

## Lab2 – L 形柱の作成

March 2010 by M. Harada  
Updated by Ryuji Ogasawara  
Last modified: 5/30/2024

<C#>C#バージョン</C#>

**目的:**この実習では、ファミリ API の基本を学習します。学習する項目は次のとおりです。

- 参照面を追加する
- パラメータを追加する
- 寸法を追加する

**タスク:** Lab1 で定義したコマンドを拡張し、L 字形のプロファイルを持つ柱を作成します。前の実習では、事前に定義された参照面、パラメータおよび寸法を使用しました。L-形のプロファイルを持つ柱を定義するために、独自の参照とパラメータ、寸法を加える必要があります。

- (0) Lab1 で定義したコマンド クラスを利用します。これは、この実習の開始点になります。ファミリ エディタと「柱(メートル単位).rft」テンプレートを使用し続けます
- (1) L 字形の「厚み」を計測する 2 つの参照面を追加する (例、“OffsetH” と “OffsetV”)
- (2) L 字形の「厚み」を定義する 2 つのパラメータを追加する(例、“Tw”と”Td”)
- (3) L 字形の「厚み」を計測する寸法をつかして、新しいパラメータでそれらにラベルを付ける(例、“Tw”と”Td”)

図 1 は、この実習に定義する予定の L-形の柱のイメージを示します。

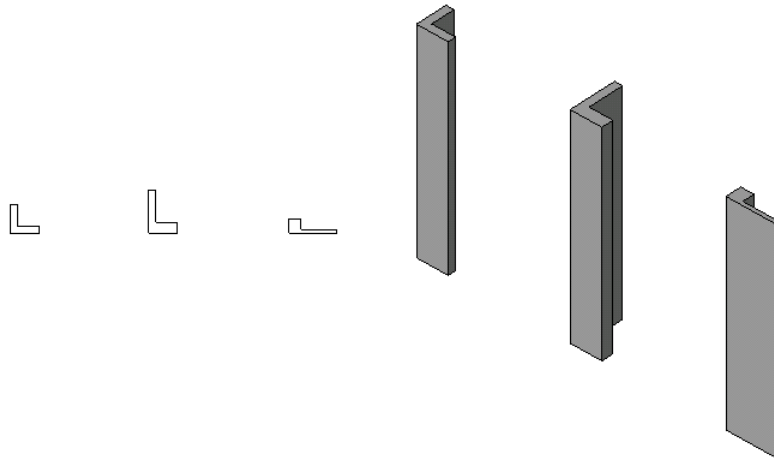
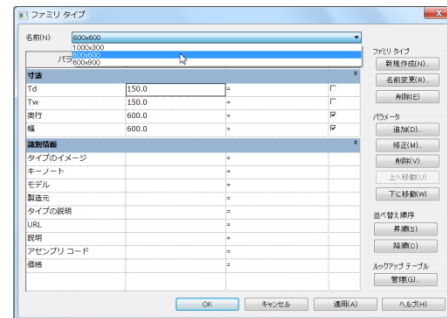
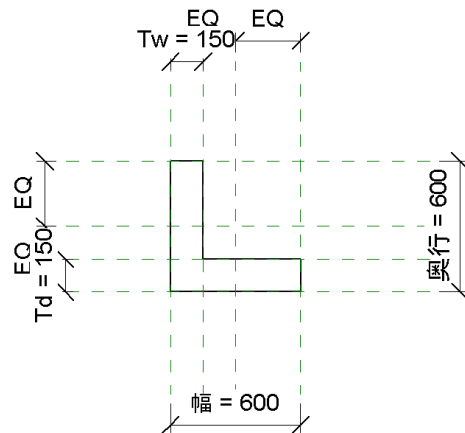


図 1.Lab2 で作成する L 字形プロファイルを持った柱ファミリ

この実習の実装と確認の手順は、下記のとおりです:

1. 別の外部コマンドを定義する
2. 参照面を追加する
3. L-形プロファイルで押し出しソリッドを作成する
4. `addAlignments()` を更新する
5. パラメータを追加する
6. 寸法を追加する
7. `addTypes()` を更新する

## 8. 作成した柱をテストする

付録 A.Lab2 で使われるヘルパー関数

## 1. 別の外部コマンドを定義する

Lab1 で定義したコマンドを拡張します。新しいクラスを定義するために Lab1 をコピーするか、Lab1 自体を拡張することができます(後者の場合、Lab1 の完了状態のプロジェクトをバックアップすることをお勧めします)。

1.1 Lab1 からコマンド クラスをコピーし、Lab2 で作業するための新しいクラスを定義してください。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: **2\_ColumnLShape.cs**
- コマンド クラス名: **RvtCmd\_FamilyCreateColumnLShape**

(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)

```
<C#>
[Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)]
[Autodesk.Revit.Attributes.Regeneration(Autodesk.Revit.Attributes.RegenerationOption.Manual)]
class RvtCmd_FamilyCreateColumnLShape : IExternalCommand
{
    ...
}
</C#>
```

## 2. 参照面を追加する

単純な L 字形のプロファイルを定義します。このために、2 つの参照面を追加していきます。

- 参照面「OffsetH」－ 水平、「正面」参照面の 150mm 上に
- 参照面「OffsetV」－ 垂直、「左」参照面から 150mm 右に

## 2.1 次の関数をクラスに付け加えてください:

```
<C#>
// =====
// (1.1) add reference planes
// =====
void addReferencePlanes()
{
    //
    // we are defining a simple L-shape profile like the following:
    //
    // 5 tw 4
    // +-+
    // | | 3      h = height
    // d | +---+ 2
    // +-----+ td
    // 0         1
    // 6 w
    //
    //
    // we want to add ref planes along (1) 2-3 and (2)3-4.
    // Name them "OffsetH" and "OffsetV" respectively. (H for horizontal, V
for vertical).
    //
    double tw = mmToFeet( 150.0 ); // thickness added for Lab2. Hard-
coding for simplicity.
    double td = mmToFeet( 150.0 );

    //
    // (1) add a horizontal ref plane 2-3.
    //
    // get a plan view
    View pViewPlan = findElement( typeof( ViewPlan ), "下基準レベル" ) as
View;

    // we have predefined ref plane: Left/Right/Front/Back
    // get the ref plane at Front, which is aligned to line 2-3
    ReferencePlane refFront = findElement( typeof( ReferencePlane ), "正面
" ) as ReferencePlane;

    // get the bubble and free ends from front ref plane and offset by td.
    //
    XYZ p1 = refFront.BubbleEnd;
    XYZ p2 = refFront.FreeEnd;
    XYZ pBubbleEnd = new XYZ(p1.X, p1.Y + td, p1.Z);
    XYZ pFreeEnd = new XYZ(p2.X, p2.Y + td, p2.Z);
    // create a new one reference plane and name it "OffsetH"
    //
    ReferencePlane refPlane =
_rvtDoc.FamilyCreate.NewReferencePlane( pBubbleEnd, pFreeEnd, XYZ.BasisZ,
pViewPlan );
    refPlane.Name = "OffsetH";

    //
}
```

```

// (2) do the same to add a vertical reference plane.
//

// find the ref plane at left, which is aligned to line 3-4
ReferencePlane refLeft = findElement( typeof( ReferencePlane ), "左" )
as ReferencePlane;

// get the bubble and free ends from front ref plane and offset by td.
//
p1 = refLeft.BubbleEnd;
p2 = refLeft.FreeEnd;
pBubbleEnd = new XYZ(p1.X + tw, p1.Y, p1.Z);
pFreeEnd = new XYZ(p2.X + tw, p2.Y, p2.Z);
// create a new reference plane and name it "OffsetV"
//
refPlane = _rvtDoc.FamilyCreate.NewReferencePlane( pBubbleEnd, pFreeEnd,
XYZ.BasisZ, pViewPlan );
refPlane.Name = "OffsetV";
}
</C#>

```

ここで、2つの参照面を作成しています。最初の参照面を見てみましょう。平面図を見ると、テンプレート内で「Lower Ref. Level」と名付けられていることがわかります(日本語テンプレートでは「下参照レベル」)。

新しい参照面を作成する主要なメソッドは、次のとおりです:

```
_rvtDoc.FamilyCreate.NewReferencePlane(pBubbleEnd, pFreeEnd, XYZ.BasisZ, pViewPlan)
```

BubbleEndとFreeEndが平面上の2点を決定します。BubbleEndは、平面上の原点と見なされます。3番目の引数は、切断ベクトルを示し、平面に沿って始点(BubbleEnd)と終点(FreeEnd)で定義される線分に直行するベクトルです。(注意: NewReferencePlane2と呼ばれる同様のメソッドも存在します。これは、平面上で3点をとります)。

このケースでは、既存の参照面のオフセットを作成して、既存の参照面から2つの端点座標をコピーするアプローチをとり、Y方向にオフセット値を加えています。

2.2 メインコマンドの Execute() 関数の IsRightTemplate() 関数を呼び出す後で、かつ、createSolid() 呼び出しの前の位置で、addReferencePlanes() 関数を呼び出してください。

```

<C#>
if( !isRightTemplate( BuiltInCategory.OST_Columns ) )
{
    Util.ErrorMsg( "Please open 柱(メートル単位).rft" );
    Return Result.Failed;
}

// (1.1) add reference planes

```

```

addReferencePlanes();

// (1.2) create a simple extrusion. This time we create a L-shape.
Extrusion pSolid = createSolid();
</c#>

```

2.3 この時点で、コードをビルドし、実行して確認することができます。

### 3. L 字形プロファイルで押し出しを作成します

前の実習では、長方形のプロファイルを押し出してソリッドを定義しました。この実習では、L 字形のプロファイルを使用します。前回との違いは、プロファイルの頂点を定義する部分だけです。このため、押し出し用コードは同じままで結構です。

3.1 次の関数をクラスに追加してください。このコードは、L 字形のプロファイルを定義します:

```

<c#>
// =====
// (1.2a) create a simple L-shaped profile
// =====
CurveArrArray createProfileLShape()
{
    //
    // define a simple L-shape profile
    //
    // 5 tw 4
    // +-+
    // | | 3      h = height
    // d | +---+ 2
    // +-----+ td
    // 0         1
    // 6 w
    //

    // sizes (hard coded for simplicity)
    // note: these need to match reference plane. otherwise, alignment
won't work.
    // as an exercise, try changing those values and see how it behaves.
    //
    double w = mmToFeet( 600.0 ); // those are hard coded for simplicity
here. in practice, you may want to find out from the references)
    double d = mmToFeet( 600.0 );
    double tw = mmToFeet( 150.0 ); // thickness added for Lab2
    double td = mmToFeet( 150.0 );

    // define vertices
    //
    const int nVerts = 6; // the number of vertices

    XYZ[] pts = new XYZ[] {
        new XYZ(-w / 2.0, -d / 2.0, 0.0),

```

```

        new XYZ(w / 2.0, -d / 2.0, 0.0),
        new XYZ(w / 2.0, (-d / 2.0) + td, 0.0),
        new XYZ((-w / 2.0) + tw, (-d / 2.0) + td, 0.0),
        new XYZ((-w / 2.0) + tw, d / 2.0, 0.0),
        new XYZ(-w / 2.0, d / 2.0, 0.0),
        new XYZ(-w / 2.0, -d / 2.0, 0.0) }; // the last one is to make the
loop simple

// define a loop. define individual edges and put them in a curveArray
//
CurveArray pLoop = _rvtApp.Create.NewCurveArray();
for( int i = 0; i < nVerts; ++i )
{
    Line line = Line.CreateBound( pts[i], pts[i + 1] );
    pLoop.Append( line );
}

// then, put the loop in the curveArrArray as a profile
//
CurveArrArray pProfile = _rvtApp.Create.NewCurveArrArray();
pProfile.Append( pLoop );
// if we come here, we have a profile now.

return pProfile;
}
</c#>

```

ここでは、L 字形状のサイズと頂点と同様に、コードを単純化するためにオフセット値をハードコーディングしています。この段階でそれらの固定値を定義する主な目的は、参照で位置合わせを設定するためです。これらの値にパラメータを割り当てれば、それらを再定義することができます。

3.2 定義したプロファイルを使用して、押し出しソリッドを作成します。createSolid() 関数を参照して、createProfileRectangle() 関数の呼び出しを createProfileLShape() 関数呼び出しに置き換えてください:

```

<c#>
// =====
// (1.2) create a simple solid by extrusion with L-shape profile
// =====
Extrusion createSolid()
{
    //
    // (1) define a simple L-shape profile
    //
    //CurveArrArray pProfile = createProfileRectangle();
    CurveArrArray pProfile = createProfileLShape(); // Lab2

    ...
}
</c#>

```

3.3 この時点で、コードをビルドし、実行して確認することができます。

## 4. addAlignment()を更新する

位置合わせを行う関数を修正する必要があります。長方形のプロファイルでは、6つの対応参照面への6つの面を位置合わせさせました。先程定義したL字形プロファイルでは、2つの面が追加されています。位置合わせの基本概念は同じままです。ただし、findFace()ヘルパー関数は、面をより正確に識別するために3つ目のパラメータとして参照面を利用するように拡張する必要があります。

4.1 addAlignments() 関数を見つけて、次のコードで関数を更新してください:

```
<C#>
// =====
// (2.1) add alignments
// =====
void addAlignments( Extrusion pBox )
{
    //
    // (1) we want to constrain the upper face of the column to the "Upper
Ref Level"
    //

    // which direction are we looking at?
    //
    View pView = findElement( typeof( View ), "正面" ) as View;

    // find the upper ref level
    // findElement() is a helper function. see below.
    //
    Level upperLevel = findElement( typeof( Level ), "上基準レベル" ) as
Level;
    Reference ref1 = upperLevel.GetPlaneReference();

    // find the face of the box
    // findFace() is a helper function. see below.
    //
    PlanarFace upperFace = findFace( pBox, new XYZ( 0.0, 0.0, 1.0 ) ); //
find a face whose normal is z-up.
    Reference ref2 = upperFace.Reference;

    // create alignments
    //
    _rvtDoc.FamilyCreate.NewAlignment( pView, ref1, ref2 );

    //
    // (2) do the same for the lower level
    //

    // find the lower ref level
    // findElement() is a helper function. see below.
    //

```



```

    Level lowerLevel = findElement( typeof( Level ), "下基準レベル" ) as
Level;
    Reference ref3 = lowerLevel.GetPlaneReference();

    // find the face of the box
    // findFace() is a helper function. see below.
    PlanarFace lowerFace = findFace( pBox, new XYZ( 0.0, 0.0, -1.0 ) ); //
find a face whose normal is z-down.
    Reference ref4 = lowerFace.Reference;

    // create alignments
    //
    _rvtDoc.FamilyCreate.NewAlignment( pView, ref3, ref4 );

    //
    // (3) same idea for the Right/Left/Front/Back
    //
    // get the plan view
    // note: same name maybe used for different view types. either one
should work.
    View pViewPlan = findElement( typeof( ViewPlan ), "下基準レベル" ) as
View;

    // find reference planes
    ReferencePlane refRight = findElement( typeof( ReferencePlane ), "右" )
as ReferencePlane;
    ReferencePlane refLeft = findElement( typeof( ReferencePlane ), "左" )
as ReferencePlane;
    ReferencePlane refFront = findElement( typeof( ReferencePlane ), "正面
" ) as ReferencePlane;
    ReferencePlane refBack = findElement( typeof( ReferencePlane ), "背面" )
as ReferencePlane;
    ReferencePlane refOffsetV = findElement( typeof( ReferencePlane ),
"OffsetV" ) as ReferencePlane; // added for L-shape
    ReferencePlane refOffsetH = findElement( typeof( ReferencePlane ),
"OffsetH" ) as ReferencePlane; // added for L-shape

    // find the face of the box
    // Note: findFace need to be enhanced for this as face normal is not
enough to determine the face.
    //
    PlanarFace faceRight = findFace( pBox, new XYZ( 1.0, 0.0, 0.0 ),
refRight ); // modified for L-shape
    PlanarFace faceLeft = findFace( pBox, new XYZ( -1.0, 0.0, 0.0 ) );
    PlanarFace faceFront = findFace( pBox, new XYZ( 0.0, -1.0, 0.0 ) );
    PlanarFace faceBack = findFace( pBox, new XYZ( 0.0, 1.0, 0.0 ),
refBack ); // modified for L-shape
    PlanarFace faceOffsetV = findFace( pBox, new XYZ( 1.0, 0.0, 0.0 ),
refOffsetV ); // added for L-shape
    PlanarFace faceOffsetH = findFace( pBox, new XYZ( 0.0, 1.0, 0.0 ),
refOffsetH ); // added for L-shape

    // create alignments

```

```

//
_rvtDoc.FamilyCreate.NewAlignment( pViewPlan, refRight.GetReference(),
faceRight.Reference );
_rvtDoc.FamilyCreate.NewAlignment( pViewPlan, refLeft.GetReference(),
faceLeft.Reference );
_rvtDoc.FamilyCreate.NewAlignment( pViewPlan, refFront.GetReference(),
faceFront.Reference );
_rvtDoc.FamilyCreate.NewAlignment( pViewPlan, refBack.GetReference(),
faceBack.Reference );
_rvtDoc.FamilyCreate.NewAlignment( pViewPlan, refOffsetV.GetReference(),
faceOffsetV.Reference ); // added for L-shape
_rvtDoc.FamilyCreate.NewAlignment( pViewPlan, refOffsetH.GetReference(),
faceOffsetH.Reference ); // added for L-shape
}
</c#>

```

ここまでで、ヘルパー関数 findFace() には 2 つのバージョンが出来ました:

- `PlanarFace findFace(Extrusion pBox, XYZ normal)`
- `PlanarFace findFace(Extrusion pBox, XYZ normal, ReferencePlane refPlane)`

最初の関数は以前のものと同じです。2 つめの関数は、3 番目の引数として、位置合わせのために必要とする参照面を利用します。この関数には、与えられた法線で、面が与えられた参照面上にあるかチェックする機能が加えてあります。今回は、面を識別するために、2 つめのバージョンの関数を使用する必要があります。法線として「右」、「背面」、「OffsetH」、「OffsetV」の参照面に沿う面は、面を決定するには不十分です。完全なコードは、このドキュメントの終わりに記載されています(付録 A)。このクラスの終わりにコピー&ペーストしてください。

面を見つけるルーチン以外は、ロジックの残りは同じままです。参照面「OffsetH」と「OffsetV」で追加の位置合わせを持つことができました。

4.2 ここで、コードはビルドし、実行して確認することができます。

## 5. パラメータを追加する

次に、2 つのパラメータ「Tw」および「Td」を追加する必要があります。その後で、L 字形プロファイルの厚み寸法として、これらのパラメータを関連づけます。

5.1 次の関数をクラスに加えてください:

```

<c#>
// =====
//   (3.1) add parameters
// =====
void addParameters()
{

```

```

// parameter group for Dimension is PG_GEOMETRY in API
//
FamilyParameter paramTw = _rvtDoc.FamilyManager.AddParameter(
    "Tw", BuiltInParameterGroup.PG_GEOMETRY, ParameterType.Length,
    false );

FamilyParameter paramTd = _rvtDoc.FamilyManager.AddParameter(
    "Td", BuiltInParameterGroup.PG_GEOMETRY, ParameterType.Length,
    false );

// give initial values
//
double tw = mmToFeet( 150.0 ); // hard coded for simplicity
double td = mmToFeet( 150.0 );
_rvtDoc.FamilyManager.Set( paramTw, tw );
_rvtDoc.FamilyManager.Set( paramTd, td );
}
</C#>

```

パラメータを加えるために、私たちは、Family Manager クラスの addParameter() メソッドを使用します:

```

_rvtDoc.FamilyManager.AddParameter( "Tw",
    BuiltInParameterGroup.PG_GEOMETRY, ParameterType.Length, False)

```

最初の引数はパラメータの名前で、第2引数は、タイプ ダイアログでパラメータがどこに表示されるかを決定するパラメータ グループです。今回のケースでは、PG\_Geometry を指定し、「幅」と「奥行」と同じように、パラメータは「寸法」の下に表示するようにします。第3引数は、パラメータのタイプです。ここでは、「Tw」を「長さパラメータ」として設定しています。最後の引数は、パラメータがインスタンス・パラメータかファミリ・パラメータかを示すフラグです。

設定値は、ここまでの実習で用いたものと同じです:

```

_rvtDoc.FamilyManager.Set(paramTw, tw)

```

5.2 メインの コマンド関数から addParameters() を呼び出してください:

```

<C#>
public Result Execute(
    ExternalCommandData commandData,
    ref string message,
    ElementSet elements )
{
    ...

    // (2) add alignment
    addAlignments(pSolid);

    // (3.1) add parameters
    addParameters();
}

```

```
...
}
</c#>
```

5.3 この時点で、コードをビルドし、実行して確認することができます。

## 6. 寸法を追加する

下記のような定義で、参照面間に 2 つの寸法を追加して、各パラメータのラベルを作成します:

- 「左」と「OffsetV」の間の寸法 — パラメータ「Tw」
- 「正面」と「OffsetH」の間の寸法 — パラメータ「Td」

6.1 次の関数をコマンド クラスに追加します:

```
<c#>
// =====
//   (3.2) add dimensions
// =====
void addDimensions()
{
    // find the plan view
    //
    View pViewPlan = findElement( typeof( ViewPlan ), "下基準レベル" ) as
View;

    // find reference planes
    //
    ReferencePlane refLeft = findElement( typeof( ReferencePlane ), "左" )
as ReferencePlane;
    ReferencePlane refFront = findElement( typeof( ReferencePlane ), "正面
" ) as ReferencePlane;
    ReferencePlane refOffsetV = findElement( typeof( ReferencePlane ),
"OffsetV" ) as ReferencePlane; // OffsetV is added for L-shape
    ReferencePlane refOffsetH = findElement( typeof( ReferencePlane ),
"OffsetH" ) as ReferencePlane; // OffsetH is added for L-shape

    //
    // (1) add dimension between the reference planes 'Left' and 'OffsetV',
and label it as 'Tw'
    //

    // define a dimension line
    //
    XYZ p0 = refLeft.FreeEnd;
    XYZ p1 = refOffsetV.FreeEnd;
    Line pLine = Line.CreateBound( p0, p1 );
```

```

        // define references
        //
        ReferenceArray pRefArray = new ReferenceArray();
        pRefArray.Append( refLeft.GetReference() );
        pRefArray.Append( refOffsetV.GetReference() );

        // create a dimension
        //
        Dimension pDimTw = _rvtDoc.FamilyCreate.NewDimension( pViewPlan, pLine,
pRefArray );

        // add label to the dimension
        //
        FamilyParameter paramTw = _rvtDoc.FamilyManager.get_Parameter( "Tw" );
        pDimTw.FamilyLabel = paramTw;

        //
        // (2) do the same for dimension between 'Front' and 'OffsetH', and
labe it as 'Td'
        //

        // define a dimension line
        //
        p0 = refFront.FreeEnd;
        p1 = refOffsetH.FreeEnd;
        pLine = Line.CreateBound( p0, p1 );

        // define references
        //
        pRefArray = new ReferenceArray();
        pRefArray.Append( refFront.GetReference() );
        pRefArray.Append( refOffsetH.GetReference() );

        // create a dimension
        //
        Dimension pDimTd = _rvtDoc.FamilyCreate.NewDimension( pViewPlan, pLine,
pRefArray );

        // add label to the dimension
        //
        FamilyParameter paramTd = _rvtDoc.FamilyManager.get_Parameter( "Td" );
        pDimTd.FamilyLabel = paramTd;
    }
</C#>

```

水平寸法と垂直寸法の2つの寸法を追加しています。ここでは、水平寸法について解説しますが、基本的に垂直寸法にも同じロジックを当てはめることができます。

寸法を作成する主なメソッドには次のメソッドを使用します:

```
_rvtDoc.FamilyCreate.NewDimension(pViewPlan, pLine, pRefArray)
```

最初の引数はビューです。今回の場合、平面図を指定します。第 2 引数は、寸法の初期位置です。ここで、2 つの参照面から 1 つを使用します。3 番目は、次のコードが示すような、「左」と「OffsetV」参照を含む、参照の配列です:

```
ReferenceArray pRefArray = new ReferenceArray();
pRefArray.Append( refLeft.GetReference());
pRefArray.Append( refOffsetV.GetReference());
```

下記では、前のセクションで定義したパラメータ「Tw」でラベルを追加します:

```
FamilyParameter paramTw = _rvtDoc.FamilyManager.get_Parameter( "Tw" );
pDimTw.Label = paramTw;
```

6.2 メインのコマンド関数から addDimensions() を呼び出してください:

```
<C#>
public Result Execute( ...
{
    ...

    // (3.1) add parameters
    addParameters();

    // (3.2) add dimensions
    addDimensions();

    ...
}
</C#>
```

6.3 ここで、コードをビルドし、実行して確認することができます。

## 7. addTypes() の更新

タイプを定義する際に考慮しなければならない、さらに 2 つのパラメータがあります。addType() 関数を更新していきましょう。今回、L 字形の厚みを定義する 2 つの追加引数 “Tw” と “Td” を用意します。「幅」、「深さ」、「Tw」および「Td」に対応する寸法名で、次の値を持つ 2 つのタイプを加えましょう:

- “600 x 900” - 600 x 900 x 150 x 225
- “1000 x 300” - 1000 x 300 x 250 x 75
- “600 x 600” - 600 x 600 x 150 x 150

## 7.1 下記の関数をクラスに加えてください:

```
<C#>
// add one type (version 2)
//
void addType( string name, double w, double d, double tw, double td )
{
    // get the family manager from the current doc
    FamilyManager pFamilyMgr = _rvtDoc.FamilyManager;

    // add new types with the given name
    //
    FamilyType type1 = pFamilyMgr.NewType( name );

    // look for 'Width' and 'Depth' parameters and set them to the given
value
    //

    // first 'Width'
    //
    FamilyParameter paramW = pFamilyMgr.get_Parameter( "幅" );
    double valW = mmToFeet( w );
    if( paramW != null )
    {
        pFamilyMgr.Set( paramW, valW );
    }

    // same idea for 'Depth'
    //
    FamilyParameter paramD = pFamilyMgr.get_Parameter( "奥行" );
    double valD = mmToFeet( d );
    if( paramD != null )
    {
        pFamilyMgr.Set( paramD, valD );
    }

    // let's set 'Tw' and 'Td'
    //
    FamilyParameter paramTw = pFamilyMgr.get_Parameter( "Tw" );
    double valTw = mmToFeet( tw );
    if( paramTw != null )
    {
        pFamilyMgr.Set( paramTw, valTw );
    }
    FamilyParameter paramTd = pFamilyMgr.get_Parameter( "Td" );
    double valTd = mmToFeet( td );
    if( paramTd != null )
    {
        pFamilyMgr.Set( paramTd, valTd );
    }
}
</C#>
```

これら2つのパラメータの設定以外に、変更箇所はありません。同じロジックは「Tw」および「Td」の設定にも当てはまります。

7.2 下記でaddType()を更新してください:

```
<C#>
// =====
//      (3.3) add types
// =====
void addTypes()
{
    //  addType(name, Width, Depth, Tw, Td)
    //
    addType( "600x900", 600.0, 900.0, 150.0, 225.0 );
    addType( "1000x300", 1000.0, 300.0, 250.0, 75.0 );
    addType( "600x600", 600.0, 600.0, 150.0, 150.0 );
}
<C#>
```

## 8. 作成した柱をテストする

ビルドしてテスト用に実行する準備が整いました。

テスト用に次のような行を Revit のアドイン マニフェスト ファイルに付け加えることができます (新しいコマンドを追加するか、Lab1から置き換えることができます)。もちろん、お使いの環境と一致するよう、必要な調節を行ってください。

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<RevitAddIns>

  <AddIn Type="Command">
    <Assembly>FamilyCs.dll</Assembly>
    <AddInId>FC5E150A-967B-4cc9-A7B0-3AA29C5DA9D9</AddInId>
    <FullName>FamilyCs.RvtCmd_FamilyCreateColumnLShape</FullName>
    <Text>Family Labs Create L-Shape Column</Text>
    <Description>Family API lab 2 to create L-shaped column</Description>
    <VisibilityMode>NotVisibleInProject</VisibilityMode>
    <AccessibilityClassName>Revit.Samples.SampleAccessibilityCheck </AccessibilityClassName>
    <VendorId>ADNP</VendorId>
    <VendorDescription>Autodesk, Inc. www.autodesk.com</VendorDescription>
  </AddIn>

</RevitAddIns>
```

「柱(メートル単位).rft」テンプレートを使用して、ファミリ エディタを起動してください。



コマンドを実行すると、柱のプロファイルがL字形を示すことがわかります。柱をテストしてみてください:

- 平面図に2つの寸法が追加されたことを確認できますか?
- 寸法は正しくラベル付けされていますか?
- ファミリタイプ ダイアログで「寸法」の下2つの追加パラメータ「Tw」と「Td」が表示されますか?
- 3つのタイプが正しく作成されていますか?
- 異なるタイプを選択して適用すると、それに沿って柱のサイズが変更されますか?
- プロジェクトに作成したファミリを追加してください。柱は期待した「振る舞い」を持っていますか?

次の実習では、ここで作成した柱ファミリ上に、計算式とマテリアルを追加していきます。

## 付録 A. Lab2 で使われるヘルパー関数

Lab2 では、もう 1 つのヘルパー関数を追加しました。必要に応じて下記のコードをコピー&ペーストで作成中のコードに貼り付けてください。

- findFace () – バージョン 2 です。与えられた押し出しソリッドで、与えられた法線を持ち、与えられた参照面上に沿う平面を見つけます。

```
<C#>
// =====
// helper function: given a solid, find a planar
// face with the given normal (version 2)
// this is a slightly enhanced version of the previous
// version and checks if the face is on the given reference plane.
// =====
PlanarFace findFace( Extrusion pBox, XYZ normal, ReferencePlane
refPlane )
{
    // get the geometry object of the given element
    //
    Options op = new Options();
    op.ComputeReferences = true;
    GeometryElement geomObjs = pBox.get_Geometry( op );

    // loop through the array and find a face with the given normal
    //
    foreach( GeometryObject geomObj in geomObjs )
    {
        if( geomObj is Solid ) // solid is what we are interested in.
        {
            Solid pSolid = geomObj as Solid;
            FaceArray faces = pSolid.Faces;
            foreach( Face pFace in faces )
            {
                PlanarFace pPlanarFace = (PlanarFace) pFace;
                // check to see if they have same normal
            }
        }
    }
}
```

```

        if( ( pPlanarFace != null ) && pPlanarFace.
FaceNormal.IsAlmostEqualTo( normal ) )
        {
            // additionally, we want to check if the face is on the
reference plane
            //
            XYZ p0 = refPlane.BubbleEnd;
            XYZ p1 = refPlane.FreeEnd;
            Line pCurve = Line.CreateBound( p0, p1 );
            if( pPlanarFace.Intersect( pCurve ) ==
SetComparisonResult.Subset )
            {
                return pPlanarFace; // we found the face
            }
        }
    }
}

// will come back later as needed.
//
//else if (geomObj is Instance)
//{
//}
//else if (geomObj is Curve)
//{
//}
//else if (geomObj is Mesh)
//{
//}
}

// if we come here, we did not find any.
return null;
}
</c#>

```