**Revit Family API Labs**

**Lab3 – Add Formulas and Materials**

March 2010 by M. Harada

Last updated, Date : May 3, 2021

C# version

**Objective:** In this lab, we will learn how to:

* add a formula
* add materials

**Tasks:** We’ll extend the command defined in the previous labs and add simple formulas and materials to the L-shape column family.

1. Take the command class which we have defined in the Lab2. This will be the starting point of this lab. We’ll continue using the Family Editor and "Metric Column.rft" template.
2. Make two parameters defined as formulas:
   * Tw = Width / 4.0
   * Td = Depth / 4.0
3. Assign a material. We’ll do this by adding instance parameter of material group to the family, and associate it with material parameter of the solid.

Figure 1 shows the image of the materials applied to a column (left) and Family Type dialog showing the simple formulas applied and a parameter under Materials and Finishes added (right).



Figure 1. The materials applied to a column (left) and Family Type dialog showing   
the formulas and a parameter under Materials and Finishes (right).

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

1. [Define Another External Command](#defineAnotherExternalCommand)
2. [Add Formulas](#addFormulas)
3. [Add Materials](#assignMaterials)
4. [Test Your Column](#testYourColumn)
5. **Define Another External Command**

We’ll be extending the command we have defined in Lab2. 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 Lab2 and define a new one to work on in Lab3. Let’s name them as:
* File name: **3\_ColumnFormulaMaterial.cs (or .vb)**
* Command class name: **RvtCmd\_FamilyCreateColumnFormulaMaterial**

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

class RvtCmd\_FamilyCreateColumnFormulaMaterial : IExternalCommand

{

...

}

**2.** **Add Formulas**

We are going to add two very simple formulas to parameters Tw and Td. We’ll set these parameters so that they always stay one fourth of Width and Depth respectively:

* Tw = Width / 4.0
* Td = Depth / 4.0

2.1 Add the following function to the class:

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

// (4.1) add formula

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

public void addFormulas()

{

// we will add the following fomulas

// Tw = Width / 4.0

// Td = Depth / 4.0

//

// first get the parameter

FamilyManager pFamilyMgr = \_rvtDoc.FamilyManager;

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

FamilyParameter paramTd = pFamilyMgr.get\_Parameter( "Td" );

// set the formula

pFamilyMgr.SetFormula( paramTw, "Width / 4.0" );

pFamilyMgr.SetFormula( paramTd, "Depth / 4.0" );

}

Here the key method is this:

pFamilyMgr.SetFormula(paramTw, "Width / 4.0")

The actual definition of formula is passed through as a string, and it is identical to the UI. So it is straight forward.

2.2 Call addFormulas() function from your main command function Execute():

...

// (3.3) add types

addTypes();

// (4.1) add formula

addFormulas();

...

2.3 Your code should build and run. You can test the formula at this point.

* Check the Type dialog. Do you see the formulas added for Tw and Td?
* While you “flex” types, does Tw and Td changes while staying one fourth of the Width and Depth?

**3.** **Add Materials**

Let’s move on and add the materials now. In this section, we will assign a material to the solid. We assume the targeted material exist in the family template for now.

*Note: Currently, we cannot access or modify properties in Render Appearance which is critical for the appearance of the solid [SPR#155053]. In future, we probably want to extend this functionality to create a new material. For now, please embed in the template through UI.*

For this exercise, we will assign “Glass” material, which we know is already in the template. (although it may sound like an unrealistic material for a column.)

3.1 Add the follow function to the class:

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

// (4.2) add materials

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

public void addMaterials( Extrusion pSolid )

{

// We assume Material type "Glass" exists. Template "Metric Column.rft" include "Glass",

// which in fact is the only interesting one to see the effect.

// In practice, you will want to include in your template.

//

// To Do: For the exersize, create it with more appropriate ones in UI, then use the name here.

//

// (1) get the materials id that we are intersted in (e.g., "Glass")

//

Material pMat = findElement( typeof( Material ), "Glass" ) as Material;

if( pMat != null )

{

ElementId idMat = pMat.Id;

// (2a) this add a material to the solid base. but then, we cannot change it for each column.

//

//pSolid.Parameter("Material").Set(idMat)

// (2b) add a parameter for material finish

//

// this time we use instance parameter so that we can change it at instance level.

//

FamilyManager pFamilyMgr = \_rvtDoc.FamilyManager;

ForgeTypeId builtinParamGroup = new ForgeTypeId(BuiltInParameterGroup.PG\_MATERIALS.ToString());

ForgeTypeId parametertype = new ForgeTypeId(SpecTypeId.Reference.Material.ToString());

FamilyParameter famParamFinish = pFamilyMgr.AddParameter( "ColumnFinish",builtinParamGroup,parametertype, true );

// (2b.1) associate material parameter to the family parameter we just added

//

Parameter paramMat = pSolid.get\_Parameter( "Material" );

pFamilyMgr.AssociateElementParameterToFamilyParameter( paramMat, famParamFinish );

// (2b.2) for our convenience, let's add another type with Glass finish

//

addType( "Glass", 600.0, 600.0 );

pFamilyMgr.Set( famParamFinish, ref idMat );

}

}

This function does the following:

1. Get the element id of the material “Glass”
2. Add an instance parameter of the parameter group Material. (e.g., “ColumnFinish”)
3. Associate material parameter of the solid to the parameter “ColumnFinish”.
4. Add a new type called “Glass” which set the material to “Glass”

You can use the helper function findElement() to find the targeted material.

To add an instance parameter, you will set the forth parameter as True. This time we are parameter of type Materials. And we add it under “Materials” parameter group:

pFamilyMgr.AddParameter("ColumnFinish", BuiltInParameterGroup.PG\_MATERIALS, ParameterType.Material, true)

Now, the key method is this:

pFamilyMgr.AssociateElementParameterToFamilyParameter(paramMat, famParamFinish)

We associate the materials parameter to the family parameter that we have just defined. By doing this, we will have an ability to change materials as an instance parameter when you load in the project.

Finally, we add one more type with materials “Glass” finish.

3.2 Call addMaterials() function from your main command function Execute():

...

// (4.1) add formula

addFormulas();

// (4.2) add materials

addMaterials(pSolid);

...

**4.** **Test Your Column**

Your code is ready to build and run for testing.

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 2). 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 202x\Family Labs\FamilyLabsCS\bin\Debug\FamilyLabsCS.dll</Assembly>

<AddInId>74FEF9E3-81E2-44c8-B137-0C9EE48F68A4</AddInId>

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

<Text>Family API 3 CS - Define Formula and Material</Text>

<Description>Family API lab 3 to create L-shaped column with formula and material</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, examine the following:

* (if you haven’t done so in section 2 above) Check the Type dialog. Do you see the formulas added for Tw and Td?
* (if you haven’t done so in section 2 above) While you “flex” types, does Tw and Td changes while staying one fourth of the Width and Depth?
* Check the Type dialog. Do you see a new parameter “ColumnFinish” defined under “Material and Finishes” group?
* Do you see a new type “Glass” created?
* Goes to 3D view and set as Model Graphics to show Shading. Apply “Glass” type. Do you see the change in the appearance?
* Try also loading to a project. Can you change the material at instance level? Does your column “behave” well overall?

In the next lab, we will be adding Visibility controls to the column family.

Autodesk Developer Network