**Revit API Intro Labs**

**Lab 5 – Model Creation**

Created by M. Harada, July 2010  
Updated by DevTech AEC WG  
Last modified:

**<VB.NET>**VB.NET Version**</VB.NET>**

**Objective:** In this lab, we will learn how to create a Revit model. We’ll learn how to:

* Create instances of architectural elements, such as walls, doors, windows and roofs

**Tasks:** We’ll write a command that create a simple “house” composed of four walls with a rectangular footprint, one door, three windows and a roof.

1. Create four walls with a rectangular footprint.
2. Add a door to the first wall
3. Add windows to the rest of the walls
4. Add a sloped roof over the walls

Figure 1 shows the sample images of output after running the command that you will be defining in this lab:

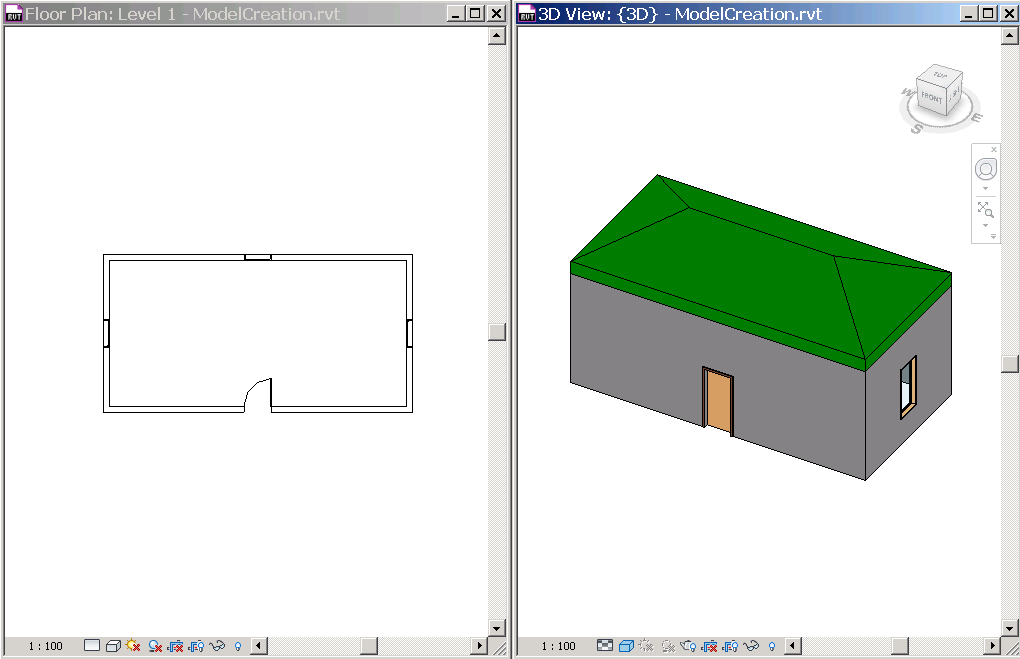


Figure 1. we’ll create a simple house composed of four walls, a door, windows and a roof.

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

1. Define a New External Command
2. Create Walls
3. Add a Door
4. Add Windows
5. Add a Sloped Roof
6. Summary
7. **Define A New External Command**

We’ll add another external command to the current project.

* 1. Add a new file and define another external command to your project. Let’s name them as follows:
* File name: **5\_ModelCreation.vb (or .cs)**
* Command class name: **ModelCreation**

**Additional Namespace:**

* Autodesk.Revit.DB.Structure (this is for StructuralType for creating a family instance.)

**Additional Considerations:**

We’ll be using the method the following methods from the ElementFiltering class:

* ElementFiltering.FindFamilyType()
* ElementFiltering.FindElement()
  1. Like we did in the previous labs, define member variables, e.g., m\_rvtApp and m\_rvtDoc, to keep DB level application and document respectively. e.g., :

**<VB.NET>**

'' Model Creation - learn how to create elements

<Transaction(TransactionMode.Automatic)> \_

Public Class ModelCreation

Implements IExternalCommand

**'' member variables**

**Dim m\_rvtApp As Application**

**Dim m\_rvtDoc As Document**

Public Function Execute(ByVal commandData As ExternalCommandData, \_

ByRef message As String, \_

ByVal elements As ElementSet) \_

As Result \_

Implements IExternalCommand.Execute

'' Get the access to the top most objects.

**Dim rvtUIApp As UIApplication = commandData.Application**

**Dim rvtUIDoc As UIDocument = rvtUIApp.ActiveUIDocument**

**m\_rvtApp = rvtUIApp.Application**

**m\_rvtDoc = rvtUIDoc.Document**

'' ...

Return Result.Succeeded

End Function

End Class  
**</VB.NET>**

1. **Create Walls**

In the previous lab, we have used application object to create a new geometry element:

* m\_rvtApp.Create.NewLineBound(point1, point2)

To create a new instance of a model element, we can do the similar. The difference is that you will use a Document object instead of an Application object to call Create. There are five overloaded methods to create a wall. (cf. Developer Guide, pp 121.) For example, the following creates a new rectangular profile wall with the default wall style at the give level:

* m\_rvtDoc.Create.NewWall(baseCurve, level, isStructural)

A base curve can be defined by NewLineBound using two end points like we have done before. You can use ElementFiltering.FindElement() to find the level where you want to place a wall, for example, “Level 1”.

If you want to constrain the top of the wall to the upper level, or to set “Top Constrains” to “Level 2”, you can additionally set the parameter after you have create a new wall; the corresponding of BuiltInParameter of “Top Constrains” is WALL\_HEIGHT\_TYPE:

* aWall.Parameter(BuiltInParameter.WALL\_HEIGHT\_TYPE).Set(level2.Id)

Below is a sample code to create four walls at arbitrary location. This code works well if you use DefaultMetric.rte template, where you have “Level 1” and “Level 2” predefined.

**<VB.NET>** '' create four walls

Function CreateWalls() As List(Of Wall)

'' hard coding the size of the house for simplicity

Dim width As Double = mmToFeet(10000.0)

Dim depth As Double = mmToFeet(5000.0)

'' get the levels we want to work on.

'' Note: hard coding for simplicity. Modify here you use a different  
 '' template.

Dim level1 As Level = \_   
 ElementFiltering.FindElement(m\_rvtDoc, GetType(Level), "Level 1")

If level1 Is Nothing Then

TaskDialog.Show("Revit Intro Lab", "Cannot find (Level 1). Maybe you use a different template? Try with DefaultMetric.rte.")

Return Nothing

End If

Dim level2 As Level = \_   
 ElementFiltering.FindElement(m\_rvtDoc, GetType(Level), "Level 2")

If level2 Is Nothing Then

TaskDialog.Show("Revit Intro Lab", "Cannot find (Level 2). Maybe you use a different template? Try with DefaultMetric.rte.")

Return Nothing

End If

'' set four corner of walls.

'' 5th point is for combenience to loop through.

Dim dx As Double = width / 2.0

Dim dy As Double = depth / 2.0

Dim pts As New List(Of 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.

Dim isStructural As Boolean = False

'' save walls we create.

Dim walls As New List(Of Wall)(4)

'' loop through list of points and define four walls.

For i As Integer = 0 To 3

'' define a base curve from two points.

Dim baseCurve As Line = \_

m\_rvtApp.Create.NewLineBound(pts(i), pts(i + 1))

'' create a wall using the one of overloaded methods.

Dim aWall As Wall = \_

m\_rvtDoc.Create.NewWall(baseCurve, level1, isStructural)

'' set the Top Constraint to Level 2

aWall.Parameter(BuiltInParameter.WALL\_HEIGHT\_TYPE).Set(level2.Id)

'' save the wall.

walls.Add(aWall)

Next

'' This is important. we need these lines to have shrinkwrap working.

m\_rvtDoc.Regenerate()

m\_rvtDoc.AutoJoinElements()

Return walls

End Function  
**</VB.NET>**

**Regeneration of Graphics**

One thing that we want to get your attention is these two lines at the end of the function:

m\_rvtDoc.Regenerate()

m\_rvtDoc.AutoJoinElements()

By adding those two lines, Revit will first update graphics information of individual walls, then performs auto-join or shrink-wraps the corners of walls.

**Exercise:**

* Implement a function that creates four walls forming a rectangular footprint, and returns the list walls created. You can use arbitrary location to place wall for this exercise.

1. **Add a Door**

To create a wall, we have used a method NewWall(), which is designated function to create a wall. When creating an instance of component family, such as a door and a window, you will need to use the method NewFamilyInstance(). There are 9 overloaded method for NewfamilyInstance(). Which one to use depends on what kind of family instance you would like to create (e.g., point based, curve based), and the condition that you would like to create (e.g., constrained to reference, free standing). The section 12.3.5 (pp 140) of Developer Guide lists applicable form of methods by the element categories with the description of when to use which form. Please refer to it for more detail.

Here to add a door, as an example, we will use the following form of NewFamilyInstance():

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

The below is an example of a function that adds a door to the given wall. In this example, we place a door at the center of the given wall as a host. The door is constrained to the bottom of the wall:

**<VB.NET>**

'' add a door to the center of the given wall.

'' cf. Developer Guide p140. NewFamilyInstance() for Doors and Window.

Sub AddDoor(ByVal hostWall As Wall)

'' hard coding the door type we will use.

'' e.g., "M\_Single-Flush: 0915 x 2134mm

Const doorFamilyName As String = "M\_Single-Flush"

Const doorTypeName As String = "0915 x 2134mm"

Const doorFamilyAndTypeName As String = \_

doorFamilyName + ": " + doorTypeName

'' get the door type to use.

Dim doorType As FamilySymbol = \_

ElementFiltering.FindFamilyType(m\_rvtDoc, GetType(FamilySymbol), \_

doorFamilyName, doorTypeName, BuiltInCategory.OST\_Doors)

If doorType Is Nothing Then

TaskDialog.Show("Revit Intro Lab", "Cannot find (" + \_

doorFamilyAndTypeName + "). Maybe you use a different template? Try with DefaultMetric.rte.")

End If

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

Dim locCurve As LocationCurve = hostWall.Location

Dim pt1 As XYZ = locCurve.Curve.EndPoint(0)

Dim pt2 As XYZ = locCurve.Curve.EndPoint(1)

'' calculate the mid point.

Dim pt As XYZ = (pt1 + pt2) / 2.0

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

Dim idLevel1 As ElementId = \_

hostWall.Parameter(BuiltInParameter.WALL\_BASE\_CONSTRAINT).AsElementId

Dim level1 As Level = m\_rvtDoc.Element(idLevel1)

'' finally, create a door.

Dim aDoor As FamilyInstance =

m\_rvtDoc.Create.NewFamilyInstance( \_

pt, doorType, hostWall, level1, StructuralType.NonStructural)

End Sub

**</VB.NET>**

1. **Add Windows**

Creating an instance of a window is basically the same as a door. You can use the same NewFamilyInstance() method:

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

Only one additional consideration is to add the sill height to it. Otherwise, the window stays at the bottom of the wall. You can set it by setting the corresponding parameter:

* aWindow.Parameter(BuiltInParameter.INSTANCE\_SILL\_HEIGHT\_PARAM).Set(sillHeight)

The below is an example of a function that adds a window to the given wall. In this example, we place a window at the center of the given wall as a host. The door is constrained to the bottom of the wall at the sill height of 915mm:

**<VB.NET>**

'' add a window to the center of the wall given.

Sub AddWindow(ByVal hostWall As Wall)

'' hard coding the window type we will use.

'' e.g., "M\_Fixed: 0915 x 1830mm

Const windowFamilyName As String = "M\_Fixed"

Const windowTypeName As String = "0915 x 1830mm"

Const windowFamilyAndTypeName As String = \_

windowFamilyName + ": " + windowTypeName

Dim sillHeight As Double = mmToFeet(915)

'' get the door type to use.

Dim windowType As FamilySymbol = \_

ElementFiltering.FindFamilyType(m\_rvtDoc, GetType(FamilySymbol), \_

windowFamilyName, windowTypeName, BuiltInCategory.OST\_Windows)

If windowType Is Nothing Then

TaskDialog.Show("Revit Intro Lab", "Cannot find (" + \_

windowFamilyAndTypeName + "). Please use DefaultMetric.rte?")

End If

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

Dim locCurve As LocationCurve = hostWall.Location

Dim pt1 As XYZ = locCurve.Curve.EndPoint(0)

Dim pt2 As XYZ = locCurve.Curve.EndPoint(1)

'' calculate the mid point.

Dim pt As XYZ = (pt1 + pt2) / 2.0

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

Dim idLevel1 As ElementId = \_   
 hostWall.Parameter(BuiltInParameter.WALL\_BASE\_CONSTRAINT).AsElementId

Dim level1 As Level = m\_rvtDoc.Element(idLevel1)

'' create a window.

Dim aWindow As FamilyInstance = m\_rvtDoc.Create.NewFamilyInstance( \_

pt, windowType, hostWall, level1, StructuralType.NonStructural)

'' set the sill height

aWindow.Parameter(BuiltInParameter.INSTANCE\_SILL\_HEIGHT\_PARAM). \_

Set(sillHeight)

End Sub

**</VB.NET>**

**Exercise:**

* Implement a function that takes a wall, and add a door or a window to the given wall. You can hard code the family type of a door or window for this exercise.

1. **Add a Roof**

By now, you have a clear idea about distinction between creating an instance of a system family and a component family. As a roof is a system family, you will need to use designated method. Roof has two kinds of methods NewFootPrintRoof() and NewExtrusionRoof(). Section 11.3 (pp 125) of Developer Guide has the detailed description about the usage. “NewRoof” SDK sample demonstrates the usage of both footprint and extrusion roofs.

Here, let’s take a look at the footprint version of the roof.

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

The last argument, curveMapping, is ModelCurveArray data type. You will pass an empty model curve array, and the function fills in with the corresponding curves of the roof created. We use this curve to further set the properties of the curve, such as slope angle.

To make a roof sloped, you will need to set the angle at each edge of the footprint model curve created for the roof. Once you have created a roof, you will loop through the curve mapping that has been returned from the NewFootPrintRoof() method, and set two values DefineSlope() and SlopeAngle() for an appropriate values:

* aRoof.DefineSlope(modeCurve) = True
* aRoof.SlopeAngle(modelCurve) = <some angular value>

Here is a skeletal form of creating a footprint roof and setting a sloop to it:

**<VB.NET>**

'' create a roof.

Dim aRoof As FootPrintRoof = m\_rvtDoc.Create.NewFootPrintRoof( \_

footPrint, level2, roofType, mapping)

'' set the slope

For Each modelCurve As ModelCurve In mapping

aRoof.DefinesSlope(modelCurve) = True

aRoof.SlopeAngle(modelCurve) = 0.5

Next

**</VB.NET>**

Below is a sample code to create a footprint roof over the four walls that have been passed in. This code works well if you use DefaultMetric.rte template, where you have “Basic Roof: Generic – 400mm” defined. (Note: we are defining the 4 corner of the roof with thickness of the walls considered. Otherwise, the roof will be placed based on the center lines of the walls.)

**<VB.NET>**

'' add a roof over the rectangular profile of the walls we created.

Sub addRoof(ByVal walls As List(Of Wall))

'' hard coding the roof type we will use.

'' e.g., "Basic Roof: Generic - 400mm"

Const roofFamilyName As String = "Basic Roof"

Const roofTypeName As String = "Generic - 400mm"

Const roofFamilyAndTypeName As String = \_

roofFamilyName + ": " + roofTypeName

'' find the roof type

Dim roofType As RoofType = \_

ElementFiltering.FindFamilyType( \_

m\_rvtDoc, GetType(RoofType), roofFamilyName, roofTypeName)

If roofType Is Nothing Then

TaskDialog.Show("Revit Intro Lab", "Cannot find (" + \_

roofFamilyAndTypeName + "). Please use DefaultMetric.rte.")

End If

'' 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.

Dim wallThickness As Double = walls(0).Width

Dim dt As Double = wallThickness / 2.0

Dim dts As New List(Of 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

Dim footPrint As New CurveArray()

For i As Integer = 0 To 3

Dim locCurve As LocationCurve = walls(i).Location

Dim pt1 As XYZ = locCurve.Curve.EndPoint(0) + dts(i)

Dim pt2 As XYZ = locCurve.Curve.EndPoint(1) + dts(i + 1)

Dim line As Line = m\_rvtApp.Create.NewLineBound(pt1, pt2)

footPrint.Append(line)

Next

'' get the level2 from the wall

Dim idLevel2 As ElementId = walls(0).Parameter( \_

BuiltInParameter.WALL\_HEIGHT\_TYPE).AsElementId

Dim level2 As Level = m\_rvtDoc.Element(idLevel2)

'' footprint to morel curve mapping

Dim mapping As New ModelCurveArray

'' create a roof.

Dim aRoof As FootPrintRoof = \_

m\_rvtDoc.Create.NewFootPrintRoof(footPrint, level2, roofType, \_

mapping)

'' setting the slope

For Each modelCurve As ModelCurve In mapping

aRoof.DefinesSlope(modelCurve) = True

aRoof.SlopeAngle(modelCurve) = 0.5

Next

End Sub

**</VB.NET>**

**Exercise:**

* Implement the function takes a list of wall and place a roof over it. For this exercise, you can assume that you will get four walls which form a rectangular footprint.
* Call functions that you have defined to create walls, windows, doors and a roof all together and make a command that create simple “house”.

Note: as of 2012 releases, you can create the most of elements through Revit API. However, you may still encounter some objects types that are not exposed yet to create by API. Stairs and railings are example of such object types.

1. **Summary**

In this lab, we learned how to create a Revit model. We have learned how to:

* Create instances of architectural elements, such as walls, doors, windows and roofs.

Autodesk Developer Network