 – モデル生成

Created by M. Harada, July 2010

Updated by DevTech AEC WG

Last modified: 6/9/2017

<C#>C#バージョン</C#>

**目的:**この実習では、Revit モデルを作成する方法を学習していきます。学習する項目は次のとおりです。

- 建築要素(壁、ドア、窓、屋根)のインスタンスを作成する

**タスク:**長方形のフットプリントに 4 つの壁と 1 つのドア、3 つの窓と屋根から構成されるシンプルな「家」を作成するコマンドを記述します。

1. 長方形のフットプリントで 4 つの壁を作成する
2. 1 つ目の壁にドアを追加する
3. 残りの壁に窓を追加する
4. 壁の上に傾斜屋根を加える

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

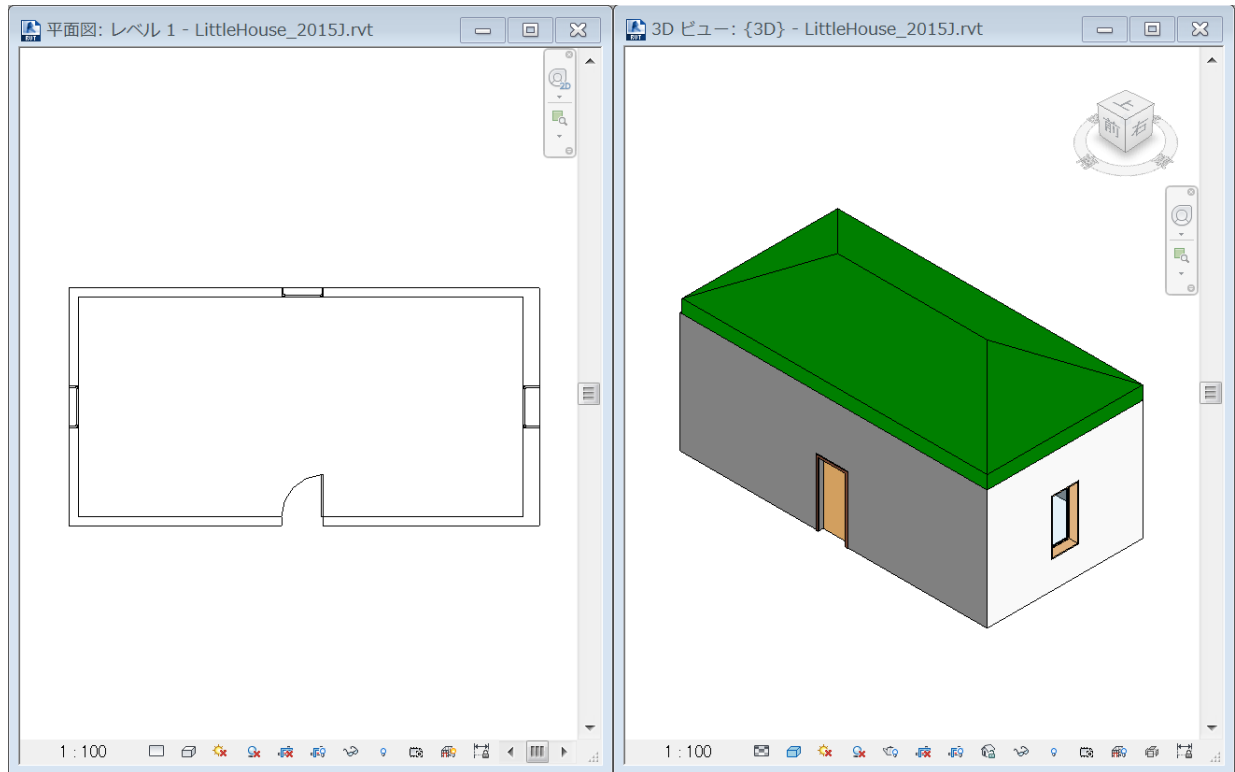


図 1. 4 つの壁とドア、窓と屋根から構成されるシンプルな家を作成

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. 壁を作成する
3. ドアを追加する
4. 窓を追加する
5. 傾斜屋根を加える
6. 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: **5\_ModelCreation.cs**
- コマンドクラス名: **ModelCreation**

追加する名前空間:

- Autodesk.Revit.DB.Structure(StructuralType のファミリ インスタンスの作成用)

追加の留意事項:

ElementFiltering クラスから次の関数を使用します:

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

1.2 前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それぞれ保持する m\_rvtApp と m\_rvtDoc を定義します。下記はその例です:

```
<C#>

// Model Creation - learn how to create elements

[Transaction(TransactionMode.Manual)]

public class ModelCreation : 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. 壁を作成する

前の実習では、新しいジオメトリ要素を作成するためにアプリケーション オブジェクトを使用しました:

- `Line.CreateBound(point1, point2)`

壁の新しいインスタンスを作成するために、Wall クラスの静的な “Create” メソッドを使用することができます。また、最初の引数としてドキュメントを渡すことができます。このメソッドには、壁を作成するための5つのオーバーロードが存在します(Revit API 開発者用ガイドの[「壁」項目](#)を参照して下さい)。例えば、下記は引数に渡したレベルに、既定の壁スタイルで新しい長方形のプロファイル(外形線)を作成します:

- `Wall.Create(doc, baseCurve, level, isStructural)`

ベース カーブは2つのエンドポイントを使用して、`Line.CreateBound`で定義することができます。壁を配置したい場所で、レベルを見つけるために `ElementFiltering.FindElement()` を使用することができます(例えば “レベル 1”)。

上部の壁を上レベルに拘束したい場合、もしくは“上部の拘束”を”レベル 2” に設定したい場合には、新しい壁を作成した後で、パラメータを追加で設定することができます。; “上部の拘束” のBuiltInParameter に対応するのはWALL\_HEIGHT\_TYPEです:

- `aWall.Parameter(BuiltInParameter.WALL_HEIGHT_TYPE).Set(level2.Id)`

```

<C#>
// create four walls

public List<Wall> CreateWalls()
{

```

```

// hard coding the size of the house for simplicity

double width = mmToFeet(10000.0);
double depth = mmToFeet(5000.0);

// get the levels we want to work on.
// Note: hard coding for simplicity. Modify here you use
// a different template.

Level level1 = (Level)ElementFiltering.FindElement(
    m_rvtDoc, typeof(Level), "レベル 1", null);

if (level1 == null)
{
    TaskDialog.Show("Revit Intro Lab", "Cannot find (Level 1). Maybe you
use a different template? Try with DefaultMetric.rte.");
    return null;
}

Level level2 = (Level)ElementFiltering.FindElement(
    m_rvtDoc, typeof(Level), "レベル 2", null);

if (level2 == null)
{
    TaskDialog.Show("Revit Intro Lab", "Cannot find (Level 2). Maybe you
use a different template? Try with DefaultMetric.rte.");
    return null;
}

// set four corner of walls.
// 5th point is for convenience to loop through.

double dx = width / 2.0;
double dy = depth / 2.0;

List<XYZ> pts = new List<XYZ>(5);
pts.Add(new XYZ(-dx, -dy, 0.0));
pts.Add(new XYZ(dx, -dy, 0.0));
pts.Add(new XYZ(dx, dy, 0.0));
pts.Add(new XYZ(-dx, dy, 0.0));
pts.Add(pts[0]);

// flag for structural wall or not.
bool isStructural = false;

// save walls we create.
List<Wall> walls = new List<Wall>(4);

```

```

// loop through list of points and define four walls.
for (int i = 0; i <= 3; i++)
{
    // define a base curve from two points.
    Line baseCurve = Line.CreateBound(pts[i], pts[i + 1]);
    // create a wall using the one of overloaded methods.
    Wall aWall = Wall.Create(m_rvtDoc, baseCurve, level1.Id,
isStructural);
    // set the Top Constraint to Level 2
    aWall.get_Parameter(BuiltInParameter.WALL_HEIGHT_TYPE).
        Set(level2.Id);
    // save the wall.
    walls.Add(aWall);
}

// This is important. we need these lines to have shrinkwrap working.
m_rvtDoc.Regenerate();
m_rvtDoc.AutoJoinElements();

return walls;
}
</C#>

```

## グラフィックスの再描画

この関数の終わりの2行には注意が必要です:

```

m_rvtDoc.Regenerate();
m_rvtDoc.AutoJoinElements();

```

これら2行を加えることによって、Revitは最初に個々の壁のグラフィックス情報を更新して、その後、壁のコーナーの自動調整を実行します。

### 実習:

- 長方形のフットプリントを形成する4つの壁を作成して、作成された壁のリストを返す関数を実装してください。  
この実習では、壁を任意の位置に配置できます

## 3. ドアを追加する

ドアや窓のようなコンポーネント ファミリのインスタンスを作成する際には、NewFamilyInstance() メソッドを使用する必要があります。NewfamilyInstance() には、12のオーバーロード メソッドがあります。どのメソッドを使用するかは、どの種類のファミリ インスタンスを作成したいかに依って変わります (例えば、参照に拘束されるか、ま

たは、自由に配置できるかなど)。Revit API 開発者用ガイド の「[ファミリインスタンス](#)」項目には、どのメソッド形式をどのような状況で利用するかというコメントと共に、要素カテゴリ毎に適用可能なオーバーロードメソッドの一覧を確認できます。より詳細な情報に関しては、そちらを参照してください。

下記は、ドアを追加する例です。ここでは、NewFamilyInstance() の次の形式を使用します:

- m\_rvtDoc.Create.NewFamilyInstance(xyzLocation, aFamilySymbol, hostObject, level, structuralType)

下記は、与えられた壁にドアを追加する関数の例です。この例では、ホストとして与えられた壁の中心にドアを配置しています。ドアは壁の下部に拘束されます(日本用のテンプレートを使ったプロジェクト上で実行する際には、ハードコードされたファミリ名を日本語版に適合させる必要があります):

```
<C#>
// add a door to the center of the given wall.
// cf. Developer Guide p140. NewFamilyInstance() for Doors and Window.

public void AddDoor(Wall hostWall)
{
    // hard coding the door type we will use.
    // e.g., "M_Single-Flush: 0915 x 2134mm

    const string doorFamilyName = "片側フラッシュ";
    const string doorTypeName = "0915 x 2134mm";
    const string doorFamilyAndTypeName =
        doorFamilyName + ": " + doorTypeName;

    // get the door type to use.

    FamilySymbol doorType =
        (FamilySymbol)ElementFiltering.FindFamilyType(
            m_rvtDoc, typeof(FamilySymbol), doorFamilyName, doorTypeName,
            BuiltInCategory.OST_Doors);

    if (doorType == null)
    {
        TaskDialog.Show("Revit Intro Lab", "Cannot find (" +
            doorFamilyAndTypeName + "). Maybe you use a different template? Try with
            DefaultMetric.rte.");
    }

    // get the start and end points of the wall.

    LocationCurve locCurve = (LocationCurve)hostWall.Location;
    XYZ pt1 = locCurve.Curve.GetEndPoint(0);
    XYZ pt2 = locCurve.Curve.GetEndPoint(1);
    // calculate the mid point.
    XYZ pt = (pt1 + pt2) / 2.0;

    // we want to set the reference as a bottom of the wall or level1.
```

```

ElementId idLevel1 = hostWall.get_Parameter(
    BuiltInParameter.WALL_BASE_CONSTRAINT).AsElementId();
Level level1 = (Level)m_rvtDoc.GetElement(idLevel1);

// finally, create a door.

FamilyInstance aDoor =
    m_rvtDoc.Create.NewFamilyInstance(
        pt, doorType, hostWall, level1, StructuralType.NonStructural);
}
</C#>

```

## 4. 窓を追加する

窓のインスタンスの作成は、基本的にドアと同じです。同じ `NewFamilyInstance()` メソッドを使用してかまいません:

- `m_rvtDoc.Create.NewFamilyInstance(xyzLocation, aFamilySymbol, hostObject, level, structuralType)`

検討すべき追加の項目は、敷居の高さを窓に加える点です。そうしないと、窓は壁の底に配置されてしまいます。対応するパラメータのセットによりそれをセットすることができます:

- `aWindow.Parameter(BuiltInParameter.INSTANCE_SILL_HEIGHT_PARAM).Set(sillHeight)`

下記は、与えられた壁に窓を追加する関数の例です。この例では、ホストとして与えられた壁の中心に窓を配置しています。窓は敷居高さ915mmで壁の最下段に拘束されます。

```

<C#>
// add a window to the center of the wall given.

public void AddWindow(Wall hostWall)
{
    // hard coding the window type we will use.
    // e.g., "M_Fixed: 0915 x 1830mm

    const string windowFamilyName = "固定";
    const string windowTypeName = "0915 x 1830 mm";
    const string windowFamilyAndTypeName =
        windowFamilyName + ": " + windowTypeName;
    double sillHeight = mmToFeet(915);

    // get the door type to use.

    FamilySymbol windowType =
        (FamilySymbol)ElementFiltering.FindFamilyType(
            m_rvtDoc, typeof(FamilySymbol), windowFamilyName, windowTypeName,
            BuiltInCategory.OST_Windows);
}

```



```

if (windowType == null)
{
    TaskDialog.Show("Revit Intro Lab", "Cannot find (" +
        windowFamilyAndTypeName +
        "). Try with DefaultMetric.rte.");
}

// get the start and end points of the wall.

LocationCurve locCurve = (LocationCurve)hostWall.Location;
XYZ pt1 = locCurve.Curve.GetEndPoint(0);
XYZ pt2 = locCurve.Curve.GetEndPoint(1);
// calculate the mid point.
XYZ pt = (pt1 + pt2) / 2.0;

// we want to set the reference as a bottom of the wall or level1.

ElementId idLevel1 =
    hostWall.get_Parameter(BuiltInParameter.WALL_BASE_CONSTRAINT).
    AsElementId();
Level level1 = (Level)m_rvtDoc.GetElement(idLevel1);

// finally create a window.

FamilyInstance aWindow = m_rvtDoc.Create.NewFamilyInstance(
    pt, windowType, hostWall, level1, StructuralType.NonStructural);

// set the sill height
aWindow.get_Parameter(BuiltInParameter.INSTANCE_SILL_HEIGHT_PARAM).
    Set(sillHeight);
}
</C#>

```

#### 実習:

- 引数として壁をとり、ドアまたは窓を与えられた壁に追加する関数を実装してください。  
ドアや窓のファミリタイプはハードコードして結構です。

## 5. 屋根を加える

システムファミリとコンポーネントファミリのインスタンスの作成方法を学習しました。屋根はシステムファミリであるため、指定のメソッドを使用する必要があります。屋根には2種類のメソッド、NewFootPrintRoof() と NewExtrusionRoof() が用意されています。Revit API 開発者用ガイドの「[屋根](#)」項目では、使用方法に関する詳細な記述があります。また、「NewRoof」SDKサンプルでは、フットプリントと押し出し屋根の両方の使用方法を確認することができます。

今回は、屋根のフットプリントの作成に次のメソッドを使用します。

- m\_rvtDoc.Create.NewFootPrintRoof(footprintCurve, level, roofType, curveMapping)

最後の引数 `curveMapping` は、`ModelCurveArray` データ型です。空のモデル カーブ配列を渡すと、関数が作成された屋根のカーブをそれに代入します。勾配角度のようなカーブのプロパティを設定するために、このカーブを後で使用する場合があります。

傾斜屋根を作成するために、屋根用に作成されたフットプリント モデル カーブの各エッジで角度を設定する必要があります。屋根を作成したら、`NewFootPrintRoof()` メソッドから返されたカーブマッピングをループすることで、適切な値を `DefineSlope()` と `SlopeAngle()` に設定します:

- `aRoof.DefineSlope(modelCurve) = True`
- `aRoof.SlopeAngle(modelCurve) = <some angular value>`

下記は、フットプリント屋根を作成して、傾斜を設定するスケルトン コードです:

```
<C#>
// create a roof.
FootPrintRoof aRoof = m_rvtDoc.Create.NewFootPrintRoof(
    footPrint, level2, roofType, out mapping);

// set the slope
foreach (ModelCurve modelCurve in mapping)
{
    aRoof.set_DefinesSlope(modelCurve, true);
    aRoof.set_SlopeAngle(modelCurve, 0.5);
}
</C#>
```

下記は、渡された4つの壁の上にフットプリント屋根を作成するサンプルコードです。(注意:壁の厚さで屋根の4つのコーナーを定義しています。そうでないと、屋根は壁の中心線に基づいて配置されます)。

```
<C#>
// add a roof over the rectangular profile of the walls we created.

public void addRoof(List<Wall> walls)
{
    // hard coding the roof type we will use.
    // e.g., "Basic Roof: Generic - 400mm"

    const string roofFamilyName = "標準屋根";
    const string roofTypeName = "一般 - 400 mm";
    const string roofFamilyAndTypeName =
        roofFamilyName + ": " + roofTypeName;

    // find the roof type

    RoofType roofType = (RoofType)ElementFiltering.FindFamilyType(
        m_rvtDoc, typeof(RoofType), roofFamilyName, roofTypeName, null);

    if (roofType == null)
    {
        TaskDialog.Show("Revit Intro Lab", "Cannot find (" +
```

```

roofFamilyAndTypeName + "). Maybe you use a different template? Try with
DefaultMetric.rte.");
}

// wall thickness to adjust the footprint of the walls
// to the outer most lines.
// Note: this may not be the best way.
// but we will live with this for this exercise.

double wallThickness = walls[0].Width;
double dt = wallThickness / 2.0;
List<XYZ> dts = new List<XYZ>(5);
dts.Add(new XYZ(-dt, -dt, 0.0));
dts.Add(new XYZ(dt, -dt, 0.0));
dts.Add(new XYZ(dt, dt, 0.0));
dts.Add(new XYZ(-dt, dt, 0.0));
dts.Add(dts[0]);

// set the profile from four walls

CurveArray footprint = new CurveArray();
for (int i = 0; i <= 3; i++)
{
    LocationCurve locCurve = (LocationCurve)walls[i].Location;
    XYZ pt1 = locCurve.Curve.GetEndPoint(0) + dts[i];
    XYZ pt2 = locCurve.Curve.GetEndPoint(1) + dts[i + 1];
    Line line = Line.CreateBound(pt1, pt2);
    footprint.Append(line);
}

// get the level2 from the wall

ElementId idLevel2 = walls[0].get_Parameter(
    BuiltInParameter.WALL_HEIGHT_TYPE).AsElementId();
Level level2 = (Level)m_rvtDoc.GetElement(idLevel2);

// footprint to morel curve mapping

ModelCurveArray mapping = new ModelCurveArray();

// create a roof.

FootPrintRoof aRoof = m_rvtDoc.Create.NewFootPrintRoof(
    footprint, level2, roofType, out mapping);

// setting the slope
foreach (ModelCurve modelCurve in mapping)
{
    aRoof.set_DefinesSlope(modelCurve, true);
    aRoof.set_SlopeAngle(modelCurve, 0.5);
}
}
</C#>

```

### 実習:

- 壁のリストをとって、その上の屋根を配置する関数を実装します。ここでは、長方形のフットプリントを形成する 4 つの壁が渡されるものとします。
- 壁、ウィンドウ、ドア、屋根をすべて一緒に作成して、シンプルな家を作成するコマンドを定義してみましょう。

## 6. サマリ

この実習では、Revit モデルを作成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。

- 建築要素 (壁、ドア、窓、屋根) のインスタンスを作成する

Autodesk Developer Network