

Revit Family Content You Decide

Paul F. Aubin—is the author of many Revit book titles including the Aubin Academy Series and his "deep dive" into the Revit family editor: Renaissance Revit. He also authors dozens of video titles at lynda.com (now LinkedIn Learning) covering Revit topics from beginner to advanced. Paul is an independent architectural consultant providing Revit content creation, implementation, training and support services. His career of nearly 30 years, includes experience in design, production, CAD management, coaching and training. Paul is an active member of the Autodesk user community and has been a top-rated speaker at conferences such as Autodesk University, BILT and Midwest University for many years; and he recently gave the keynote address to the Campus FM Technology Association annual conference. Paul is an associate member of the AIA, an Autodesk Expert Elite and an Autodesk Certified Professional. He lives in Chicago with his wife and their three children currently attending universities around the country.

Visit: www.paulaubin.com

Follow: @paulfaubin

Class Description

If you use Revit in your work every day, you have no doubt discovered how important it is to have good family content at your disposal. When you do, things move quite smoothly and when you don't, they can be quite the opposite. Inevitably, regardless of how good your library is, we all need to visit the family editor from time to time. In this session, I thought it would be fun to let you decide. Come to the session with ideas, we'll take a vote at the start and if your idea is chosen, I'll build your family content suggestion live in real-time as I discuss how and why I am performing each step and decision. If you have ever wanted to be a fly on the wall of someone else's office as they work through a content creation problem, here is your chance. We'll cover approach, design and strategy and many modeling, and parametric family creation techniques. The final file will be made available to all attendees following the conference.

Learning Objectives

- Learn how to approach the design of a custom piece of family content
- Understand the basics required to get successful flexible components
- Understand the sometimes-delicate relationship between functionality and complexity
- Learn about planning, references, labels, parameters, geometry, nesting and much more.

We have limited time in the live session. It is my hope that one of you will suggest a good example that we can work on (and complete) in the session. If so, then we'll be building it on the fly and you can consider these examples as bonus material. However, if we have difficulty coming up with a good live example for the session, then I have included a few possibilities here. Enjoy.

Rotating North Arrow

A Generic Annotation with built-in rotation parameter

The default north arrow families included with Revit are simple Generic Annotation families. You place them with the Symbol tool (from the Annotation tab) and then rotate them manually with the rotate tool. This combined with Paste Aligned to Selected Views works fine in most cases to create and place a north arrow on all required plan sheets. The trouble with rotating the north arrow symbol itself however occurs when text is added to the symbol (like the letter “N” or the word “North.”) (see Figure 1).

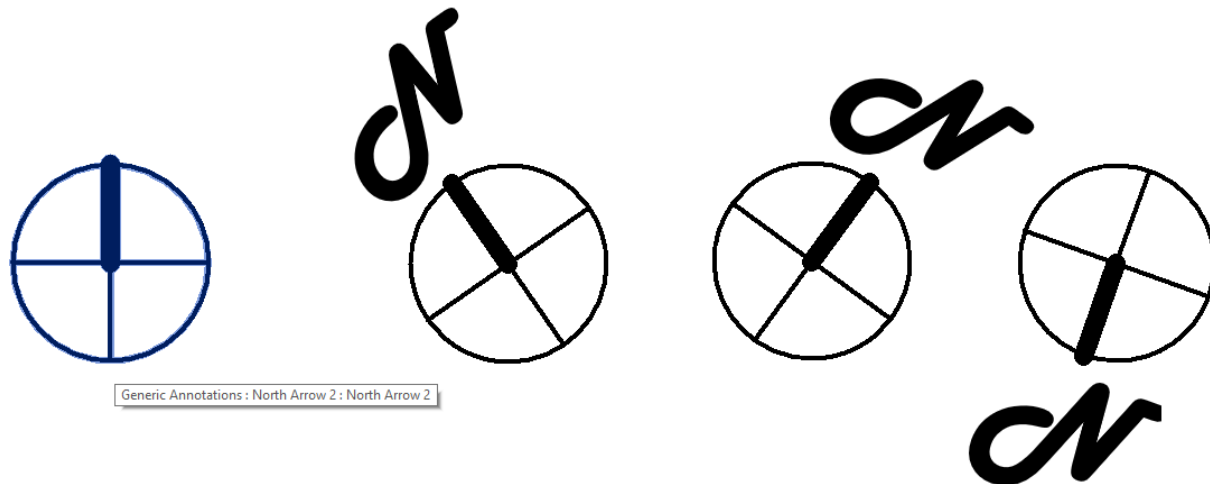


Figure 1 –Standard Generic Annotation North Arrow with Embedded text does not rotate well

Since you are rotating the entire symbol, the text also rotates. The “Keep Readable” setting is only a partial improvement. It will prevent the text from ever being upside down like the example on the far right of the figure, but there is no setting that will keep the text always upright. If you require this behavior, your choices are to place the text separately from the north arrow (not ideal) or build a rotation parameter into the north arrow geometry. Following is an example.

Features Explored:

- Working with Generic Annotation families
- Nested Labels vs. Embedded text
- Using Trigonometry to control rotation
- Understanding limitations when using Linework vs Filled Regions
- Bonus: Use Dynamo to rotate!

Planning

Start with determining what features you need your north arrow to have. Here we will focus on two main aspects: the rotation of the symbol graphics and the letter “N.” We want the position of the letter N to stay fixed relative to the symbol and not rotate.

Create the Text

To create the letter “N” you can just use text. But if you need it to move parametrically, you might want to consider a nested Generic Annotation instead. This will give you a nice stable insertion point that you can snap dimensions to. These dimensions can be labeled making it possible to adjust the position of the text. This is a nice feature if you want the size of the north arrow to flex. The location of the text can flex with the size of the geometry (see Figure 2).

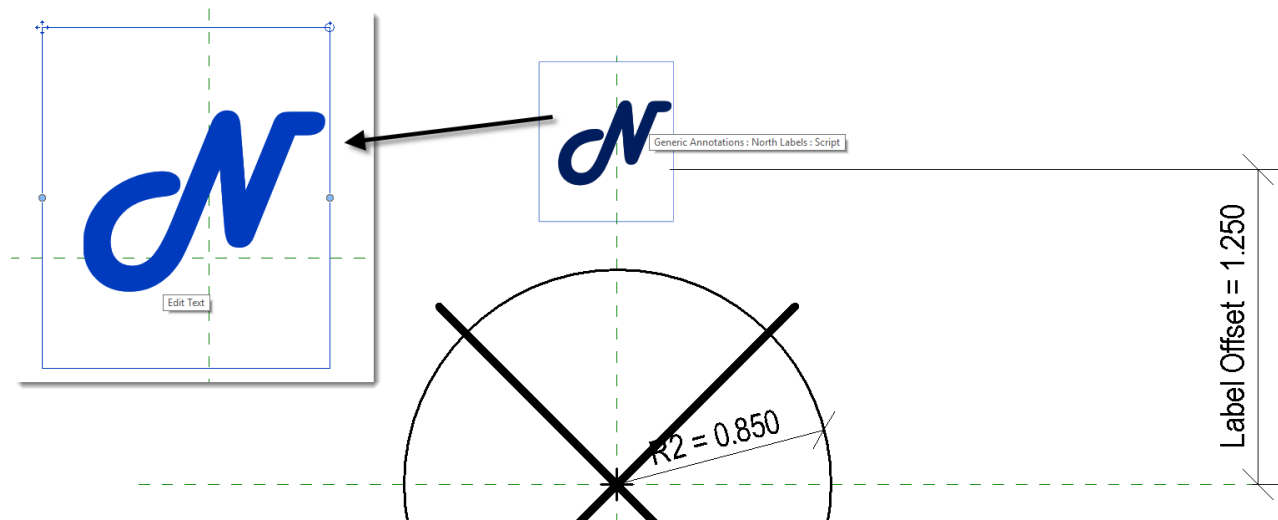


Figure 2 –If you want a flexible size symbol, and adjustable text, use a nested generic annotation

In the example in the figure, the text is placed just above the reference planes in the nested family.

Generic Annotation Characteristics

Generic Annotation families adjust to the scale of the view in which they are inserted, which is one of the reasons they are the best choice for symbols like this. However, there are some limitations in the generic annotation family template that make them also a challenge to work with.

- You cannot insert in any other family except another generic annotation.
- They contain only two reference planes (at the insertion point) and the reference plane tool is not available in the family editor for generic annotations.
- Reference lines are available, which can be useful, but they are not always the best replacement for reference planes.

Controlling Rotation

One of the most common ways to control rotation in a family is to use Reference Lines with a labeled angular dimension. Since it was noted above that reference lines are available in this type of family, this is certainly an option. However, rotation past 180 degrees and at 90-degree increments like 90 and 270 can often cause the family to break. It is not clear exactly why this is

the case, but if you want to use reference lines to control rotation in this family, you will need to allow for special cases like 90, 180 and 270. This can add complexity to the family.

Alternatively, you can forgo angle parameters and use trigonometry instead. This does require some basic knowledge of using trig in Revit and will also take a little effort to test and troubleshoot to get working, but in the end, I think it is much more stable than the angular parameters and much less likely to break.

Visibility

Regardless of the approach you use for rotation, both methods mentioned will need to be coupled with some visibility parameters. This is because we will need to hide and show different geometry in the family at different times.

Using Trigonometry

You can apply an angular dimension and label this with an angular parameter, but instead consider creating a Reference Line that is constrained diagonally across a flexible rectangle. By flexing the length and width of the rectangle, you can control the angle of the diagonal. The diagonal creates a right triangle with the sides of the rectangle. Trigonometry is all about right triangles! And the formulas needed are quite simple in this case (see Figure 3). The red angular dimension in the figure is just for reference. It is not required in the family.

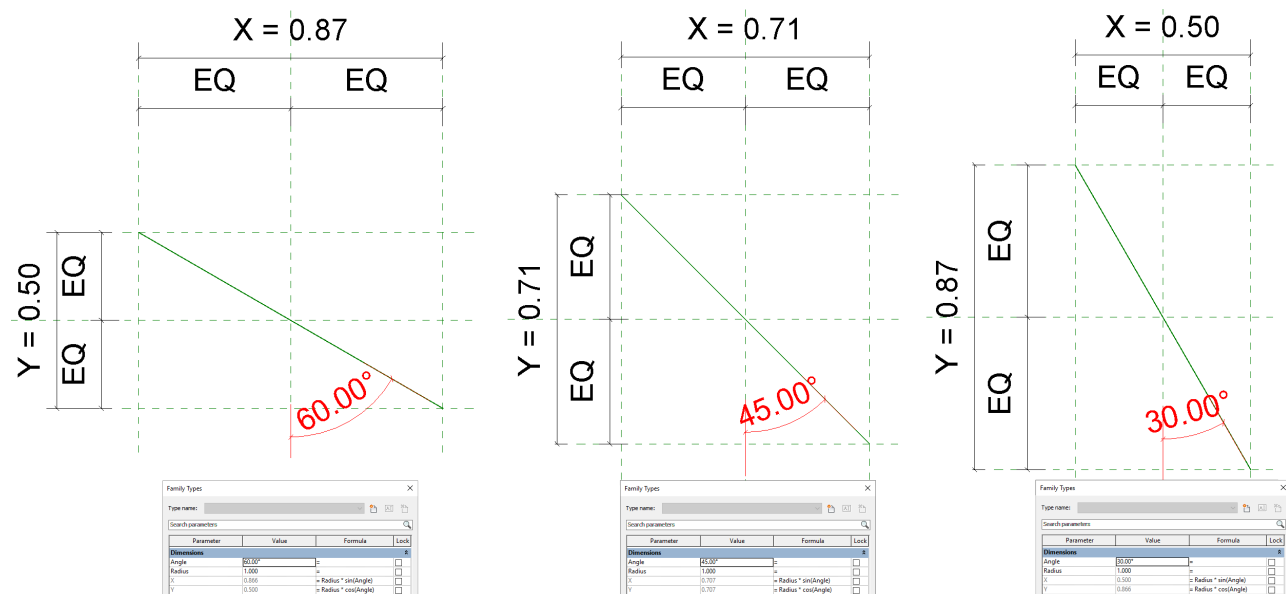


Figure 3—Use trigonometry instead of angle parameters to control the rotation of the diagonal line

So, this is helpful, but we need to be able to rotate around a full 360 degrees. Angles greater than 90 would make one or both of our dimensions negative. This will cause the labeled dimensions to fail. So rather than reference the dimensions to the centerlines, add two more reference planes and measure from an outside corner. Then do the math to compensate for this shift. This will keep all the numbers positive and thereby allow for a full circular rotation (see Figure 4).

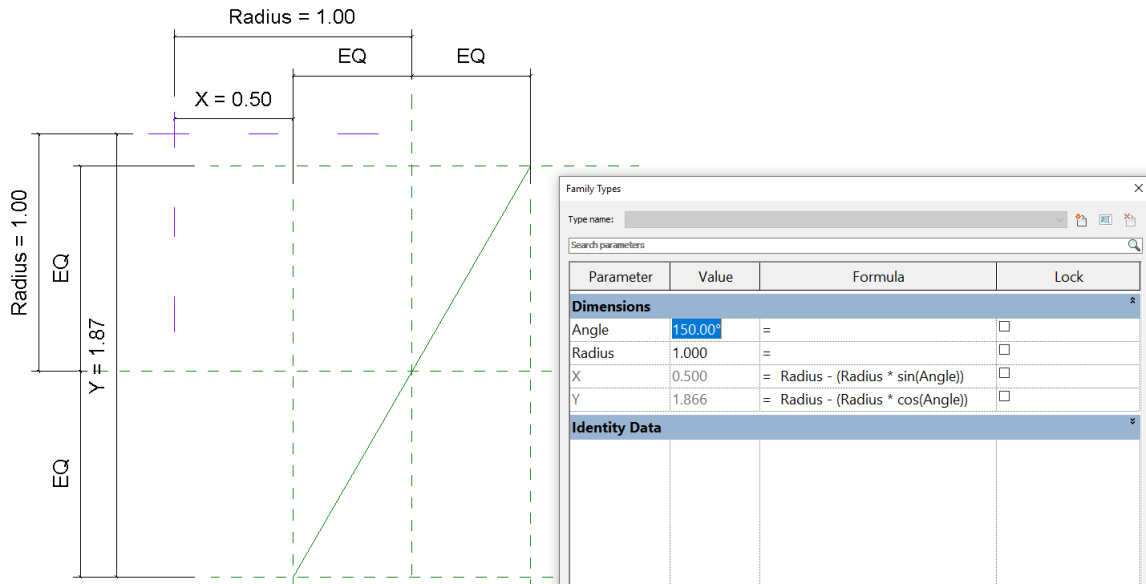


Figure 4 – To prevent labeled dimensions from failing, measure from an outside corner instead

With a rig like this, you can use any angle, positive or negative and it will rotate fully and will not fail. You can then associate geometry to the work plane of the reference line and it will rotate as well.

The Add Extra Reference Planes to a Generic Annotation Family Hack

There is just one small problem with the above strategy. You cannot add reference planes to a Generic Annotation family. At least not in the traditional way... Turns out that you can copy them from another family and paste into your generic annotation. But not just any family, and, you have to do it just right. I'll give a shout out here to John Pierson of Parallax Team here. I learned this trick from his AU class last year. You can see the class here:

<https://www.autodesk.com/autodesk-university/class/Flexible-Families-Learn-Make-Your-Families-Stretch-and-Flex-Never-2017>

First, you have harvest the required reference plane from a family in the out-of-the-box architectural template. So create a new project from the default architectural template installed with the US Imperial installation of Revit. Expand Families then Annotation Symbols. Right-click and Edit the View Title family. In that family, use VG, turn on Reference Planes and then you will be able to select, and copy the reference planes to your clipboard! You can then paste these into a Generic Annotation family.

Not sure how long it took John to figure that out, but I am not sure I want to know. We'll just say thanks John!

Now, there is one small "gothcha." When you paste these reference planes, their "Is Reference" and "Defines Origin" behaviors will not be editable. This means when you paste, they will try to take over as your origin point. Therefore, you must do another step in the work around to overcome this. You can simply paste them a second time. The second copy will become the new origin. Then select the first set pasted (which are not the origin any longer) and copy them again. Now you can create a second (and final) Generic Annotation family and paste them there and they will not mess with the origin this time. You can close all other families without saving. They were all

just temporary files.

Once you have these reference planes in the Generic Annotation family, you can make additional copies as required to complete the rig noted above. The final looks like Figure 5.

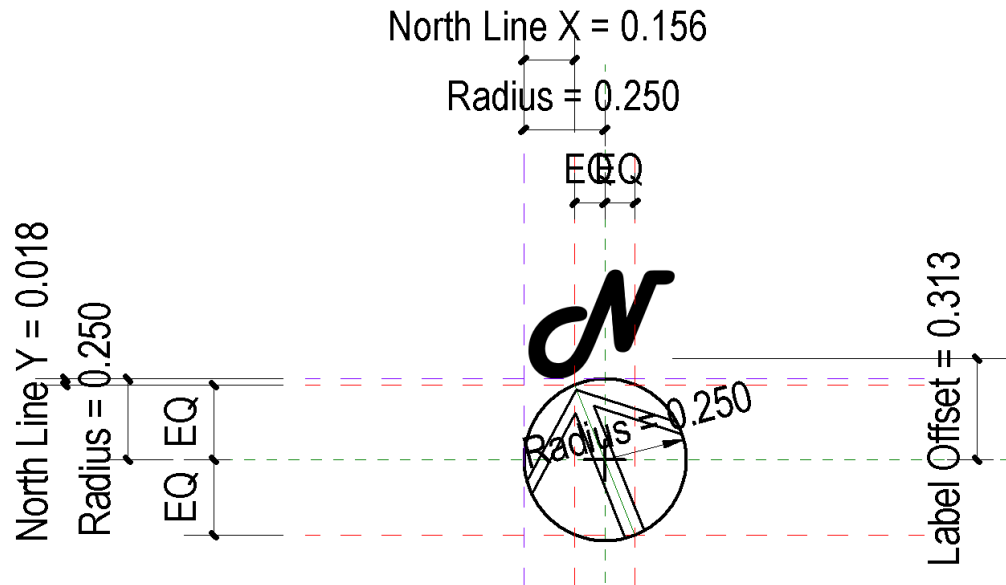


Figure 5 –Purple and Red Reference Planes are pasted from the OTB View Title family

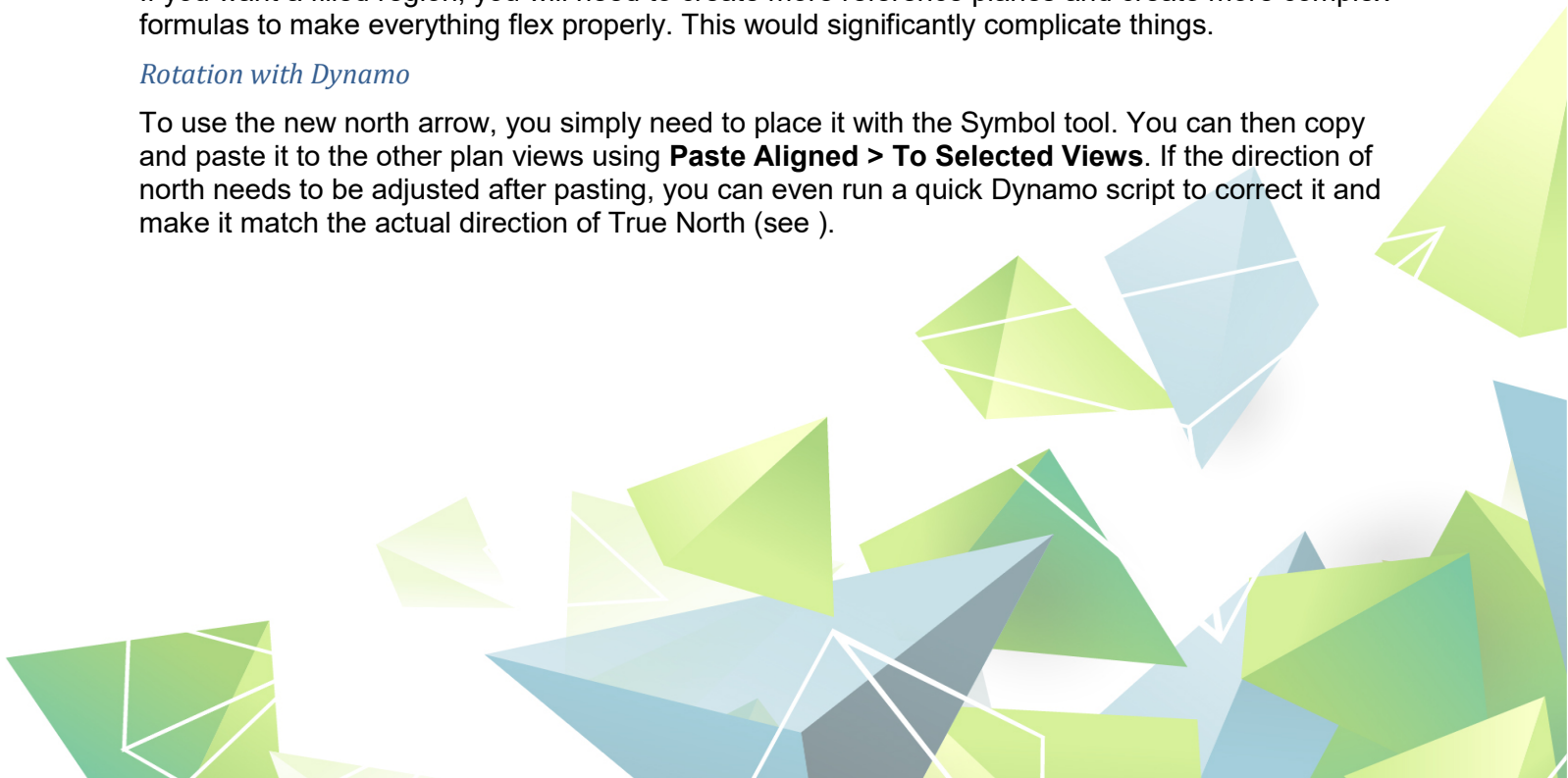
Filled Region “Gotcha”

Sadly in my tests, a filled region will not respect the reference line work plane. So when you rotate, its geometry either ignores the rotation or becomes distorted. Therefore, the figure shows some linework added for the north arrow geometry using simple lines. Lines do respect the work plane of the Reference Line. So, they rotate just fine. The Letter “N” is a nested Generic Annotation. It does not rotate, which was the intended design of this north arrow.

If you want a filled region, you will need to create more reference planes and create more complex formulas to make everything flex properly. This would significantly complicate things.

Rotation with Dynamo

To use the new north arrow, you simply need to place it with the Symbol tool. You can then copy and paste it to the other plan views using **Paste Aligned > To Selected Views**. If the direction of north needs to be adjusted after pasting, you can even run a quick Dynamo script to correct it and make it match the actual direction of True North (see).



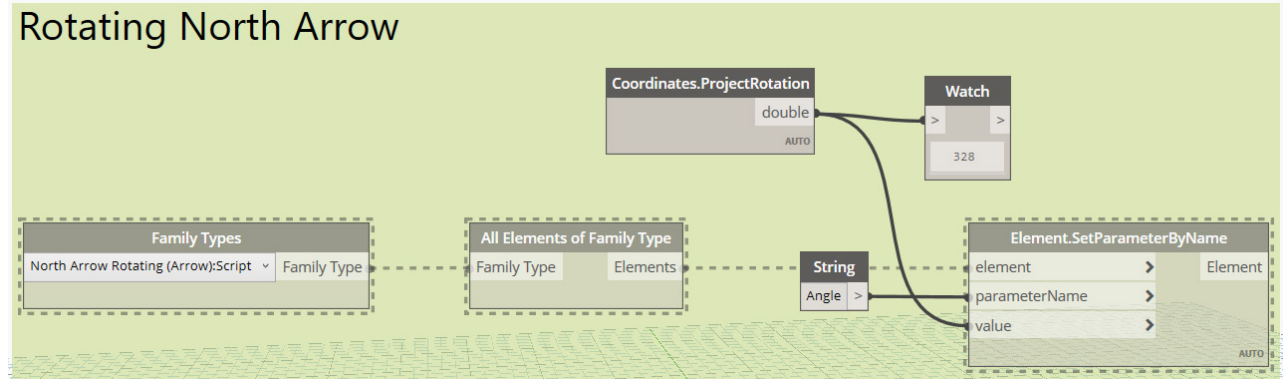


Figure 6 –Use Dynamo to query True North and rotate all instances of the north arrow

Parametric Bar Scale

Graphical bar scale that adjusts parametrically

This is another Generic Annotation family. The family itself is a simple graphical scale. The challenge here is creating a single family that can be used for any view scale. This is particularly vexing since one of the features of Generic Annotation families is that they adjust to the scale of the view. But this behavior is built-in and affects the graphics of the family only. You cannot access the scale-dependent behavior to adjust what values the labels in the family report. In other words, there is no way to make the Generic Annotation use the scale of its host view as a multiplier to the text labels displayed within it. We can overcome this easily with a formula, but we still must input a scale factor to make the whole thing work. This is unfortunate, but we can use two additional features to lessen the impact of the manual aspects of this solution: Dynamo and Lookup Tables. Therefore, the main point of this example will be showcasing these two tools.

Features Explored:

- Generic Annotation
- Nested labels
- Formulas to drive scales based on a Scale Factor
- Adjustable length
- Use Dynamo to tie to view scales

Building the geometry

If you looked at the previous example, we will leverage some of the same techniques here. Namely, the trick to copy and paste additional Reference Planes into our Generic Annotation family. This will make it much easier to make the family flexible. But there are other methods that can be used instead if you would prefer not to copy and paste reference planes. Please refer to the topic “The Add Extra Reference Planes to a Generic Annotation Family Hack” above for details about copying and pasting reference planes.

Following the procedure above, start by pasting in one in each direction. You will be able to further copy these as needed. Start with the basic layout (see Figure 7).

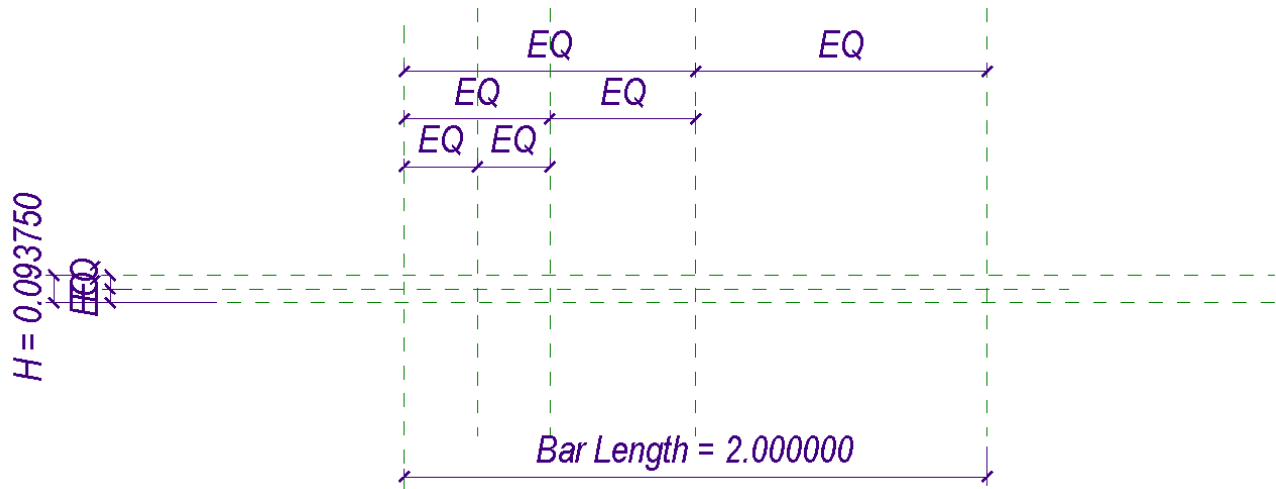


Figure 7 –Copy and paste reference planes from the OTB View Title family as noted above. Add dimensions

Use mostly equality dimensions to constrain the reference planes. Establish two parameters for the overall length of the bar scale called: **Bar Length** and another for the height called: **H**. These are both type-based.

Create two filled regions based on these reference planes. Use the <Invisible Lines> line style to make the edges nice and crisp. A solid black fill works well, or feel free to customize the fill pattern if you wish (see Figure 8).

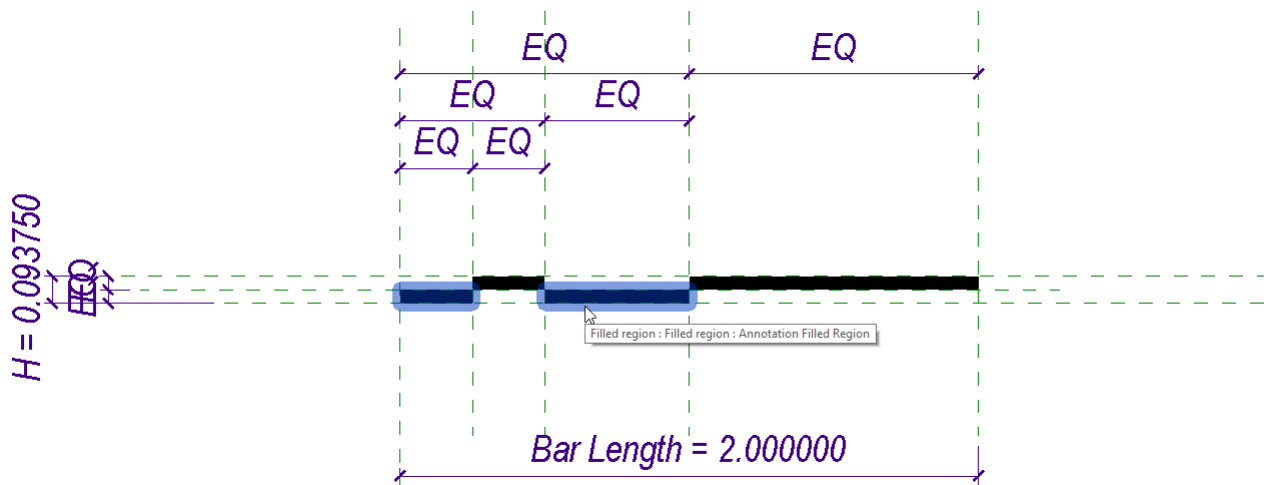


Figure 8 –Build filled regions constrained to the reference planes

Make sure that all the edges in the filled regions are locked to the reference planes to ensure that they flex properly.

If you wish, you can add additional linework and any required reference planes and dimensions to constrain it (see Figure 9).



Figure 9 – Optionally add additional linework and reference planes to constrain them

Using Nested Generic Annotations for the Labels

At this point, you can add Labels for the text portion of the symbol. However, if you want a little more control over the final symbol, consider creating a nested Generic Annotation family instead. The nested generic annotations will give you an insertion point that you can align and dimension to. This means you will be able to ensure that the location of each label remains lined up as desired and will flex with the family as it flexes.

Create a new Generic Annotation family. Add a label in the middle of the screen and then edit it. You will need four copies of this family. The first three will need a new parameter named: **LabelNo** (see step 1 in Figure 10). This should be a Length parameter and set to Type-based. Add this to the label and then edit its unit format (see step 3 in Figure 10), choose Decimal Feet with 1 decimal place with the unit symbol for feet.

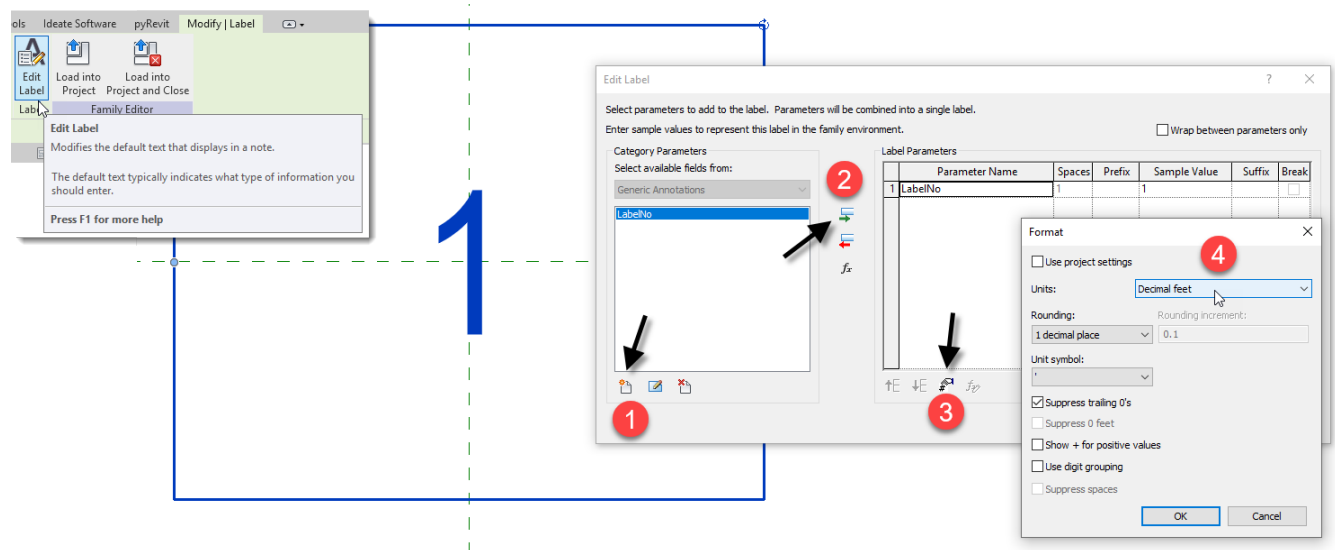


Figure 10 – Create new Generic Annotation with a single label parameter

Edit the Family Types in this family and make three types called: **Start**, **Mid** and **End**. Save this family as: Scale Breaks and load it into the bar scale generic annotation family. Place one of each type randomly for now above the graphics.

Return to the label family. Save the second version as: **Zero Label**. Delete two of the types, rename the remaining one to: **Zero**. Edit the label and edit the unit format. Set the unit symbol to None and complete and save this family. Load it into the bar scale family and place two copies near the left side of the graphics.

Return to the Zero Label family, save it as: **Reference Label**. Rename the type in this family to: **Reference Length**. Edit the units format of this label and set it to Decimal Inches. Save the family, load it into the bar scale family and place one instance near the lower right.

The last copy of the family needs a new label. It might be easier to start this one from scratch as we need to delete everything anyhow. Build it exactly like the others except that you want the parameter to be a Text parameter this time. So, in step 1 above, be sure to change the Type of Parameter to: **Text** before adding it to the label. The units format step will not be necessary for this label. Name the type in the family to: **Scale**, save the file as: **Scale Label** and load it into the bar scale family and place it below the graphics.

Finally, you can use Align and lock to move each label family instance and lock it to an appropriate reference plane (see Figure 11).

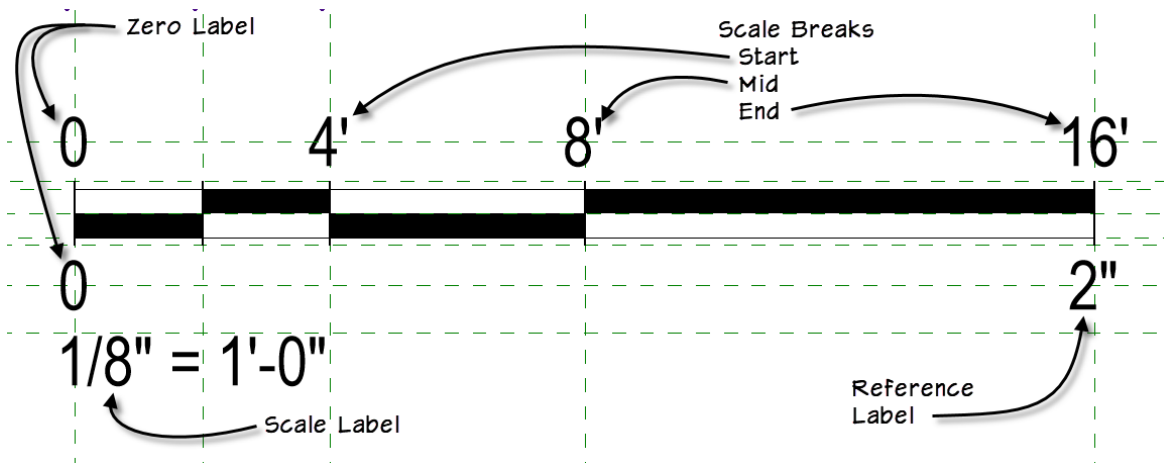


Figure 11 –Position all labels with align and lock

Associate Family Parameters to Nested Labels

Now that you have all the label families placed and positioned, you can associate parameters in the host bar scale family to drive the values of the nested label parameters. To do this, click on one of the label families onscreen. On the Properties palette, click the Edit Type button. In the “Type Properties” dialog, you will see the nested label parameter. Click the small Associate Family Parameter icon to the far right next to the label parameter (see Figure 12).

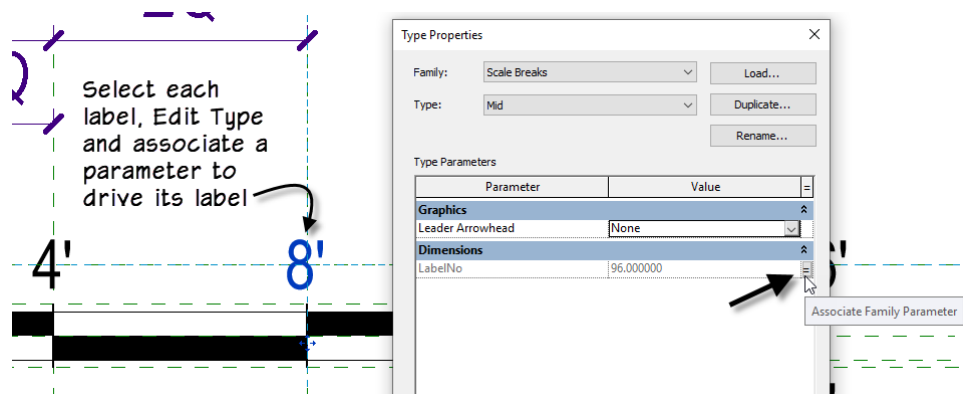


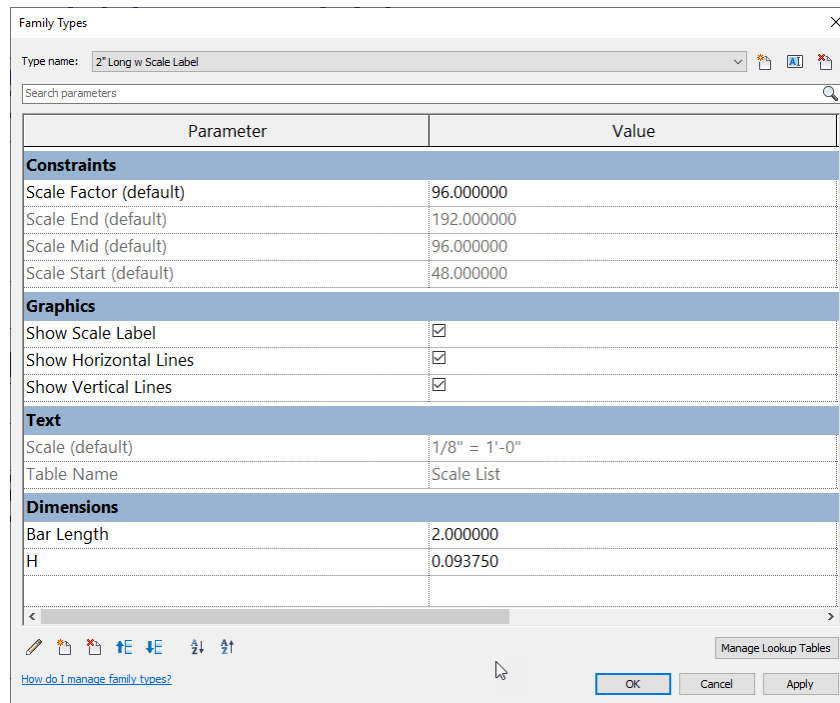
Figure 12 –Associate family parameters to the nested labels

In the “Associate Family Parameter” dialog, click the New Parameter icon at the bottom and create a new parameter to drive this parameter. Click OK to dismiss all dialogs.

Repeat this process for the Start, Mid, End and Scale labels. Make each of these parameters instance-based. Repeat once more for the Reference label, but make it type-based. For the zero labels, no need to associate a parameter. Just ensure that the value is set to 0. These will stay zero permanently.

Add remaining parameters

Edit “Family Types” and add a few additional parameters. At a minimum, we will need a Number instance-based parameter called: **Scale Factor** and a Text parameter called: **Table Name**. Optionally, you can add some Yes/No parameters to control the display of various graphical pieces of the family. For example, in Figure 13 I have added checkboxes to toggle the linework and the scale label. If you add these, be sure to remember to associate these parameters to the Visible setting of these elements on the Properties palette.



The screenshot shows the 'Family Types' dialog box for a family named '2" Long w Scale Label'. It contains several sections of parameters:

Parameter	Value
Constraints	
Scale Factor (default)	96.000000
Scale End (default)	192.000000
Scale Mid (default)	96.000000
Scale Start (default)	48.000000
Graphics	
Show Scale Label	<input checked="" type="checkbox"/>
Show Horizontal Lines	<input checked="" type="checkbox"/>
Show Vertical Lines	<input checked="" type="checkbox"/>
Text	
Scale (default)	1/8" = 1'-0"
Table Name	Scale List
Dimensions	
Bar Length	2.000000
H	0.093750

At the bottom, there are buttons for 'OK', 'Cancel', and 'Apply', along with a 'Manage Lookup Tables' button and a link 'How do I manage family types?'.

Figure 13 –Add the remaining required parameters in Family Types

For now, use the figure to input some basic values for each parameter. We’ll add formulas to improve this next.

Using Formulas to drive the labels

As noted at the start of this example, it is unfortunate, but this solution relies on a Scale Factor parameter. I tried many ways to extract this value intelligently from the model, and while it is taggable in a view title, the value of that parameter is not accessible to formulas in the family editor. At the end of this example, we’ll use Dynamo to help smooth this over. But sadly, I found no way to make it completely automated.

Let’s focus for now on the formulas we can write. The scale breaks are simple arithmetic based on

the Scale Factor and Bar Length parameters:

Parameter	Formula
Scale End	(Bar Length / 1) * Scale Factor
Scale Mid	(Bar Length / 2) * Scale Factor
Scale Start	(Bar Length / 4) * Scale Factor

Using a Lookup Table for the scale list

The scale label is just a text parameter. This means if we leave it as-is, we would be required to type in the scale manually each time we change it. This is of course not ideal. The trouble is, finding a formula type that will give us what we need. It turns out there is an easy way to build a list and match it up to the value of the Scale Factor parameter. This means that a single parameter will drive all the other flexible values in the family! The trick is to use a lookup table!

Lookup tables are lists of values stored in a delimited text file. This file is embedded directly in the family and can be referenced formulaically. Lookup tables can contain as many values or combinations of values that you need. In this case, we have a reasonably finite list of scales that are typically used in architectural drawings. Therefore, we simply need to build that list in Excel in the proper format and then save it as a CSV file. Here is the data that needs to be input into Excel. Column A will contain the scales formatted exactly the way you want them to appear in the scale label. Column B is the Scale Factor that will be used to lookup the values in column A. Column A has no header (notice it is blank). Column B must be formatted correctly:

- **SF** The name of this column of data (SF is short for Scale Factor in this case)
- **NUMBER** The data format (options include, NUMBER, LENGTH, AREA, etc.)
- **GENERAL** The unit format for this data (other examples include: INCHES, FEET, etc.)

A double hashtag is used as a delimiter in the header.

Column A	Column B
	SF##NUMBER##GENERAL
12" = 1'-0"	1
6" = 1'-0"	2
3" = 1'-0"	4
1 1/2" = 1'-0"	8
1" = 1'-0"	12
3/4" = 1'-0"	16

1/2" = 1'-0"	24
3/8" = 1'-0"	32
1/4" = 1'-0"	48
3/16" = 1'-0"	64
1/8" = 1'-0"	96
3/32" = 1'-0"	128
1/16" = 1'-0"	192
3/64" = 1'-0"	256
1/32" = 1'-0"	384
1/64" = 1'-0"	768
1" = 10'-0"	120
1" = 20'-0"	240
1" = 30'-0"	360
1" = 40'-0"	480
1" = 50'-0"	600
1" = 60'-0"	720
1" = 80'-0"	960
1" = 100'-0"	1200
1" = 160'-0"	1920
1" = 200'-0"	2400
1" = 300'-0"	3600
1" = 400'-0"	4800

After typing this information into Excel, save the file as: **Scale List.csv**. Make sure to save as a CSV file, not an Excel workbook.

Reopen “Family Types.” At the bottom right corner of the dialog, click the Manage Lookup Tables button and then click Insert and browse to the CSV file, just created (steps 1 and 2 in Figure 14). Recall the text parameters we created above. The Table Name parameter will be used to point to CSV file. The Scale parameter will perform the lookup function.

(see Figure 14).

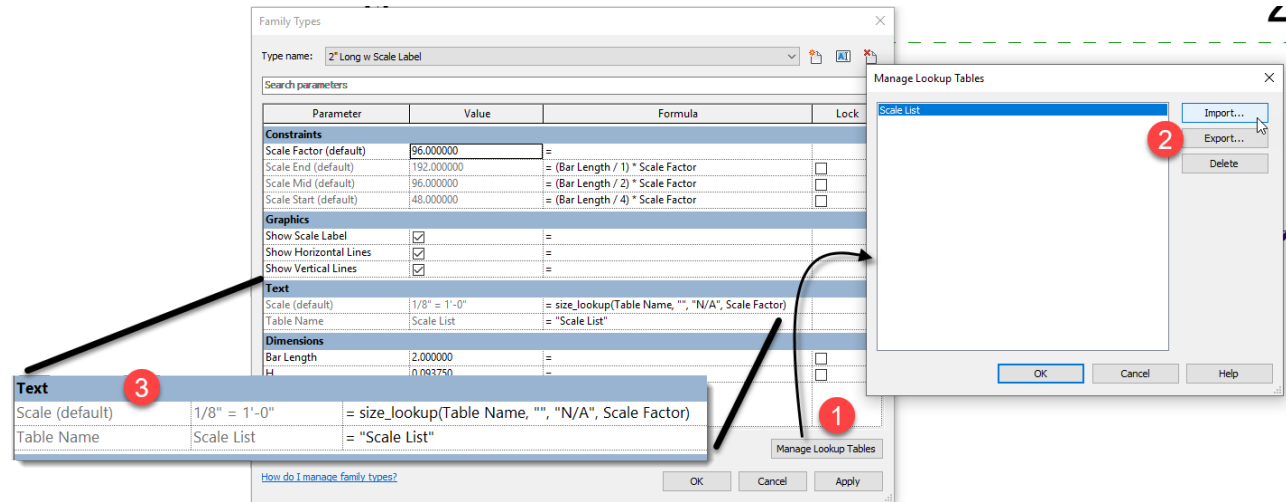


Figure 14 –Add the remaining required parameters in Family Types

In the formula field next to Table Name, input the name (in quotes) of the CSV file. Do not include the extension, so it is: “Scale List”. This makes the values read-only. For the formula in the Scale parameter, type:

size_lookup(Table Name, "", "N/A", Scale Factor)

size_lookup is a built-in formula type in Revit. It will take a minimum of four arguments.

size_lookup(Lookup Table Name, Value to Return, Return Value if Lookup not found, 1st Value to Lookup, Optional – 2nd Value to Lookup, Etc.)

Lookup Table Name—As noted above, you can use a parameter to record the name of the table as we did above. In that case, you put the name of the parameter here. If you do not create the parameter, you can reference the name of the lookup table file directly, but in that case, you must include the extension (.csv) after the name. If you have the parameter name as we do here, you do not include an extension in the formula or in the value of the parameter.

Value to Return—This is the name of the column from the lookup table that contains the values you want returned if a match is found. This is the name of the column header in the CSV file, *not* a parameter in the “Family Types” dialog. The formula we are using here does not include a name. It has empty quotes ("") instead. If you use empty quotes as we have done here, it returns values from column A. This is how you get a lookup table to return text values!

Return Value if Lookup not found—The next argument is what should be reported if Revit cannot find the value you are looking for. In this case, I have input “N/A”, but you can put anything you like here, even another formula! Just remember , in this case we must put it in quotes since the format of the **Scale** parameter is Text.

1st Value to Lookup—Finally, the last argument(s) is/are the values you want to search for in your lookup table. In our case, we are looking for the value of **Scale Factor** parameter. For example, if Scale Factor is set to: 96, it will return the corresponding value in column A: 1/8" = 1'-0".

Our lookup table is very simple. We are only looking for one value, but lookup tables can look for several values. To do so, you would add more arguments to the formula and add corresponding columns of data to the CSV file.

Input the formula and then close "Family Types." To test the family, try alternate scale factors. As long as they occur on your list in the CSV, you should get the corresponding scale in the text field. Also, the formulas will adjust the scale breaks and report the correct numbers.

Bonus—Use Dynamo to Query and Set Scale Factors

The family will work well and only requires a single parameter to be modified when used on a view of a different scale. But as you can imagine, it is likely that you may forget to check this and set the correct scale factor, rendering the scale shown in the graphic scale incorrect. To help ensure that this does not happen, we can make a simple Dynamo script (see Figure 15).

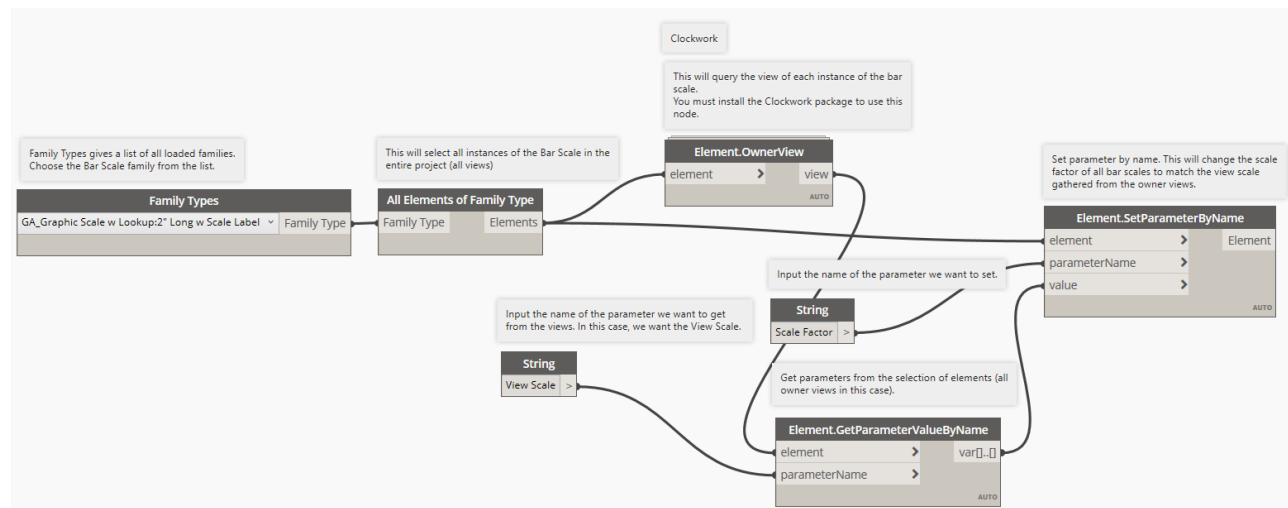


Figure 15 –Dynamo Script that reads and sets the proper scale factor for all instances of the bar scale family

This script simply gathers a list of all the bar scales used in the project. From that list, it uses a node from the Clockwork package to query the scale of each owner view. Once we have the scales, we can use them to set the value of the Scale Factor parameter in all of the bar scales.

Conditionally Colored Tags

Adjust the color of tags parametrically

I saw this question in the Autodesk forum not too long ago:

"Is it possible to get the tags themselves to change color based on a parameter or even if I have to setup a separate tag for each color but would really like my tags to show in color to help service my client better."

Post: <https://forums.autodesk.com/t5/revit-architecture-forum/how-to-create-colored-tags/m-p/8525161#M225719>

I took a stab at trying to solve this. I came up with a half solution. The main issue is that while phasing parameters are built-in to Revit, they are not exposed to families. So, we cannot reference the phase of elements in formulas. Despite this, I offered a solution that should prove acceptable.

Features Explored:

- Custom Tag
- Shared Parameters for Tags
- Using text formula to conditionally control label display

I achieved limited success with the text portion of the tag, not so much with the graphics. So, if you can live with just the text changing color, then I can offer a partial solution.

Ideally, we would want this to be able to read the phasing parameters, but sadly, those are not exposed to tags. So, we do have to rely on some checkbox (Yes/No) parameters to make this work.

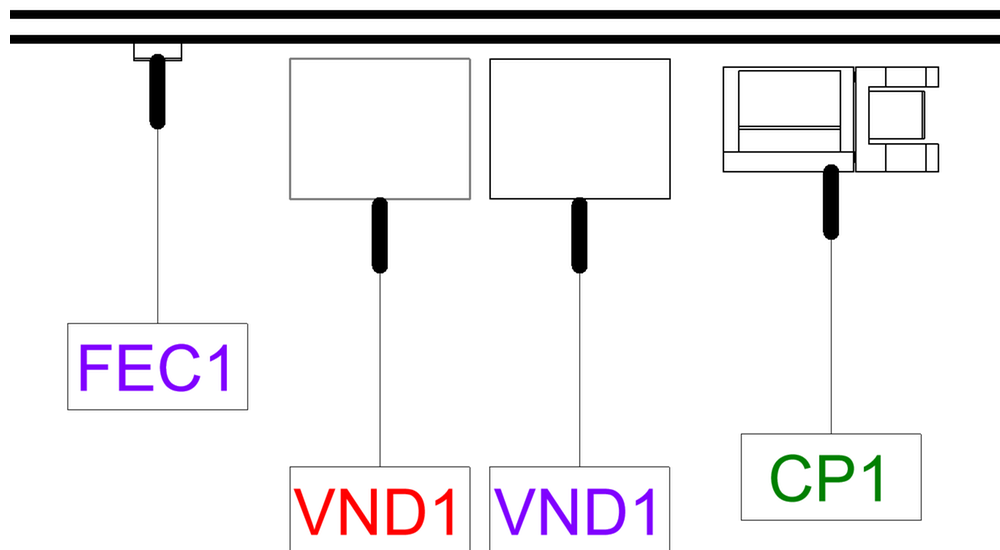


Figure 16 – Tags with colored labels that automatically react to the Yes/No checkboxes

So, here is what you need:

1. Create or edit your Shared Parameter file. (If you are not sure how to do this, you can look online or here in the forums. Plenty of resources, but to show in tags, custom parameters must be shared).
 - Make three Yes/No parameters. I called them "New", "Existing" and "Relocated"
2. Load these into a project and create three Project Parameters that use these three shared parameters. Assign them to Specialty Equipment. I made them Instance parameters and chose to let them vary by group instance. (Project Parameters automatically get assigned to all elements of the chosen category in a project. You do NOT need to edit each family).
3. Edit or create a Specialty Equipment tag. Make three copies of the label. Create three text types - one of each desired color and assign to each label. Place them right on top of each

other.

4. Edit each label. At the bottom of the Edit Label dialog, click the Add Parameter icon. Add the three Yes/No check-boxes from the shared parameter file in step 1.
5. Create a Calculated Parameter (middle of the dialog). Make it a text parameter and input the following formula: **if(Existing, Type Mark,"")**
 - What this means is this: It will look at the Yes/No parameter "Existing". If it is checked (set to yes), then it will display the Type Mark parameter value. If it is unchecked (set to no) it will use empty quotes "" or in other words, fill in a blank text value.

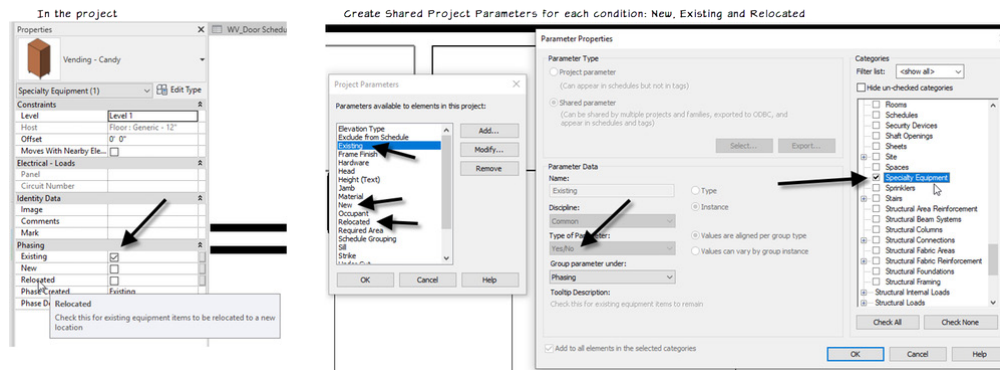


Figure 17 – Add shared Project Parameters as instance parameters to the Specialty Equipment category

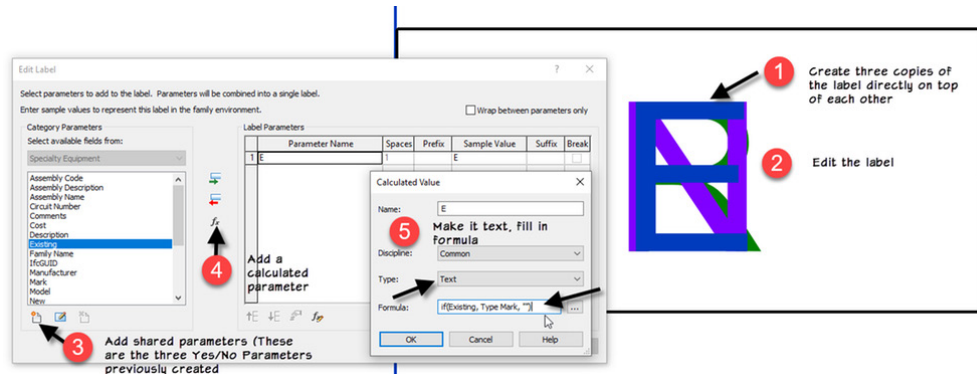


Figure 18 – Use calculated value to format the labels in the tag

- Repeat this for the other two labels substituting the appropriate formulas:
 - **if(New, Type Mark,"")**
 - **if(Relocated, Type Mark,"")**

Once everything is configured, if you select a piece of tagged specialty equipment, check one of the check-boxes on the Properties palette and the label will change color.

Sliding Panel Door with Variable Quantity of Panels

Use formulas to determine quantity of panels and position them appropriately

This one comes from a recent Revit training session. I was at the client and they wanted to build a custom sliding door that had a variable number of door panels with parameters controlling the maximum panel size and amount of overlap. We will focus on just the panels in this one. So we'll start with the out-of-the-box door template and build it from scratch.

Features Explored:

- Nested Door Panel Family
- Parametric Array
- Formula controlling panel size and position

Strategy

The basic strategy will be as follows: Create the panel geometry. Load it into a family made from the default door family template. Array the panel and configure its position and quantity parametrically. Create formulas and flex.

Building the Door Panel Family

For the panel, we will start with a Generic Model and then convert it to a Door. We want to make it a Door so that we can use the standard door subcategories, but we will start with a Generic Model so that it will not require a host.

Create a new family and point to the Generic Model family template (See Figure 19).

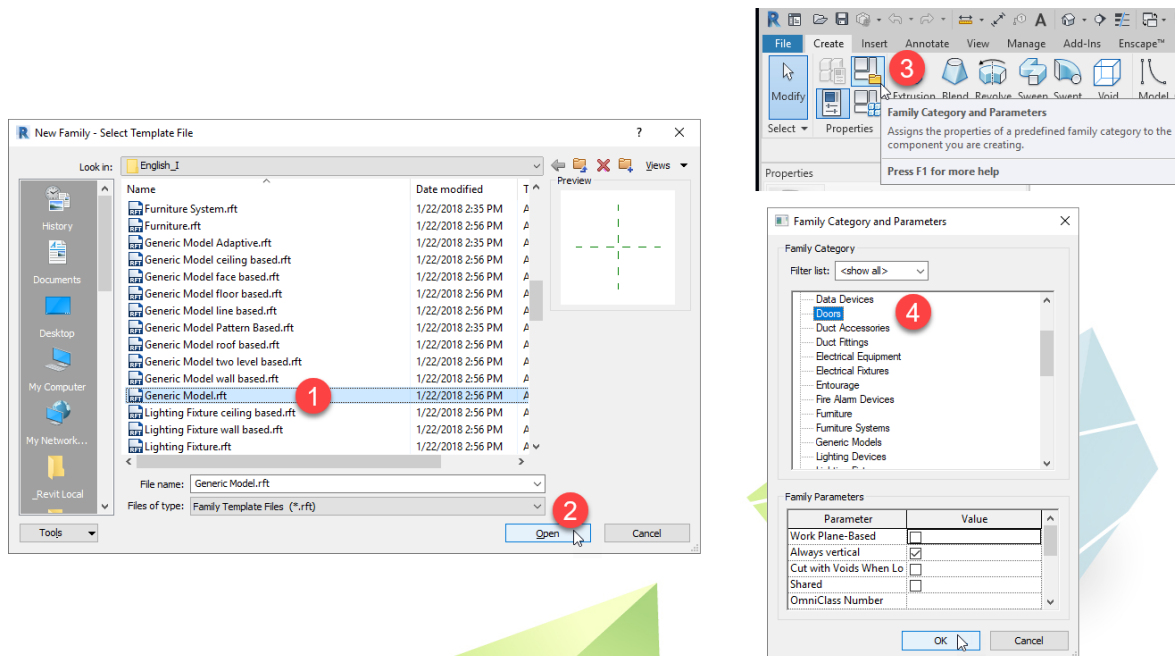


Figure 19 – Start with a Generic Model template and then change to Door

Click the Family Category and Parameters button, choose Doors and then click OK.

In the Front view, add two vertical and one horizontal Reference Planes. Name them Right, Left and Top. Add dimensions as shown in Figure 20.

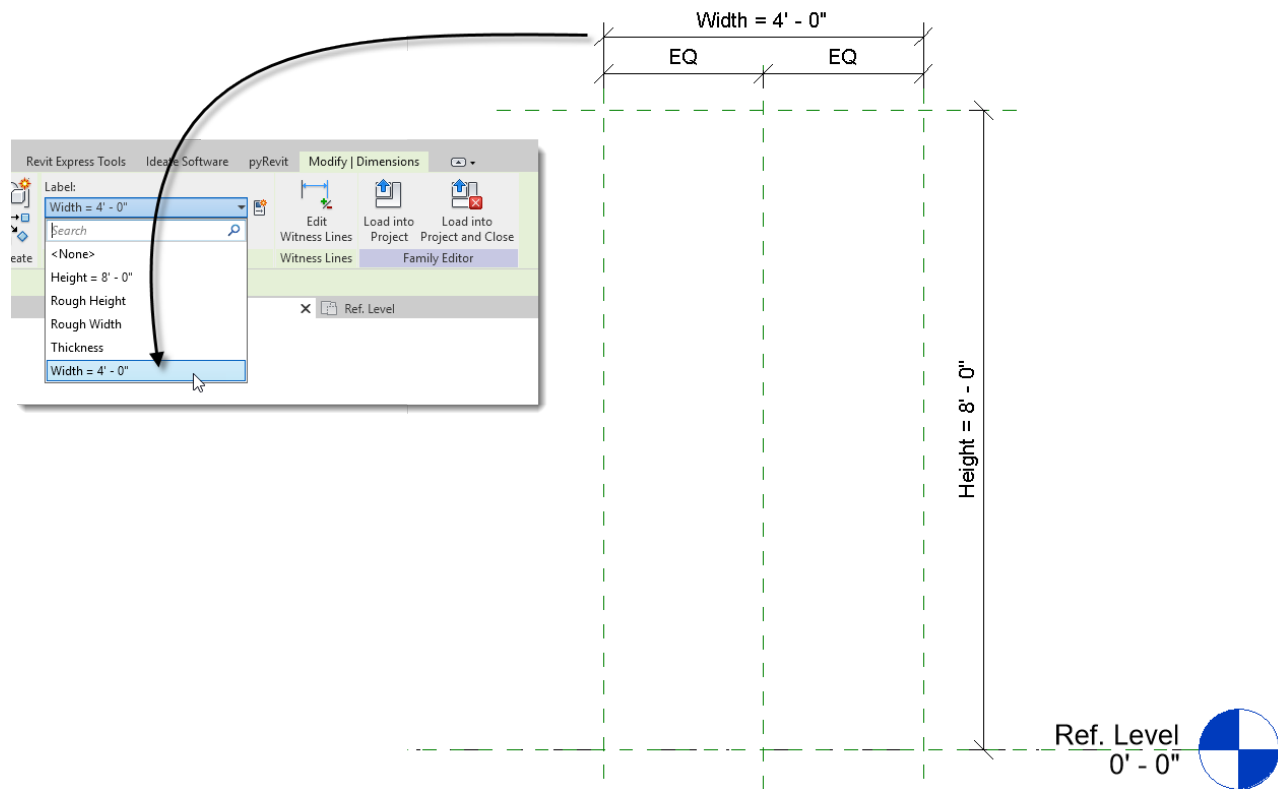


Figure 20 – Add reference planes and dimensions

Select and label the horizontal dimension with the Width parameter. Label the vertical dimension with the Height parameter. (These parameters appeared when we changed the category).

Stay in the elevation and create an extrusion. Use the rectangle shape and snap it to the reference planes you just drew and dimensions. Close all four padlocks before finishing the sketch. Keep the new extrusion selected and on the Properties palette, set the Extrusion End to: **1"** and set the Extrusion Start to: **-1"**. This makes the extrusion 2" thick. Optionally, you can assign parameters to these values using the Associate Family Parameters buttons, then create parameters to drive them. In this case, we are keeping it simple and going with a fixed thickness for our door panel.

Another option you can try on your own is making a more complex door panel with stiles, rails and glazing. We'll keep this one flush here.

Let's create a material parameter however. With the extrusion still selected, click the Associate Family Parameters button next to Material. Click the New Parameter icon at the bottom, call it **Door Panel** and then OK out of all dialogs (see Figure 21).

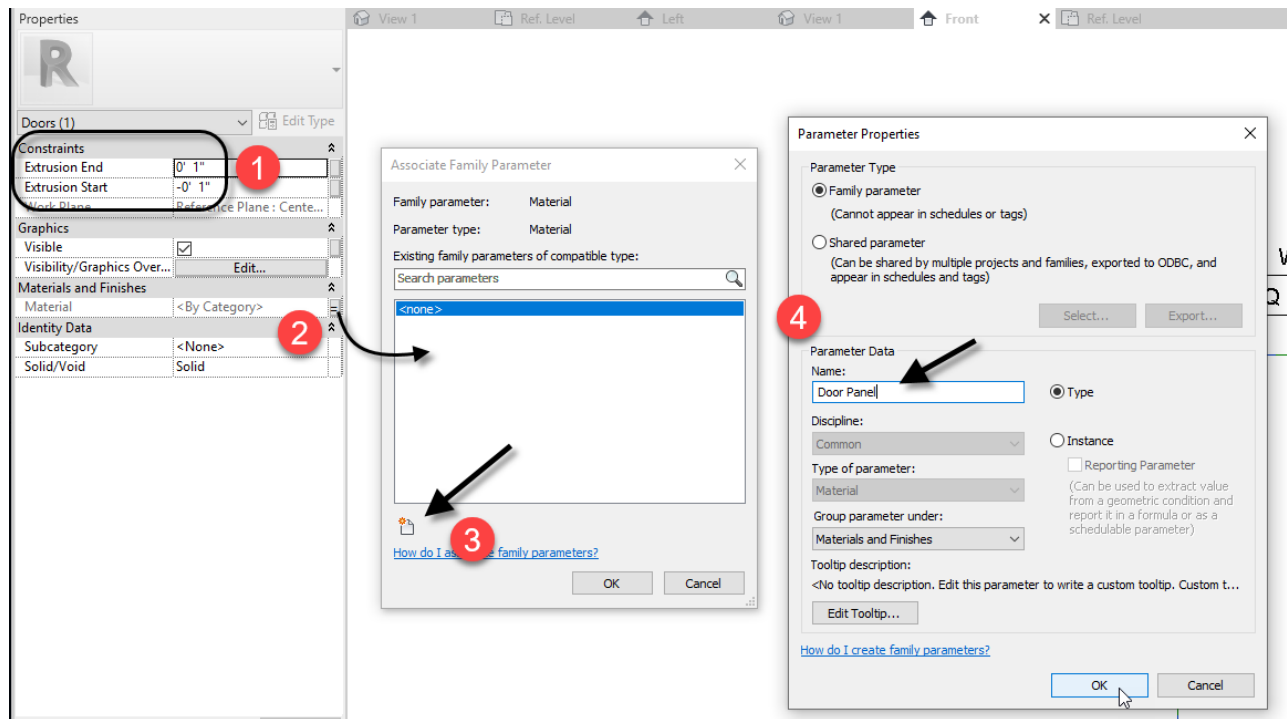


Figure 21 – Adjust thickness and make a material parameter

Save this family as: **Flush Door Panel**. Close the file.

Build the Door Family

Create another new family, this time choosing the default *Door.rft* family template. Once again, there is much we could do when creating a custom door family. In this example, we will be focused only on placing the panels and making sure that they flex properly. If you want to customize the frame or trim or create any symbolic graphics, you can explore that on your own later.

On the Create tab of the ribbon, click the Component tool. When prompted to load a family, click Yes and then load the family you just created and saved. Place it anywhere in the plan view. Do not try to place it precisely yet (see Figure 22).

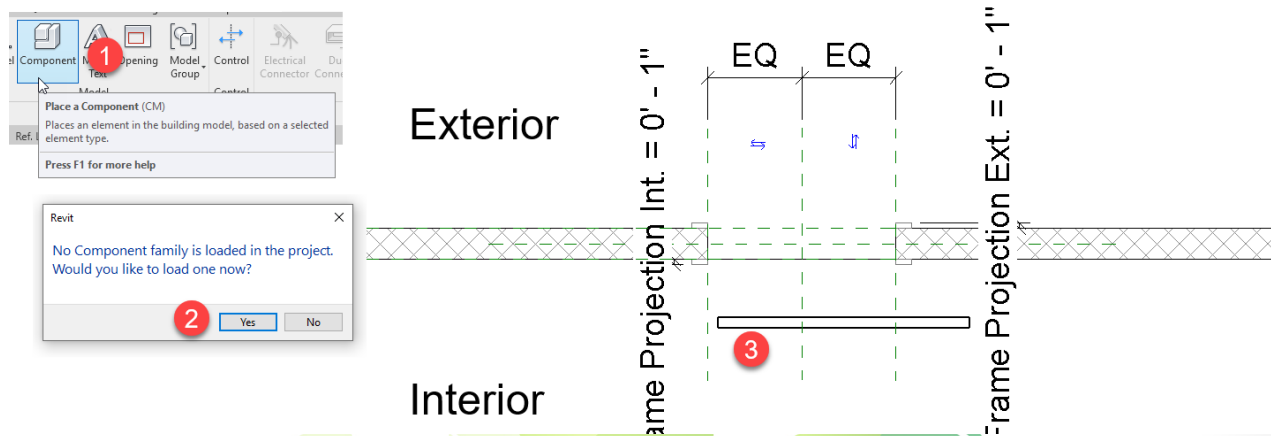


Figure 22 – Place an instance of the door panel in the new door family

Save this new family as: **Multi-Slider Door**.

Using the Align tool, align the door panel to the reference plane at the exterior face of the wall. Don't lock it (see Figure 23).

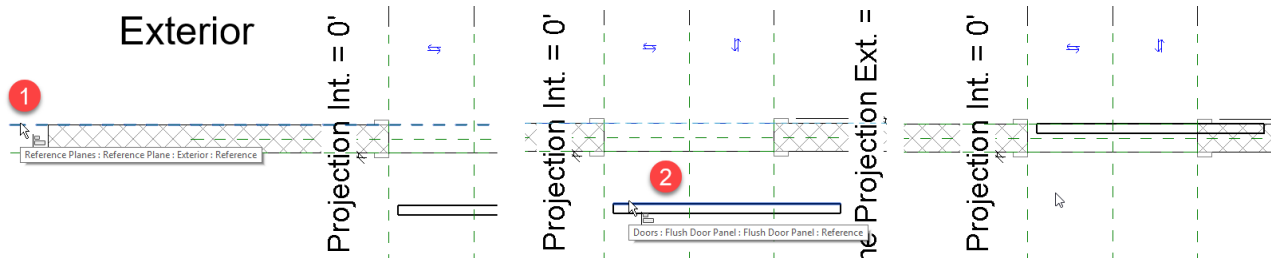


Figure 23 – Align the door panel to move it flush to the exterior face of the wall

Select the wall onscreen. Edit the temporary dimension that appears and make it: **25' - 0"**. Open the "Family Types" dialog. Flex the Width parameter and make it: **20' - 0"** and then click OK. Drag the two Frame Projection labeled dimensions off to either side to get them clear of the door opening. Move the door panel onscreen a little closer to the left side of the opening. Don't snap it yet. Align the left edge of the door panel to the left reference plane to move the door panel to the left edge of the opening (see Figure 24).

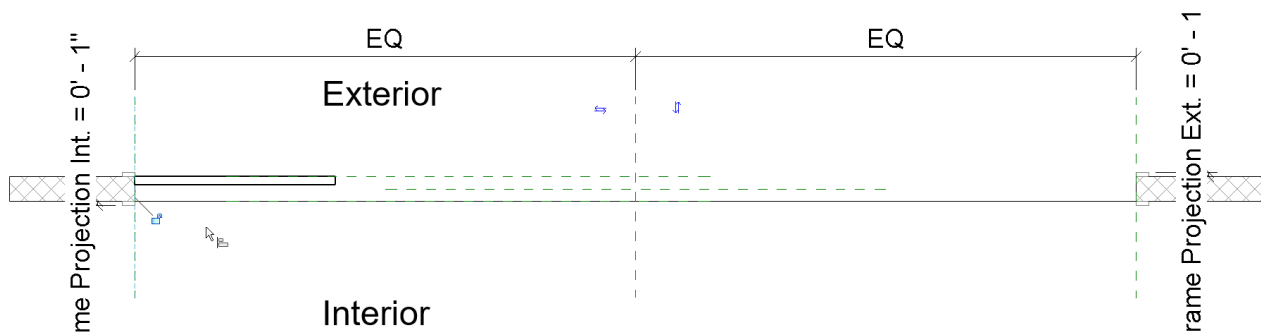


Figure 24 – Dimension the door panel and move it flush to the exterior face of the wall

Select everything onscreen. On the ribbon, click the filter button. Uncheck Reference Planes and then click OK. Hold down the SHIFT key and deselect the door panel, then using the Temporary Hide/Isolate pop-up (sun glasses) choose: **Hide Element** (see Figure 25).

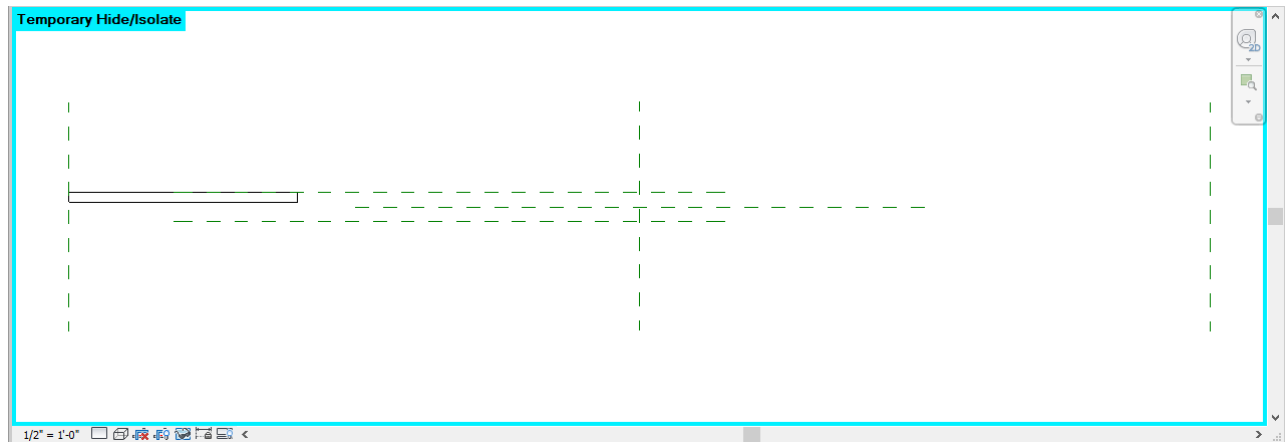


Figure 25 – Isolate just the door panel and the reference planes

Select the door panel. On the Modify tab, click the Array button. Pick the start point near the door panel, click the second point toward the right reference plane. It does not have to be precise. For now, leave the array count at 2 (see Figure 26).

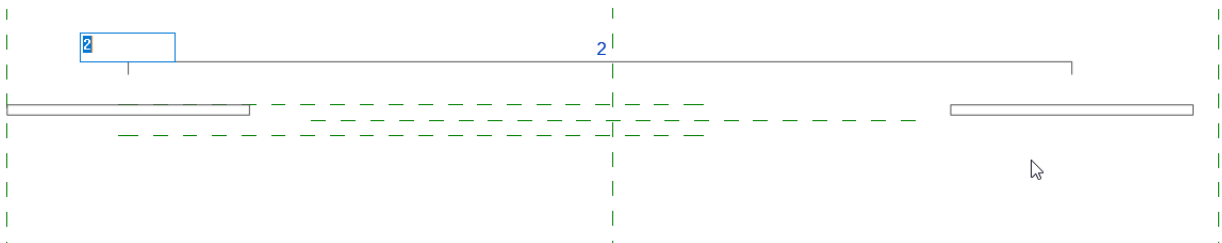


Figure 26 – Create an array from the door panels

Add dimensions: from the left reference plane to the centerline of the left door panel, from the right reference plane to the center of the right door panel and from the Exterior reference plane to the center of each panel (2 dimensions, both measured from the Exterior reference plane). The dimension from the left panel center to the exterior reference plane will be 1" (since we aligned that panel above). Simply lock this dimension. For the horizontal dimension to the left panel, label it with a new parameter called: **End Panel Offset**. On the right side, label the vertical dimension (it is also likely 1" currently) with a new parameter called **Offset from Exterior Wall Face** (see Figure 27).

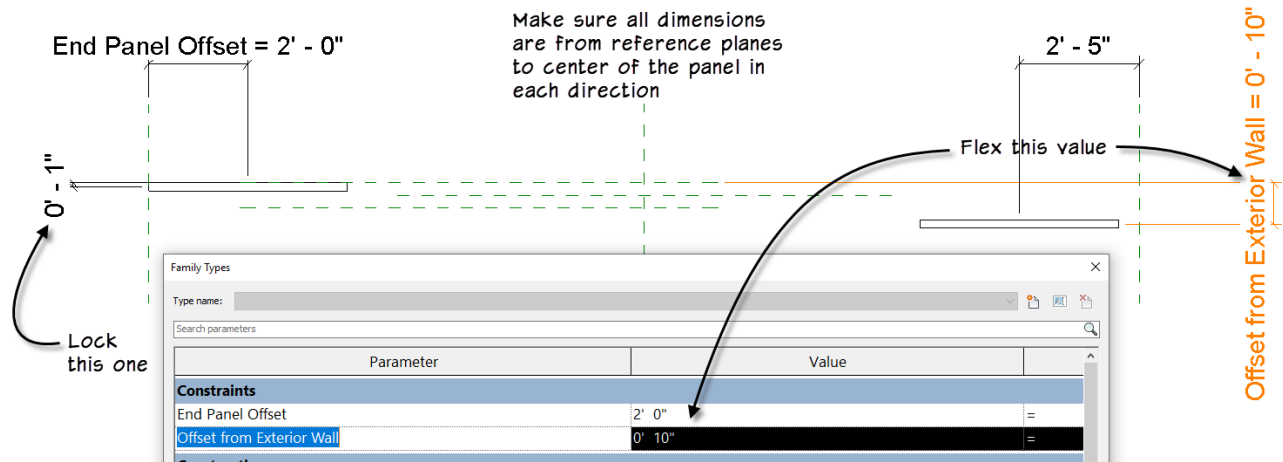


Figure 27 – Add dimensions and parameters to control positions of first and last panel

Make these two type-based parameters and place them in the Constraints grouping. The panel on the left is in the correct position, the one on the right still needs adjusting. Start by flexing the value of the new parameter. Change the value of **Offset from Exterior Wall Face** to: **10"**. This is just a temporary value for now. We'll assign the permanent value later. But for now we just want to be sure that the panel move in toward the interior side of the door. To set the position of the right panel horizontally, dismiss the "Family Types" dialog and assign the **End Panel Offset** parameter to the dimension on the right. This will move the panel and flush it against the reference plane on the right.

Select one of the panels. You will see a dimension containing the quantity of the array (currently 2). Select this dimension line. This is the array. On the Options Bar, its parameters will display. Uncheck the "Append to End" checkbox. Then click the Label drop-down and choose Add Parameter (see Figure 28).

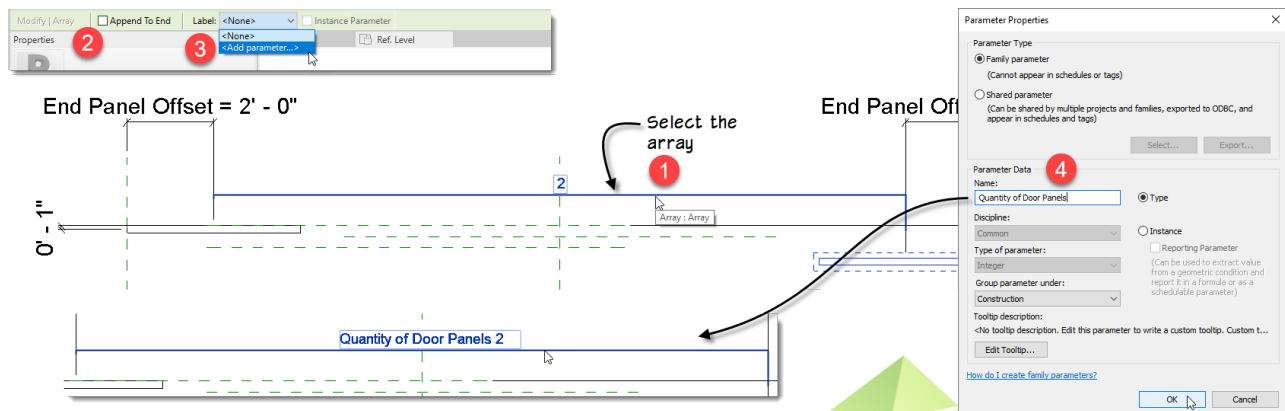


Figure 28 – Label the array dimension

Name it: **Quantity of Door Panels**, make it type-based and place it in the Construction grouping. Open "Family Types" and test it out. Try a quantity of 4 or 5 (see Figure 29).

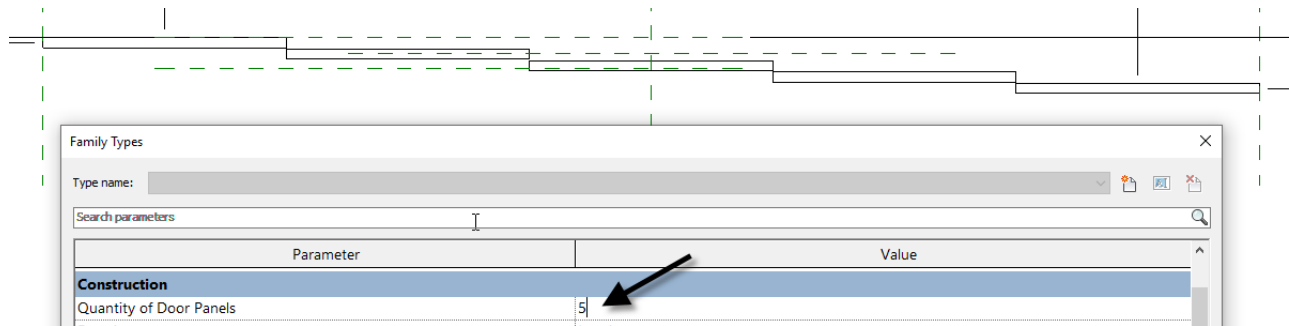


Figure 29 – Flex the quantity

While in “Family Types” let’s make a few more length parameters. In the Constraints grouping, add: **Actual Panel Width**. In the Dimensions grouping, add: **Maximum Panel Width** and **Panel Overlap**. Make one material parameter called: **Door Panel Material**. We’ll use this later. Make all four type-based.

Apply Parameters to the Door Panel

Now that we have the parameters required, we need to hook them up to the nested family geometry. The easiest way to do this is to locate the family on the Families branch of the Project Browser. Expand the Doors branch and then the door family. Right-click the type and choose: **Type Properties**. Use the Associate Family Parameter icon next to the Door Panel to assign the material parameter. Repeat for the Width, choose the **Actual Panel Width** parameter and assign the **Height** parameter to Height (see Figure 30).

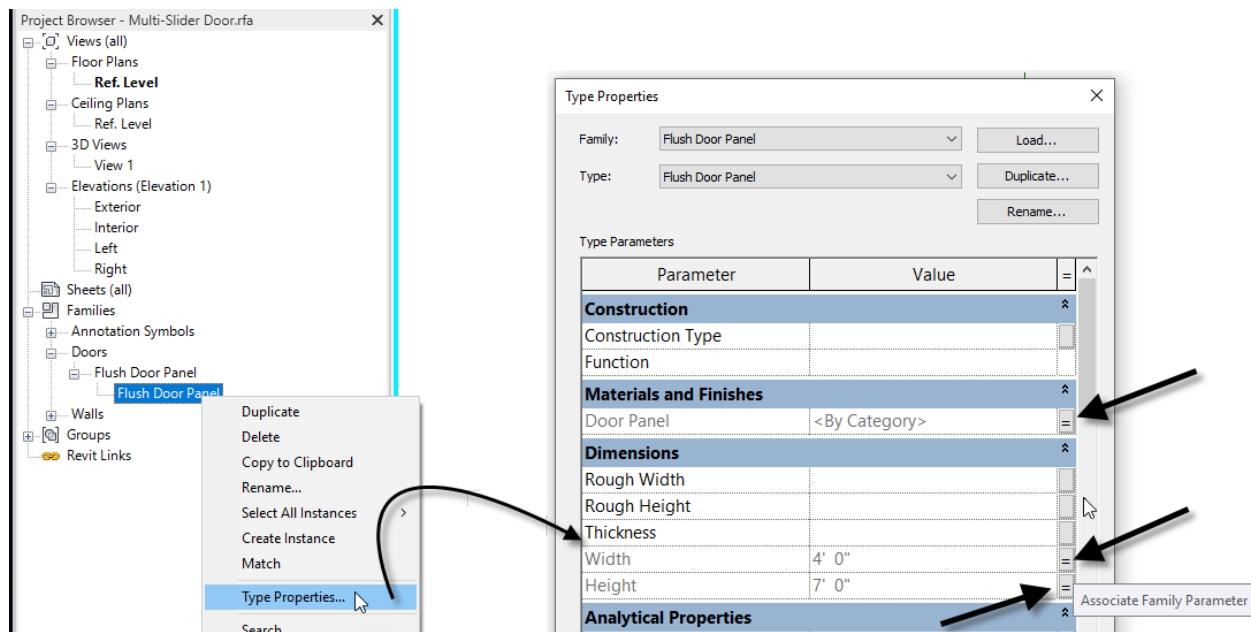


Figure 30 – Associate the parameters with the nested panel family

Configure the Formulas

We are now ready to put it all together with some formulas to ensure that everything flexes properly. For the **Maximum Panel Width** set it to: **4'-0"**, for the **Panel Overlap**, set it to: **2"**. These

are starting values. We can edit them later and the family will flex accordingly.

Here are the formulas to input:

Parameter	Formula
Actual Panel Width	$(\text{Width} + ((\text{Quantity of Door Panels} - 1) * \text{Panel Overlap})) / \text{Quantity of Door Panels}$
End Panel Offset	$\text{Actual Panel Width} / 2$
Offset from Exterior Wall	$(\text{Quantity of Door Panels} * 2") - 1"$
Quantity of Door Panels	$\text{if}((\text{Width} / \text{Maximum Panel Width}) < 2, 2, (\text{Width} / \text{Maximum Panel Width}))$

The order that you apply the formulas can matter. In this case, you can add them in the order shown in the table. Click Apply after each formula to see the results and make sure there are no errors. With each formula, you are adding more constraints to the behavior. Let's go through what each formula does:

Actual Panel Width—This parameter starts with the built-in Width parameter. The Width parameter is the size of the overall door opening. We could simply divide this by the number of panels. But then they would not overlap at all. Usually in a door like this, you will want them to overlap slightly. We created a parameter for the overlap which is currently set to 2". So, the next part of the formula takes the quantity of panels (currently 5) and subtracts 1. This is because we don't need overlap on the ends. So, if you look at how many panels need to overlap, it is quantity minus one. Take this new quantity (now 4) and multiply it by the **Panel Overlap** parameter. Finally, divide this number by the original quantity of panels. This makes the panels wider than they would be if not considering the overlap which allows them to overlap as required.

End Panel Offset—This one is simple. Since the dimension is to the centerline, we simply divide the Actual Panel Width calculated above by 2.

Offset from Exterior Wall—The panels stagger as they are duplicated. More panels mean this number needs to grow. However, since we opted to set the thickness of the panel to a fixed size of 2", we can simply build this directly into the formula. So just multiply the quantity by 2". But since the dimension goes to the centerline of the panel, we subtract 1" from the total.

Quantity of Door Panels—We are doing two things in this formula. One of the most common ways that an array can break in a family is if the quantity drops below the minimum; 2 for a linear array, 3 for a radial one. But we are also incorporating a maximum door panel size. So, we divide the overall Width by the Maximum Panel Width. Then we check this number to see if it is less than the minimum of 2. If it is, we force the number to 2 to keep the array from breaking. Otherwise, we simply allow the quantity calculated.

In some families that use arrays, you will require a condition where the quantity can be 1. To do that, you would need the array, plus a standalone element in the same place. You then add visibility parameters. When the array count goes below two, you force it to two as shown here, but then you also hide the array. And in its place, you display the single element instead. So, the formula would ensure that the array doesn't break and would control which one shows, the array

or the single element. But never both at the same time. We don't have to worry about that here because a door like this would never need fewer than two panels and would often have more than two. So we can skip that.

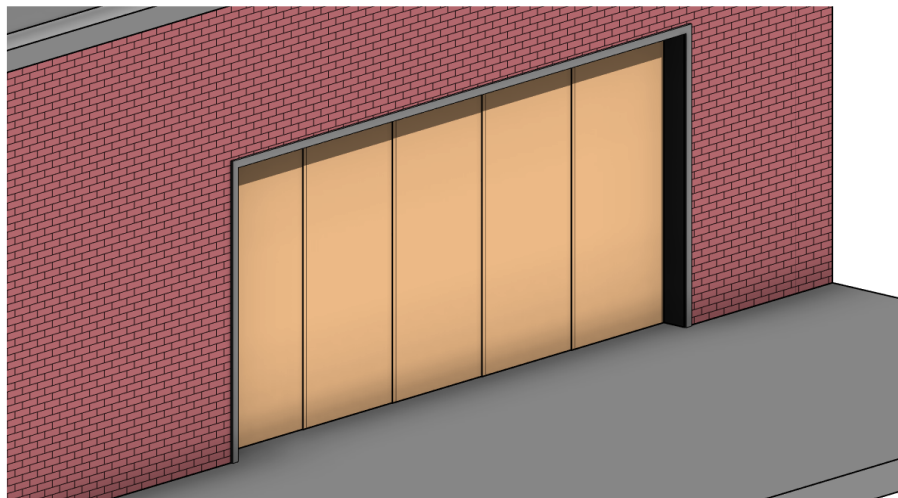
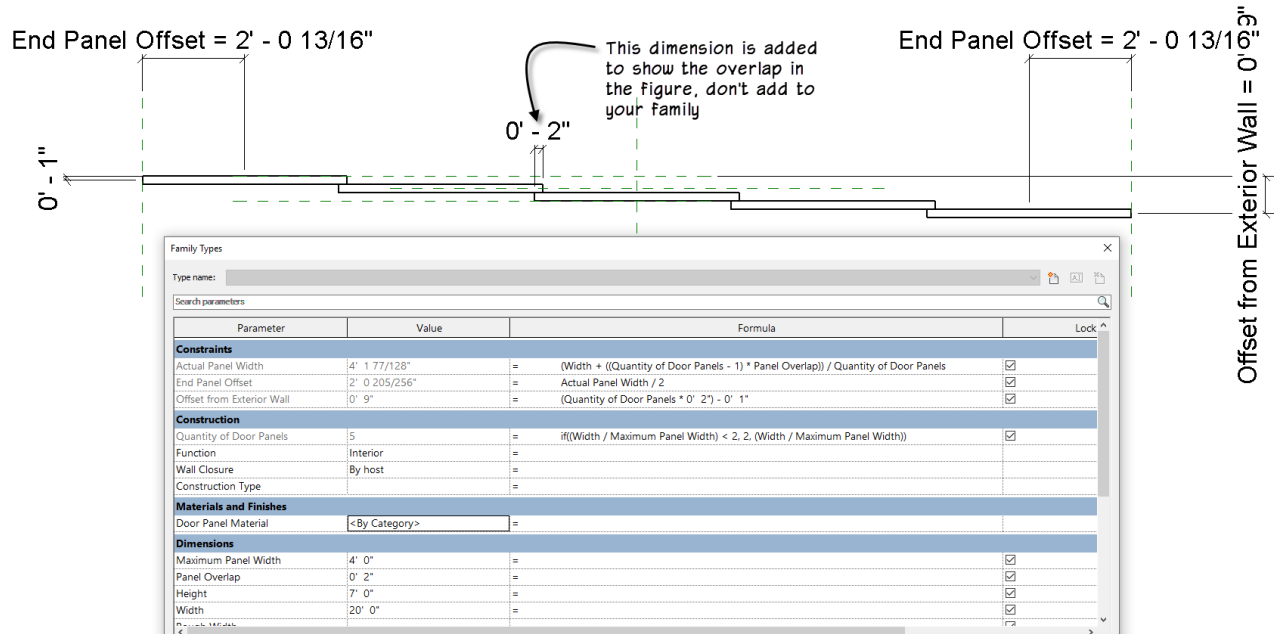


Figure 31 – The completed family

Naturally there are plenty more enhancements you could make. So feel free to experiment further.