**Revit Family API Labs**

**Lab1 – Create Rectangular Column**

March, 2010 by M. Harada

Last updated, Date : April 30, 2015

VB.NET version

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

* check the family environment
* create a simple solid using extrusion
* set alignments
* add types

**Tasks:** We’ll define a command that creates a column family which has a rectangular profile, and add three types with dimensional variations:

1. Take "Metric Column.rft" as a template. We can assume the user has opened the correct template. But within the command, we check if the user has chosen a right template.
2. Define a rectangular profile and create a simple box solid using extrusion.
3. Add alignments between each face and corresponding reference plane.
4. Define types with dimensional variations.

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





Figure 1. A column family with a rectangular profile we will be creating in Lab1.

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

1. [Define an External Command](#defineExternalCommand)
2. [Check the Validity of the Document Context](#checkValidityOfDocumentContext)
3. [Create a Simple Solid with Extrusion](#createSimpleSolidWithExtrusion)
4. [Add Alignments](#addAlignments)
5. [Add Types](#addTypes)
6. [Test Your Column](#testYourColumn)

[Appendix A. Helper Functions Used in Lab1](#AppendixAHelperFunctions)

1. **Define an External Command** 
   1. Create a new Visual Studio project of your choice (VB.NET or C#). Add references. Define a new external command. Let’s name them as follows:

* Solution name: **FamilyLabs**
* Project name: **FamilyLabsVb**
* File name: **1\_ColumnRectangle.vb (or .cs)**
* Command class name: **RvtCmd\_FamilyCreateColumnRectangle**

(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.)

We will need the following reference at least:

* System
* System.Core (this is for LINQ query)
* Revit API
* RevitAPIUI

In the project [Properties] >> [References] >> [Imported namespaces], in addition to the defaulted setting, add the following:

* System.Linq
* Autodesk.Revit
* Autodesk.Revit.DB
* Autodesk.Revit.UI
* Autodesk.ApplicationServices

* 1. Keep the top level object access for Revit application and document as member variables, e.g., \_rvtApp, and \_rvtDoc respectively. The code should look like below:

<Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)> \_

<Autodesk.Revit.Attributes.Regeneration(Autodesk.Revit.Attributes.RegenerationOption.Manual)> \_

Public Class RvtCmd\_FamilyCreateColumnRectangle

    Implements IExternalCommand

    '' member variables for top level access to the Revit database

    ''

    Dim \_rvtApp As Application

    Dim \_rvtDoc As Document

    ''  command main

    ''

    Public Function Execute( \_

        ByVal commandData As ExternalCommandData, \_

        ByRef message As String, \_

        ByVal elements As ElementSet) \_

        As Result \_

        Implements IExternalCommand.Execute

        ''  objects for the top level access

        ''

        \_rvtApp = commandData.Application.Application

        \_rvtDoc = commandData.Application.ActiveUIDocument.Document

'' ...

''

        ''  finally, return

Return Result.Succeeded

End Function

End Class

**2.** **Check the Validity of Document Context**

Our command with family API works only in the context of Family editor. We will first check the validity of the document context that we are current in. Let’s define a function isRightTemplate()to check this. The function isRightTemplate()takes BuiltInCategory as an argument.

* 1. Add the following function to the class:

Function isRightTemplate(ByVal targetCategory As BuiltInCategory) \_

As Boolean

'' This command works in the context of family editor only.

''

If Not \_rvtDoc.IsFamilyDocument Then

MsgBox("This command works only in the family editor.")

Return False

End If

'' Check the template for an appropriate category here if needed.

''

Dim cat As Category = \_rvtDoc.Settings.Categories.Item(targetCategory)

If \_rvtDoc.OwnerFamily Is Nothing Then

MsgBox("This command only works in the family context.")

Return False

End If

If Not cat.Id.Equals(\_rvtDoc.OwnerFamily.FamilyCategory.Id) Then

MsgBox("Category of this family document does not match the context required by this command.")

Return False

End If

'' if we come here, we should have a right one.

Return True

End Function

This function checks:

* If we are in a family document. You can check \_rvtDoc.IsFamilyDocument for this.
* If the template is for defining a column. The category of current document can be checked by looking at \_rvtDoc.OwnerFamily.FamilyCategory.Id.

If the above are satisfied, the function returns True, otherwise False.

2.2 Call isRightTemplate() function from your main command function Execute(). If it is not a right template, the command will halt:

        \_rvtApp = commandData.Application.Application

        \_rvtDoc = commandData.Application.ActiveUIDocument.Document

'' (0) This command works in the context of family editor only.

'' We also check if the template is for an appropriate category if needed.

'' Here we use a Column(i.e., Metric Column.rft) template.

'' Although there is no specific checking about metric or imperial, our lab only works in metric for now.

''

If Not isRightTemplate(BuiltInCategory.OST\_Columns) Then

MsgBox("Please open Metric Column.rft")

Return IExternalCommand.Result.Failed

End If

...

**3.** **Create a Simple Solid with Extrusion**

Now that we have a valid template, let’s start creating a simple solid. Here we define a rectangular profile, extrude with a given height.

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

    '' ============================================

    ''   (1.1) create a simple rectangular profile

    '' ============================================

    Function createProfileRectangle() As CurveArrArray

        ''

        ''  define a simple rectangular profile

        ''

        ''  3     2

        ''   +---+

        ''   |   | d    h = height

        ''   +---+

        ''  0     1

        ''  4  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.

        ''

        Dim w As Double = mmToFeet(600) ' hard coded for simplicity here. in practice, you may want to find out from the references)

        Dim d As Double = mmToFeet(600)

        ''  define vertices

        ''

        Const nVerts As Integer = 4 '' the number of vertices

        Dim pts() As XYZ = {New XYZ(-w / 2, -d / 2, 0), New XYZ(w / 2, -d / 2, 0), New XYZ(w / 2, d / 2, 0), New XYZ(-w / 2, d / 2, 0), New XYZ(-w / 2, -d / 2, 0)} ' the last one is to make the loop simple

        ''  define a loop. define individual edges and put them in a curveArray

        ''

        Dim pLoop As CurveArray = \_rvtApp.Create.NewCurveArray

        Dim lines(nVerts - 1) As Line

        For i As Integer = 0 To nVerts - 1

            lines(i) = Line.CreateBound(pts(i), pts(i + 1))

            pLoop.Append(lines(i))

        Next

        ''  then, put the loop in the curveArrArray as a profile

        ''

        Dim pProfile As CurveArrArray = \_rvtApp.Create.NewCurveArrArray

        pProfile.Append(pLoop)

        ''  if we come here, we have a profile now.

        Return pProfile

    End Function

There is one helper function we use in this function, i.e.,

mmToFeet()

Revit uses feet. You will need to convert unit whenever you define dimensions through API. The code for **mmToFeet**()is attached at the end of this doc, the section, [Appendix A](#AppendixAHelperFunctions). Please copy it and paste to the end of this class.

We are hard-coding the actual size of rectangle as well as vertices of rectangular shape for simplicity and for the readability of the code for our learning purpose. The size actually comes from the distance between the reference planes that are predefined in the column family template. If you are using a different template, you will need to adjust those values. A profile is defined as a CurveArrArray (or a collection of curve arrays).

This may be a good time to closer look at your template. Open the template and observe the following:

* Dimensions used in the template.
* The names of the reference planes that are predefine for the template.
* Looks for these in Front view as well as Plan view.

What do you see there? You may want to note those as we’ll reference them throughout the labs.

3.2 Using the profile we have just defined, we then create a solid from extrusion. Add the following function to your class code:

'' ============================================

'' (1) create a simple solid by extrusion

'' ============================================

Function createSolid() As Extrusion

''

'' (1) define a simple rectangular profile

''

'' 3 2

'' +---+

'' | | d h = height

'' +---+

'' 0 1

'' 4 w

''

Dim pProfile As CurveArrArray = createProfileRectangle()

''

'' (2) create a sketch plane

''

'' we need to know the template. If you look at the template (Metric Column.rft) and "Front" view,

'' you will see "Reference Plane" at "Lower Ref. Level". We are going to create an extrusion there.

'' findElement() is a helper function that find an element of the given type and name. see below.

''

Dim pRefPlane As ReferencePlane = findElement(GetType(ReferencePlane), "Reference Plane") ' need to know from the template

Dim pSketchPlane As SketchPlane = SketchPlane.Create(\_rvtDoc, pRefPlane.Plane)

'' (3) height of the extrusion

''

'' once again, you will need to know your template. unlike UI, the alightment will not adjust the geometry.

'' You will need to have the exact location in order to set alignment.

'' Here we hard code for simplicity. 4000 is the distance between Lower and Upper Ref. Level.

'' as an exercise, try changing those values and see how it behaves.

''

Dim dHeight As Double = mmToFeet(4000) '' distance between Lower and Upper Ref Level.

'' (4) create an extrusion here. at this point. just an box, nothing else.

''

Dim bIsSolid As Boolean = True ' as oppose to void.

Dim pSolid As Extrusion = \_rvtDoc.FamilyCreate.NewExtrusion(bIsSolid, pProfile, pSketchPlane, dHeight)

Return pSolid

End Function

There is the second helper function we use in this function, i.e.,

findElement()

This helper function finds an element of the given type and the name. You can use this, for example, to find a ReferencePlane, Level or View. The full code is attached at the end of this doc, the section, [Appendix A](#AppendixAHelperFunctions). Please copy it and paste to the end of this class.

At the bottom of the above code, you see a call to a method,

\_rvtDoc.FamilyCreate.NewExtrusion ().

This is the main method we use to define an extrusion. It takes Solid/Void flag, a profile, a sketch plane and a height as argument. We use one of the predefined reference planes to define a sketch plane (i.e., “Reference Plane” in the above code or in our template).

Once again, we have hard-coded the height information here. This comes from the template and is the distance between the lower and upper reference level. (Check this in the Front view of the template, for example.)

3.3 The call above function from your main command function. createSolid() returns an object of type Extrusion:

Public Function Execute(ByVal commandData As ExternalCommandData,

...

'' (0) This command works in the context of family editor only.

...

If Not isRightTemplate(BuiltInCategory.OST\_Columns) Then

MsgBox("Please open Metric Column.rft")

Return IExternalCommand.Result.Failed

End If

'' (1) create a simple extrusion. just a simple box for now.

Dim pSolid As Extrusion = createSolid()

'' We need to regenerate so that we can build on this new geometry

\_rvtDoc.Regenerate()

...

3.4. Your code is ready to build and run at this point to test if your solid is created correctly. If you would like, go ahead and see how your solid look like at this point.

You can create an .addin manifest file with the information like the following, and add it to the location that Revit would recognize. (I’m assuming that you are familiar with this by now.) Make necessary adjustment to match with your environment, of course. One thing you may notice is that we have set the visibility mode as “NotVisibleInProject”. This is because our command is specifically designed to work in Family Editor mode and not in a Revit project.

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

<RevitAddIns>

<AddIn Type="Command">

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

<AddInId>99781C81-1B5D-4a4d-ADEC-69B2F8B87511</AddInId>

<FullClassName>FamilyLabsVB.RvtCmd\_FamilyCreateColumnRectangle</FullClassName>

<Text>Family API 1 - Create Rectangular Column</Text>

<Description>Family API lab 1 to create rectangular 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 may find the column you just create does not quite behave as you intended. This is as expected at this stage. A family is a parametric object. What we have just defined is an initial state of the model. We’ll be adding more to it to make our column behave in parametric manner.

**4.** **Add Alignments**

The next step is to add alignment constraint between each face of the solid and corresponding reference planes. This is needed to make our column behaves in parametric manner; when we want our column to adjust its sizes when the user changes its dimensions.

4.1 Add the following function to the class. This function adds six alignments: one for each of six faces of box-shape:

'' ============================================

'' (2) add alignments

'' ============================================

Sub addAlignments(ByVal pBox As Extrusion)

''

'' (1) we want to constrain the upper face of the column to the "Upper Ref Level"

''

'' which direction are we looking at?

''

Dim pView As View = findElement(GetType(View), "Front")

'' find the upper ref level

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

''

Dim upperLevel As Level = findElement(GetType(Level), "Upper Ref Level")

Dim ref1 As Reference = upperLevel.PlaneReference

'' find the face of the box

'' findFace() is a helper function. see below.

''

Dim upperFace As PlanarFace = findFace(pBox, New XYZ(0, 0, 1)) ' find a face whose normal is z-up.

Dim ref2 As Reference = 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.

''

Dim lowerLevel As Level = findElement(GetType(Level), "Lower Ref. Level")

Dim ref3 As Reference = lowerLevel.PlaneReference

'' find the face of the box

'' findFace() is a helper function. see below.

''

Dim lowerFace As PlanarFace = findFace(pBox, New XYZ(0, 0, -1)) ' find a face whose normal is z-down.

Dim ref4 As Reference = 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.

Dim pViewPlan As View = findElement(GetType(ViewPlan), "Lower Ref. Level")

'' find reference planes

Dim refRight As ReferencePlane = findElement(GetType(ReferencePlane), "Right")

Dim refLeft As ReferencePlane = findElement(GetType(ReferencePlane), "Left")

Dim refFront As ReferencePlane = findElement(GetType(ReferencePlane), "Front")

Dim refBack As ReferencePlane = findElement(GetType(ReferencePlane), "Back")

'' find the face of the box

Dim faceRight As PlanarFace = findFace(pBox, New XYZ(1, 0, 0))

Dim faceLeft As PlanarFace = findFace(pBox, New XYZ(-1, 0, 0))

Dim faceFront As PlanarFace = findFace(pBox, New XYZ(0, -1, 0))

Dim faceBack As PlanarFace = findFace(pBox, New XYZ(0, 1, 0))

'' 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)

End Sub

There is the third helper function we use in this function, i.e.,

findFace()

This helper function finds a planar face with the given normal from an extrusion solid. The full code is attached at the end of this doc, the section, [Appendix A](#AppendixAHelperFunctions). Please copy it and paste to the end of this class. (Kind note: You will also need to copy isEqual() as it is used in findface().)

Let’s focus on the first portion the code where we add an alignment between the top face of the solid and the reference plane “Upper Ref Level”. Once you understand one, the rest should be more or less the same.

m\_rvtDoc.FamilyCreate.NewAlignment(pView, ref1, ref2)

This is the main method to create a new alignment. It takes a view, and two references as arguments. As we do in UI, to align the top face to the upper reference plane, we look at the model from a side. We use Front view here. findElement() and findFace() are helper functions.

One thing to note is that unlike UI, API method NewAlignment will not automatically calculate and adjust the geometry of the model. Here is the excerpt from the RevitAPI.chm file:

*“These references must be already geometrically aligned (this function will not force them to   
 become aligned).”*

You will need to make sure that they are at the same location before you call NewAlignment.

4.2 Call addAlignments(pSolid) from your main command execute:

Public Function Execute(ByVal commandData As ExternalCommandData, ByRef

...

'' (1) create a simple extrusion. just a simple box for now.

Dim pSolid As Extrusion = createSolid()

'' (2) add alignment

addAlignments(pSolid)

...

4.3 Your code should build and run at this point. Your column should respond when you change a reference; for example, when you move the level.

**5.** **Add Types**

Let’s add a couple of types now, for example, ones with dimensions corresponding to “Width” and “Depth” to:

* 600 x 900
* 1000 x 300
* 600 x 600

5.1 Add the following functions to the class:

'' ============================================

'' (3) add types

'' ============================================

Sub addTypes()

'' addType(name, Width, Depth)

''

addType("600x900", 600.0, 900.0)

addType("1000x300", 1000.0, 300.0)

addType("600x600", 600.0, 600.0)

End Sub

'' add one type

''

Sub addType(ByVal name As String, ByVal w As Double, ByVal d As Double)

'' get the family manager from the current doc

Dim pFamilyMgr As FamilyManager = \_rvtDoc.FamilyManager

'' add new types with the given name

''

Dim type1 As FamilyType = pFamilyMgr.NewType(name)

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

''

'' first 'Width'

''

Dim paramW As FamilyParameter = pFamilyMgr.Parameter("Width")

Dim valW As Double = mmToFeet(w)

If paramW IsNot Nothing Then

pFamilyMgr.Set(paramW, valW)

End If

'' same idea for 'Depth'

''

Dim paramD As FamilyParameter = pFamilyMgr.Parameter("Depth")

Dim valD As Double = mmToFeet(d)

If paramD IsNot Nothing Then

pFamilyMgr.Set(paramD, valD)

End If

End Sub

The key class that you need to get hold of is a Family Manager object of the given document. You can access it through:

m\_rvtDoc.FamilyManager

Once you have a family manager, you can create a new type using NewType method:

pFamilyMgr.NewType(name)

You can access to a parameter of your interest, using (e.g.):

pFamilyMgr.Parameter("Width")

Then sets its value, using:

pFamilyMgr.Set(paramW, valW)

The above code defines three types.

5.2 Call addTypes() from your main command function:

Public Function Execute(ByVal commandData As ExternalCommandData, ByRef

...

'' (2) add alignment

addAlignments(pSolid)

'' (3) add types

addTypes()

'' finally, return

Return Result.Succeeded

End Function

5.3 Your code should be ready to build and run.

**6.** **Test Your Column**

Your code is ready to build and run at this point to test if your column is created correctly.

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

After running a command, go to the type dialog, check to see if three types are created. Apply each of them, and see if your column changes its size accordingly.

In the next lab, we will modify the profile of the column and learn how to add reference planes, parameters, and dimensions.

**Appendix A. Helper Functions Used in Lab1**

In the Lab1, we use the following helper functions. Copy and paste from the code below to your code as required.

* findFace() - given an extrusion solid, find a planar face with the given normal.
* findElement() - find an element of the given type and the name. You can use this, for example to find Reference or Level with the given name.
* isEqual() - determine if two vectors are about the same. (this is used in findFace())
* mmToFeet() - convert unit from millimeter to feet.

''============================================

''

'' Helper functions

''

''============================================

#Region "Helper Functions"

'' ============================================

'' helper function: find a planar face with the given normal

'' ============================================

Function findFace(ByVal pBox As Extrusion, ByVal normal As XYZ) As PlanarFace

'' get the geometry object of the given element

''

Dim op As New Options

op.ComputeReferences = True

Dim geomObjs As IEnumerable(Of GeometryObject) = pBox.Geometry(op).AsEnumerable()

'' loop through the array and find a face with the given normal

''

For Each geomObj As GeometryObject In geomObjs

If TypeOf geomObj Is Solid Then '' solid is what we are interested in.

Dim pSolid As Solid = geomObj

Dim faces As FaceArray = pSolid.Faces

For Each pFace As Face In faces

Dim pPlanarFace As PlanarFace = pFace

If Not (pPlanarFace Is Nothing) Then

If pPlanarFace.Normal.IsAlmostEqualTo(normal) Then '' we found the face

Return (pPlanarFace)

End If

End If

Next

ElseIf TypeOf geomObj Is GeometryInstance Then

'' will come back later as needed.

ElseIf TypeOf geomObj Is Curve Then

'' will come nack later as needed.

ElseIf TypeOf geomObj Is Mesh Then

'' will come back later as needed.

Else

'' what else do we have?

End If

Next

'' if we come here, we did not find any.

Return Nothing

End Function

'' ============================================

'' helper function: find an element of the given type and the name.

'' You can use this, for example, to find Reference or Level with the given name.

'' ============================================

Function findElement(ByVal targetType As Type, ByVal targetName As String) As Autodesk.Revit.DB.Element

'' get the elements of the given type

''

Dim collector = New FilteredElementCollector(\_rvtDoc)

collector.WherePasses(New ElementClassFilter(targetType))

'' parse the collection for the given name

'' using LINQ query here.

''

Dim targetElems = From element In collector Where element.Name.Equals(targetName) Select element

Dim elems As List(Of Element) = targetElems.ToList()

If elems.Count > 0 Then '' we should have only one with the given name.

Return elems(0)

End If

'' cannot find it.

Return Nothing

End Function

'' ============================================

'' convert millimeter to feet

'' ============================================

Function mmToFeet(ByVal mmVal As Double) As Double

Return mmVal / 304.8 '' \* 0.00328;

End Function

#End Region

Autodesk Developer Network