Lab 5 – Model Creation

Created by M. Harada, July 2010
Updated by DevTech AEC WG
Last modified: 10/28/2023

<C#>C# Version**</C#>**

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:

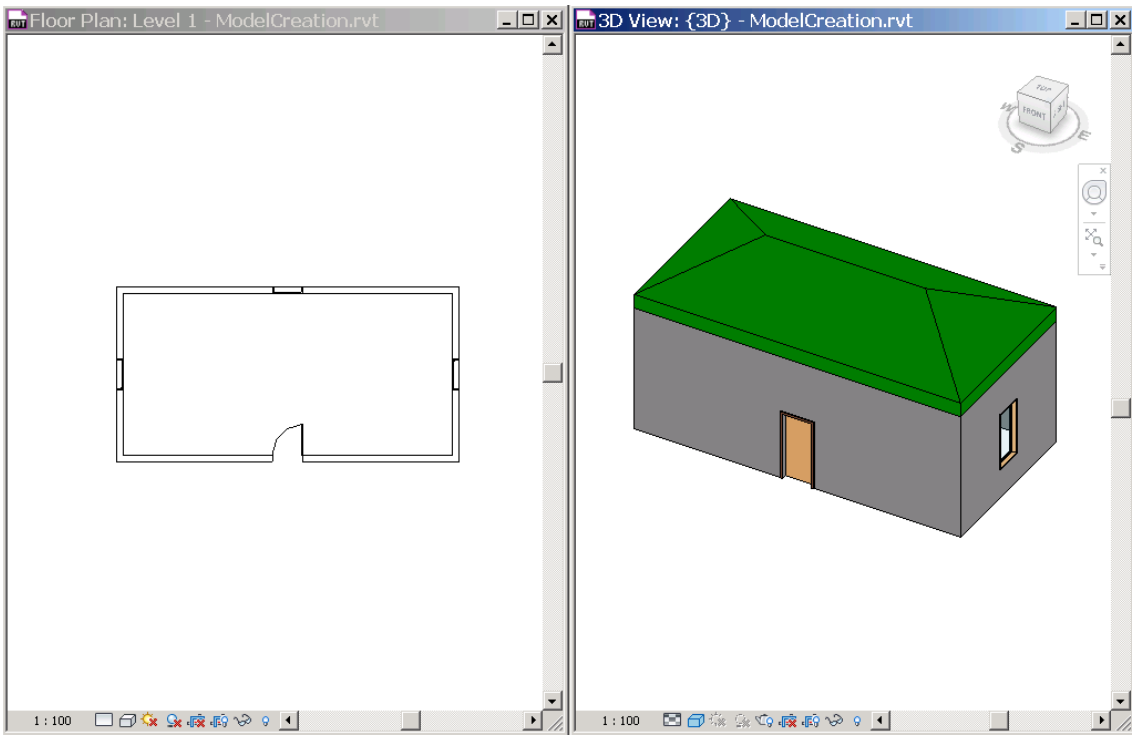

Figure 1. we'll create a simple house composed of four walls, a door, windows and a roof.

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

1. Define A New External Command

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

1.1 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 following methods from the ElementFiltering class:

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

1.2 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., :

<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. Create Walls

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

- Line.CreateBound(point1, point2)

To create a new instance of a wall, the static “Create” method in the Wall class can be used and the document passed in as the first parameter. There are five overloaded methods to create a wall. (Refer Wiki page). For example, the following creates a new rectangular profile wall with the default wall style at the given level:

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

A base curve can be defined by Line.CreateBound 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.

<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), "Level 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), "Level 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 combenience 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#>
```

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.

3. 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 12 overloaded methods 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 Revit API Developer Guide on Autodesk Help 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:

<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 = "Single-Flush"; // "M_Single-Flush"
      const string doorTypeName = "30\" x 80\""; // "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. 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:

```
<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 = "Fixed"; // "M_Fixed"
```

```
      const string windowTypeName = "16\" x 24\""; // "0915 x 1830mm"
      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#>
```

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.

5. 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. Revit API Developer Guide on Autodesk Help 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:

```
<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#>
```

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.)

```
<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 = "Basic Roof";
      const string roofTypeName = "Generic - 9\""; // "Generic - 400mm"
      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;

      // 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#>
```

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”.

6. 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.