MIDWEST UNIVERSITY 2018

Family Editor—Beyond the basics

411-Part 1

511-Part 2

Paul F. Aubin, www.paulaubin.com

Paul F. Aubin is best known as the author of many books and video training courses at lynda.com for Revit and other Autodesk tools. He has 28 years of experience in the Architectural industry and has worn many hats in that period, from designer, to CAD Manager, technologist, and trainer. He continues many of these in his current role as an independent Architectural consultant based in Chicago. Paul is the founder and host of ChiNamo; the Chicago Dynamo Community.

Class Description

Success with the family editor is so much more than creating simple objects with flexible dimensions. This two-part session will dispense with the basics and jump right into the deep end of the pool! I will share with you several real content creation examples that I have built for my clients over the years. Each will explore family creation concepts that go beyond the basic box or simple flexible family. If you want to up you family creation game, join me for this information packed two-part session.

Learning Objectives

- Understand how to make smooth 3D forms.
- Model complex forms both with native Revit geometry and leveraging imported mesh geometry.
- Use robust formulas to drive parametric values and incorporate invisible parameters to enhance functionality.
- Control nested families with family types parameters and custom drop-down lists.

Introduction

To get the fullest potential from Revit, you must be comfortable with the family editor and creating custom family content. There are so many techniques worthy of our attention. Naturally I had to be selective. I like each one of the techniques discussed herein for different reasons. While some of the techniques complement each other, many do not. Rather than attempt a continuous narrative, this session is just a pot luck of family editor goodness. So, sit back and enjoy. You are sure to find something useful for your next content creation project.

411—Family Editor—Beyond the basics, Part 1

In Part 1 we will explore some modeling techniques and look at different kinds of geometry. We'll discuss the use of voids and solids in the service of creating smooth geometry. Explore the choice or face-based, work plane-based and non-hosted families. Lighting fixture families offer lots of opportunities for innovation and we'll look at some of my favorite features to add to such families. We'll wrap up part one with a look at some highly specialized families created using both solid modeling and imported mesh modeling techniques.

Geometry

First let's review some of the basics: In the traditional family editor, we have five forms to work with: Extrusion, Blend, Revolve, Sweep and Swept Blend (See Figure 1).

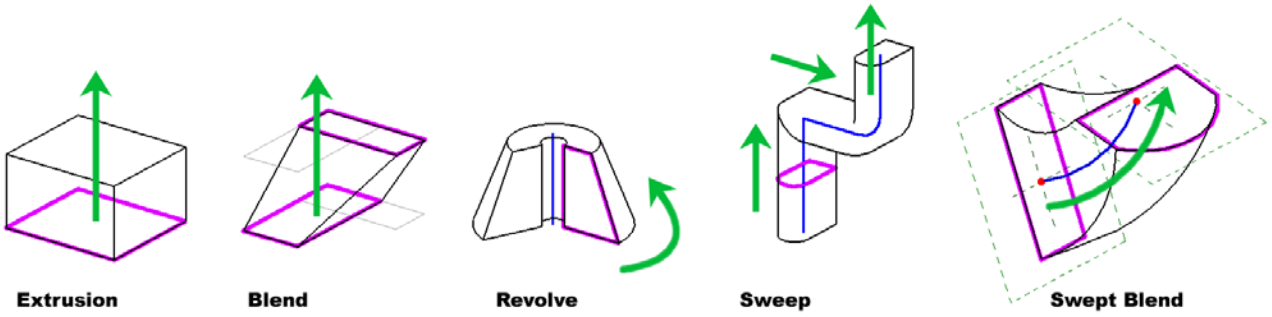


Figure 1—Forms in the traditional family editor

All five are sketch-based solids (or voids). But if you analyze them carefully, you find that any form you can make with an extrusion can also be created by a sweep with a single straight-line path. Likewise, a single straight-line path with two profiles in a swept blend can mimic anything created by a blend. And in most cases, you can mimic nearly any form created by a revolve using a sweep with a circular path. This means that if you are clever, you can create nearly all forms using sweeps and swept blends exclusively. Why would you want to do this? To leverage Profile families! The Sweep and Swept Blend are the only two that support the use of Profile families in place of the sketch. So, in situations where using a profile would be desirable over a sketch, you will want to use a sweep or swept blend instead of the other three. Using forms that support profiles can be advantageous for a few reasons:

- Profiles are separate families.
- You can build them independently of the other geometry in a family.
- This eliminates the possibility that the sketch will flex unexpectedly based on unintended relationships to other nearby forms (Automatic sketch dimensions).
- Profiles can be drawn as a single static form or can use parameters and formulas.
- Profiles can be reused in many families. They can be loaded and reloaded as needed in as many families as needed.

Many families I will be showing you today make use of sweeps or swept blends with profiles.

Pick Path

In addition to supporting profiles, both swept forms also support the “Pick Path” method of creating their path. This means that you can easily associate the shape of a sweep or swept blend with the edges of another form using the Pick Path method. This is an easy way to make a parametrically flexible path without requiring lots of parameters or constraints. Pick Path automatically follows the shape of the picked geometry and can follow a 2D or 3D path. Sketched paths can only be 2D (in a single work plane).

Making Smooth Corners

It is often desirable to make smoothly rounded corners on the geometry in a family. Unlike some 3D programs, Revit does not support filleting 3D geometry. Therefore, you will have to build your solids with smooth corners and/or use voids to “knock off” the sharp corners. The trick is making sure that the filleted edges are smooth without seams showing. To do this, you need to make sure that the fillet edge is tangent to any other nearby edges. Sometimes this is hard to do when relying on voids. Sometimes voids can hurt performance as well. So, I usually try to build the form with solids first and then use voids only where necessary. Figure 2 shows a good example:

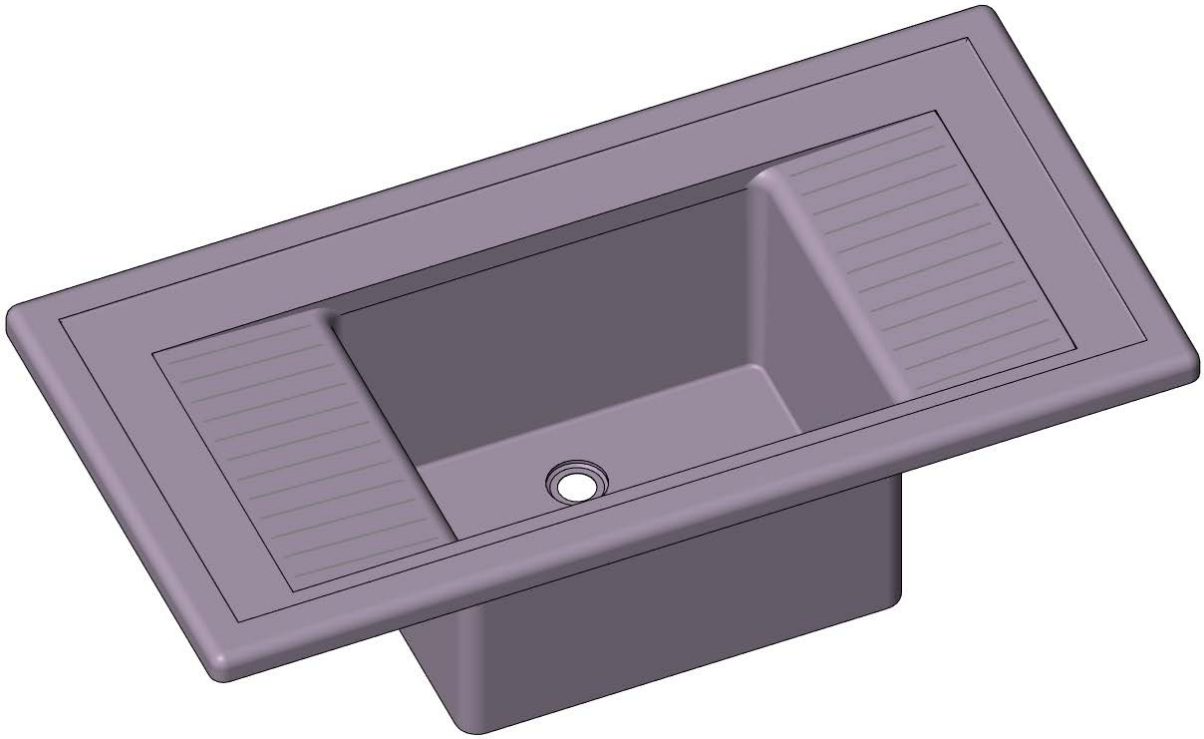


Figure 2—Object with smooth edges created from several joined forms

The outer edges of this molded countertop and sink use a sweep along a path that has a small radius at each corner. The challenge with this approach is that your sweep profile cannot be equal to or larger than the smallest radius on the path (see Figure 3). In my experience, allowing about 1/32" extra is usually enough to avoid the dreaded “Can't create sweep” error (see Figure 4).

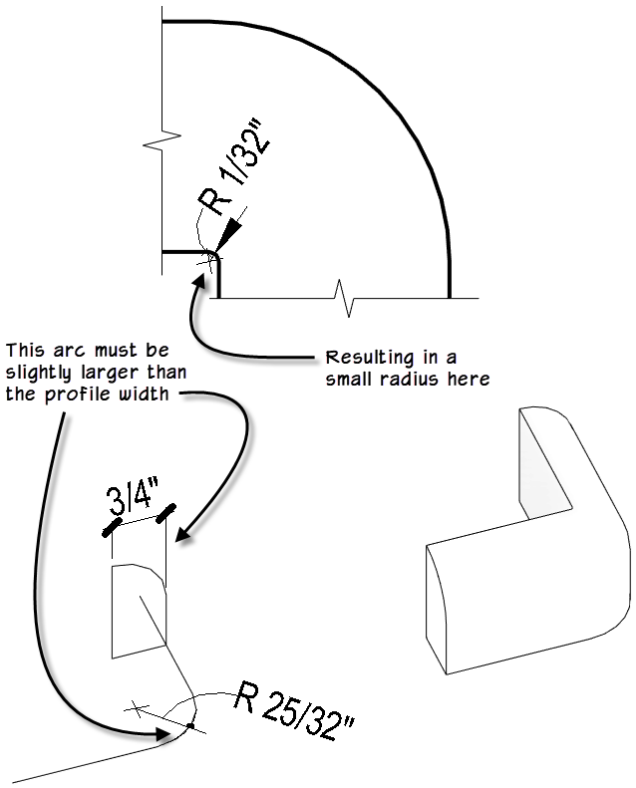


Figure 3—Profile cannot be larger than the smallest radius in the path

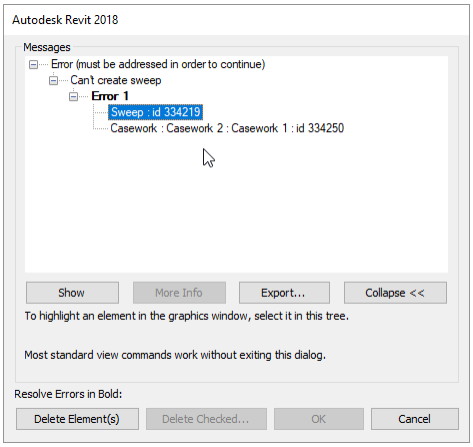


Figure 4—Errors while modelling are frustrating

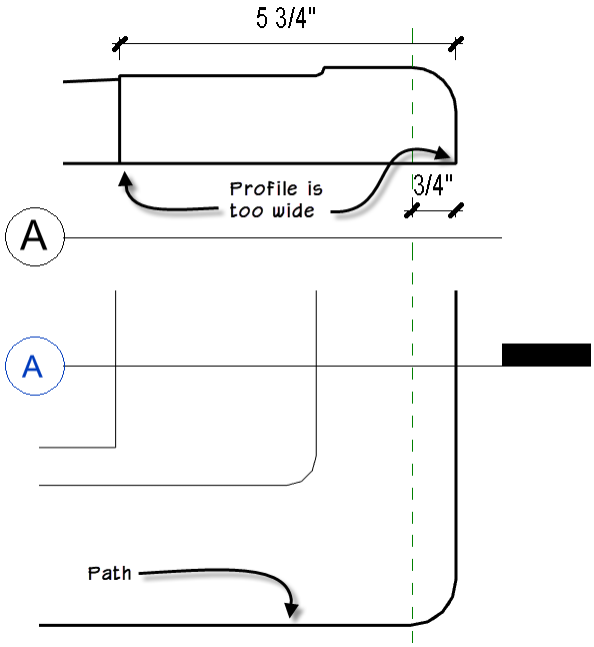


Figure 5—Sometimes a single profile must be broken into smaller pieces

So, if your profile is bigger than the radius of your path, you will have to break it up into smaller profiles and then create more than one solid element (see Figure 5). You can then use join geometry to put them all together. But this is often harder than it sounds…

In this example, the reference plane marks where the first profile is cut. There are several solids joined together to make the overall form (see Figure 6).

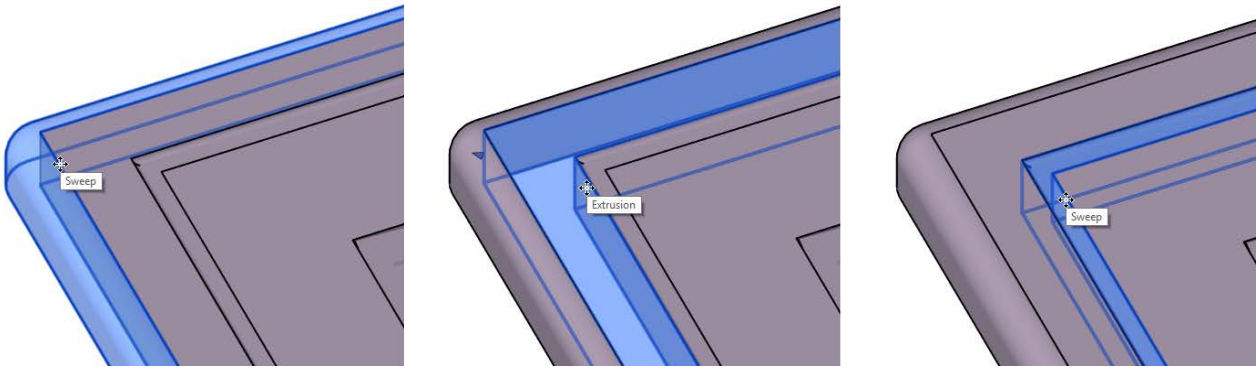


Figure 6—Create the final form from a series of solids (Seams show only while solids are selected)

To complete the inner portion, another sweep is used for the main shape of the sink bowl. However, this time, the path is on the inside edge meaning that even if the arcs on the path are very small, you can end up with a large solid form around the outside edge if the profile shape is large (see Figure 7).

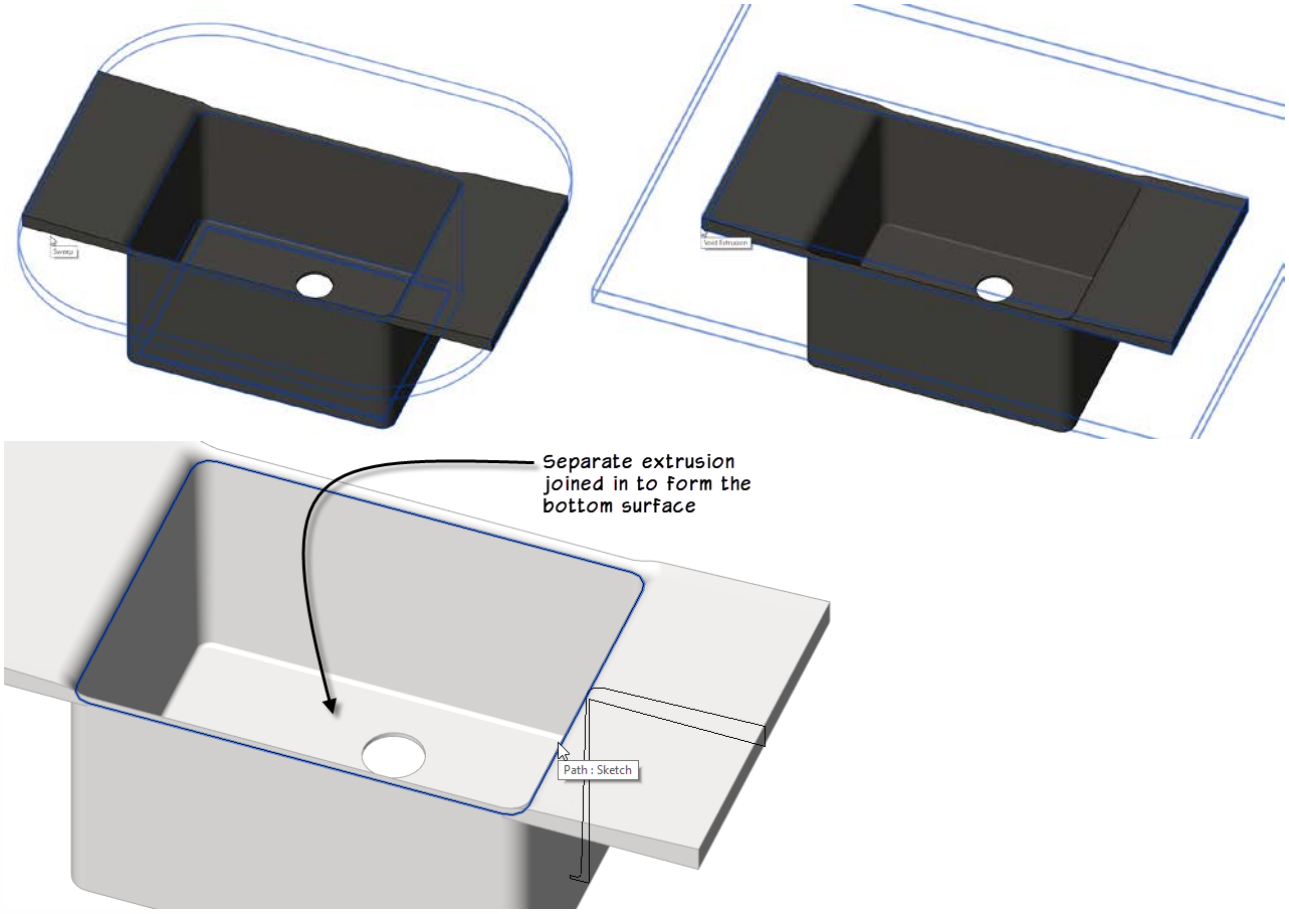


Figure 7—Create a sweep and then use a void to cut portions of it away

It took me a few attempts to successfully join the results from the bowl geometry with the apron around the sink. Join Geometry can be quite picky. If you cut the geometry very precisely where they meet and try to have simple forms at the points where they touch

(like a simple straight edges), then you will have a much better chance of joining the forms (see Figure 8).

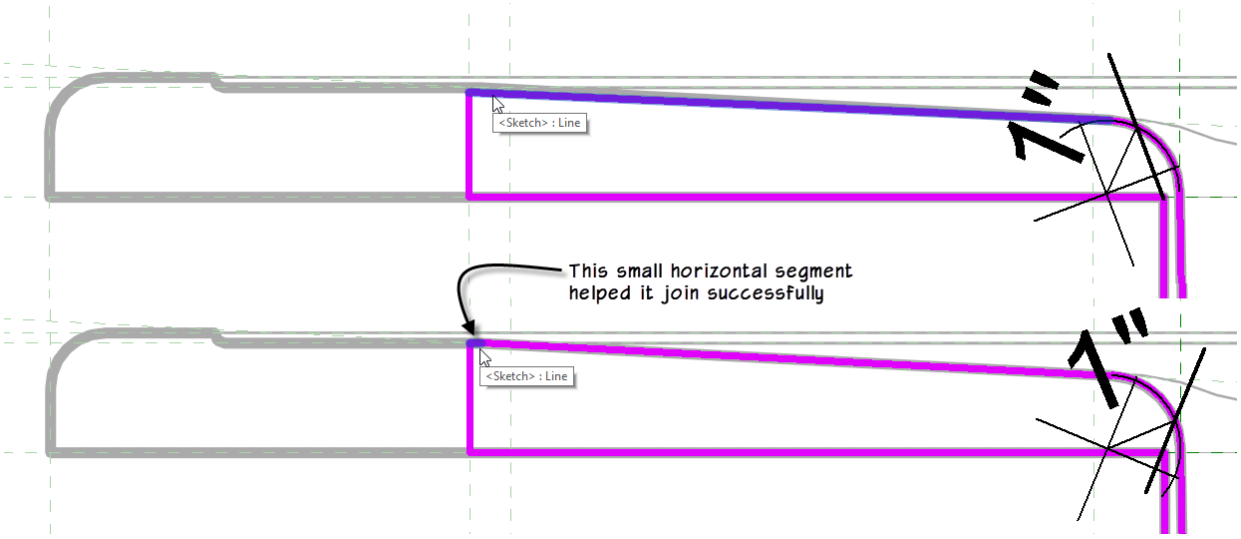


Figure 8—It takes some effort sometimes to get complex forms to join successfully

This cleanup sink family uses all solids. The smooth edges are achieved by ensuring that the fillet corners in the sketches are all tangent on both sides (the tangent lock helps maintain these). An alternative is to use voids to "knock off" the sharp edges. Here is an example piece of furniture that uses that approach (see Figure 9).

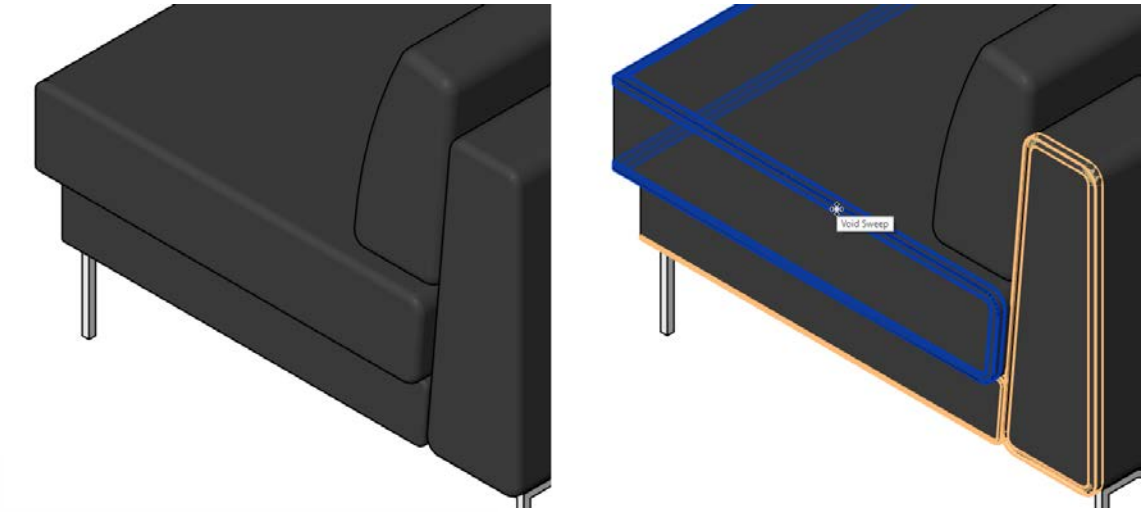


Figure 9—Sharp edges on solid forms can be rounded off using voids whose path is picked from the edges

Revit furniture is notorious for appearing hard and stiff. Softening the edges with this technique can help to make furniture that is more appealing and feels more natural.

Face-based and Work Plane-based Families

Face-based families are very popular. They have many useful features to be sure (hosting to a face instead of an object, they work through linked files, they offer flexibility in

orientation), however, they also come with some undesirable ones too (inflexibility in default orientation and icon previews being the biggest, but also issues with using embedded 2D graphics).

Consequently, I tend to limit my use of face-based families to those situations where:

1). I absolutely need a host, and
2). I need the host's category to be variable (like walls or columns) and or flexibility in physical orientation (like horizontal or vertical for example).

Otherwise, I tend to explore other options. This might include dedicated hosts (wall-based, ceiling-based), using work plane-based or simply leaving it un-hosted (example shown in the "Face-based or Freestanding" topic in Part 2 below).

Voids and Always Vertical

For this example, I have an item that uses both face-based and work plane-based in the same piece of content. It is a face-based sink element (that you place on the face of a countertop). There is an embedded void in the family that automatically engages and cuts the countertop. Meanwhile, there is also a nested work plane-based faucet family within it.

The difference between the face and work plane-based here is subtle.

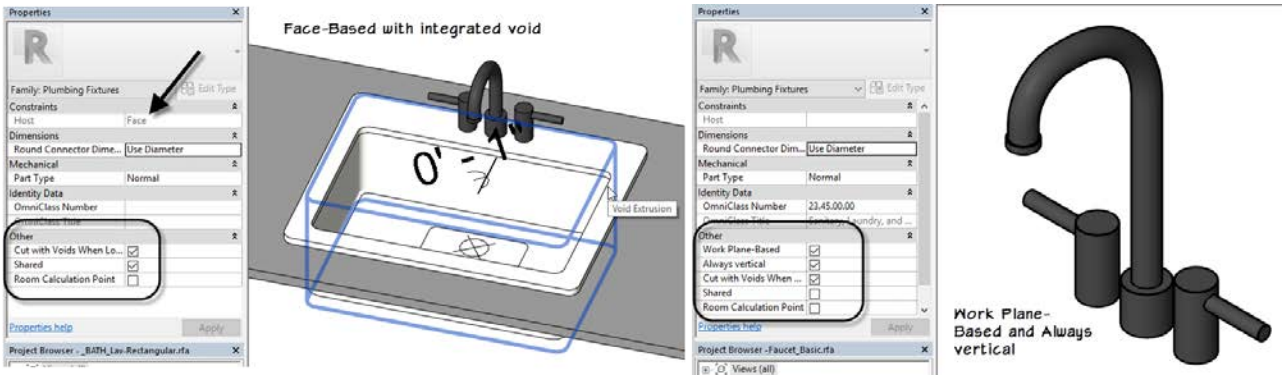


Figure 10—Combining face-based and work plane-based in the same piece of content

Face-based can be inserted on nearly any face in your model, including faces of linked geometry. That was not really needed here; a sink will always be inserted on a horizontal plane, but face-based families also support nested voids. So, if you add a void to the face-based family and use it to cut the face host inside the family, the family will attempt to automatically engage the void with the host object when you insert it in a project. For this to work, the face you insert it on must be a category that supports voids. This is typically cutable geometry (more on this in the "Nested Family Visibility (Cutable or non-cutable)" topic in Part 2 below) like system families, casework or columns for example. If you placed it on a piece of furniture, it would not cut the void into the furniture element.

Work Plane-based gives you behavior like face-based for placement but does *not* automatically engage voids. (You can add voids to them and use cut geometry separately in some cases however). Despite this apparent limitation, the nice thing about Work Plane-based is that you can enable it for nearly *any* category and you can couple it with the "Always vertical" checkbox. This makes the orientation of content that uses these

features much more predictable than an equivalent face-based item. So here I favored Work Plane-based for the faucet to gain control over the orientation and its vertical offset (from the floor) and Face-based for the sink to take advantage of the automatically applied void to cut the sink hole. So I do not use just one or the other. I think you should always seek to use the right tool for the job. Below in the "Face-based or Freestanding" topic in Part 2, we'll see an example where I chose to leave it un-hosted. All options have their usefulness under the correct circumstances.

Preview Orientation

Another common place for the use of face-based families is in ceiling mounted items like lighting fixtures. Face-based in this case is helpful in that if the host face (usually a ceiling) moves, the light or other ceiling mounted fixture will move accordingly. But, since it is not explicitly ceiling-based, it is flexible; allowing the host to be any surface such as a ceiling, a roof or even a reference plane. And even more importantly, this host face can be in a linked model! The biggest disadvantages when using face-based ceiling items are: the orientation of the icon preview on the type selector and the need to change the default placement option (on the ribbon) each time you run the component command and choose a face-based ceiling element (see Figure 11).

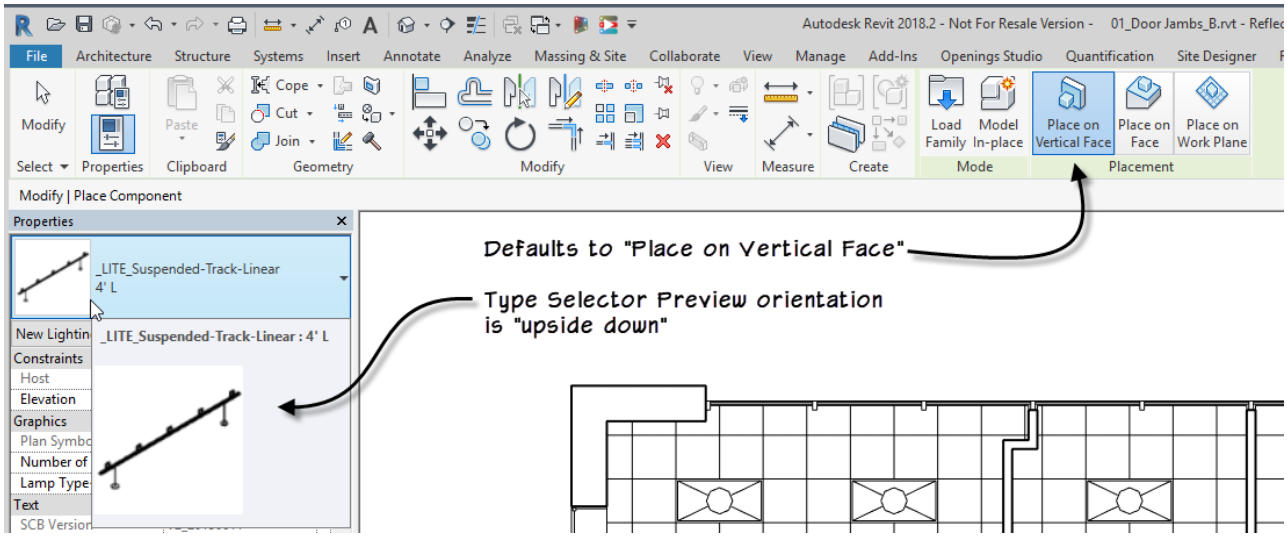


Figure 11—Using Face-based Ceiling Families

Otherwise, when placing a face-based ceiling item in reflected ceiling plans and using the "Place on Face" option, you will usually get good results. However, there is one significant exception to this. If you use the Basic Ceiling rather than the Compound Ceiling family, the face orientation will point up instead of down. This means that your face-base lighting fixtures will be point the wrong way! (see Figure 12).

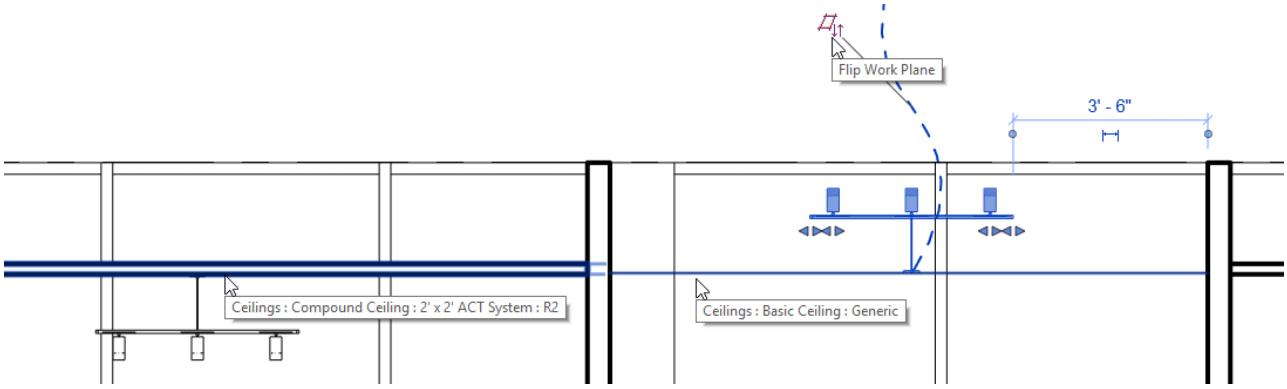


Figure 12—Basic Ceilings do not work well with face-based content

In such cases, you can flip the work plane by selecting the component and then clicking the Flip Work Plane control that appears. But you'll have to do this for each item you place.

These are minor inconveniences in most cases, so typically the pros of using a face-based family for ceiling content outweigh the cons. And while we cannot change these default behaviors, we can improve the preview orientation a bit; at least the one that shows when opening the family or in the load family dialog. (The preview shown on the type selector cannot be edited sadly).

The easiest way to adjust the preview is to orbit the 3D view to the desired orientation using standard view navigation techniques. In that 3D view, right-click the ViewCube and choose: **Save View**. (This might prompt you to rename the 3D view). Then save as the family. In the save as dialog, click the Options button. Choose the modified 3D view as the thumbnail preview source, check the "Regenerate if view/sheet is not up-to-date" checkbox and then click OK (see Figure 13).

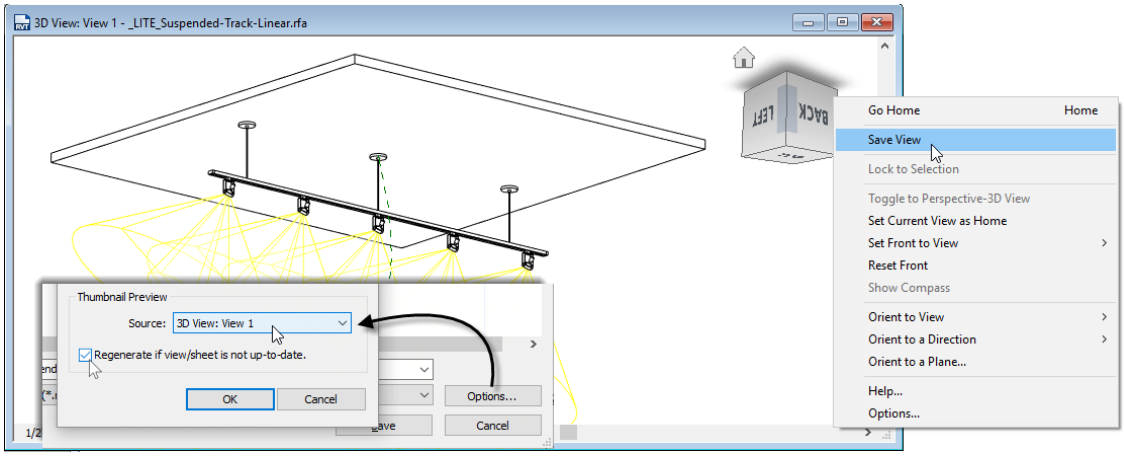


Figure 13—Customize the preview

You will see this preview in the open and load family dialogs.

Lighting Families

On the subject of lighting families, I have several other features I usually like to incorporate. We have already discussed using Face-based. Here are some of the others:

2D Symbols

If you want to add symbolic geometry to your lighting or other face-based family, you will need to consider the orientation carefully. For items that will be placed on the ceiling, the orientation of the 2D symbolic elements can be placed in the Ref Level plan view of the family without issue. This is because they will remain parallel to the ceiling plan when inserted. However, for wall mounted fixtures like wall sconces, the Ref Level plan will be parallel to the wall face instead. This means that to get 2D graphics to show in ceiling plans, you will have to add them to one of the elevation views (usually the front or back) instead (see Figure 14).

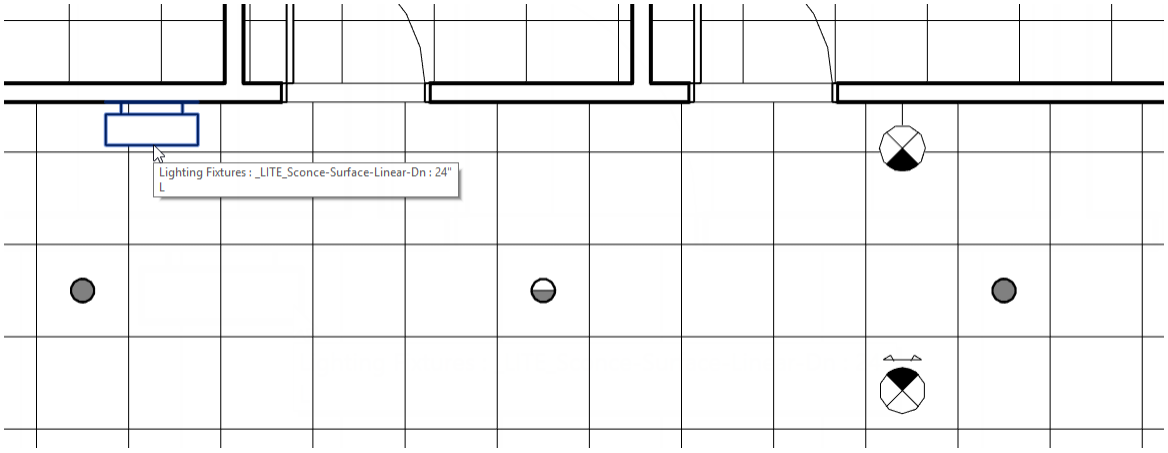


Figure 14—Vertically oriented (wall mounted) light fixtures with embedded 2D graphics

Grid Modules

Acoustical tile ceilings are very common. To make it easier to place light fixtures on these ceilings, you can add a pair of reference planes in each direction and drive them with grid size parameters. Set these parameters to match the size of the ceiling grid (i.e. 2x2 or 2x4). Then you can use the align tool to easily center fixtures on the grid modules (see Figure 15).

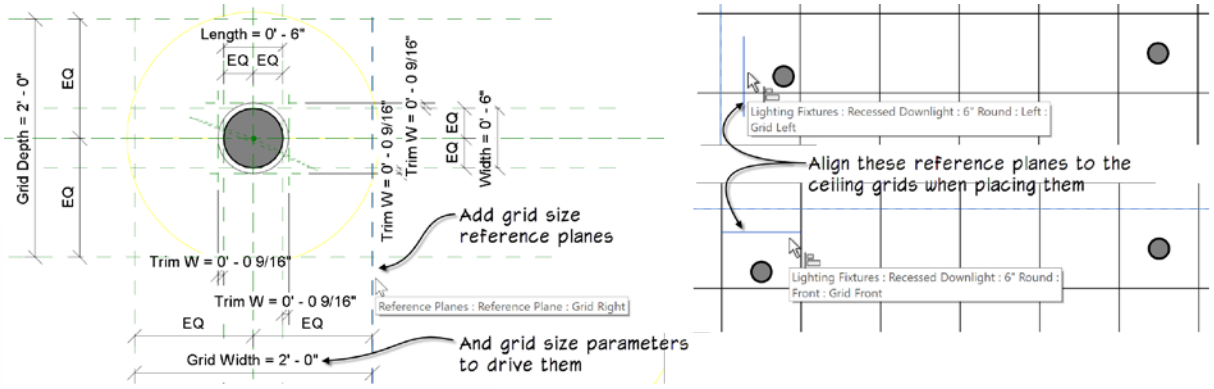


Figure 15—Add grid module reference planes and parameters for ceiling grid centering

Optional Void Cutting Ceiling

Many hosted ceiling elements have voids integrated into them. This allows the fixture to be recessed into the ceiling plane. This can have an impact on performance in large files (see Figure 16).

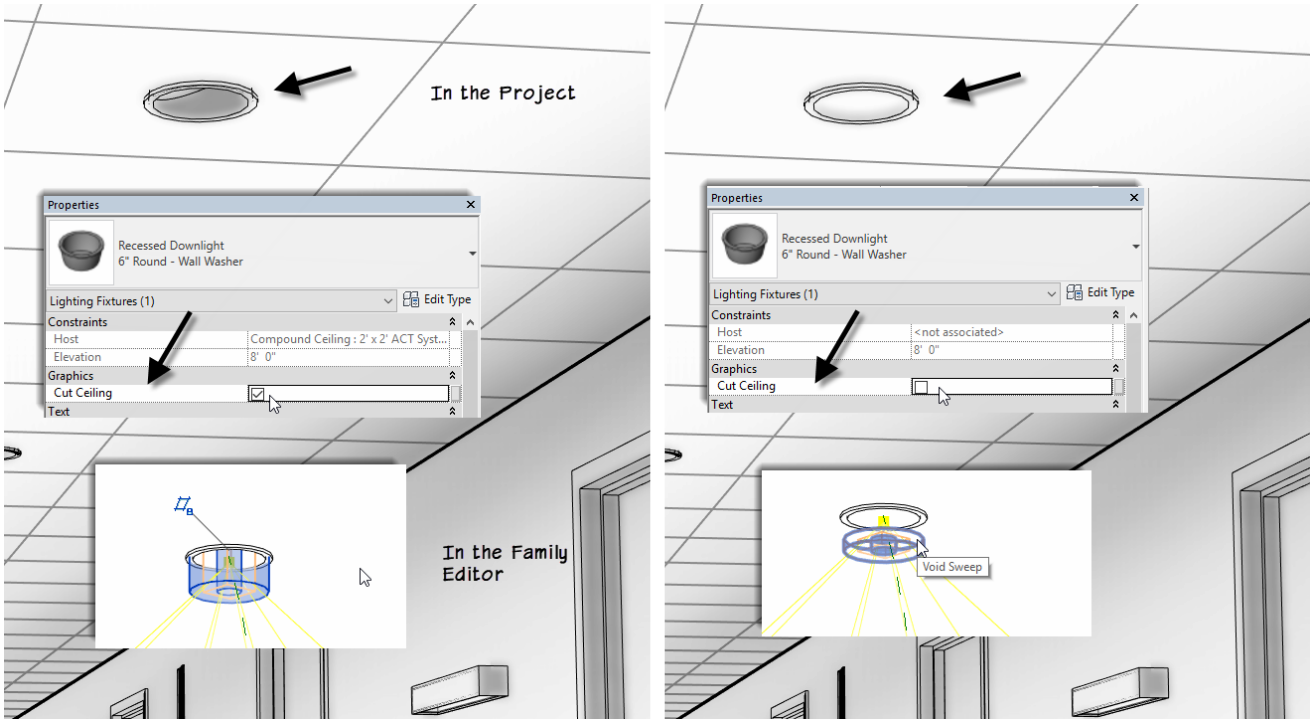


Figure 16—Create a void that moves away from the surface when "Cut Ceiling" is unchecked

To deal with this, you can use a Yes/No parameter to drive the size of the void element. When checked, the void will be big enough to engage the surface host. When unchecked, it will move away from the face and disengage[1].

Version Identifier

Sometimes it can be helpful to have parameters with "fixed" values. You might do this if you are using a constant value in a formula. Or if you want to add fixed manufacturer information. In many of my families that I build for clients, I like to add a "Version ID" parameter. This is just a text field that I input a value to keep track of what version of the family is being used. If you issue a version of the family to the team, and then later update this family, this version ID can be a useful way to tell if the team is using the latest version. Simply select the family and check the ID on Properties.

To make this ID "read only," add the version ID in quotes in the formula field in the family editor. This way, it will show up in the project, but will not be editable. You can use this trick for any parameter that requires a fixed value (see Figure 17).

[1] Unfortunately, this does not always work as expected. Once you uncheck the void, it will not reengage with the surface when rechecking the box. So, it works well to disengage the void, but not as well to reengage it.

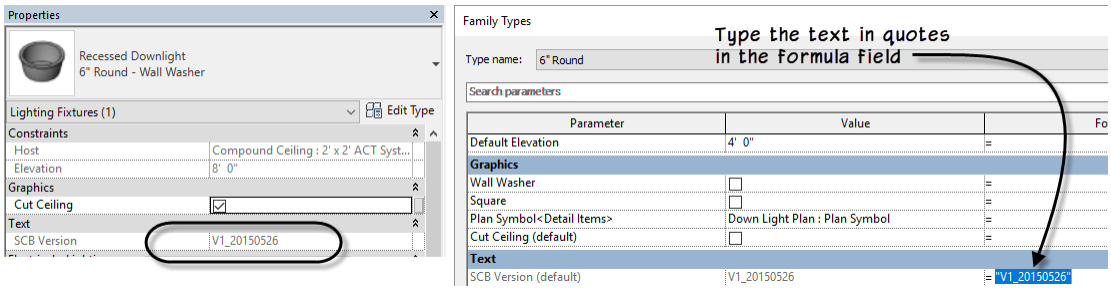


Figure 17—Create a "read only" text value for a version ID

Square or Round?

Sweeps have some unique features. One of my favorites is Trajectory Segmentation. I am always pleased with myself when I can utilize it. When toggled on, this feature affects the path of your sweep. Specifically, it affects the curved segments of the path and renders them in segments instead of a smooth curve. It is listed in the help as an MEP feature. They show an example like the one at the top of Figure 18. While this is certainly a valid use for it, what I find most exciting about the feature is that it can controlled by parameters!

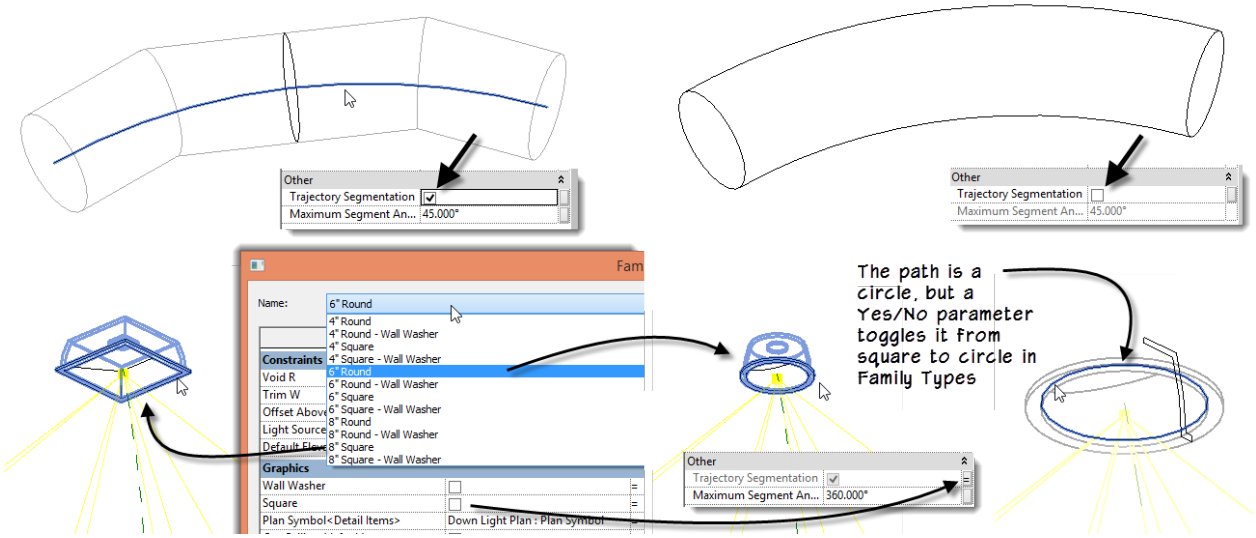


Figure 18—Trajectory segmentation can be set manually or with a parameter

Looking at the bottom of the figure, I am showing an example where I used this feature to create a recessed downlight fixture. The same fixture can flex to different sizes and using trajectory segmentation, can be either round or square. The path for the sweep is a circle (shown at the right). Then the trajectory segmentation feature is linked to a family parameter (Yes/No) that you can toggle on or off. When it is on, we get a square. When it is off it is circular. Because the path is a circle, these are the only two options. If you use an arc for your path, like the example, at the top of the figure, you can also change the Maximum segmentation angle setting to get more or fewer segments along the curve. Smaller numbers give more segments.

Wall Washer

The light source included in lighting fixture families has angle parameters that can be

driven by parameters. This makes it easy to create a fixture that can toggle from a normal spotlight to a wall washer. Just add a checkbox (Yes/No) parameter to toggle Wall Washer on and off. Then use this in an IF formula to control the angle of the Tilt Angle built-in parameter (see Figure 19).

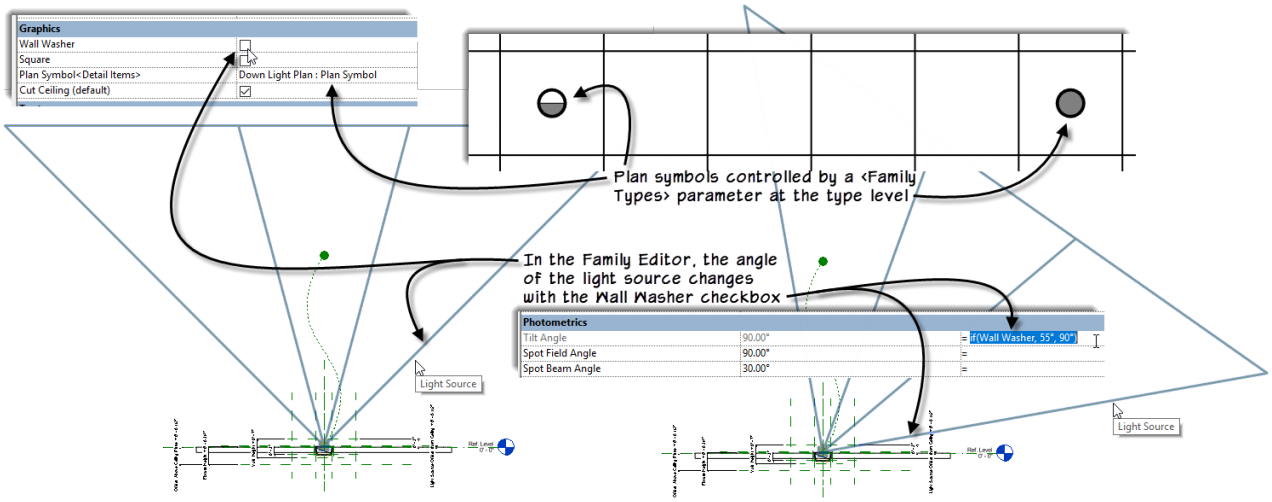


Figure 19—Use a Yes/No parameter combined with an IF formula to drive the angle of the light fixture

Two things to note about this example: First, notice that the figure shows the Front elevation of the family and its orientation is “upside down” due its being a face-based family. Notice that the ceiling is at the bottom; so, when you define your geometry and behaviors, you have to take this into account. Second, the symbol used in the reflected ceiling plan views is a 2D Detail Item family nested into this lighting fixture family. It is possible to setup additional parameters and formulas to make the correct symbol toggle automatically based on the Yes/No checkbox. (I will show an example of this in the “Make a “List” Parameter” topic below in Part 2). However, in this case, since all of this is type-based, I thought it better to manually designate each choice at the type level instead of doing it formulaically. Sometimes, we can get carried away with the formulas and “smarts” to the detriment of the overall experience. Once the family types are defined and saved, there is little reason for them to change. Therefore, there is little risk involved in setting the <Family Types> parameter value manually for an example like this.

Complex Specialized Models

Last year I participated in a reality capture workshop in Volterra Italy. (I will be returning next month for a second one!) In the workshop, we used laser scanners and photogrammetry to capture several buildings, historical sites and artifacts. From this we came away with nearly a terabyte of point clouds and raw data. Over the intervening months, I have been slowly building Revit models (when time permits) of some of the sites using the point cloud data (see Figure 20 and Figure 21).

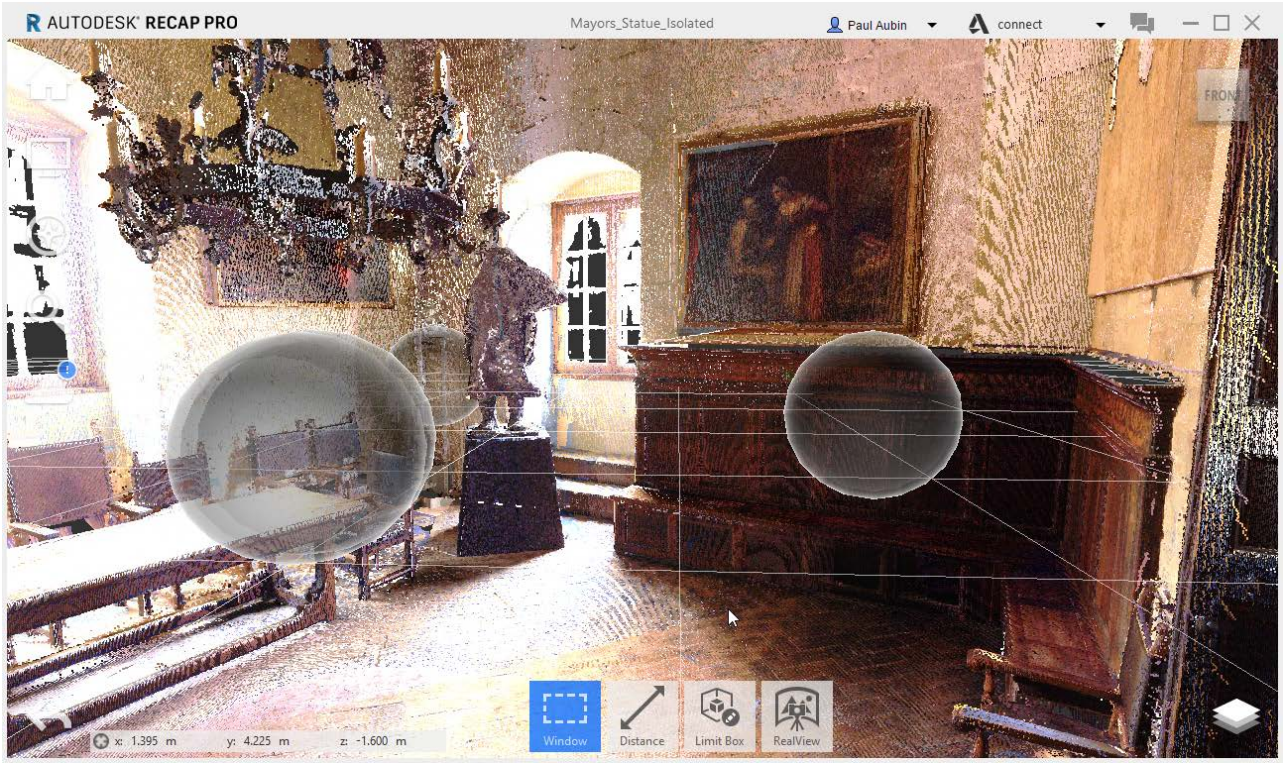


Figure 20—Point cloud of the Volterra City Hall; Mayor's Office

There are two basic approaches that I have been taking to achieve this:

- Linking in the Point Cloud and tracing over it with Revit geometry.
- Creating mesh models from the point cloud and using them directly in Revit.

Figure 21—Many custom families in the Mayor of Volterra's office

Point Clouds in Revit

When working directly on the point cloud in Revit, you must link the point cloud into a project file. Revit does not support point clouds in the family editor. This is unfortunate since there are many times when it would be useful to be able to do so. To overcome this limitation, I have used two techniques:

- Link a point cloud into a temporary project, use it to take measurements and build the family separately with the data captured from the measurements.
- Link a point cloud into a temporary project, create an in-place family and then build relative to the point cloud. In-place geometry can then be copied and pasted into a component family while in-place edit mode is active.

Both methods can be effective, but they also both have challenges as well. I tend to rely on the first method more often. I will often take measurements and then physically print out views of the model with these dimensions to help me model separately in the family editor (see Figure 22). While perhaps not very "green," it helps to have them printed. Minimizes the amount of back and forth between multiple screens and views.

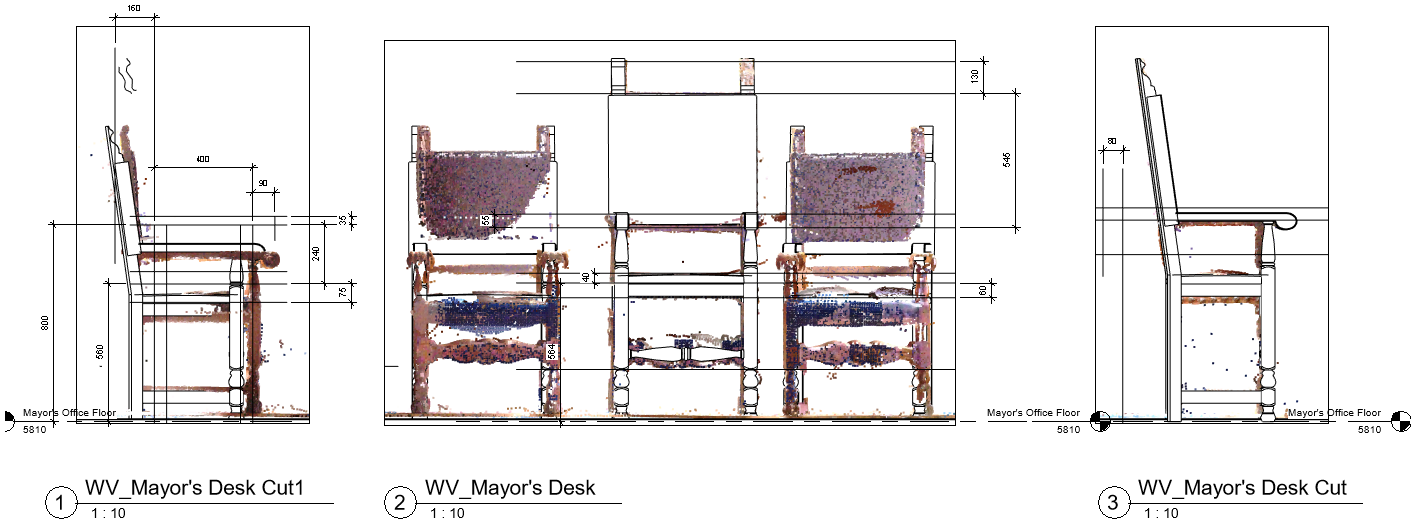


Figure 22—Add dimensions on working views and use to assist in modeling

The figure shows a flexible family superimposed over the point cloud that was measured and printed out to help build it. These measured views are used in conjunction with photographs of the same items. This method allows for good results. There are places where compromises must be made. Particularly in areas where the point cloud might be missing information or distorted as sometimes happens if the line of sight was compromised during scanning.

Chandelier

The chandelier in the mayor's office presented a nice modeling challenge. It was complex, had some organic detailing combined with regular geometry and it had to be a light fixture. The resulting family uses simple modeling techniques and instead relies on nested families to create the complexity. The parent family is a lighting fixture (Category), but if you edit it, you will find that the Light Source option is turned off. If you want a light fixture

with several separate lamps, create nested families for the individual lamps instead. Be sure to check the Light Source checkbox in the nested families instead of the parent family. But if you want these nested light fixtures to function as light fixtures in the host project, they *must* also be set to “Shared” families (see Figure 23).

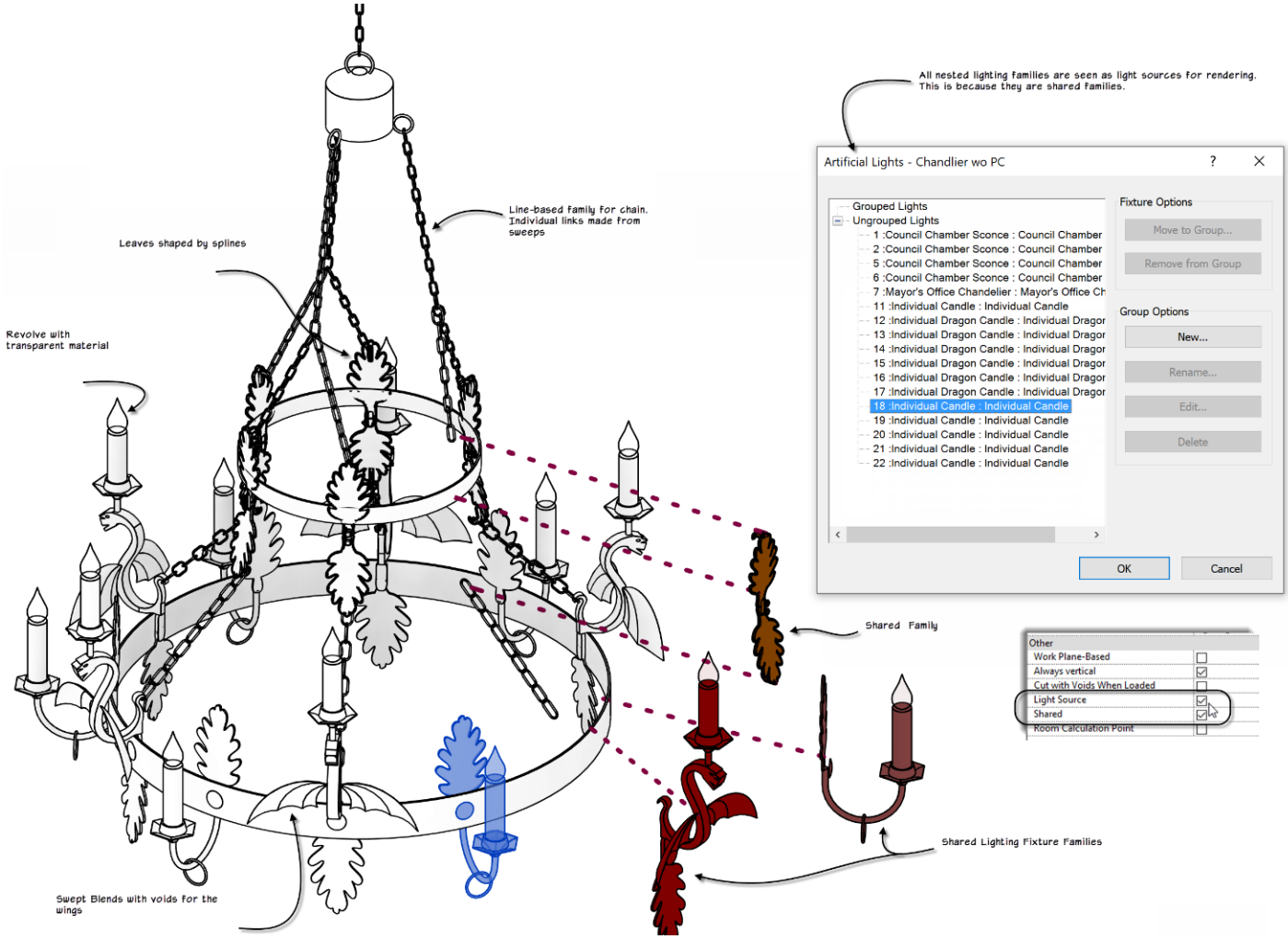


Figure 23—Chandelier family

There are a few other points of note in this family. The chains are line-based families with a nested parametric array to repeat the chain links as the family stretches to longer and shorter sizes. To draw them at the proper 3D location and orientation, create work planes with reference planes and/or reference lines.

Work planes are important to many of the other details as well: the revolve used for the flame-shaped light bulbs (made from a revolve), the dragon wings (made from swept blends and voids), the leaves (made from extrusions using splines for the freeform edges), and finally each layer of the chandelier are built on reference planes defining each level.

Trajectory segmentation shows up again in the nested light fixtures to create the small hexagonal base of the candles. Voids are kept to a minimum but are utilized for the facial features of the dragons and as noted already to help sculpt the shape of their wings.

Imported Geometry

There are many organic forms in the Volterra dataset. There are plenty of statues and sculptures and there are some flags and other fabric items like tablecloths and draperies. If you want the model to look authentic and realistic, you need an efficient way to deal with such items. For the backs of the chairs and the tablecloths in the council chamber, I relied on sweeps and swept blends whose profiles and paths rely heavily on splines to give them their organic forms. This approach can work well; but can also be quite time consuming.

There are two alternatives:

Use the massing environment. (The flags in the mayor's office are built this way) or import mesh geometry directly into the family editor.

To comment briefly on the flags, the main feature that is utilized to create their form is the support for lofted forms in the massing family editor. You can select two or more shapes (open or closed) and create a smooth transition between each of them. If you use closed shapes, you get a solid, and open shapes will make a 3D surface. The flags use open spline shapes and make a free-flowing surface.

It is possible to use the massing environment to create nearly any organic 3D form (don't believe me, check out: http://paulaubin.com/books/renaissance-revit/), solid or surface. So, this would certainly be possible for the statues as well. But you will often pay a heavy cost in file size and performance for these forms and not all categories are supported. To say nothing of the steep learning curve associated with the mass environment or how radically it deviates from the traditional user interface. And sometimes the time and effort to do it, even if you have the expertise, can be hard to justify.

For these reasons, in the Volterra dataset, I took a different approach for the sculptures. I imported mesh models directly into the family editor! Now, as you may well be aware, there is a common rule of thumb in the family content creation world that says: "**do not use CAD imports**". Well, in this topic, I am going to be breaking this rule!

Revit creates solid geometry, and the results from it are often quite nice, but given the simplicity of the solid modeling tools in the traditional family editor, it is often quite challenging to create complex or smooth organic forms. Sometimes CAD is all you have. Or you might have mesh models created in other software; or in the case I am discussing here, you might have point clouds. Such cases are where mesh models provide a viable and compelling alternative.

The approach is a bit convoluted and involves working outside of Revit in 3ds max for some of the steps. However, since it is included with the AEC collection that most firms have, this is not a huge obstacle for many.

Here is the summary of the process:

Get a mesh

The first thing you need is a mesh. You can get these from a variety of sources. I will cover two possibilities here. The first will be creating a mesh from a point cloud. The

second will be using an existing mesh file in another file format.

Make the edges of the Mesh invisible

If you have ever tried importing a mesh into Revit, you were probably less than satisfied with the results. Usually, all the tessellation of the mesh will show up in Revit making it graphically unappealing for most drawing types (see the left side of Figure 24).

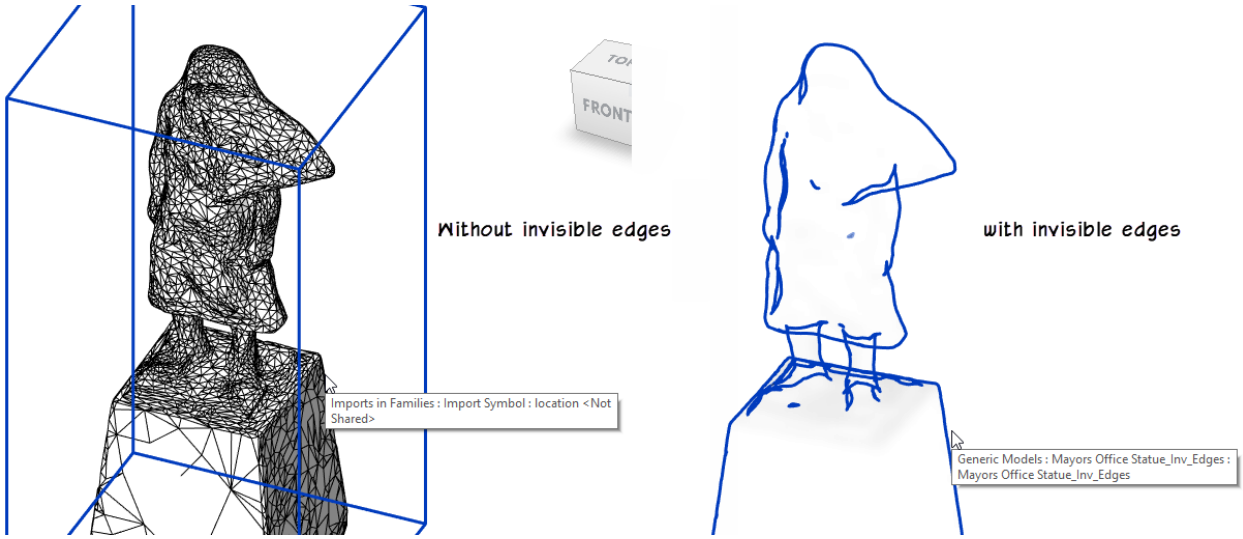


Figure 24—Mesh models show all their facets when inserted into Revit

The solution to this is to process the mesh in 3ds max first to hide the edges of the mesh. The mesh can be saved with the edges invisible and then brought into Revit yielding much nicer results (see the right side of Figure 24).

Import into Revit

Once you have a processed mesh with invisible edges, import this into a Revit family. Perform a few cleanup steps in Revit and optionally add any view-specific 2D graphics. Save the family and use it in your projects.

Detailed Procedure to Process and Import Models

Let's dig a little deeper into the process. As noted above, you first need a mesh. This can be something you create from a point cloud, create in other 3D modeling software or download from the Internet. Process will vary slightly depending on the mesh model's source. For the statue models in the Volterra dataset, I created them directly from the point cloud using the mesh service from Autodesk that is part of ReCap Pro. If you do not have access to ReCap Pro, similar functions may be available in other point cloud processing software.

ReCap Workflow

I want to quickly create a family from one of the sculptures. The first step is to isolate that portion of the point cloud in ReCap Pro. Use a Limit Box to do this (see Figure 25).

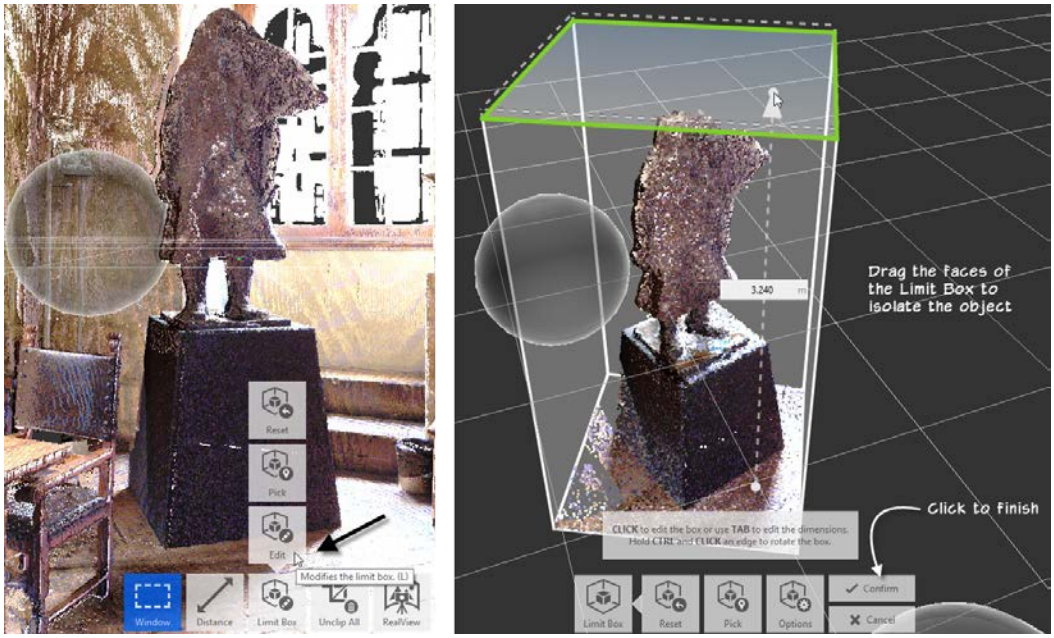


Figure 25—Isolate the object with a Limit Box

You can optionally isolate collections of points to focus on just the object and remove anything unneeded. This will involve selecting points that you want to isolate and moving them to a named group to make them easier to work with. To make the selection easier, you can adjust the point display. In this case the Intensity display mode offers nice contrast. Then use a convenient selection method such as Fence to select unneeded points. You can clip these points to hide them (see Figure 26). Repeat as required to leave just the points you need visible.

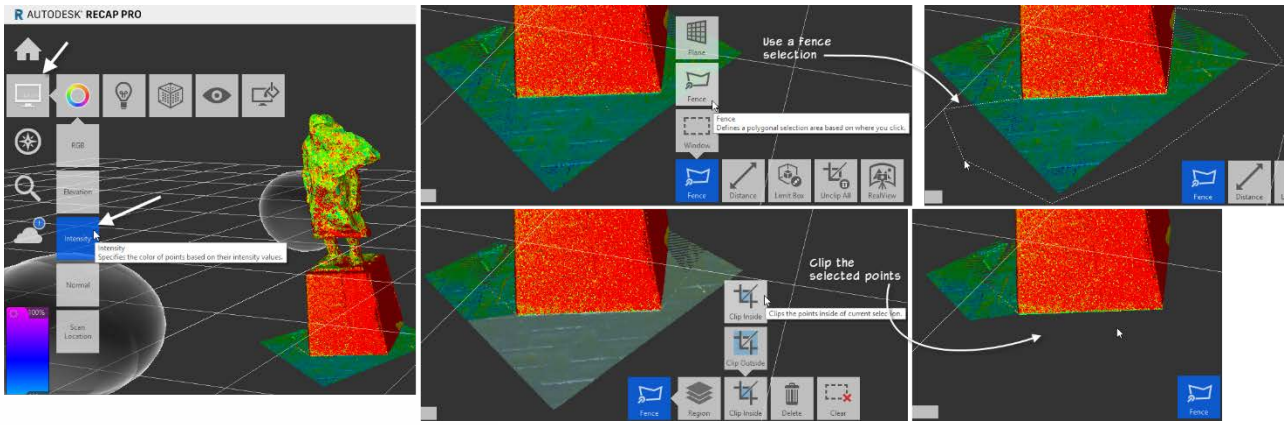


Figure 26—Use different display modes to assist in selection

There are other handy tools. You can select all the points of the statue and then create a “Scan Region” from them (like a layer). This will make it easier to return to this collection of points later. You can also create a “View State” of the current view. This will allow you to restore the selection and limit box later and zoom right to this portion of the model. None of this is required to create a mesh, but it will prove useful to have these later if you need to repeat any of the steps (see Figure 27).

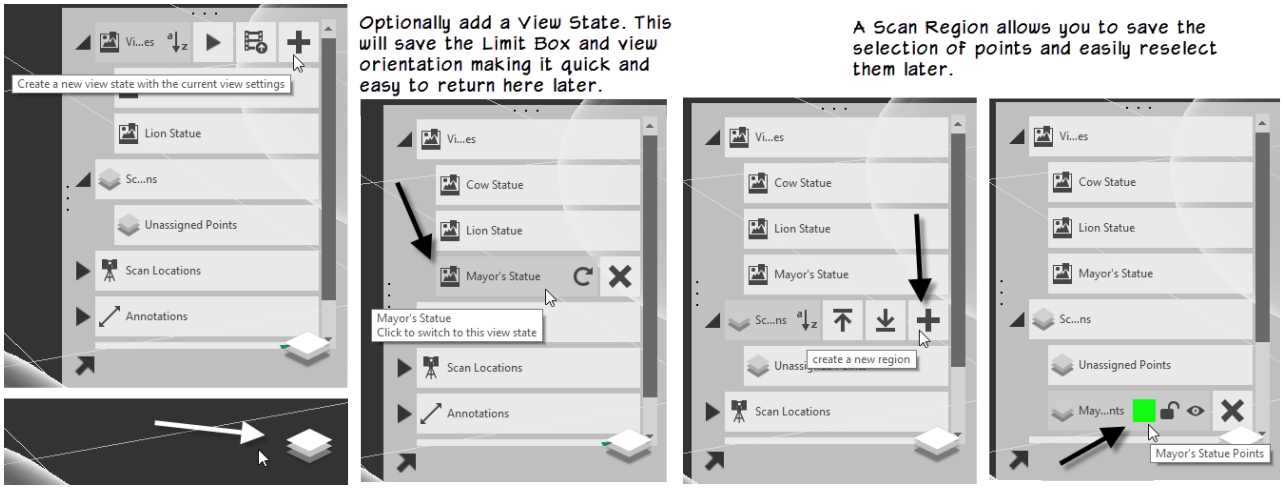


Figure 27—Save your work with View States and Scan Regions making it easy to return later

With the statue isolated, you are ready to convert it to a mesh. Use the cloud-based service for this. When you first choose Mesh from this flyout and click Start, it will prompt you to upload your project. This might take some time (sometimes a very long time). Once it is finished uploading, you will see the dialog on the right side of Figure 28. Give your mesh a name, choose the "High" quality[2] option and select all output formats. Click Submit. When it is finished, you will get an email indicating how to access the files.

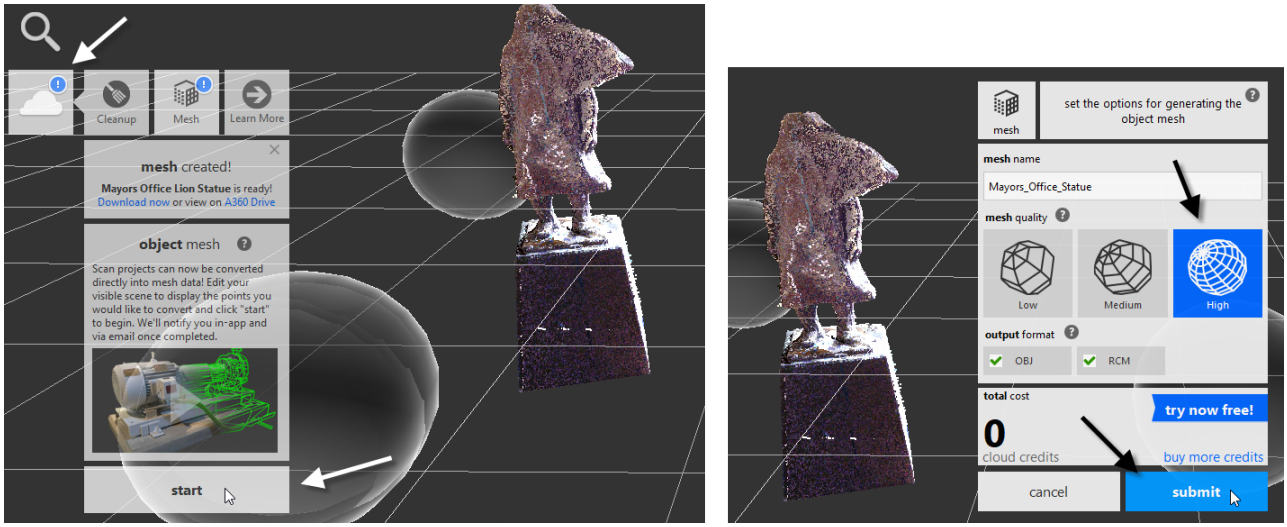


Figure 28—Point cloud with a sculpture

ReCap Photo Workflow

Depending on the coverage of your point cloud, you might have to do some post processing on the mesh. If you have ReCap Photo, you can open the RCM file directly and edit it. Otherwise, you can use the OBJ model in other mesh editing software. ReCap Photo has tools to clean up the mesh and fill holes (see Figure 29).

[2] Later in the process you might need to reduce the number of faces in the mesh. This can be done in ReCap Photo, in 3ds Max (as shown below) or you can come back here and try the "Medium" quality option here instead.

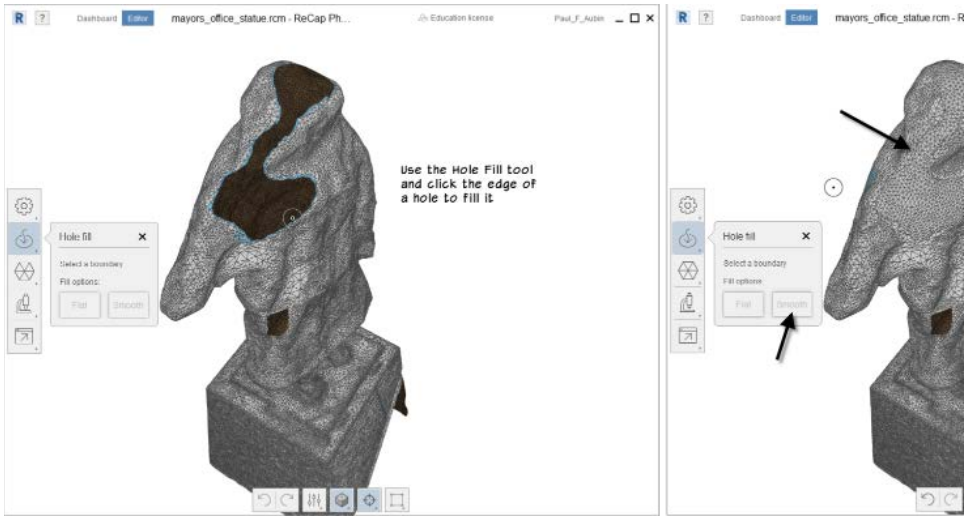


Figure 29—Fill holes and cleanup the mesh in ReCap Photo

When you are finished with the cleanup, export the model to the format of your choice, such as 3ds Max FBX format (see Figure 30).

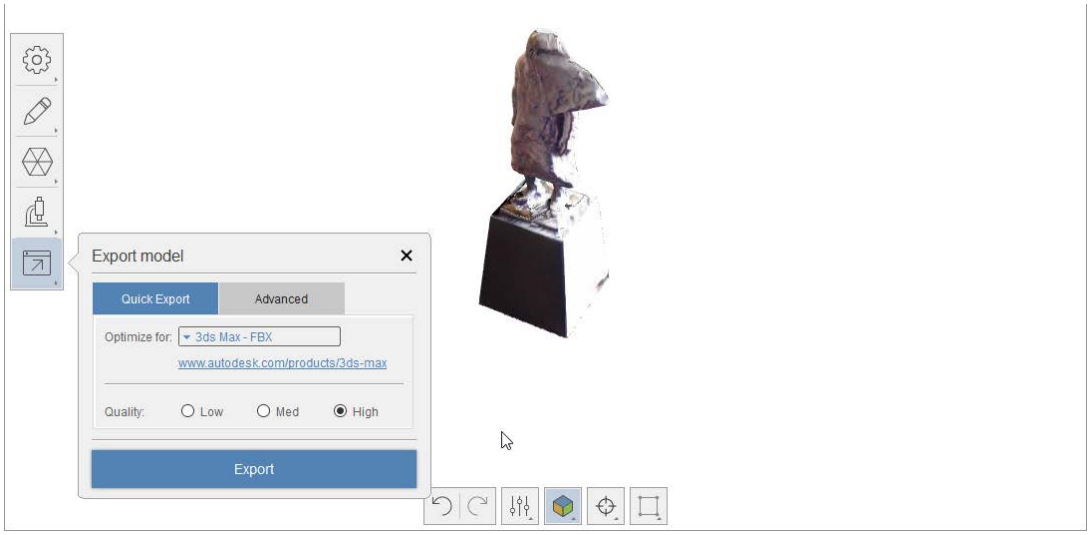


Figure 30—Export the completed mesh to 3ds Max

3ds Max Workflow

In 3ds Max, load the exported model. You may need to further process the model in 3ds Max. In my experience, the most common things to look for are orientation, scale and number of faces. There may also be some cameras and other items you don't need. Feel free to clean up and of these things (see Figure 31).

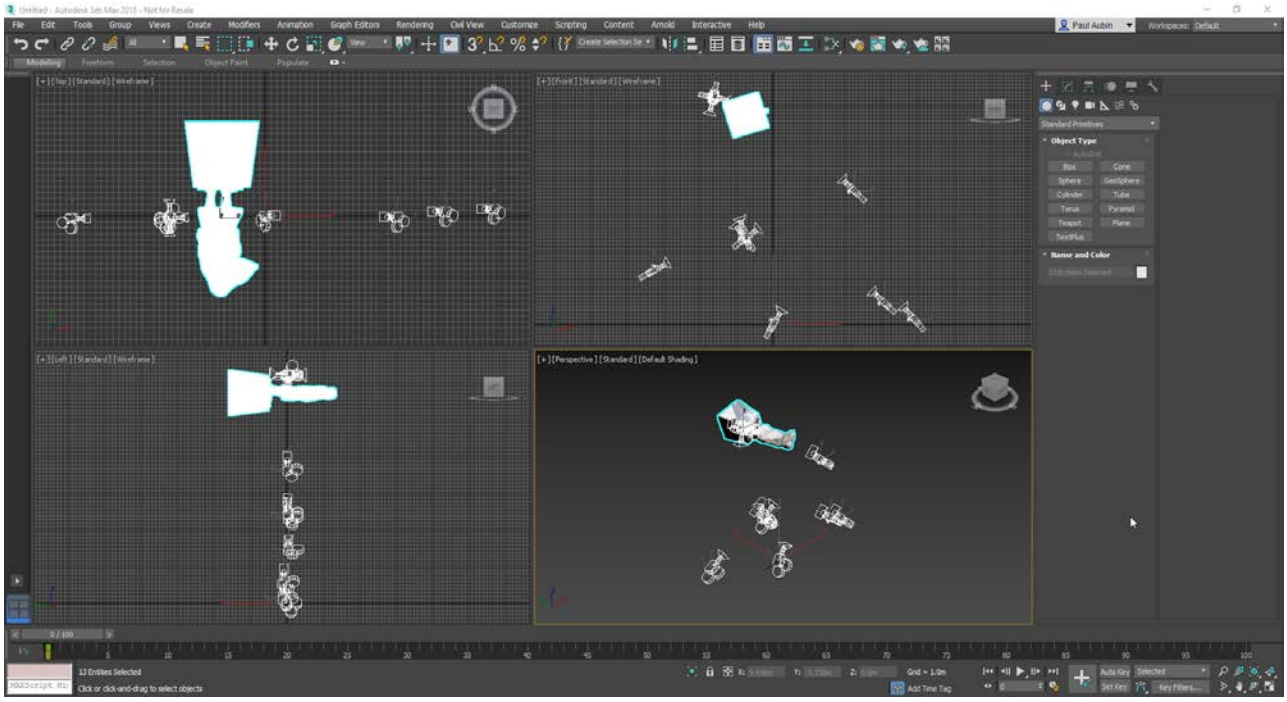


Figure 31—The Orientation, scale, number of facets and excess cameras may need attention

The most important thing you need to do is hide the facet edges. To do this and preserve their visibility when importing into Revit, we need to export using the **DXF 2004** format. This format supports up to 32,767 vertices. So, we need to make sure that the total number of vertices is within this limit (see Figure 32).

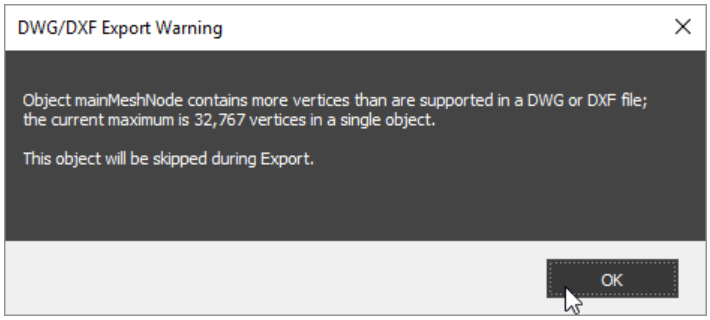


Figure 32—DXF has a maximum number of vertices allowed on export

To get the number in the allowable range, you can return to ReCap Pro, and create a new mesh using the “Medium” setting this time. But then you will have to clean up the mesh in ReCap Photo again. Instead, you can return to ReCap Photo and use the tools to decimate the mesh and then export it to Max again. Or, stay in 3ds Max and use the **ProOptimizer** modifier. This tool will help you quantify the current number of points and faces and allow you to reduce them if there are too many (see Figure 33).

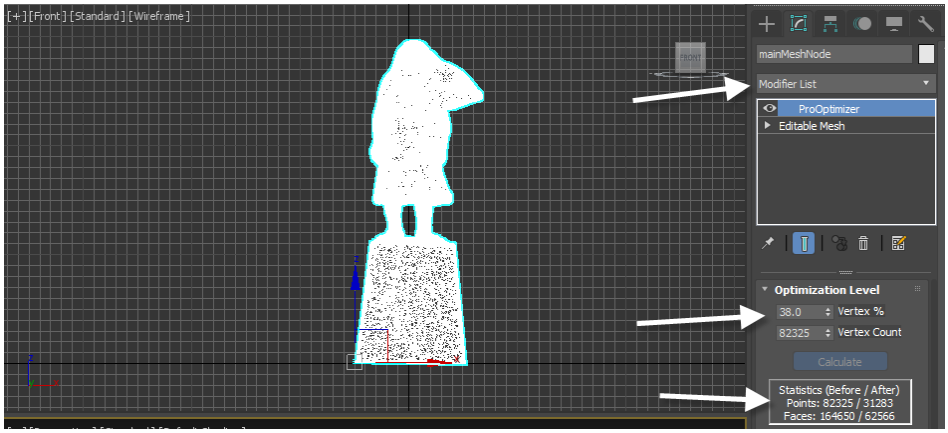


Figure 33—DXF has a maximum number of vertices allowed on export

One you have the vertices under 32K, collapse the modifier stack to return to an Editable Mesh. Then choose the edge select option. You want to select *most* of the edges. You need to leave at least a few edges visible so that you can still select the object. These can be on the bottom of the statue or some other inconspicuous area (see Figure 34).

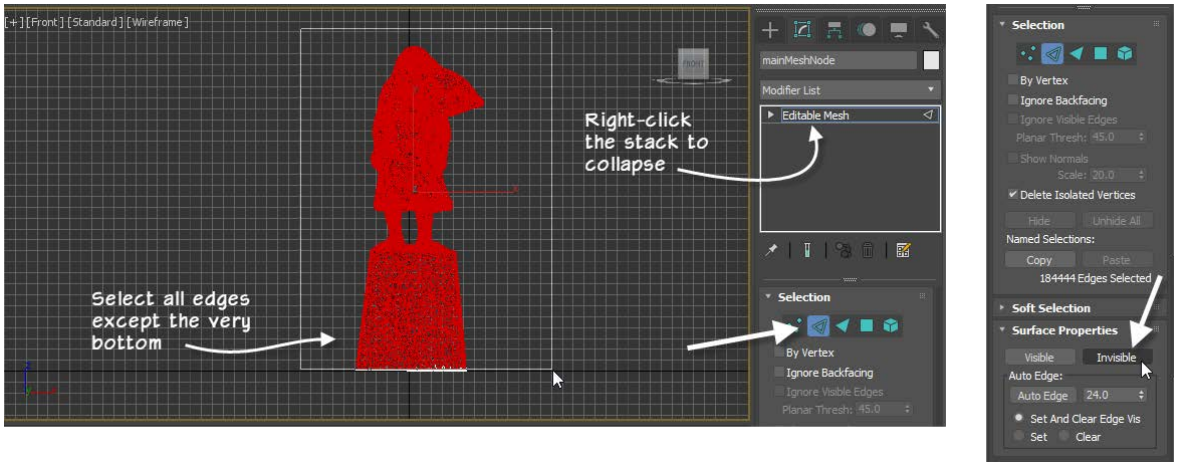


Figure 34—Collapse the stack and select edges. Make them Invisible

Click the Invisible button to make them invisible.

Alternative with Downloaded Meshes

Another common place to get meshes is from online 3D libraries. These might come in a variety of formats. 3ds Max can read most formats. Shown here is an example with an airplane mesh. In this case, you will be a little more strategic about your edge selection. Remember you need to leave at least some edges visible to allow the mesh to be selected. Unlike the statue, with an object like an airplane, there are some obvious edges that would be appropriate to leave visible. Items like the windows or other detailing. Maybe the edges of the wings or other features that would benefit from a hard edge. Sometimes models have material IDs. This can help in selection of these elements (see Figure 35).

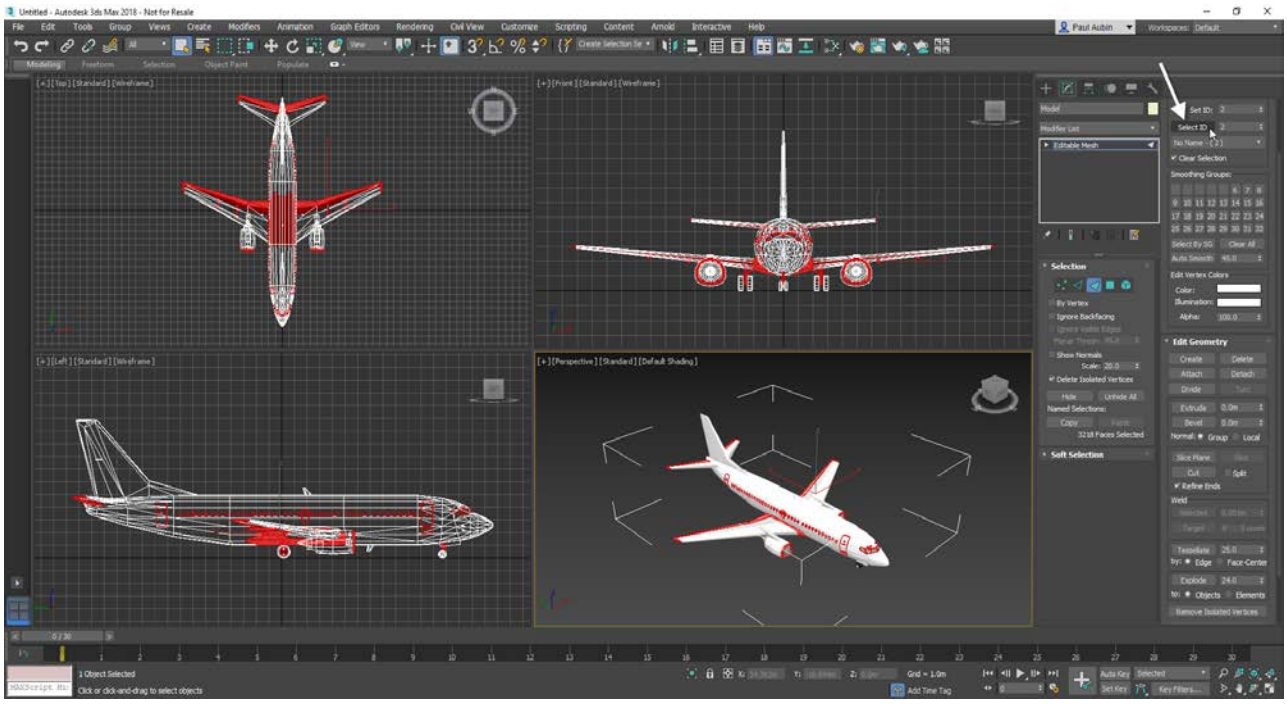


Figure 35—Use Select ID to select items by material ID

In this case, the material ID 2 grabs all the windows and some detailing on the wings. You can detach these elements to make them into separate objects. This will allow for easier selection and layering (see Figure 36).

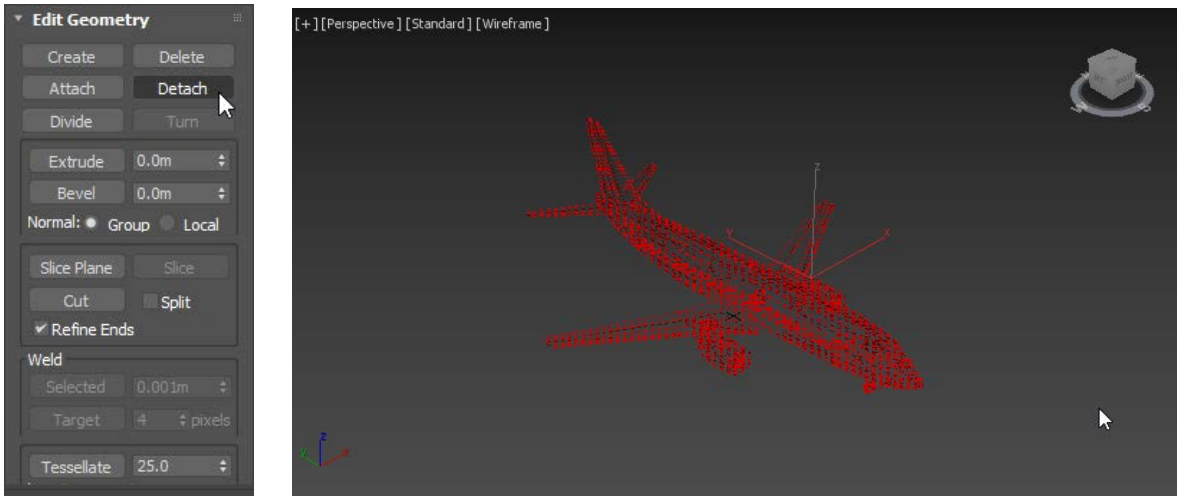


Figure 36—Detach a selection to create separate elements. Make the fuselage invisible

You can hide the detached items and then easily select the fuselage and other parts that need to be invisible and repeat the process from above. The element will disappear when not selected and will appear dashed while selected.

You can open the Layer Explorer on the Tools menu. This will allow you to see the layers in the file (see Figure 37). Right-click each layer and make it By Layer. This will allow us to change color and material once imported into Revit.

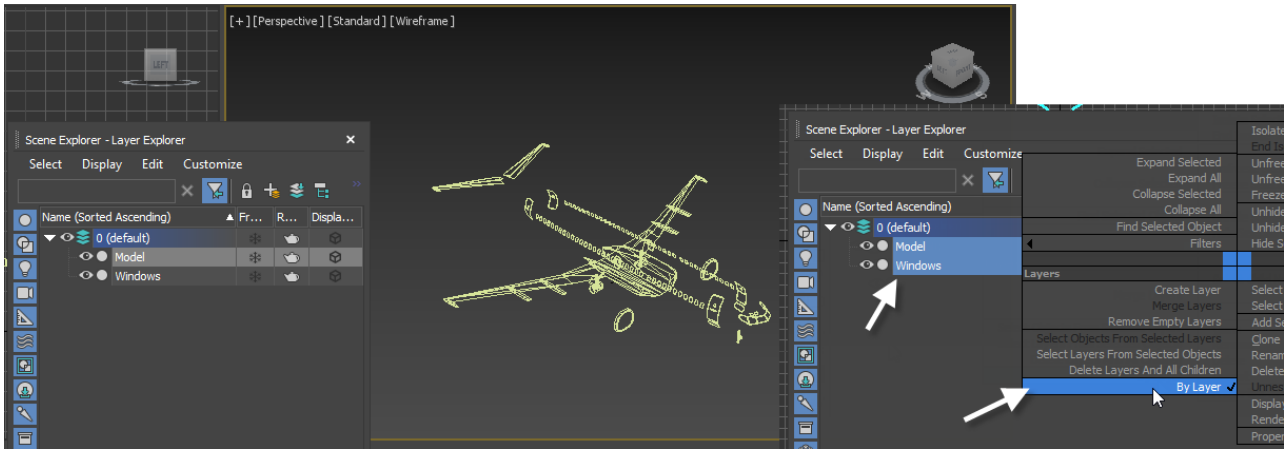


Figure 37—View the layers after processing the file

From the File menu, choose: **Export > Export**. Choose DXF for Save as Type. Browse to a location and give the file a name. In the dialog that appears, choose: **AutoCAD 2004 DXF**. This is important as this format is required to retain the invisible edges when exporting (see Figure 38).

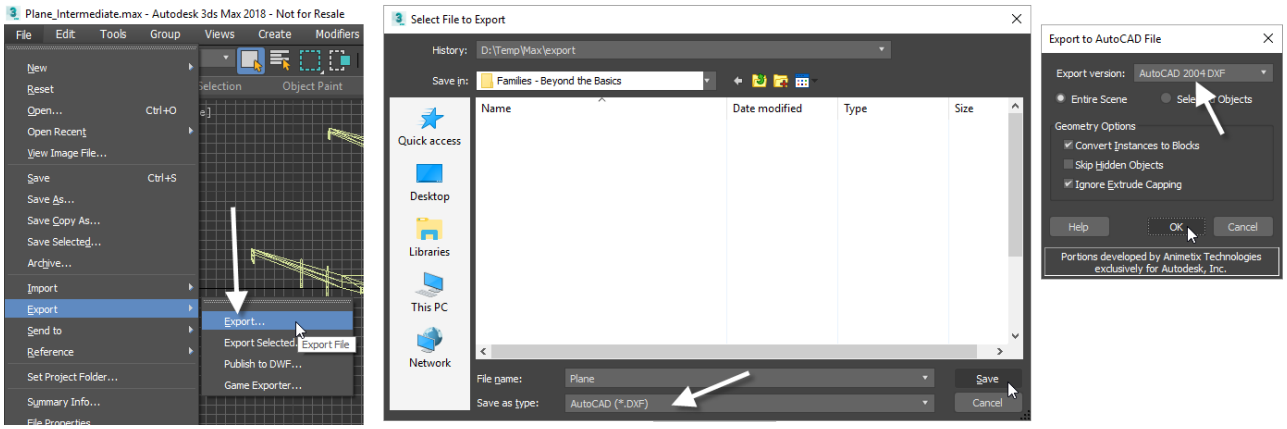


Figure 38—Export as a DWF 2004

Revit Workflow

We are finally ready to import the file into Revit. In Revit, create a new Family and choose a category and template, then import the CAD file (Figure 39).

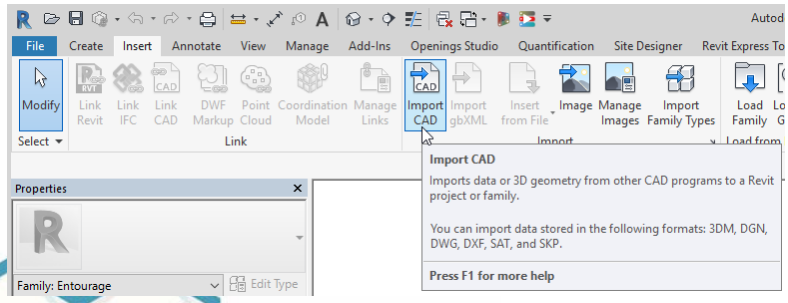


Figure 39—Import CAD into a new family

You will probably want to disable the “Correct lines that are slightly off axis” checkbox. Choose any other options as required and open the file (see Figure 40).

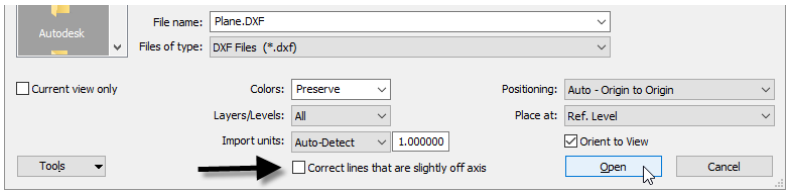


Figure 40—Configure CAD import options

When the file comes in, all the invisible edges will be preserved making for a very nice result. Any edges you left visible like the windows will still show (see Figure 41).

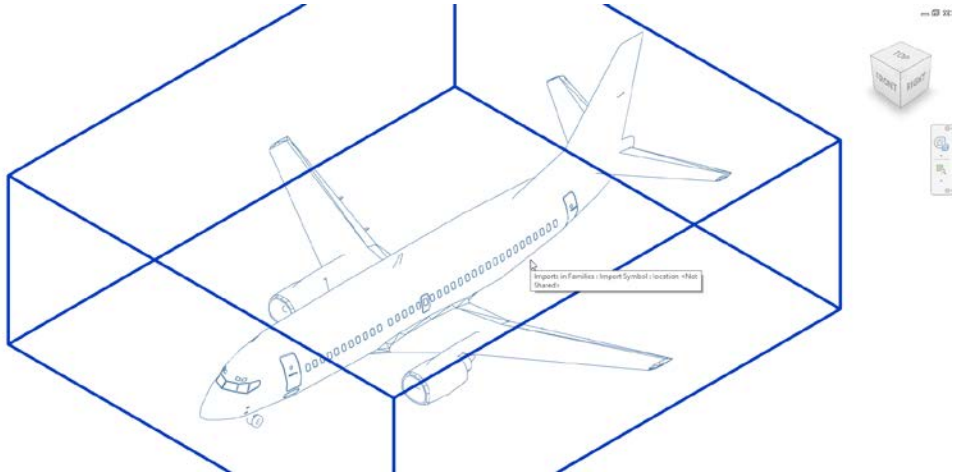


Figure 41—Invisible edges are preserved

The final step is to adjust color and materials. This is done in the Object Styles dialog. You can remove the material designation and set the color to black. You can also delete any unneeded layers and rename layer 0 if desired (see Figure 42).

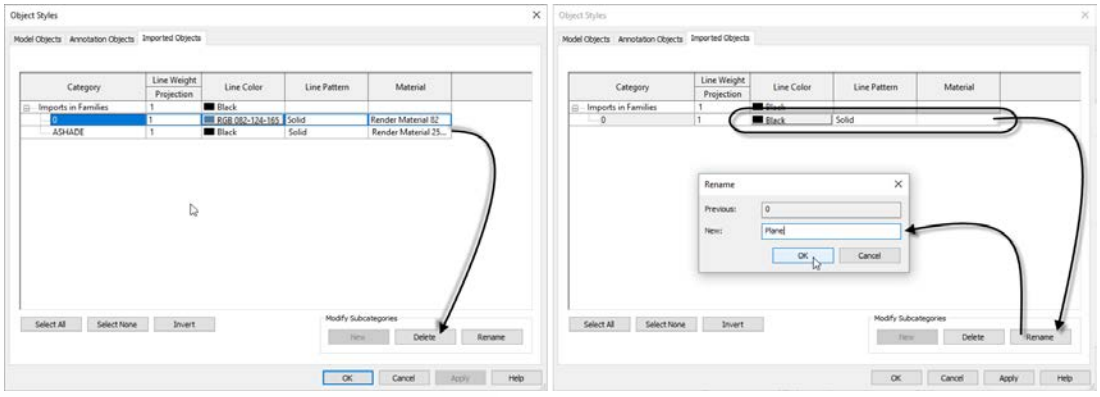


Figure 42—Adjust colors, materials and layers

If all you need is the 3D, you are done. But optionally, you can import 2D geometry into the elevation and plan views. You can add symbolic geometry, masking and filled regions as required (see Figure 43).

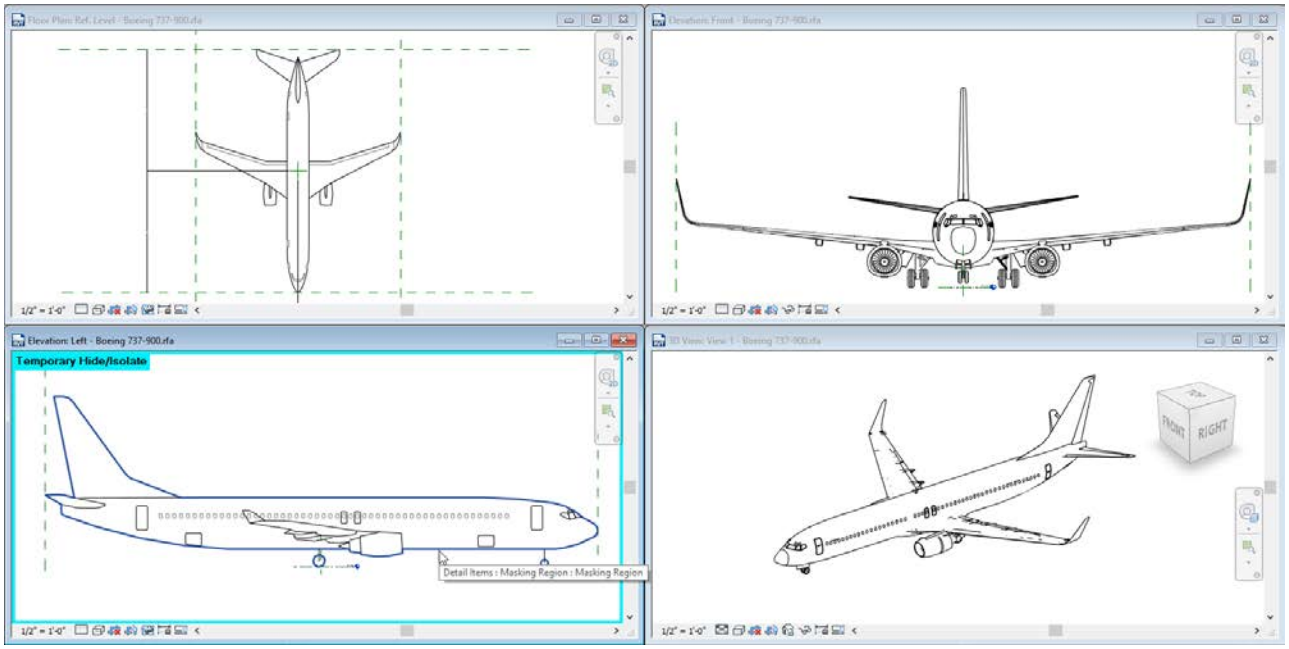


Figure 43—Optionally add symbolic geometry to the 2D views

The best part of this technique is not only do we get quite satisfactory 3D results without spending hours modeling custom and organic forms, but the files end up quite small! Take a look at the file size for that airplane! The statue looks pretty good too (see Figure 44).

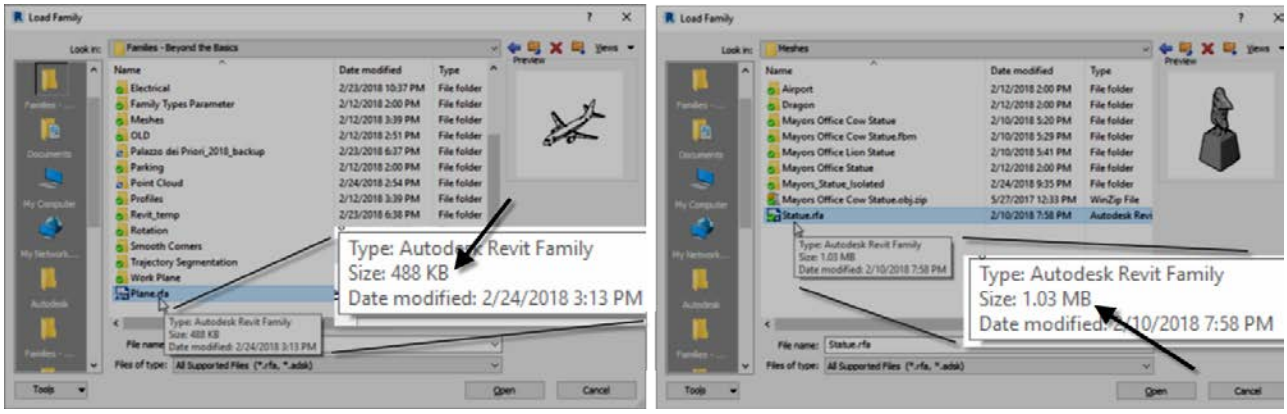


Figure 44—Mesh imports are quite compact

Naturally, these models are not parametric. You can scale them by editing their type properties, but they will not stretch or have other parametric behaviors. If you need those behaviors, you'll need to consider taking the time and effort to build a native Revit model.

511—Family Editor – Beyond the basics, Part 2

In Part 2 we will explore some additional modeling techniques with an emphasis on formula driven values. We'll discuss nested families for control of graphics and to leverage category behavior. We'll explore some formulas, throw in some trigonometry of course and we'll wrap up part two with a look at cutable behavior and using <family types> parameters to drive a list of values.

Casework

An important consideration when planning out a piece of family content is deciding how much detail to include. Should it be simple or complex? As you might expect, there are pros and cons to each approach. In general, however, it is good to take a "wholistic" approach to your content planning. Let's look at casework as an example.

I have had the opportunity to build casework libraries for several firms. Each has its own criteria and goals. While at first it may seem that casework is a straightforward problem to solve, there are many considerations and possible variations. Consider the images shown in Figure 45. The 3D geometry of the example on the left uses a simple extrusion for the entire form. It is drawn in elevation view so that the shape of the toe kick can be incorporated into the sketch. Separate door front families are nested in and placed on the front to represent doors and drawers. The example in the middle is based on the out-of-the-box families. A sweep is used to create a hollow box that sits on top of separate extrusions for the toe kick. Again, there are nested families for the door and drawer fronts. The third example on the right uses several sweeps and extrusions to create members representing each piece of wood and material in the actual construction of the case.

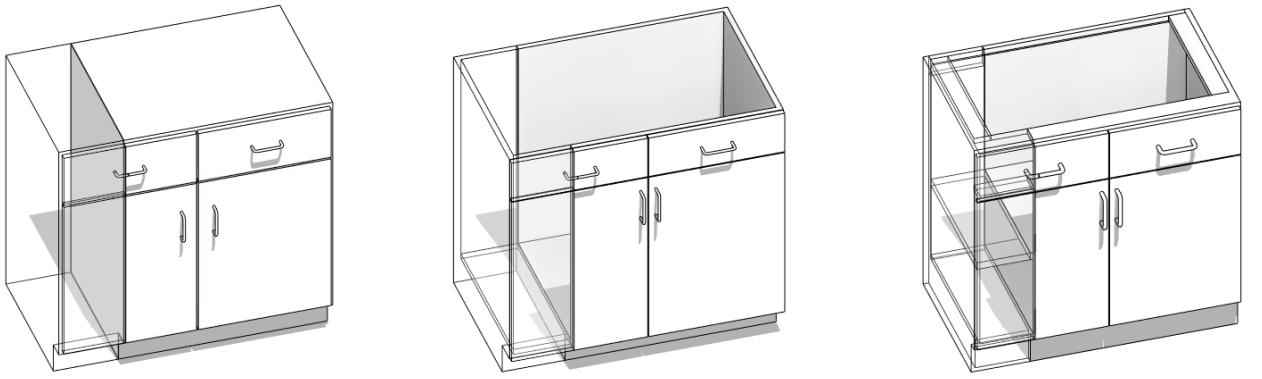


Figure 45—Casework examples with varying geometry strategies

Which of these approaches is correct? As you might expect, to answer this question we need to know more about what the specific requirements are. But ultimately, there are advantages and disadvantages to each approach, so it becomes a matter of preference.

When you look at the front of each of these units, they look nearly the same. The only time you can really tell the difference is when sectioning through them. If you do not anticipate needing to section them, then the approach on the left is perfectly fine. However, if you need a more accurate rendition of the actual case in both elevation and detail/section views, then you might favor the approach on the right. The middle can be seen as a compromise between the two.

Solid vs Void

In the case of the client that went with the approach on the left, we had to use voids for the sink base units. This is because if we left those units solid, then when you add a sink you would see the solid form from below filling the sink. By adding the void, you create an opening below that allows for the sink (see Figure 46).

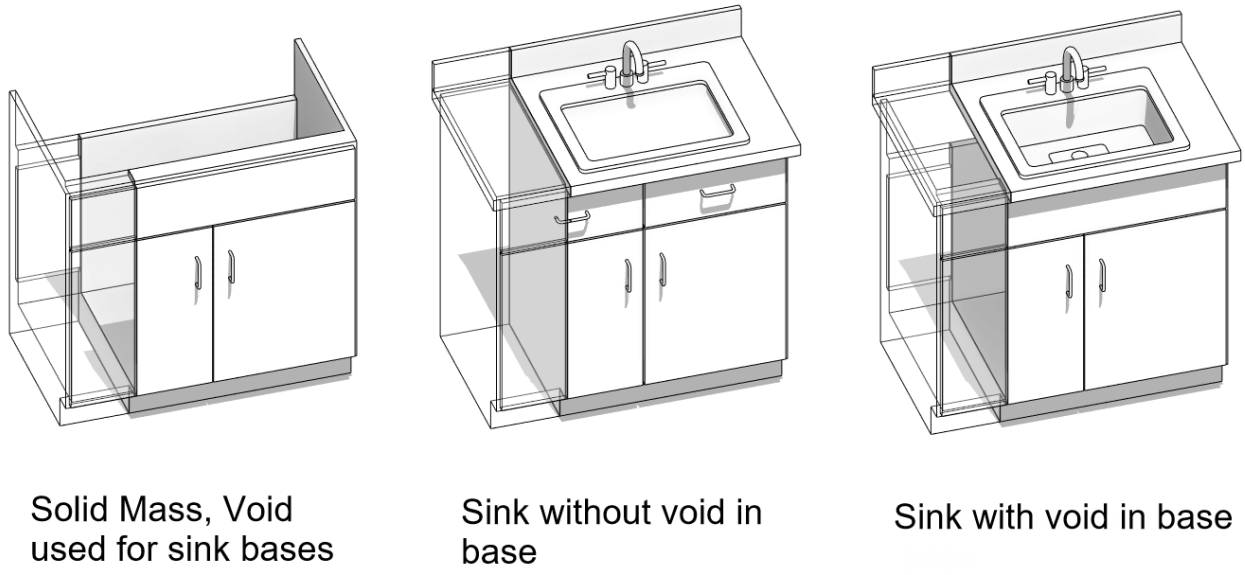


Figure 46—Voids required in sink bases to properly interact with sink

There were some similar situations when glass doors were required. While I prefer to minimize the use of voids, I did like the simplicity of this approach and the ability to have a single extrusion for most elements in this library. Using voids allowed me to keep the construction consistent across the whole library.

Internal Construction

The approach on the right in Figure 45 above has the advantage of showing a rendition that is very accurate in 2D and 3D and when you section through it. This is advantageous because you are building it as it will actually be built. This nearly eliminates any chance of having the model show inconsistencies from plans to sections to elevations. It does require a bit more effort to build this kind of model. There will often be more parameters and reference planes required. And certainly, more individual pieces of geometry will be required (see Figure 47).

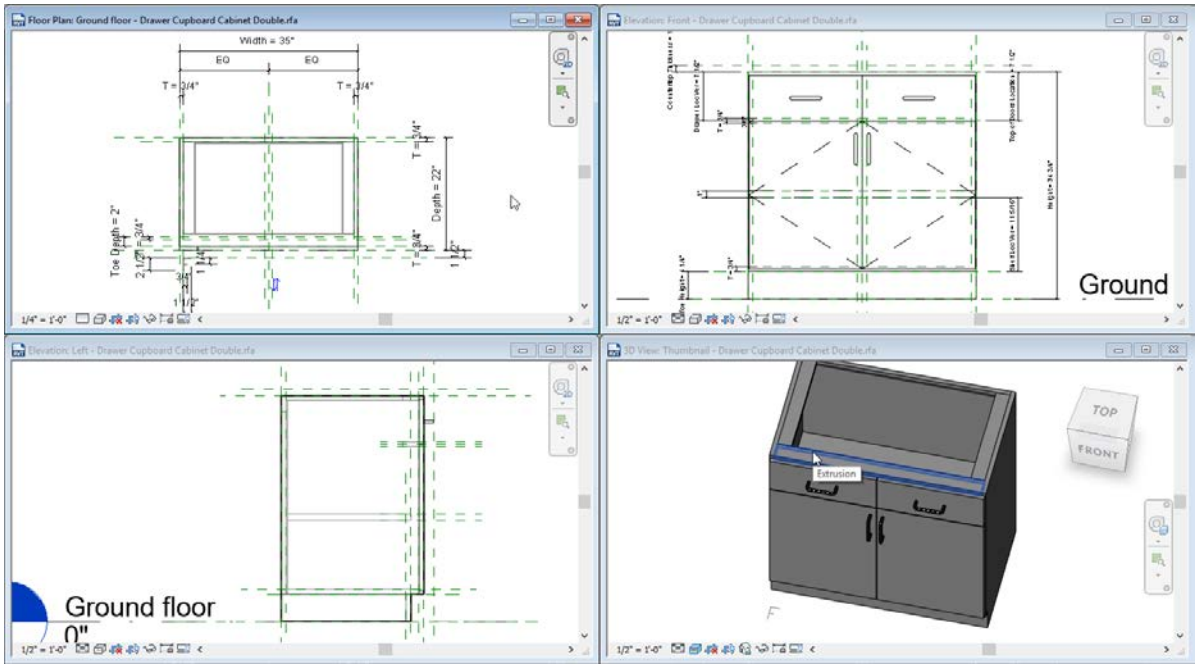


Figure 47—Building the case in individual pieces emulating its actual construction

Alternatively, you can represent the section cut using a nested 2D component. In the example shown on the left of Figure 45 above, this is what was done. Since the case is a solid extrusion, when you slice through it in section, you would not see anything but a solid. But since the Casework category supports nested 2D detail items, you can create a fully parametric 2D Detail Item family and have it show only when the instance is cut.

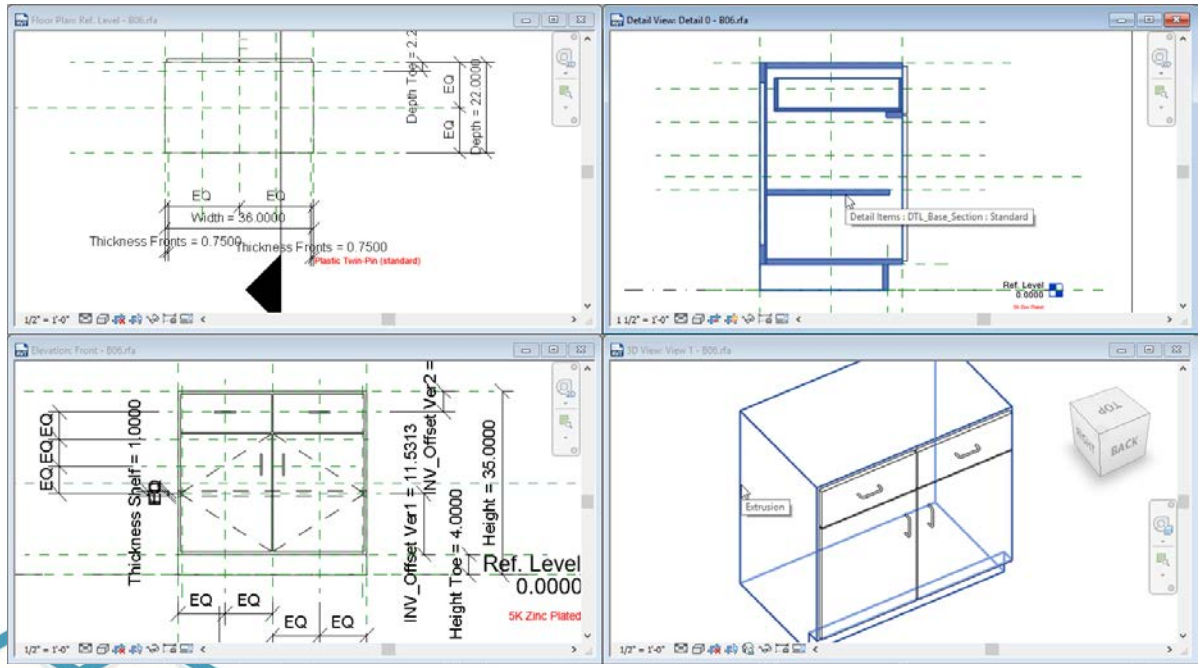


Figure 48—Us a nested 2D Detail Item family as an alternative

I've built entire libraries each way, based on this experience, I do not see an obvious advantage of one approach over the other. I think if you are consistent with whatever approach you choose, you will find that the results are quite satisfactory in any of the three methods showcased.

Zero and Negative Values

Sometimes you need a length parameter to support zero and negative values. The length parameter itself can be set to any value you like: positive, negative or zero. But depending on what that parameter is assigned to, it might break the geometry or dimensions when you try to flex to zero and negative values. As an alternative, apply the parameters to the Extrusion Start and/or End values of the extrusion. You can use zero and negative values as long as the total thickness of the extrusion remains a positive value (see Figure 49).

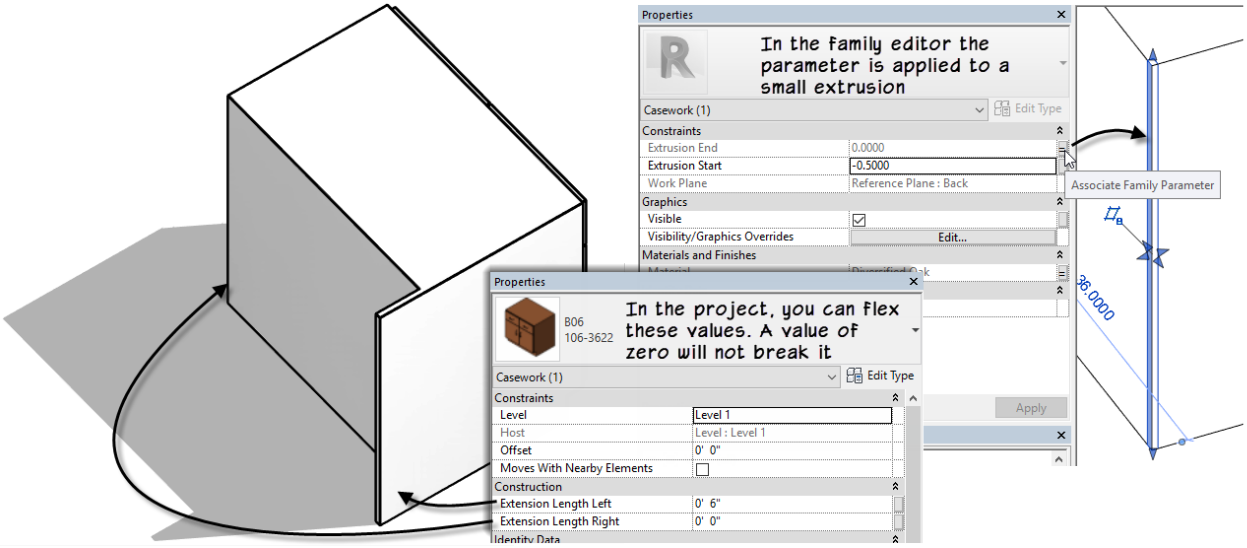


Figure 49—Parameters assigned to the Extrusion Start and End values can be any number if the total thickness remains positive

Options Lists

When building libraries for casework manufacturers, they often have many options and variations possible. While many can certainly be included in the model, many options do not need to be modeled at all. Consider options like variations in hinges or drawer sliders or changes in the construction of the case. They can be included as shared parameters that can be included on schedules. This is the most efficient way to include options. I usually like to include a tooltip in these parameters so that users will know which options to choose (see Figure 50).

Sometimes the option can also drive geometry. But the nice thing about shared parameters is that they can appear in schedules and tags regardless of whether or not they are used to drive the family's geometry.

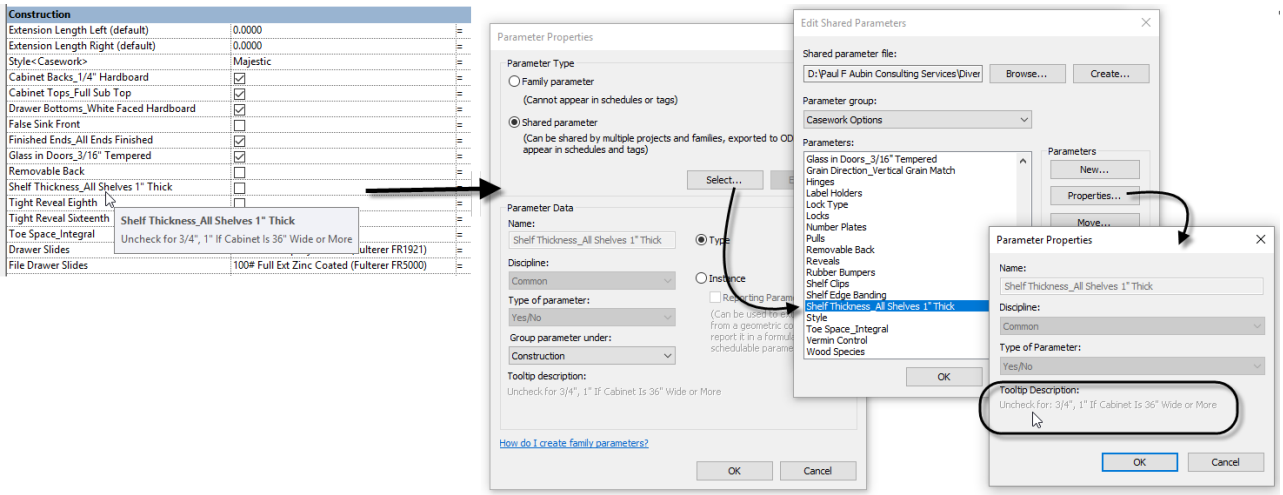


Figure 50—Name checkboxes carefully and use tooltips to show what the unchecked state is

Controlling Reveals with Formulas

All three strategies above use nested families for the door and drawer fronts. These are simple families with an extrusion or sweep for the solid panel portion of the door/drawer front. Additional families are nested in for the hardware. In many cases, there are setback and reveal requirements around all four edges of the door or drawer front. This is accomplished with reference planes and parameters in the door/drawer family. The size of the reveals is then driven by parameters in the host family to which the door/drawer family is nested (see Figure 51).

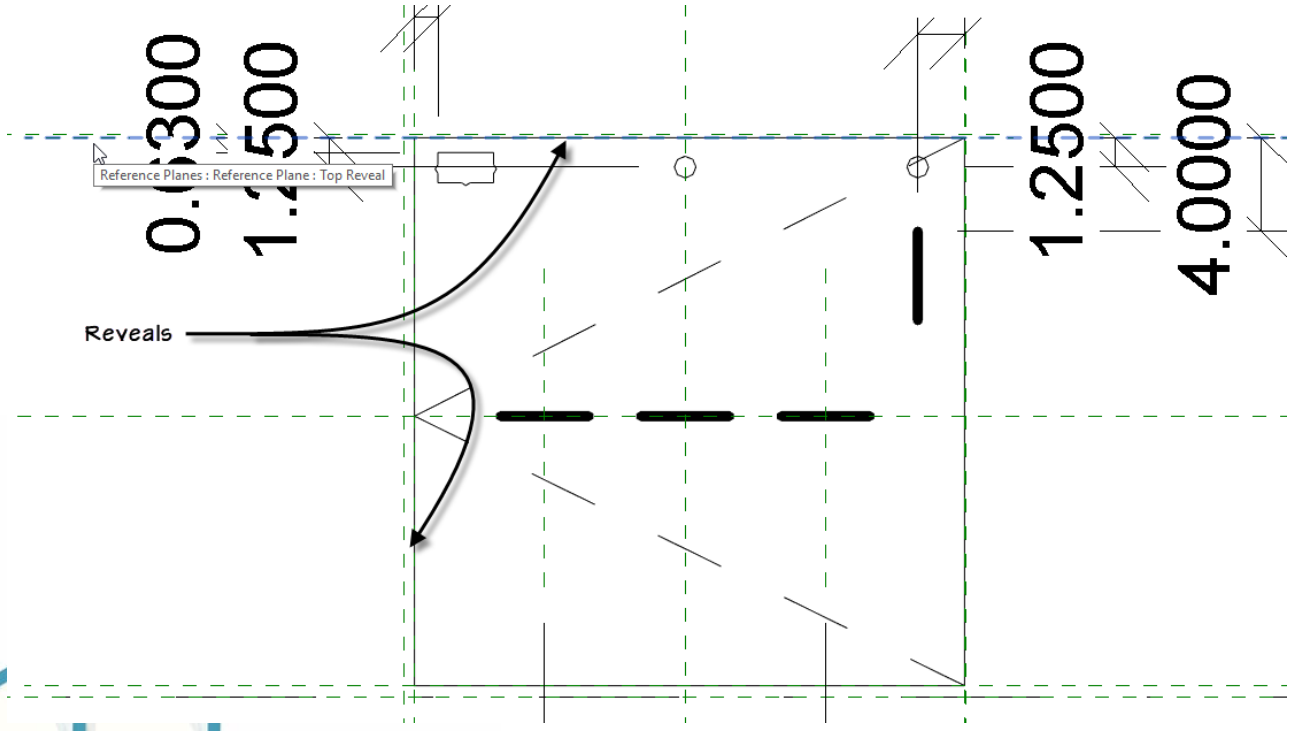
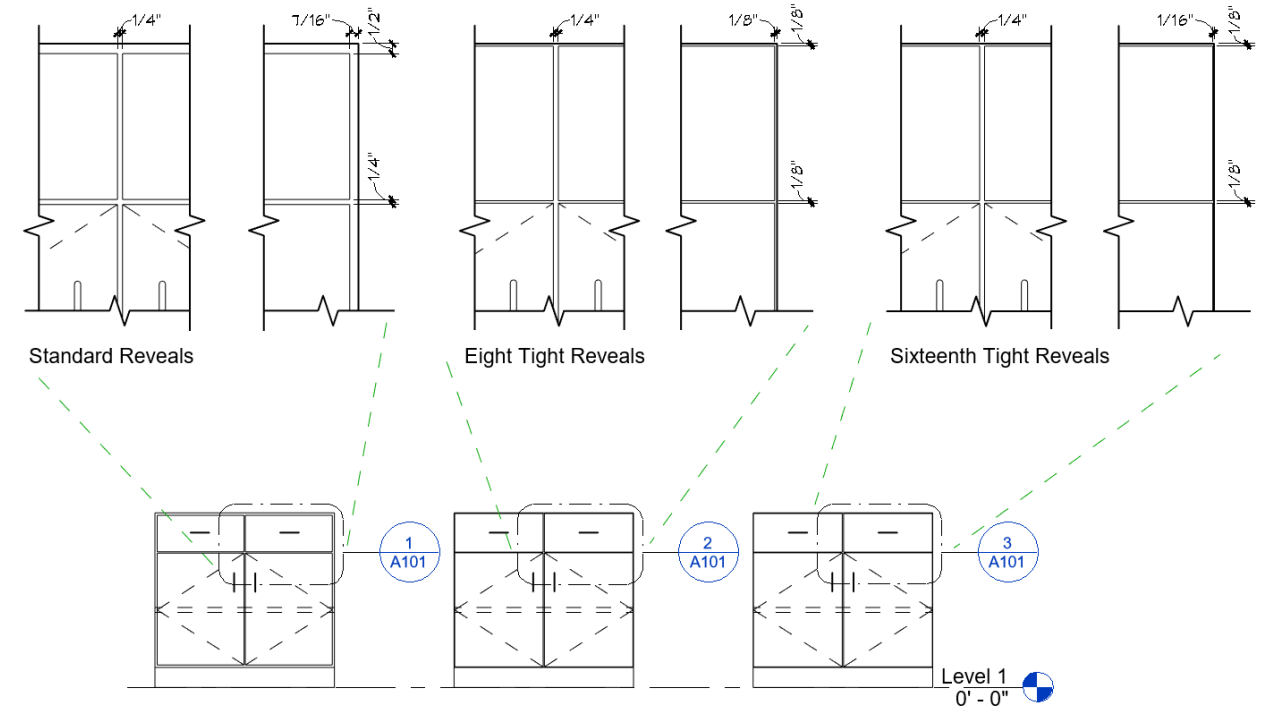


Figure 51—Geometry in family is set back from edges with reveal reference planes and parameters

In some cases, I set these values at the type level manually in the family or with a type catalog if there are many variations. However, in the case of one collection, there were very specific rules about the reveals. There are three possibilities: “Standard,” “Eight Inch Tight Reveals” and “Sixteenth Inch Tight Reveals.” Furthermore, there are different styles for the fronts. One of the styles uses different values for reveals than the others.

This could have all been controlled in a type catalog, but we decided to control it with formulas instead. Since there are more than two possibilities, we needed to nest conditional statements. The formulas look at three Yes/No parameters. One of them is a check to see if the style of the door/drawer fronts is set to the “Millennium” style. (Millennium uses different rules than the rest). If not, then it looks at two other checkboxes: Eight and Sixteenth. These two checkboxes create three possibilities: Only Eight checked, only sixteenth checked, and in the case where they are either both checked or both unchecked, the result is the same (see Figure 52).



Tight Reveal Eighth	☐
Tight Reveal Sixteenth	☐
Toe Space_Integral	☑
Drawer Slides	75# 3/4 Ext Epoxy Coated (Fulterer FR1921)
File Drawer Slides	100# Full Ext Zinc Coated (Fulterer FR5000)
Reveals	Standard Reveals
RV Top Bottom	0' 0 1/2"
RV Left Right	0' 0 7/16"
GV1	0' 0 1/8"
GH1	0' 0 1/4"

Tight Reveal Eighth	☑
Tight Reveal Sixteenth	☐
Toe Space_Integral	☑
Drawer Slides	75# 3/4 Ext Epoxy Coated (Fulterer FR1921)
File Drawer Slides	100# Full Ext Zinc Coated (Fulterer FR5000)
Reveals	Eighth Inch Tight Reveals
RV Top Bottom	0' 0 1/8"
RV Left Right	0' 0 1/8"
GV1	0' 0 1/8"
GH1	0' 0 1/8"

Tight Reveal Eighth	☐
Tight Reveal Sixteenth	☑
Toe Space_Integral	☑
Drawer Slides	75# 3/4 Ext Epoxy Coated (Fulte
File Drawer Slides	100# Full Ext Zinc Coated (Fulte
Reveals	Sixteenth Inch Tight Reveals
RV Top Bottom	0' 0 1/8"
RV Left Right	0' 0 1/16"
GV1	0' 0 1/8"
GH1	0' 0 1/8"

Parameter	Value	Formula
Tight Reveal Eighth	☐	=
Tight Reveal Sixteenth	☑	=
Toe Space_Integral	☑	=
Drawer Slides	75# 3/4 Ext Epoxy Coated (Fulterer FR1921)	=
File Drawer Slides	100# Full Ext Zinc Coated (Fulterer FR5000)	=
Reveals	Sixteenth Inch Tight Reveals	= if(and(Tight Reveal Eighth, Tight Reveal Sixteenth), "Standard Reveals", if(and(Tight Reveal Eighth, not(INV_Is Millennium)), "Eighth Inch Tight Reveals", if(and(Tight Reveal Sixteent
RV Top Bottom	0.1250	= if(and(Tight Reveal Eighth, Tight Reveal Sixteenth), 0.5", if(or(and(Tight Reveal Eighth, not(INV_Is Millennium)), and(Tight Reveal Sixteenth, not(INV_Is Millennium))), 0.125", 0.5"))
RV Left Right	0.0625	= if(INV_Is Millennium, 0.5", if(and(Tight Reveal Eighth, Tight Reveal Sixteenth), 0.4375", if(Tight Reveal Eighth, 0.125", if(Tight Reveal Sixteenth, 0.0625", 0.4375"))))
GV1	0.1250	= if(INV_Is Millennium, 0.25", 0.125")
GH1	0.1250	= if(and(Tight Reveal Eighth, Tight Reveal Sixteenth), 0.25", if(or(and(Tight Reveal Eighth, not(INV_Is Millennium)), and(Tight Reveal Sixteenth, not(INV_Is Millennium))), 0.125", 0.25")

Figure 52—Formulas with IF statements drive the sizes of the reveals

Here is the formula for the left and right reveal from the figure as an example:

if(INV_Is Millennium, 0.5", if(and(Tight Reveal Eighth, Tight Reveal Sixteenth), 0.4375", if(Tight Reveal Eighth, 0.125", if(Tight Reveal Sixteenth, 0.0625", 0.4375"))))

The first IF checks to see if the item is set to Millennium. If so, the reveal is: ½". If this is true, then no further query is performed. If not, a nested IF asks if both the eight and sixteenth checkboxes are checked. If so, we have “Standard” reveals and the value is set to: 7/16". If this is not true, then two more nested IF statements come next to ask if only one of the boxes is checked. If eight is checked, the value is set to: 1/8". If not and sixteenth is checked, then the value is assigned to: 1/16". If none of those are true, (both boxes are unchecked) then Standard Reveals are used, and it assigns a value of: 7/16".

The formulas for the top and bottom and interior conditions are similar.

I would prefer this were simpler. My number one wish for new features in the family editor is a true list parameter! (That and concatenation). And while there are work arounds to lists (see the “Make a “List” Parameter” topic below), in cases like this with a few checkboxes and some carefully tested formulas, you can arrive at almost the same user experience. The trick here is making result of both eight and sixteenth or neither eight and sixteenth equal checked equal the standard reveals. This makes it less likely the user will make a mistake. They must be sure to check *only one* of the two boxes if they don’t want standard reveals.

Profile Family Rotation

Rotation is a common challenge in many pieces of family content. There are a variety of techniques that are in common use. Using a reference line is the typical “go to” solution. This is effective for many simple examples. Here I am showcasing an alternative that can work well for some common scenarios (like door swings), but also in many other areas as well like the examples shown in Figure 53.

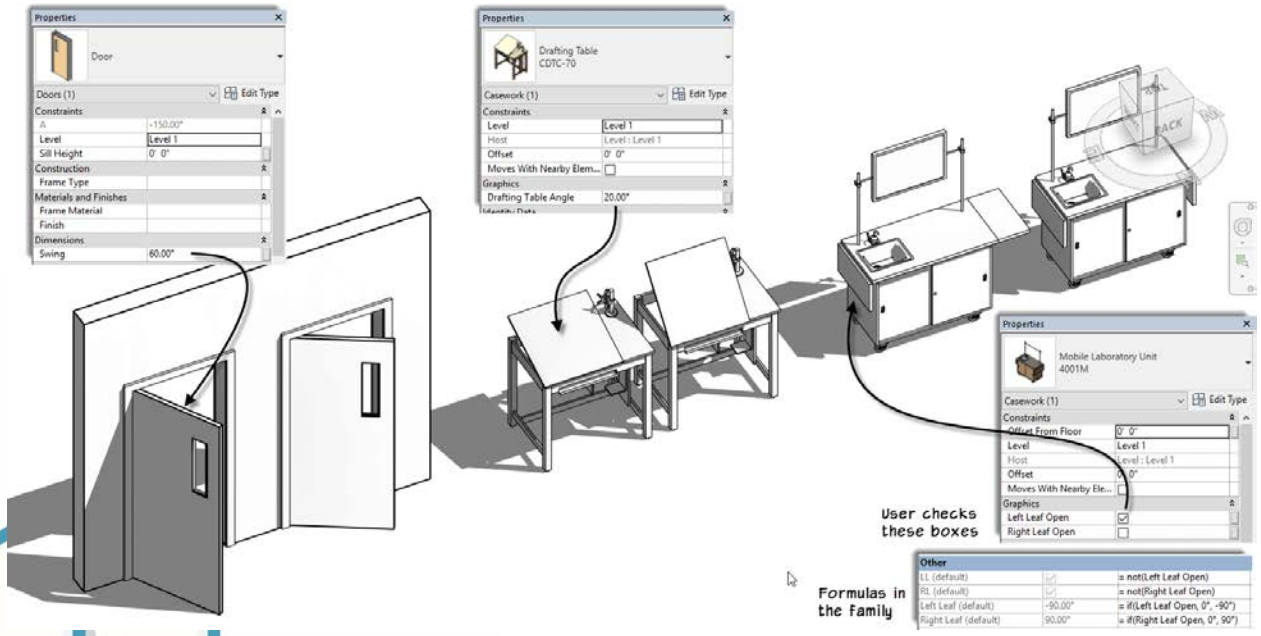


Figure 53—Using the rotation parameter of nested profile families in sweeps

The trick is to assign a parameter to the rotation parameter of the profile used to form a sweep. The example in the middle is the simplest one. A drafting table with adjustable angle. The surface of the drafting table is a sweep. The profile of the sweep is a long thin rectangle. The parameter is assigned to the Angle parameter of the sweep profile. The mobile lab table to the right uses the same approach, only the sweep is inside a nested family for counter in this one (see Figure 54).

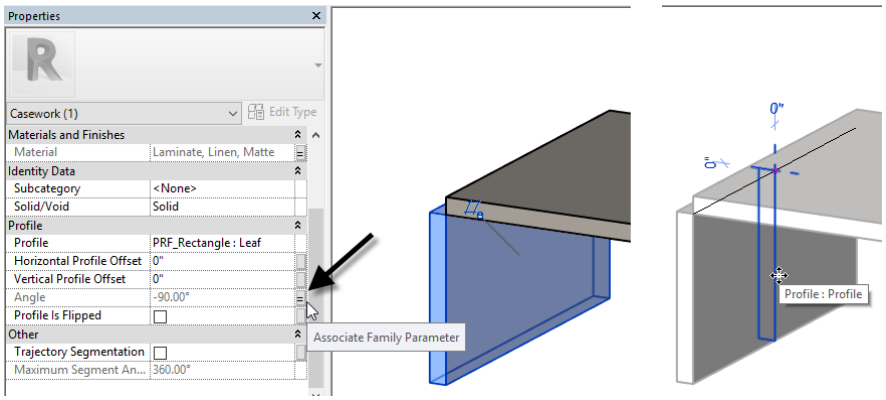


Figure 54—Sweeps have a built-in Angle parameter that can be driven with a family parameter

The door is similar but adds a second sweep for the glazing component. Both the sweep for the door panel and the glazing use the same angle parameter. There is also a void sweep in the same size as the glazing sweep to cut the door panel. The glazing sweeps use an offset built into the family to shift the profile shape from the origin. The origin remains at the hinge point of the door to allow for the proper rotation (see Figure 55).

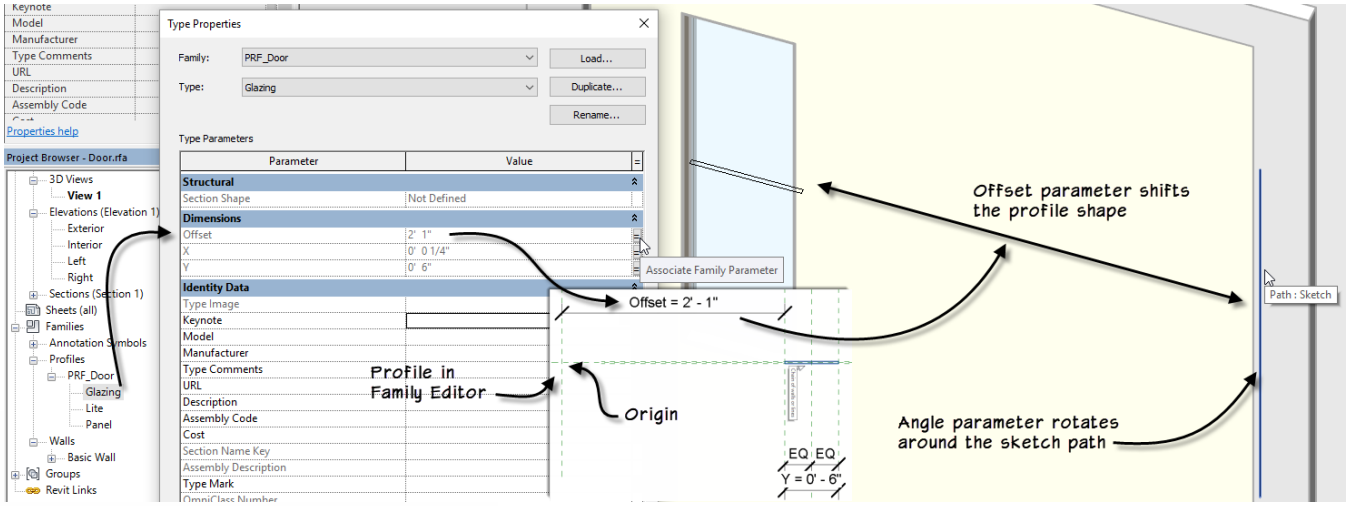


Figure 55—Add an offset in the profile family to adjust the center of rotation

Window Families

When creating a library of components, I often start by identifying the common features and requirements and then building these into a series of components and “seed” families. A seed family is just a starting point family that has as many of the features I need in it and flexing properly. Then I can save copies from that seed to make the library. I discussed

casework above. I built seeds for each of those libraries before starting. In this topic, I will show a library of windows.

Most of these windows use the same set of nested components. There is a sash family. This gets nested into another subcomponent and hosted to a reference line. This reference line can rotate the window sash and by positioning it differently, we can use the same nested family to create casement, awning, hopper and fixed windows. Since they all use the same nested sash components, they all benefit from behaviors built into the sash families. The most important feature in this case is the inclusion of several muntin patterns. Included are a single pane to up to a 5x5 pattern giving five different versions of the family (see Figure 56).

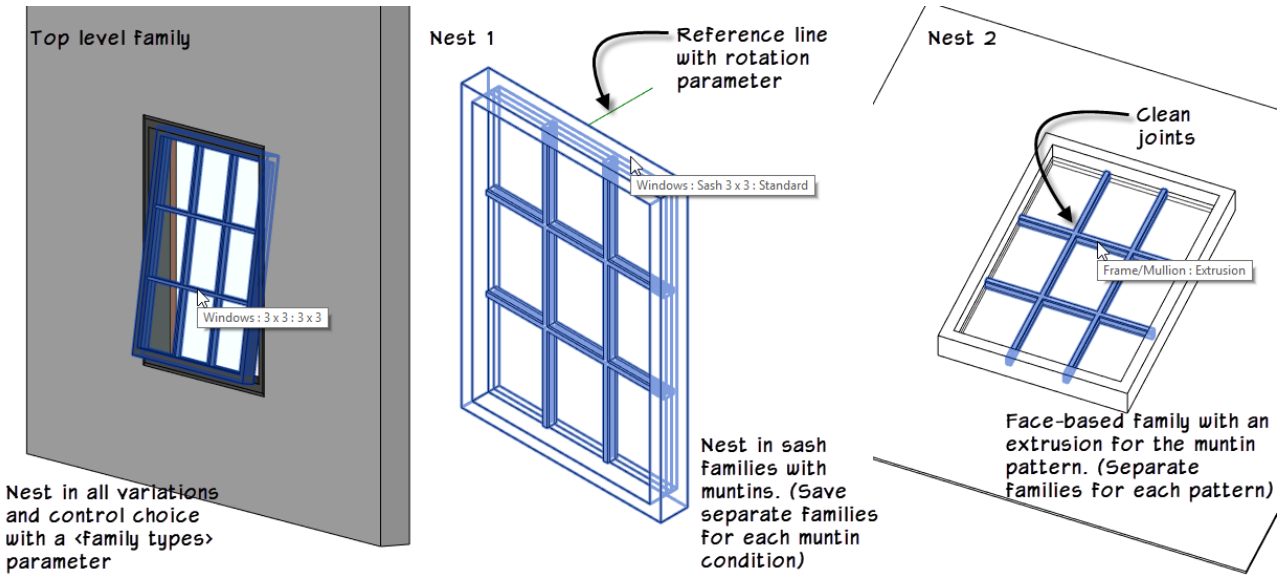


Figure 56—Add an offset in the profile family to adjust the center of rotation

I decided to build the muntin patterns separately for each configuration. In other words, I am not doing flexible arrays and trying to build it all into a single family. The main reason for this is that is by using a single extrusion for the muntin pattern, all the intersections will clean up. If they were arrays, they would overlap and show seams. The down side of building them this way is that I need to make several versions; one for each muntin pattern. We therefore limited this to the following: None, 2x2, 3x3, 4x4 and 5x5. However, there are parameters that allow the outer bays to vary from the inner ones, so they do not have to all be equal. This gives some variation to the collection (see Figure 57).

Outside Depth	0' 1"	=
Frame Width Sill	0' 2"	=
Frame Width Jamb_2	0' 2"	=
Frame Width Jamb_1	0' 2"	=
Frame Width Head	0' 2"	=
Frame Depth (default)	0' 6"	=
Sash Depth Offset	0' 0 1/2"	=
Muntin Width	0' 0 3/4"	=
Muntin Thickness	0' 0 3/4"	=
Offset from Wall Centerline (default)	0' 1"	=
Construction		
Space Vertical Lites Equally (default)	☐	=
Space Horizontal Lites Equally (default)	☐	=
Lite Width Right (default)	0' 8"	=
Lite Width Left (default)	0' 8"	=
Lite Height Top (default)	1' 0"	=
Lite Height Bottom (default)	1' 0"	=
Grille Pattern<Windows> (default)	5 x 5	=
Frame Is Centered (default)	☐	=
Frame Depth matches Wall Thickness (default)	☐	=
Wall Closure	By host	=
Construction Type		=
Graphics		
Open in 3D (default)	☑	=
Text		
SCB Version (default)	V1_20161004	= "V1
Materials and Finishes		
Window Trim Material	042_Wood - Window Trim	=
Window Muntin Material	148_Metal - Steel, Polished	=
Window Sealant Material	Rubber, Black	=
Window Glazing Material	102_Glass	=

Figure 57—Outer bays can optionally vary from the internal ones

These windows also feature 2D symbolic graphics that show in plan and elevation views. This includes support for coarse, medium and fine levels of detail (see Figure 58).

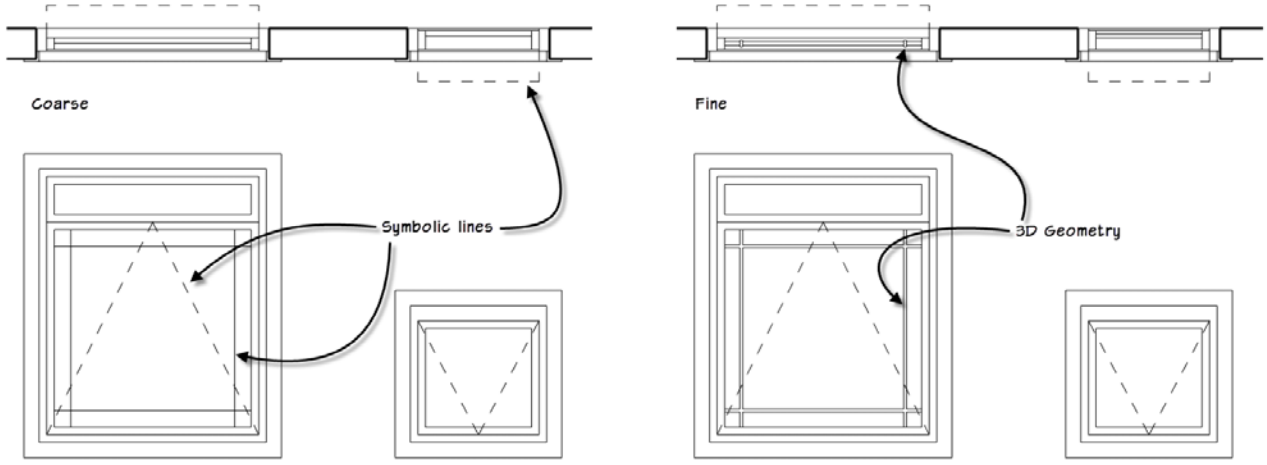


Figure 58—Symbolic linework in plan and elevation supports varying levels of detail

Finally, they have materials, sealant on the outside, trim on the inside and features that allow them to be centered or offset in plan as user needs require.

Electrical Fixtures

This topic will look at some electrical fixture families. They use a simple 3D representation in elevations and 3D. In plans, they use scale-dependent 2D symbolic graphics. They can be inserted at various mounting heights. When the height is different than the default, a label displays the height value in plans. Furthermore, there are several toggles to control the different kinds of fixtures: GFI, Dedicated Circuit, 220V, etc.

Face-based or Freestanding

In the “Face-based and Work Plane-based Families” topic in Part 1, I discussed some pros and cons of face-based families. As I noted there, these are very popular for many kinds of content. This typically includes electrical fixtures. One very nice feature of face-based families is the Default Elevation parameter. When you place a face-based family in a plan view, using the place on a vertical face option, this value will be used as its placement height. You can move the instance later as required. This default is only used when inserting in a plan view. This feature coupled with hosting behavior are attractive features for electrical fixture families.

However, most electrical fixtures also have special requirements for plan views that differ from elevations and 3D. In plan we typically want to show a symbolic representation. Furthermore, we typically want this 2D plan symbol to adjust with the scale of the view, while in elevations and 3D we typically want to see an actual 3D representation. For simple fixtures, these requirements are quite easy to satisfy. But when there are more requirements, the features and benefits begin becoming more complicated and more difficult to satisfy. The items I have here had the additional requirement of showing text in plan views to indicate the type of fixture and the mounting height above the floor.

The mounting height presents the biggest challenge. The level in a face-based family is in a different view and orientation than it will be in the project. The Default Elevation parameter is a “default” parameter. Once the item is inserted, the object will have an Elevation parameter instead. Some elements also have an Offset parameter. All of these are built-in parameters and none of them are available to tags or schedules. So, the solution many folks arrive at is creating a custom shared parameter for the mounting height instead. The trouble with that approach is that with a face-based family it is difficult to control this reliably. Furthermore, face-based families can be inserted on any face. But if you want specialized and scale-dependent 2D graphics in plan views, you must make assumptions about the insertion of these families so that you can introduce the 2D symbolic graphics in the correct view of the family to make sure they appear in the project’s plan view. Most of the time, these issues and their potential problems can be mitigated, while staying with a face-based solution. But there is another alternative: Don’t’ use face-based, instead, make the electrical fixture content non-hosted.

Non-hosted elements eliminate any of the orientation issues. They have a fixed reference level and orientation. You can place them in plans easily and use a custom shared parameter to control the mounting height. This parameter measures from the reference level and can appear in tags and schedules (see Figure 59).

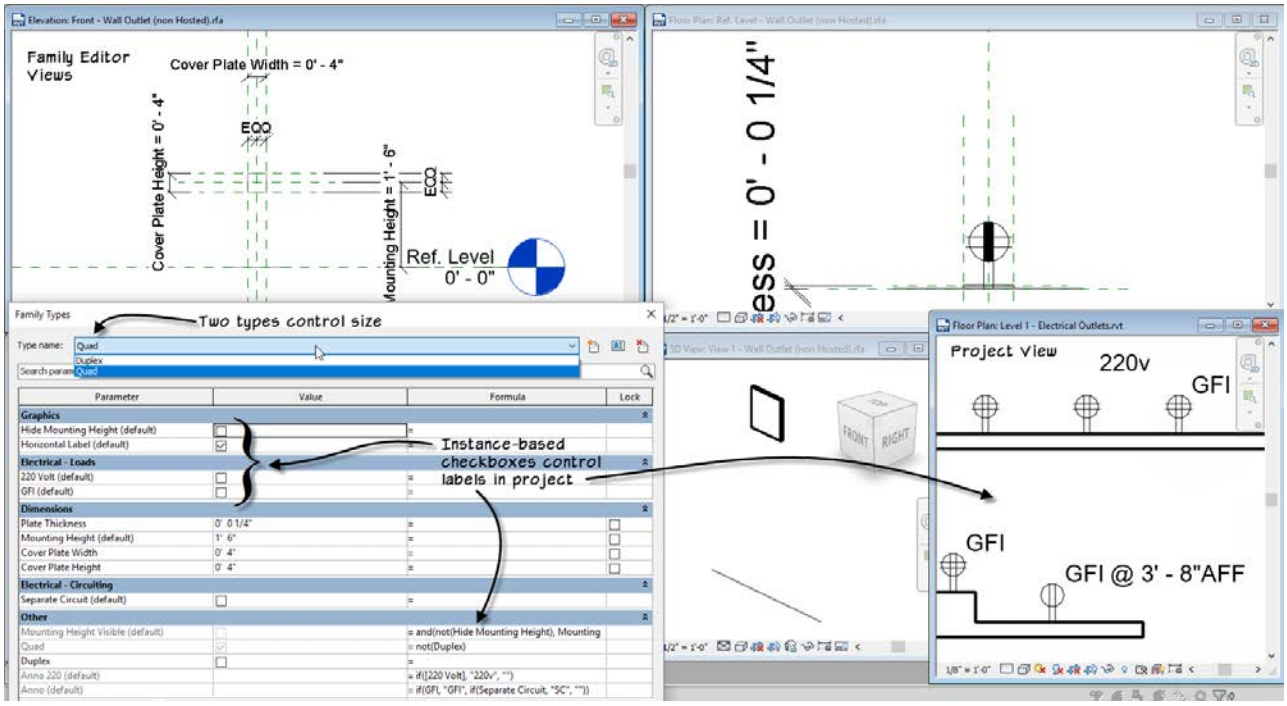


Figure 59—Electrical symbols with nested annotation families for plan views

Nested 2D Components

For the plan view, nest in one or more Generic Annotation families. Generic Annotation families respond to the scale of the view. This will allow the graphics in the nested family to adjust with the project view's scale. You can add as many custom shared parameters as required for the desired text labels (see Figure 60).

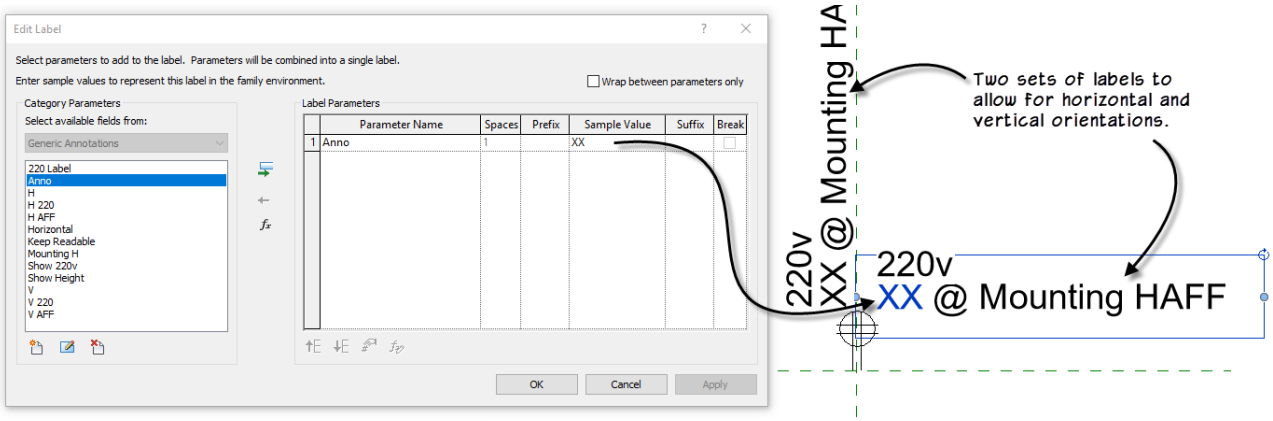


Figure 60—2D symbols created from Generic Annotation families with custom labels

Using IF, AND, OR and NOT conditional statements in the formula fields, you can control when each label displays, and which combinations are allowable.

This collection allows for all the labels required. They show in plan views, adjust to scale and react to the mounting height. However, since they are not hosted (face-based or otherwise) they can be placed anywhere without restriction. This can be advantageous in some cases (snap them to walls, columns, other surfaces, or just leave them freestanding

in open plan situations), but also means that these will *not* attach automatically to the faces of walls and other geometry. If you want them to move when the wall moves, you will need to use align and lock instead. So, like some many similar solutions, it is your job as the family author to carefully weigh the pros and cons of each approach you implement and be sure to clearly communicate the overall behavior and features to your team.

Parking Spaces

The out-of-the-box parking space family works well for simple layouts. It uses an angle parameter to change the angle of the spaces. It does not have an accessible space option. It also does not work especially well on curved layouts. Here I will show two alternative parking space families that address these issues.

General Parking Space Features

The parking space family itself has parameters to control the length and width of the space. There is also an angular parameter to control the angle of the space. In this family, that angle is used in formulas that employ some trigonometry rather than being applied directly to the geometry. The result is the same, but in many cases the trig can be more stable. This is because the calculation is performed mathematically regardless of the angle value. The space includes a stripe on both sides, but one of them is off by default. It has an instance parameter to control it. You can toggle this at just the ends that need it.

There is also a reference plane and parameter to control the minimum backup distance. You can use the align tool or dimensions with this reference plane to assist in placement. There is an accessibility symbol that can be toggled on and off in the accessible space. All families use type catalogs to manage the various types. Out-of-the-box subcategories were preserved. So certain parts of the families like guidelines and striping can have separate visibility settings. In coarse level of detail, striping changes to single line display (see Figure 61).

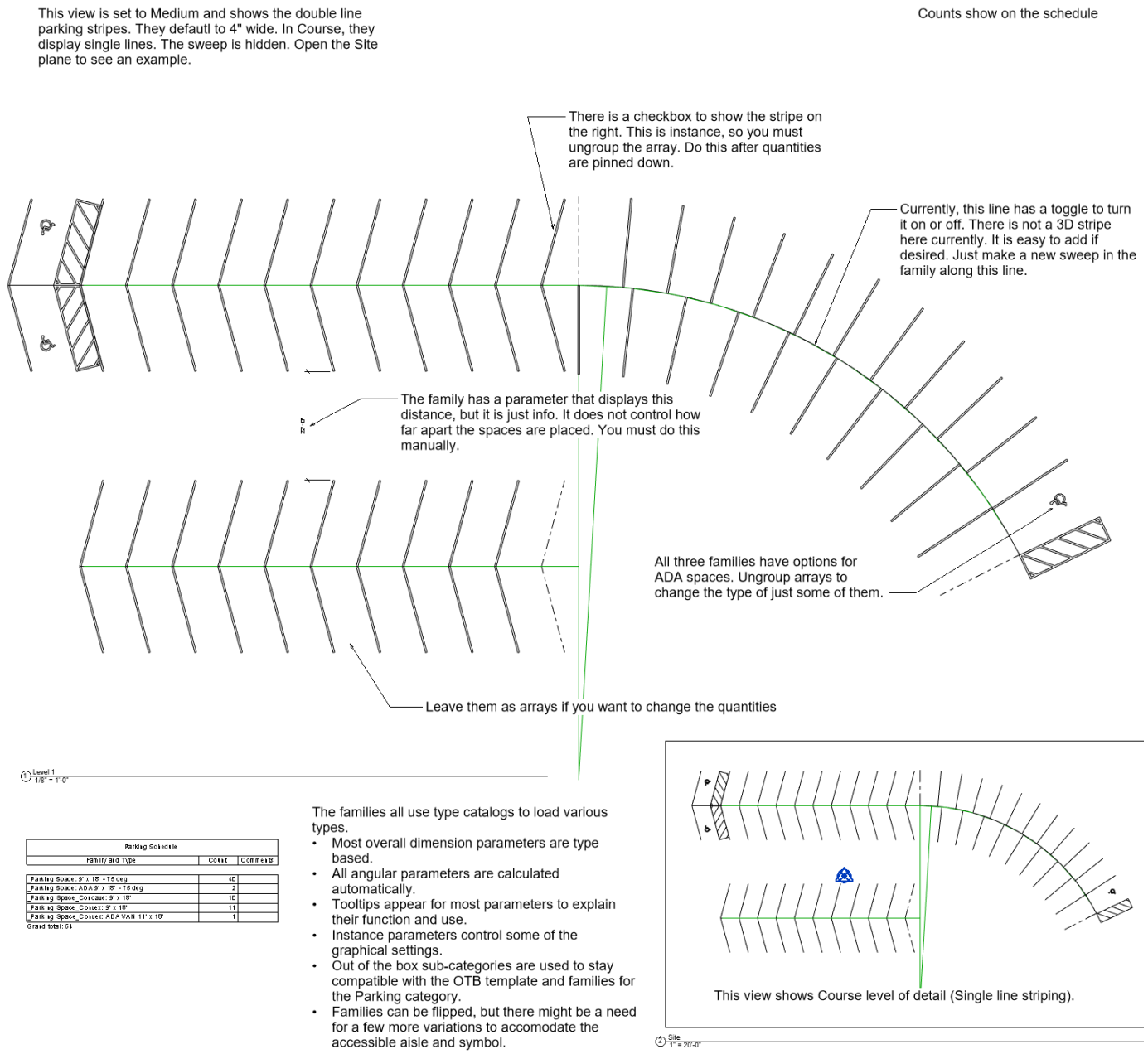


Figure 61—Parking space families

Parking Accessible Aisle

The parking space family has the two stripes, controlled by visibility toggles and simple trig formulas for the angles. The accessible aisle is a separate nested family (see Figure 62).

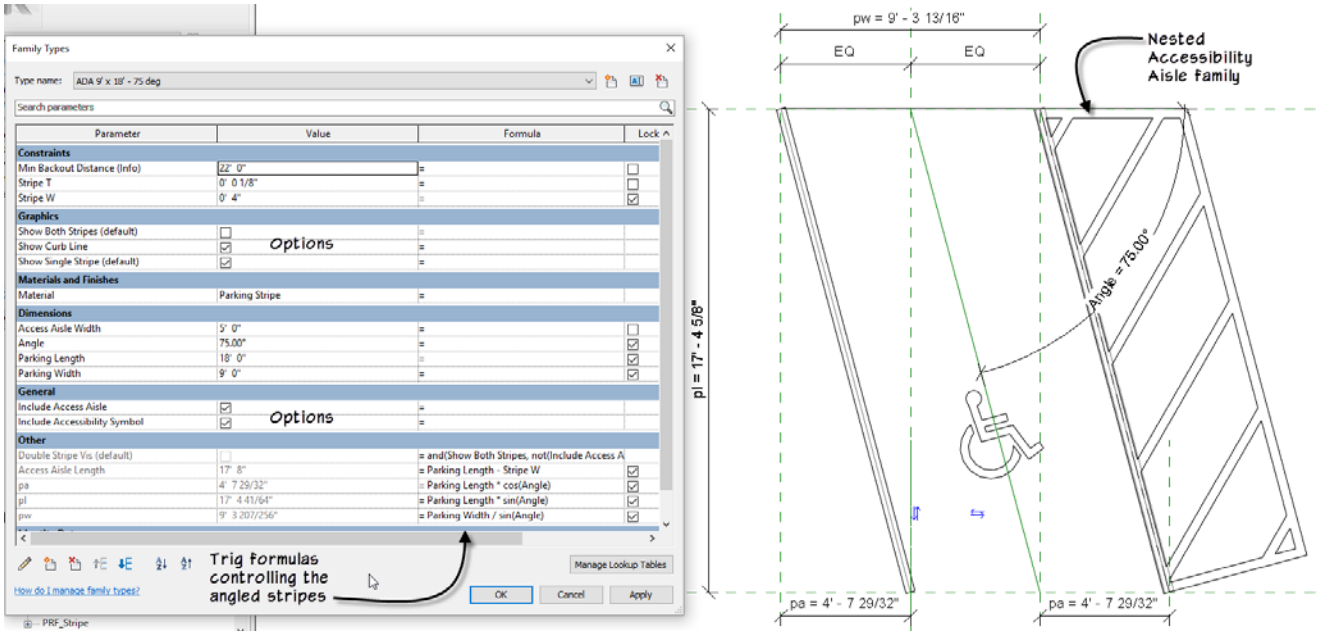


Figure 62—Parking space in the family editor

The real complexity in this family is buried inside the accessibility aisle nested family. The diagonal stripes in the aisle are at a fixed 4'-0" distance apart from one another and locked at a 45° angle to the sides of the aisle. However, when the angle of the parking space changes, the stripes near the top (particularly stripes 2 and 3) can terminate either on the opposite parallel side or on the horizontal. This condition necessitated some extra parameters, and conditional formulas (see Figure 63).

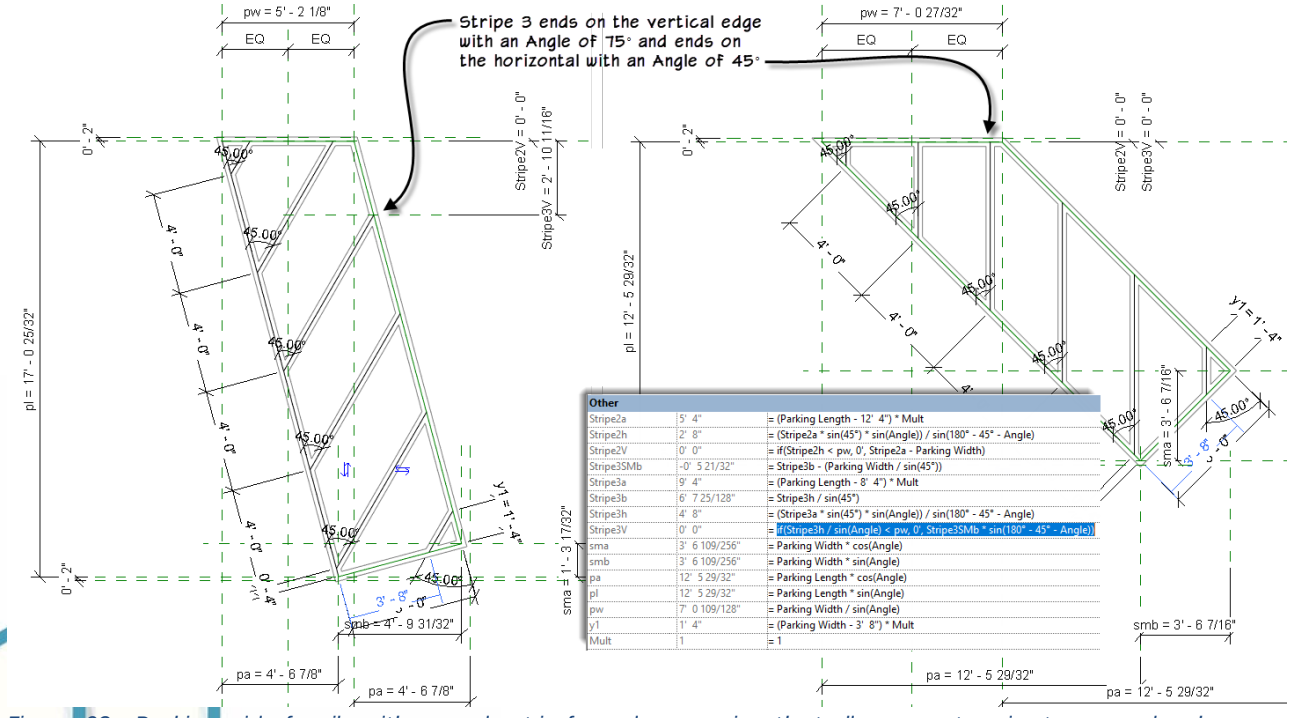


Figure 63—Parking aisle family with complex trig formulas ensuring that all sweeps terminate properly when angle is flexed

Everything is driven by the **Angle**, **Parking Length** and **Parking Width** parameters. The stripe geometry are sweeps. These use the Pick Path option and the paths are the model lines that are used for the coarse level of detail display. Therefore, the formulas are constraining these model lines (which are the centerlines of the sweeps). The collection of parameters at the bottom of the Other grouping (sma, smb, pa, pl and y1) are just like the ones in the parent family and shape the overall aisle.

The rest of the parameters and formulas serve the two IF formulas: (one for **Stripe2V** and the other for **Stripe3V**). Let's dig into the formulas for **Stripe3V** (see Table 1). Stripe3V sets the location of the right endpoint of the third stripe.

Table 1—Stripe 3 Formulas

Parameter	Formula
Stripe3V	if(**Stripe3h** / sin(Angle) < **pw**, 0', **Stripe3SMb** * sin(180° - 45° - **Angle**))
pw	**Parking Width** / sin(**Angle**)
Stripe3h	(**Stripe3a** * sin(45°) * sin(**Angle**)) / sin(180° - 45° - **Angle**)
Stripe3a	(**Parking Length** - 8' 4") * **Mult**
Mult	1
Stripe3SMb	**Stripe3b** - (**Parking Width** / sin(45°))
Stripe3b	**Stripe3h** / sin(45°)

I've colored the parameters in the table to make it easier to follow their relationships to each other. Trigonometry is about triangles[3]. We need to create these triangles strategically to help us figure out what we need from what we know. As noted above, the user inputs are the **Angle**, the **Parking Width** and the **Parking Length** parameters. We can use those in the formulas to form triangles and derive the other values we need.

Let's start with the value of **pw**. It is the same as **Parking Width** when the **Angle** is 90°. But when you rotate the aisle to another angle, **pw** becomes the hypotenuse of a triangle where we know a side and the angle. Thus, the SIN function helps us solve it.

Next, we have **Stripe3h**. If you take the third stripe and extend it to the right to make a triangle with the top edge and then consider the height of that triangle with the left side as the base, we are calling this height: **Stripe3h**. Since **Stripe3h** is non-horizontal for any value of **Angle** other than 90°, the first part of the formula uses the SIN function to find the length of the horizontal of component of this small triangle and compares to see if it is less

[3] At the end of this paper, I've provided a copy of the trigonometry cheat sheet for Revit that will assist you in determining the proper formulas based on what parts of the triangle you know.

than the value of **pw**. If so, then Stripe 3 attaches to the horizontal and the value of **Stripe3V** is zero. Otherwise, it calculates how far to shift the endpoint down. This formula has other parameters nested in it. So, let's look at those next.

The height of the main triangle is called: **Stripe3h**. This parameter's formula refers to another parameter: **Stripe3a** and some other trig functions derived from the law of sines. The law of sines can be used when you do not have a right-triangle, but you know all the angles and at least one side. This is what we have here. We know that one of the angles is 45° (this is fixed), and the other is the input value of the **Angle** parameter. And since we know that triangles are a total of 180° we can calculate the third angle as well. The law of sines says that the ratio of the angle to the side opposite the angle is equal for each such pair around the triangle. (The ratio of side a to angle A equals side b to angle B and side c to angle C).

We have the base of this non-right triangle (**Stripe3a**). **Stripe3a** is determined using simple arithmetic. It takes the length of the parking space (one of our inputs) and subtracts a fixed value from the constraints locked into the family with the dimensions noted above. The **Mult** parameter is used to prevent accidental edits. You can make a parameter a permanent constant by putting the value you want in the formula field. In this case, this is just a value of: 1. This allows us to multiply this by any other value and make the result read only. This is like the strategy noted in the "Version Identifier" topic above in Part 1 to add a read-only version ID to families.

With the length of **Stripe3a** and the angles, we can construct a formula using the law of sines, simplify it and use it to find **Stripe3h**. With that value, we can find the final values we need with simple trig functions again (see Figure 64).

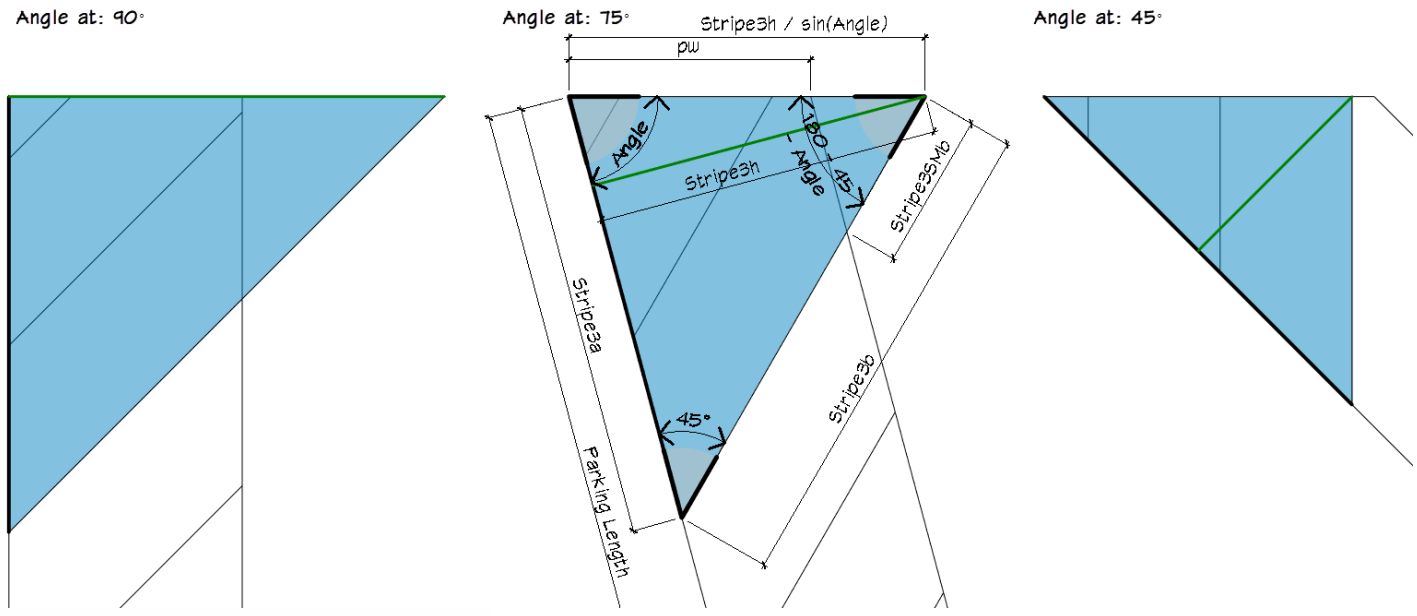


Figure 64—Understanding parking aisle formulas

Parking space following a curve

To create a parking space that follows a curve, the math is a little different. Here the goal is to have the stripes follow the radius of the curve. It is possible to create one family for this that you can toggle from concave to convex. I considered this, but I am always cognizant of creating families that are too complex. There is a fine line between complex functional families and so-called “super” families. Many of the families I have showcased in this session are right on the border to be sure. And some of you might argue that I missed the mark and have in fact created super families or excessively complex ones. However, even though the line between functional but complex and super is not empirically defined, a good way to remove complexity in a family is to simply make two families. If a toggle switch makes a major change to the family’s behavior, you might consider just doing a “save as” instead. That is what I did here. I have a convex version and a concave version. They are quite similar to one another, but since the formulas and trig already added a good deal of complexity, making two helped keep them manageable.

Comparing these two families to the straight one, there are a few major differences: The edges are not parallel to each other. They do not have the angle parameter, so they are always 90° relative to the curve. And their insertion point is centered on the parking space rather than being at the edge. Here is a look at the family (see Figure 65).

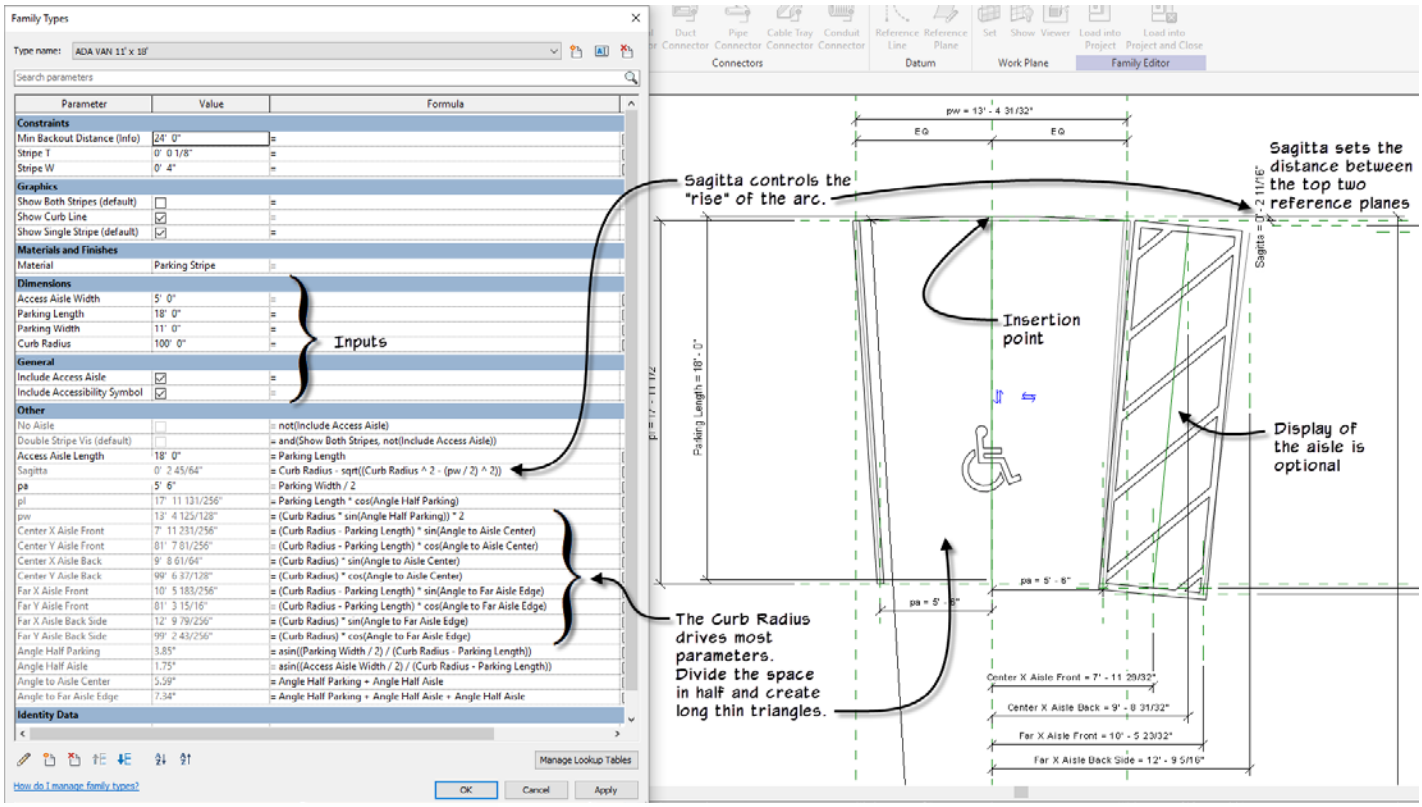


Figure 65—Parking along curves with the edges following the radius of the arc

As with the other parking family, the user has a few inputs like: **Parking Length** and **Parking Width**. This time however, instead of an Angle parameter, there is **Curb Radius**. You need two reference planes at the top. In the case of the space pictured, the top one is

the insertion point and the one below it sets the end points of the stripes. (It would be reversed in the convex version). You have a shallow arc passing between these two locations. The distance between these two, the rise of the arc, is called “Sagitta” in mathematics. You can calculate the **Sagitta** from a chord length and the radius. Both of which we have since these are the user inputs (see Figure 66).

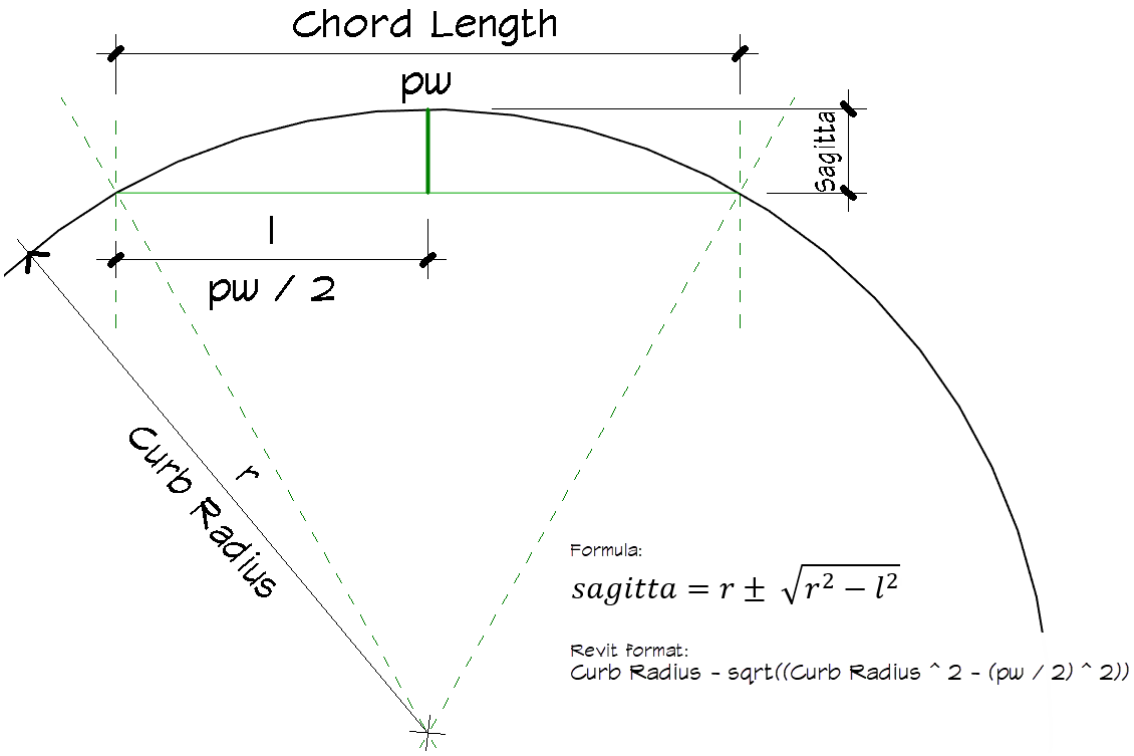


Figure 66—Understanding the formula for finding the “rise” of the curve known as the Sagitta

If you look carefully at that formula, it is just a derivation of the Pythagorean theorem. The endpoints of the stripes at the other end of the parking space form easy to identify triangles. If you refer back to the previous figure, you will see that those calculations are all based on the user inputs for parking length and width and deriving triangles from half the width of the parking space. When you add the accessible aisle, it gets slightly more complicated and we must calculate a few additional angles.

I am showing the concave curve here. The convex version is very similar. Everything is just reversed.

Shower Family

This topic will feature a shower family. There are several features built into this family that I would like to showcase here. These include graphics, cutable behavior, controlling options parametrically and a work around for creating a list of options.

Nested Family Visibility (Cutable or non-cutable)

Let’s start with the graphics. We wanted the shower pan to show as cut in sections (with a nice profile line around it). Trouble is, the Plumbing Fixture category is not cutable. So

what to do? If you insert a nested family that IS a cutable category, it will happily display as cut even while its host displays in projection. In this example, the solution was to nest in a Generic Model for the shower pan (see Figure 67). You can see which categories are cutable in Object Styles.

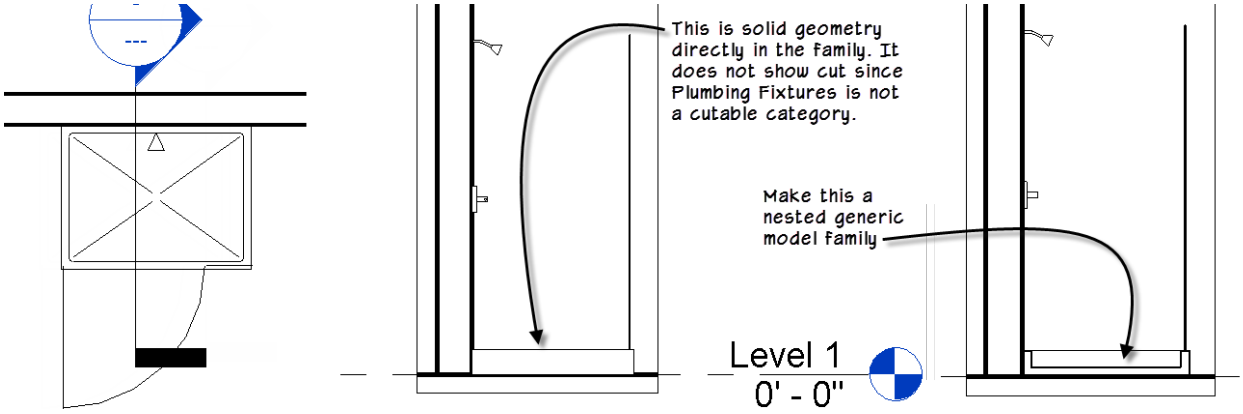


Figure 67—To make part of the family cutable, nest in a family in a cutable category

Nested Family Visibility (Shared vs Non-Shared)

When building the shower family, I had another challenge. When you cut through the shower and look in the opposite direction from the fixtures, they would still show in elevations even though the fixtures were behind the cut plane of the section line.

This is another manifestation of the same issue described in the previous tip. Remember, the Plumbing Fixture category is non-cutable. This means that when the object displays, the entire object will display (using the projection settings) regardless of whether it intersects the cut plane or not. The cut plane location does matter for cutable and non-cutable elements alike, but how the cut plane is used for each is quite different. In the case of a cutable element, if the cut plane is outside the element, but looking at it, the object will display in projection. If the cut slices through the element, then the part that is sliced will be profiled with the cut line weight from Object Styles and any part beyond the cut will display in projection. Any part behind the cut will *not* show at all. However, if the object is a non-cutable category, then the element will not be sliced. It will display as if you are seeing it in elevation regardless of where the cut line falls. For non-cutable categories, if any part of the object intersects the view extents, the entire object displays. Even if it is behind the cut plane.

The part that is potentially confusing is that the fixtures of the shower are a separate nested family. You might be left to assume that the nested family would use its own extents to determine if it should display. This is not the case. The host family's extents are used. That is unless the nested family is set to "Shared." In that case, it *does* use its own extents to determine if it displays (see Figure 68)! Confused yet?

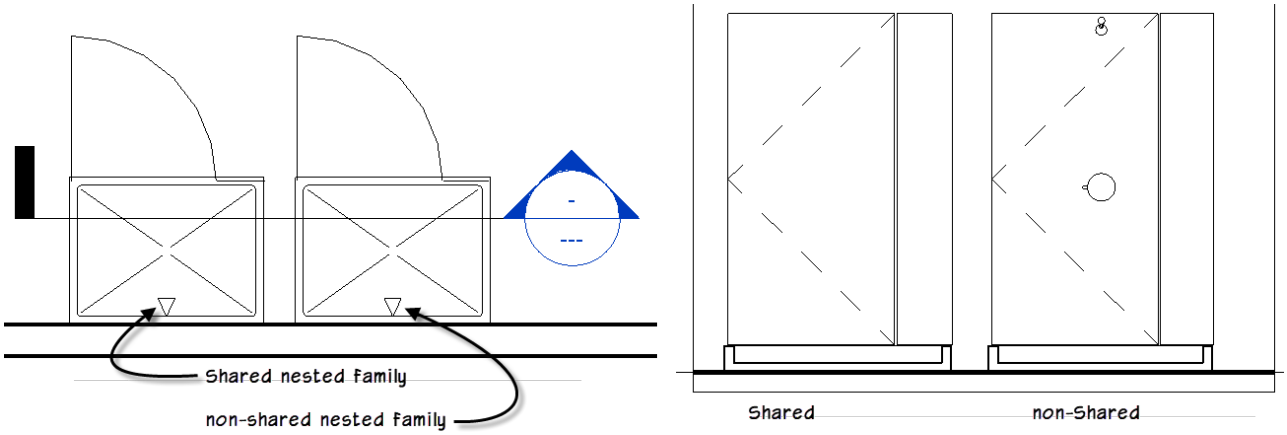


Figure 68—Shared nested families will not display if they are behind the cut plane

The bottom line is, if you use nested families and if you want the extents of the nested family (not the host) to be used when determining if it (the nested family) should display, then set the nested family to shared.

Invisible Shared Parameters

This is one of my favorites! Ever wanted to hide a parameter? Maybe you are creating a custom family and you have some detailed nested formulas (like the parking above). Parameter A is used in the formula of parameter B, which might in-turn drive parameter C. Wouldn't it be nice to not have to rely on just stashing the parameter in the "Other" grouping and hoping that your users don't mess with it? Well, guess what? You can make a parameter that shows in the family environment, can be used to drive formulas, but is hidden in the project environment!

To do this, you just perform a simple manual edit to your shared parameter file. Of course this means the parameter you want invisible must be a shared parameter. I like to create a special group in my shared parameter file just for my invisible parameters called: "Invisible Parameters." Further, I always name my invisible parameters with the prefix: **INV**_

There are two important steps to creating an invisible parameter. First, you must make the parameter invisible *before* you add it to a family. Second, you must make it invisible outside of Revit by manually editing the shared parameter TXT file[4]. Therefore, you might want to begin in a project instead of a family.

1. Open a project file (or create a temporary one) and create one or more shared parameters (Manage tab).

   Repeat for as many invisible parameters you need to create.

2. Close the project and do not save it.

[4] Since we are here at Midwest University, I should mention the wonderful tool that is part of the Revit Express Tools: Shared Parameter Manager. With this tool, you can create and modify shared parameters and enable features like invisible and non-editable switches without needing to manually edit the shared parameter file as discussed here. Not only is this a safer way to modify the shared parameter file, it is significantly simpler too! Check out the Revit Express Tools. You won't regret it. The Shared Parameter Manager is part of the CTC BIM Manager Suite.

3. Run Windows Notepad, browse to and open the shared parameter file.

The first two lines in the file will read:

```
# This is a Revit shared parameter file.
# Do not edit manually.
```

Well, we are going to ignore that. The group information comes first. Skip over that. Next you will see column headers. Look for the position of the VISIBLE column. This is the one we need to edit. Depending on your Revit version, VISIBLE might be the last column or the third column from the right. VISIBLE is a yes/no toggle and take values 0 and 1. Zero means the parameter is invisible and one means it is visible. All parameters default to: 1. Simply change any that you want to make invisible to: 0 (see Figure 69).

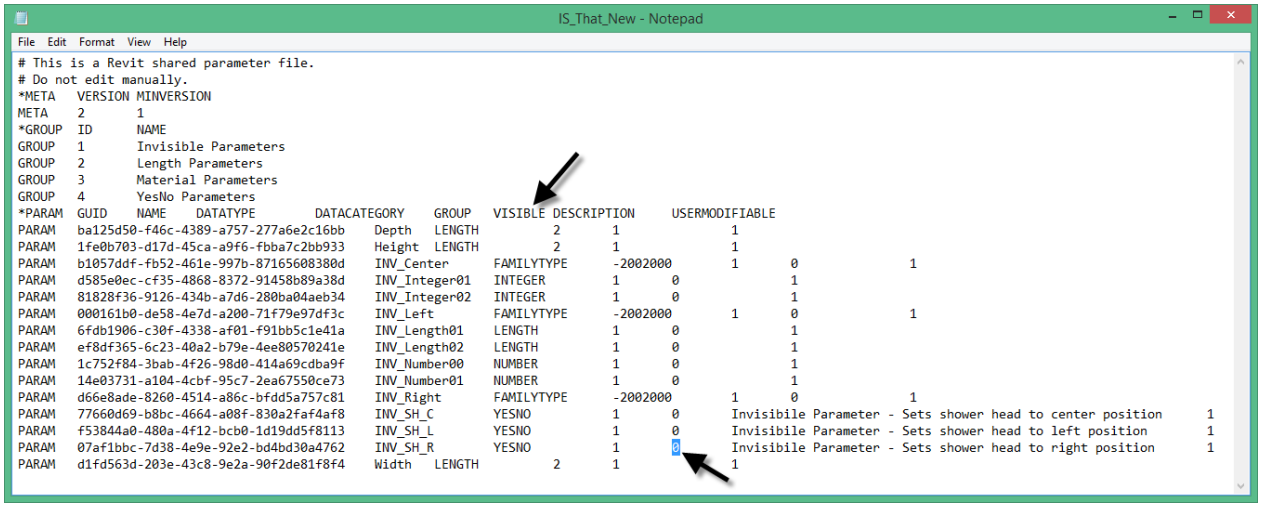


Figure 69—Set parameters you want invisible to zero in the VISIBLE column

4. When finished, close and save the shared parameter file.

Limit your edits to the VISIBLE column only[5] and you will not risk breaking the shared parameters in the file. Avoid editing the GUID or parameter names.

5. Now you can open a family and add the parameter(s) to the family.

6. Finish the family and then load it into a project.

[5] Depending on your Revit version, you might see a column called: USERMODIFIABLE. This is also a yes/no toggle with 1 equals yes and 0 equals no. If you want to create a parameter with a fixed value that cannot be edited, you can change to a 0 here. Just be sure that you create the parameter and set it the desired value first. The easiest way to do this is to create a Family parameter first. Then after you have assigned the desired value to it, you can edit the parameter and swap it out with the non-user modifiable shared parameter. The parameter will become read only immediately. This is a great way to create constant values or to protect detailed formulas that you don't want users to mess with. The CTC Shared Parameter Manager also makes it easy to create these kinds of parameters. Alternatively, you can use the same edit in Windows Notepad process discussed here.

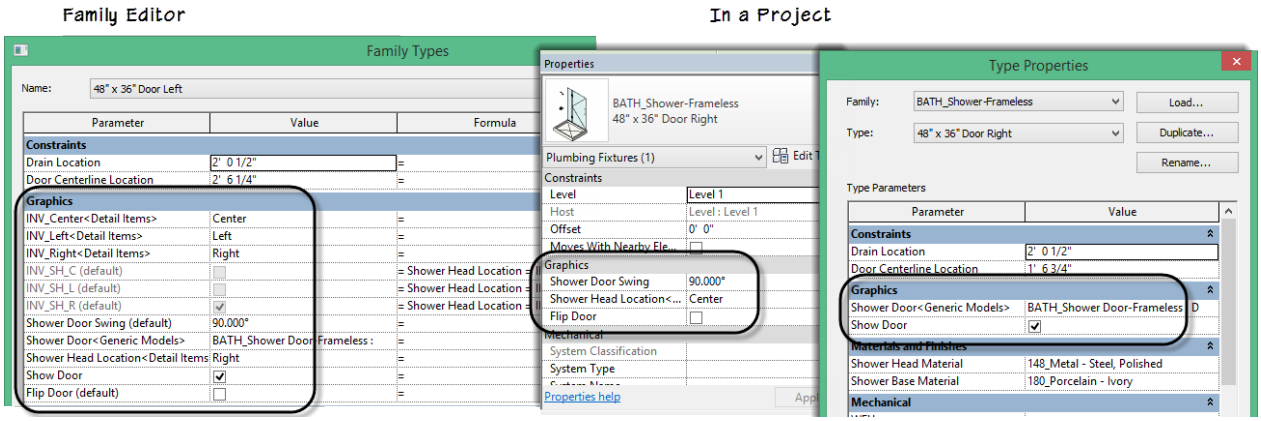


Figure 70—In the Family Editor (left) the parameters show, but in the project, they are invisible (right)

I know that you might be nervous about manually editing your shared parameter file. Well of course you will want to make a backup of it first. This way if something goes wrong, you're covered. One more note about this procedure. While the invisible parameters do not show in projects on the Properties palette, they *do* show in the list of available fields for schedules. There is no way to hide them from that list. This is one of the reasons I like to prefix the names with INV_. It makes them less appealing for users to add to schedules.

Make a "List" Parameter

I noted above that I have always wanted Revit to add a "list" parameter. By this I mean a parameter that will give a drop-down list of user-editable choices. Well, read on! Because by following this procedure you can! Sort of. For this example, I will continue with the shower family and add three possible mounting positions for the shower head and controls. You can use this technique any time you need three or more choices for an item. However, it does start to become a bit impractical if you need say five or more items, but your results may vary.

In this example, the shower head was copied to make three copies and positioned for left, right and center mounting options. We then want a list where the user can choose their mounting option. We want the list to read: Left, Right and Center. When they make their choice, we want only the selected option to appear and the other to be invisible.

Select each shower head family instance and associate a family parameter to its visibility. I used invisible parameters for these (see the previous tip). This is not required, but I think it ultimately makes a nicer solution. This is because you will not have to worry about users messing with the visibility parameters or being confused by them (see Figure 71).

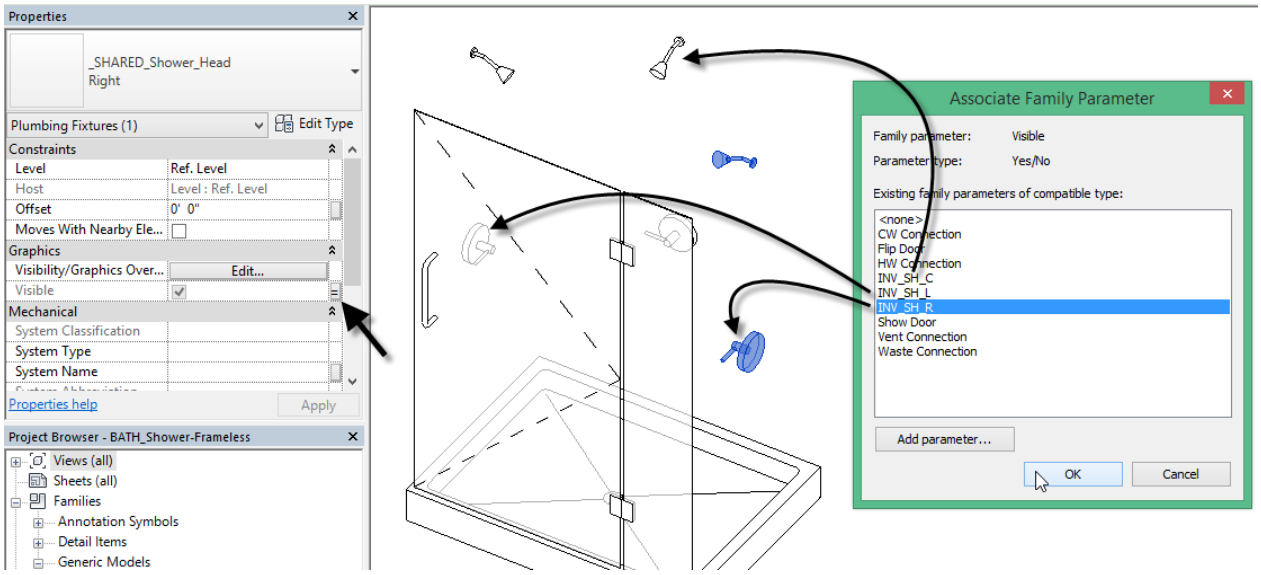


Figure 71—Assign a visibility parameter to each instance

Now that we have the three instances each controlled by its own Yes/No visibility parameter, it is time to move on and create the list. This will be achieved with **<Family Types>** parameters. We need three choices. So, we will need four <Family Types> parameters. One for the actual list, and the other three used for "comparison" in the formulas. This is how it works:

We make a <Family Types> parameter for each condition: Left, Right and Center. You set the value of each one permanently to one of these values. Then you have a fourth one that users can manipulate. In that fourth one, is where the list will appear. By choosing one of the options from the list, the formulas will do a comparison and thereby determine which item to chose and set the visibility parameters accordingly. I recommend invisible parameters for the comparison <Family Types> parameters. This will greatly reduce the possibility that an end user will inadvertently break the family. We can even use the feature (noted in the footnote above) that allows for the parameter to not be modifiable.

One last point. It doesn't really matter what category of <Family Types> parameter you use. But I recommend something that is not likely to be used elsewhere in the family. This is because the list of choices for the <Family Types> parameter will include *all* items loaded for that category. So, if you don't want extraneous items showing on the list, choose carefully. I chose Detail Items for this example.

The nice thing is that all Revit looks for is the presence of a family in the chosen category. In other words, you don't even need any geometry in the family. Just create a new Detail Item family (or whatever category you selected) and then save it. Repeat for as many items as you need on your list. In my case, I made three empty Detail Item families called: **Right**, **Left** and **Center**. If you only name the family and do NOT make any types in those families, the list will show just the family names. Otherwise, it will display both family and type in like this: Shower:Right, Shower:Left and Shower:Center.

These three empty families get loaded into the shower family. Then, following the procedures outlined in the previous topic, I made three invisible: **Family Types:Detail**

Items shared parameters. I called mine: **INV_Right**, **INV_Left** and **INV_Center**.

> **Important:** Remember to make them invisible by editing the TXT file *before* adding them to the shower family.

After making them invisible by editing the shared parameter file, add each of the invisible shared parameters to the shower family. You then want to assign the values: Set INV_Right to: Right. INV_Left to: Left and INV_Center to: Center (see Figure 72). Optionally replaced these with version of the shared parameters set to non-user modifiable.

These need to be set permanently to these values, or it will not work. That is why I recommend invisible parameters and/or non-modifiable. You can set it and forget it that way.

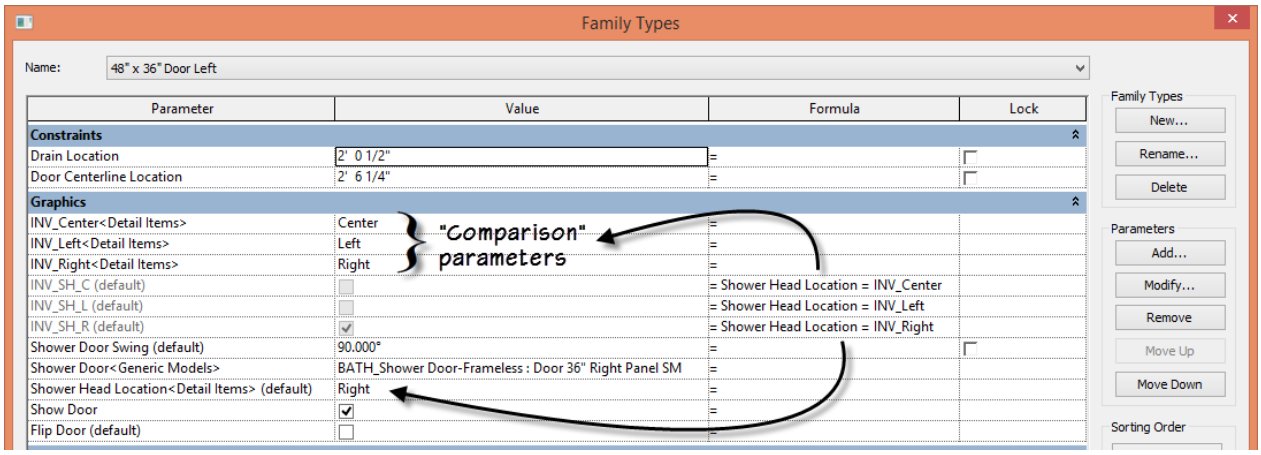


Figure 72—Set up family types to evaluate the choice from Shower Head Location and use it to set visibility

Almost there. We need one more parameter. This will be another <Family Types> that is *not* invisible and is modifiable. This is the one that will have the list from which the user will make their selection. For this I created another **Family Types:Detail Items** parameter called: **Shower Head Location**.

All four <Family Types> parameters will show a list of the three detail items we loaded into the file. The three invisible ones we permanently set to the values we needed as indicated above. When the user chooses one of the three values from the list in the Shower Head Location parameter, it will make that parameter equal to one of the three invisible comparison parameters. Using simple formulas, we can have this condition evaluated by each of the three Yes/No visibility parameters. In the formula field next to each checkbox parameter input the following:

INV_SH_C: **Shower Head Location = INV_Center**

INV_SH_L: **Shower Head Location = INV_Left**

INV_SH_R: **Shower Head Location = INV_Right**

We need the comparison parameters because we cannot specify the value we are looking for directly in the formula. So instead, we are simply asking: is the value the user's chose equal to this first one? No? how about the second or third? When it finds the one that is equal, it hides the others.

There you have it! You can make list parameters after all! It is a little tedious I grant you that. But I find it incredibly powerful. However, I would certainly prefer if the factory just adds a proper list parameter sometime soon.

As I noted, finding a good choice for the category of these list parameters is a challenge. This is particularly true if the nested families are shared. In that case, you will see all families of that category appear on the list once loaded into a project. Therefore, I highly recommend that if you use this trick, do not use a category where you are likely to have shared families. Often folks will use a category from another discipline, so if you are Architectural, try an MEP category for example.

In those casework items I discussed at the start of this paper, the client had lots of options that users can specify for their content. They wanted them to appear in lists where possible and the choices to appear on schedules. So we made shared parameters and looked for unused categories for <Family Types> lists. It turns out that some of the annotation categories were viable options (see Figure 73).

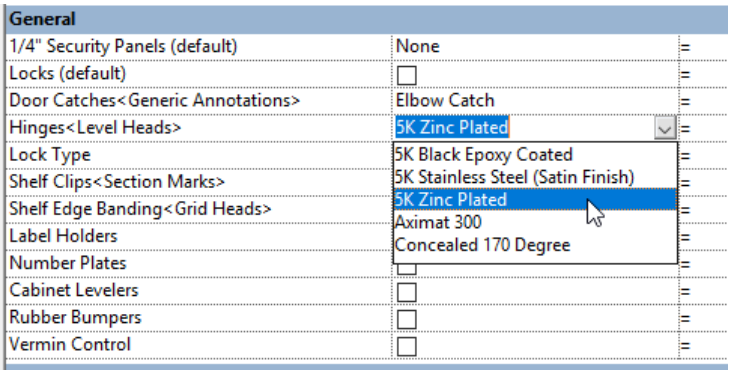


Figure 73—Using annotation categories for lists

In the figure you can see that the lists are being formed by the Generic Annotations, Level Heads, Section Marks and Grid Heads categories.

Datasets

Since most of the content showcased in these sessions was built specifically for my clients, I am not able to provide the files for download. The exception is the Volterra dataset. That project is hosted on C4R. If you are interested in taking a closer look at the file and the families it contains, please contact me after class via email. I will be happy to add you as a viewer to the C4R project.

paubin@paulaubin.com

Further Study

You can find more information and tutorials in:

***Renaissance Revit: Creating Classical Architecture with Modern Software*.** This book can be thought of as a "deep dive" into the family editor. It starts with the basics, but gets very advanced as well. The entire book is on family creation using classical architectural examples. Both the traditional and massing family editors are covered.

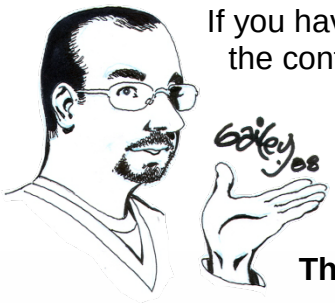
The Aubin Academy Revit Architecture: 2016 and beyond. Chapter 11 is devoted to the subject of the family editor.

The Aubin Academy Master Series: Revit MEP. Chapters 12 and 13 are devoted to the subject of the family editor.

Autodesk University courses: I have taught a family editor lab for several years at AU. I have also taught an advanced follow-up lab. Both class have papers and materials available for download from my website: www.paulaubin.com/au

Lynda.com
A LINKEDIN COMPANY

If you prefer video training, I have several Revit video courses at: **www.lynda.com/paulaubin**. Check out: *Revit Essential Training*, *Revit Family Editor*, *Revit Family Curves and Formulas* and *Revit Advanced Modeling*.



If you have any questions about this session or Revit in general, you can use the contact form at **www.paulaubin.com** to send me an email.

Follow me on Twitter: **@paulfaubin**

Thank you for attending. Please fill out your evaluation.

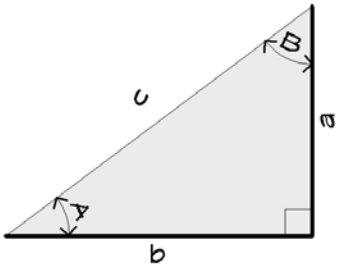
MIDWEST UNIVERSITY 2018

Trigonometry Cheat Sheet for Revit

Which parts are known?

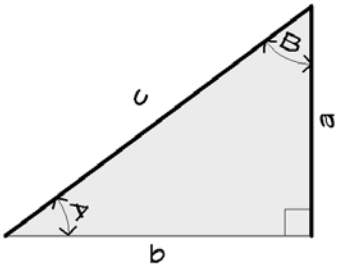
Two Sides

Known: a & b



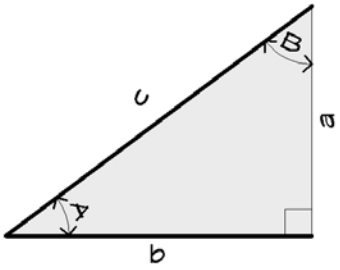
c = sqrt(a^2 + b^2)
A = atan(a / b)
B = atan(b / a)

Known: a & c



b = sqrt(c^2 - a^2)
A = asin(a / c)
B = acos(a / c)

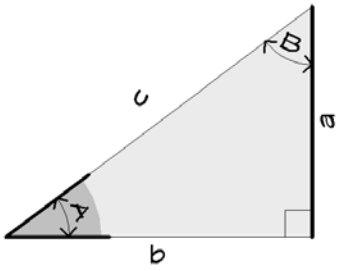
Known: b & c



a = sqrt(c^2 - b^2)
A = acos(b / c)
B = asin(b / c)

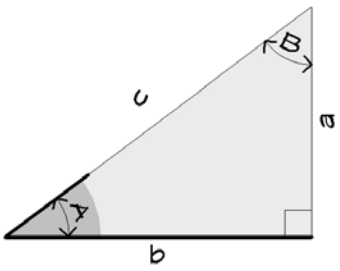
One Side & One Angle

Known: a & A



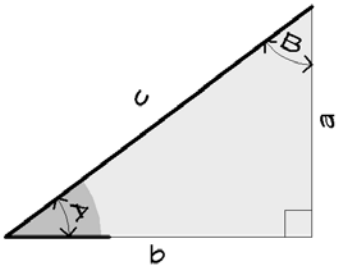
b = a / tan(A)
c = a / sin(A)
B = 90° - A

Known: b & A



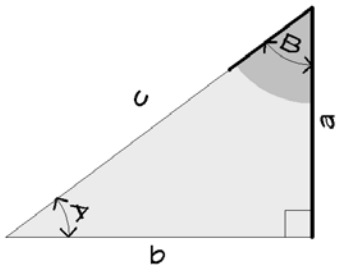
a = b * tan(A)
c = b / cos(A)
B = 90° - A

Known: c & A



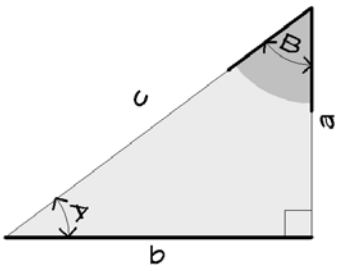
a = c * sin(A)
b = c * cos(A)
B = 90° - A

Known: a & B



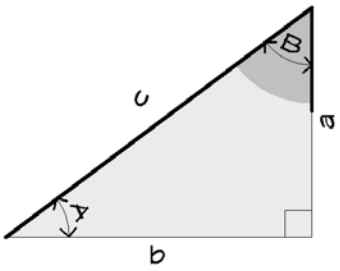
b = a * tan(B)
c = a / cos(B)
A = 90° - B

Known: b & B



a = b / tan(B)
c = b / sin(B)
A = 90° - B

Known: c & B



a = c * cos(B)
b = c * sin(B)
A = 90° - B