**Revit Family API Labs**

**Lab2 – Create L-Shape Column**

March 2010 by M. Harada

Last updated, Date : March 19, 2014

C# version

**Objective:** In this lab, we will learn the basics of family API. We’ll learn how to:

* add reference planes
* add parameters
* add dimensions

**Tasks:** We’ll extend the command we have defined in the Lab1 and creates a column that has an L-shape profile. In the previous lab, we have used predefined reference planes, parameters and dimensions. In order to define a column with an L-shaped profile, we will need to add our own references, parameters and dimensions.

1. Take the command class which we have defined in the Lab1. This will be the starting point of this lab. We’ll continue using the Family Editor and "Metric Column.rft" template.
2. Define two additional reference planes that measures “thicknesses” of the L-shape profile. (e.g., “OffsetH” and “OffsetV”).
3. Add two parameters that define the thicknesses of the L-Shape (e.g., “Tw” and “Td”).
4. Add dimensions that measures thickness of L-shape and label them with the new parameters, e.g., “Tw” and “Td”.

Figure 1 shows the image of the L-shaped columns that we are going to define in this lab.





Figure 1. A column family with L-shape profile we will be creating in Lab2.

The following is the breakdown of step by step instructions in this lab:

1. [Define Another External Command](#defineAnchoterExternalCommand)

1. [Add Reference Planes](#addReferencePlanes)
2. [Create an Extrusion with L-Shape Profile](#createExtrusionWithLShapeProfile)
3. [Update addAlignments()](#updateAddAlignment)
4. [Add Parameters](#addParameters)
5. [Add Dimensions](#addDimensions)
6. [Update addTypes()](#updateAddTypes)
7. [Test Your Column](#testYourColumn)

[Appendix A. Helper Functions Used in Lab2](#AppendixAHelperFunctionsInLab2)

1. **Define Another External Command**

We’ll be extending the command we have defined in Lab1. You can either copy it to define a new class or continue extending the existing one on top of it. (Just make sure to back up in case you need it to start again.)

* 1. Copy the command class from Lab1 and define a new one to work on in Lab2. Let’s name them as:
* File name: **2\_ColumnLShape.cs (or .vb)**
* Command class name: **RvtCmd\_FamilyCreateColumnLShape**

(Once again, you may choose to use any names you want here. When you do so, just remember what you are calling your own project, and substitute these names as needed while reading the instruction in this document.)

[Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Automatic)] [Autodesk.Revit.Attributes.Regeneration(Autodesk.Revit.Attributes.RegenerationOption.Manual)]

class RvtCmd\_FamilyCreateColumnLShape : IExternalCommand

{

...

}

**2. Add Reference Planes**

We are going to define a simple L-shape profile. In order to do so, we will add two reference planes.

* Reference plane “OffsetH” – horizontal, 150mm above the reference plane ” Front”.
* Reference plane “OffsetV” – vertical, 150 mm off to the right from the reference plane “Right”.

2.1 Add the following function to the class:

// ==============================

// (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 horizonal ref plane 2-3.

//

// get a plan view

View pViewPlan = findElement( typeof( ViewPlan ), "Lower Ref. Level" ) 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 ), "Front" ) 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 ), "Left" ) 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";

}

We are creating two reference planes. Let’s look at the first one. We are looking at the plan view, which is named as “Lower Ref. Level” in the template.

The main method to create a new reference plane is this:

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

Here, Bubble End and Free End determine two points on the plane. The Bubble End is considered as an origin of the plane. The third argument (i.e., cut vector) determines a vector, which lies on the plane and orthogonal to a line defined by bubble end and free end. (Note: there is similar method called NewReferencePlane2. This one takes three points on the plane.)

In our case, since we are making an offset of an existing reference plane, we take an approach of copying the coordinates of two end points from the existing reference plane and add the offset value in Y-direction.

2.2 Call addReferencePlanes() function from your main command function Execute() after IsRightTemplate() and before createSolid() call.

if( !isRightTemplate( BuiltInCategory.OST\_Columns ) )

{

Util.ErrorMsg( "Please open Metric Column.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();

2.3 Your code should build and run at this point.

**3.** **Create an Extrusion with L-Shape Profile**

In the previous lab, we have defined a solid by extruding a rectangular profile. In this lab, we will use a L-shape profile. Only difference in this lab is the part that defines vertices of the profile. The code for extrusion remains the same.

3.1 Add the following function to the class. This code defines a profile with a L-shape:

// ===========================================

// (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;

}

Once again, we are hard-coding the initial value for the amount of offset as well as overall size and vertices of L-shape for simplicity. The main purpose of having those canned values at this stage is set the alignment with the references. They can be redefined once we assign parameters to it.

3.2 Using the profile we have just defined, we then create a solid from extrusion. Go to createSolid(). Replace the call to createProfileRectangle() with call createProfileLShape():

// =================================================================

// (1.2) create a simple solid by extrusion with L-shape profile

// =================================================================

Extrusion createSolid()

{

//

// (1) define a simple L-shape profile

//

//CurveArrArray pProflie = createProfileRectangle();

CurveArrArray pProfile = createProfileLShape(); // Lab2

...

3.3 Your code should build and run at this point.

**4.** **Update addAlignment()**

The function that adds alignments needs to be updated. With a rectangular profile, we aligned six faces to the six corresponding reference planes. With the L-shape profile we have just defined, we have addition two more faces; one more facing back and one facing right. The basic idea of adding an alignment still remains the same. But our findFace() helper function needs enhancement to take a reference plane as the third parameter in order to identify the face more accurately.

4.1 Locate the addAlignments() function, and update it with the following code:

// ======================================

// (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 ), "Front" ) as View;

// find the upper ref level

// findElement() is a helper function. see below.

//

Level upperLevel = findElement( typeof( Level ), "Upper Ref Level" ) as Level;

Reference ref1 = upperLevel.PlaneReference;

// 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 ), "Lower Ref. Level" ) as Level;

Reference ref3 = lowerLevel.PlaneReference;

// 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 ), "Lower Ref. Level" ) as View;

// find reference planes

ReferencePlane refRight = findElement( typeof( ReferencePlane ), "Right" ) as ReferencePlane;

ReferencePlane refLeft = findElement( typeof( ReferencePlane ), "Left" ) as ReferencePlane;

ReferencePlane refFront = findElement( typeof( ReferencePlane ), "Front" ) as ReferencePlane;

ReferencePlane refBack = findElement( typeof( ReferencePlane ), "Back" ) 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.Reference, faceRight.Reference );

\_rvtDoc.FamilyCreate.NewAlignment( pViewPlan, refLeft.Reference, faceLeft.Reference );

\_rvtDoc.FamilyCreate.NewAlignment( pViewPlan, refFront.Reference, faceFront.Reference );

\_rvtDoc.FamilyCreate.NewAlignment( pViewPlan, refBack.Reference, faceBack.Reference );

\_rvtDoc.FamilyCreate.NewAlignment( pViewPlan, refOffsetV.Reference, faceOffsetV.Reference ); // added for L-shape

\_rvtDoc.FamilyCreate.NewAlignment( pViewPlan, refOffsetH.Reference, faceOffsetH.Reference ); // added for L-shape

}

The helper function findFace() now has two versions:

* PlanarFace findFace(Extrusion pBox, XYZ normal)
* PlanarFace findFace(Extrusion pBox, XYZ normal, ReferencePlane refPlane)

The first one is the same as before. The second takes a reference plane that we want intend to align as the third argument. It adds an extra checking if the face with the given normal also lies on the given reference plane. We need to use the second version to identify faces that lie reference planes on “Right”, “Back”, “OffsetH” and “OffsetV” as normal is not enough to determine the face. The full code is attached at the end of this doc, the section, [Appendix A](#AppendixAHelperFunctionsInLab2). Please copy it and paste to the end of this class.

Other than the routine to find the face, the rest of the logic remains the same. We have additional alignments with reference planes “OffsetH” and “OffsetV”.

4.2 Your code should build and run at this point.

**5.** **Add Parameters**

The next thing we need to do is to add two parameters, “Tw” and “Td”. Later, we will associate these parameters with dimensions for thickness of the L-shape profile.

5.1 Add the following functions to the class:

// ======================================

// (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 );

}

To add a parameter, we use a method addParameter() of the Family Manager class:

\_rvtDoc.FamilyManager.AddParameter( "Tw",

BuiltInParameterGroup.PG\_GEOMETRY, ParameterType.Length, False)

The first argument is the name of the parameter. The second is the parameter group, which determines where in the type dialog the parameter appears. In our case, PG\_Geometry will put our parameter under “Dimensions”, the same as “Width” and “Depth”. The third one is type of the parameter; here we are setting “Tw” as the length parameter. The last one is a flag if the parameter is Instance parameter or family parameter.

Setting the value is the same as we did in the Labs:

\_rvtDoc.FamilyManager.Set(paramTw, tw)

5.2 Call addParameters() from your main command function:

public Result Execute(

ExternalCommandData commandData,

ref string message,

ElementSet elements )

{

...

// (2) add alignment

addAlignments(pSolid);

// (3.1) add parameters

addParameters();

...

}

5.3 Your code should build and run at this point.

**6.** **Add Dimensions**

We now add two dimensions between the reference planes and label them with parameters we have just defined:

* Dimension between “Left” and “OffsetV” – parameter “Tw”
* Dimension between “Front” and “OffsetH” – parameter “Td”

6.1 add the following function to the command class:

// ======================================

// (3.2) add dimensions

// ======================================

void addDimensions()

{

// find the plan view

//

View pViewPlan = findElement( typeof( ViewPlan ), "Lower Ref. Level" ) as View;

// find reference planes

//

ReferencePlane refLeft = findElement( typeof( ReferencePlane ), "Left" ) as ReferencePlane;

ReferencePlane refFront = findElement( typeof( ReferencePlane ), "Front" ) 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.Reference );

pRefArray.Append( refOffsetV.Reference );

// 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 lable 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.Reference );

pRefArray.Append( refOffsetH.Reference );

// 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;

}

We are adding two dimensions here: horizontal one and vertical one. Let’s focus on the horizontal one now. Once you understand one, basically the same logic applies.

The main method we use to create a dimension is this one:

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

The first argument is a view. In our case, we are looking at a plan view. The second argument is the initial location of a dimension. Here we take the one end from two reference planes. The third is the array of references. In our case, it is an array that contains Left reference and OffsetV as following code shows:

ReferenceArray pRefArray = new ReferenceArray();

pRefArray.Append( refLeft.Reference );

pRefArray.Append( refOffsetV.Reference );

The following adds the label with the parameter “Tw” which has defined in the previous section:

FamilyParameter paramTw = \_rvtDoc.FamilyManager.get\_Parameter( "Tw" );

pDimTw.Label = paramTw;

6.2 Call addDimensions() from your main command function:

public Result Execute( ...

{

...

// (3.1) add parameters

addParameters();

// (3.2) add dimensions

addDimensions();

...

}

6.3 Your code should build and run at this point.

**7.** **Update addTypes()**

We have two more parameters to consider when defining types. Let’s make an updated version of addType() function. This time, it takes two additional arguments to define the thickness of L-shape, “Tw” and “Td”. Let’s add a couple of types, for example, ones with dimensions corresponding to name, “Width” , “Depth”, “Tw” and “Td” to:

* “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 Add the following function to the class:

// 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( "Width" );

double valW = mmToFeet( w );

if( paramW != null )

{

pFamilyMgr.Set( paramW, valW );

}

// same idea for 'Depth'

//

FamilyParameter paramD = pFamilyMgr.get\_Parameter( "Depth" );

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 );

}

}

Other than setting two more parameters, there is nothing new here. Exactly the same logic works for setting “Tw” and “Td”.

7.2 update addType() with the following:

// ======================================

// (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 );

}

**8.** **Test Your Column**

Your code is ready to build and run to test.

You can add lines like the following to your Revit .addin manifest file to test this. (You can either add a new command or replace with one from Lab 1). Make necessary adjustment to match with your environment, of course.

<?xml version="1.0" encoding="utf-16" standalone="no"?>

<RevitAddIns>

<AddIn Type="Command">

<Assembly>C:\Revit SDK 2013\Family Labs\FamilyLabsCS\bin\Debug\FamilyLabsCS.dll</Assembly>

<AddInId>FC5E150A-967B-4cc9-A7B0-3AA29C5DA9D9</AddInId>

<FullClassName>FamilyLabsCS.RvtCmd\_FamilyCreateColumnLShape</FullClassName>

<Text>Family API 2 CS - 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>

Remember to start with Family Editor and use "Metric Column.rft" template.

After running a command, you will see the profile of the column now shows L-shape. Examine your column:

* Do you see two dimensions added in the plan view?
* Are they labeled correctly?
* Go to the Type dialog. Do you see two additional parameters, “Tw” and “Td” under “Dimensions”?
* Do you see three types being created correctly?
* “flex it” (choose different types and apply) . Does your column changes its size accordingly?
* Try also loading to a project. Does your column “behave” well?

In the next lab, we will be adding formulas and materials to on top of the column family we have just created.

**Appendix A. Helper Functions Used in Lab2**

In the Lab2, we have added one more helper function. Copy and paste from the code below to your code as required.

* findFace() - The second version. Given an extrusion solid, find a planar face that has the given normal, and lies on the given reference plane.

// ===============================================================

// helper function: given a solid, find a planar

// face with the given normal (version 2)

// this is a slightly enhaced 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.Normal.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;

}

Autodesk Developer Network