



AUTODESK UNIVERSITY 2015

ES10490

Programming Revit MEP? Say it Ain't Dynamo!

Jason Boehning

CADLearning by 4D Technologies

Learning Objectives

- Learn what the Dynamo extension is
- Discover what visual programming is
- Discover practical examples where Dynamo can speed up processes in Revit MEP
- Discover large changes that can be made quickly with Dynamo

Description

Are you a Revit MEP software user who feels left behind in the development of Dynamo software? Well, not so fast! You can use Dynamo software to help expedite many common MEP workflows. The Dynamo extension is a program that uses visual programming, so you don't have to worry about trying to learn difficult programming languages. In this class you'll get to know the basics of Dynamo software and how it interacts with Revit software. We will also cover several examples where you can use Dynamo software to save time during the MEP design process. This lecture is ideal for Revit MEP software users who want to see what Dynamo software can do. Even better, no programming experience is required! Afterward, you will be able to implement Dynamo software immediately.

Speaker Profile

Jason Boehning is the Building Content Manager at 4D Technologies and authors Revit and Dynamo courses for AEC design professionals. After graduating from Texas A&M University in 2009, Jason helped implement Revit into the design process at an engineering firm in Houston, TX. He was later promoted to BIM Technology Manager and increased productivity for sustainable design and energy modeling using Revit. He also served as the 2011 President of the Houston Chapter of the US International Building Performance Simulation Association. In 2012, Jason decided to make a career of teaching building professionals how to use Revit and joined 4D Technologies in development of their CADLearning® products, helping create affordable training for Autodesk software. Jason is also a contributing author to several CADLearning eBooks. He is also a repeat speaker at RTC North America and Autodesk University. Jason is a Certified Professional for Revit Architecture, Revit Structure, and Revit MEP.

Table of Contents

Learning Objectives.....	1
Description.....	1
Speaker Profile	1
Table of Contents.....	2
Introduction	3
About This Paper.....	3
What is Dynamo?.....	4
What is Visual Programming?.....	5
Using Dynamo with Revit.....	8
Rename and Renumber Space Example	8
Selecting Elements.....	11
Selecting Model Elements.....	12
Select Elements by Category.....	13
Selecting System Families and Elements in a System Family	14
Selecting Loadable Family Types and Elements in a Loadable Family Type	15
Understanding Lists	16
Getting Parameters.....	16
Wall Parameters Example	18
Space Parameters Example.....	19
Exporting to Excel	21
Importing From Excel.....	21
Space Parameters from Excel Example.....	22
Setting Parameters	24
Duct Example	25
Lighting Fixture Example	25
Plumbing Fixture Example	26
Units	26
Advanced Examples	26
Setting Multiple Parameters for All Spaces	26
Setting Air Terminal Flow Based on Space Requirements	30
Conclusion.....	31



Introduction

I want to begin by letting you know that I am not a computer programmer – not even close. However, once I began using Revit, I realized that there are certain things you can program. And by program, I simply mean inputting a coded instruction to automate a task. For example, you can input formulas into family parameters. In the **Family Types** dialog, there is a **Formula** column. This allows you to program family parameters. You simply add a formula that will automatically set that parameter value.

As I became more and more comfortable with programming some simple tasks, I wanted to learn more. That's when I started trying to learn .net programming so I could interact with the Revit API. That's also when I hit a wall. I just did not have the time to learn everything that was required to be able to program effectively. Also around that time, I heard about Dynamo. I quickly found out that I could become a visual programmer in Dynamo and accomplish the same things I wanted to with traditional programming.

About This Paper

This paper is a combination of step-by-step examples and me sharing my thoughts and best practices for using Dynamo. I hope you find each very beneficial. The bolded text throughout the paper indicates User Interface components in either Revit or Dynamo. So when you see bold, you can look for it in the program's UI.



What is Dynamo?

First of all, Dynamo is developing so fast and changing so much that it's hard to say what Dynamo actually is. At the time of AU 2015, there are actually two versions of Dynamo. There is Dynamo Studio, which is a stand-alone Autodesk product, and there is the Dynamo extension, or add-in, for Revit.

Both are visual programming tools. Dynamo Studio provides a graphical programming interface for computational design. This paper does not focus on Dynamo Studio, but a great deal of the content covered in this paper also applies to Dynamo Studio. This paper focuses on the Dynamo add-in for Revit, which I will refer to simply as “Dynamo”. At this time, the add-in is free!

Now you are probably wondering – what does Dynamo do? This is actually a very difficult question to answer simply because Dynamo does so much. Essentially, it is a visual programming tool that can create its own geometry with parametric relationships, and it reads from and writes to external databases.

Revit is a database of information with parametric geometry, so Dynamo is a perfect fit to interact with Revit. Dynamo communicates with Revit through the Revit API.

To get started with Dynamo, go to www.DynamoBIM.com and download the latest version. There are actually daily releases of the program and stable releases every 2-3 months. If you are just trying it for the first time, start with a stable release. As you become more familiar with the program, you may want to download and install the daily builds so you can see what the developers are working on.

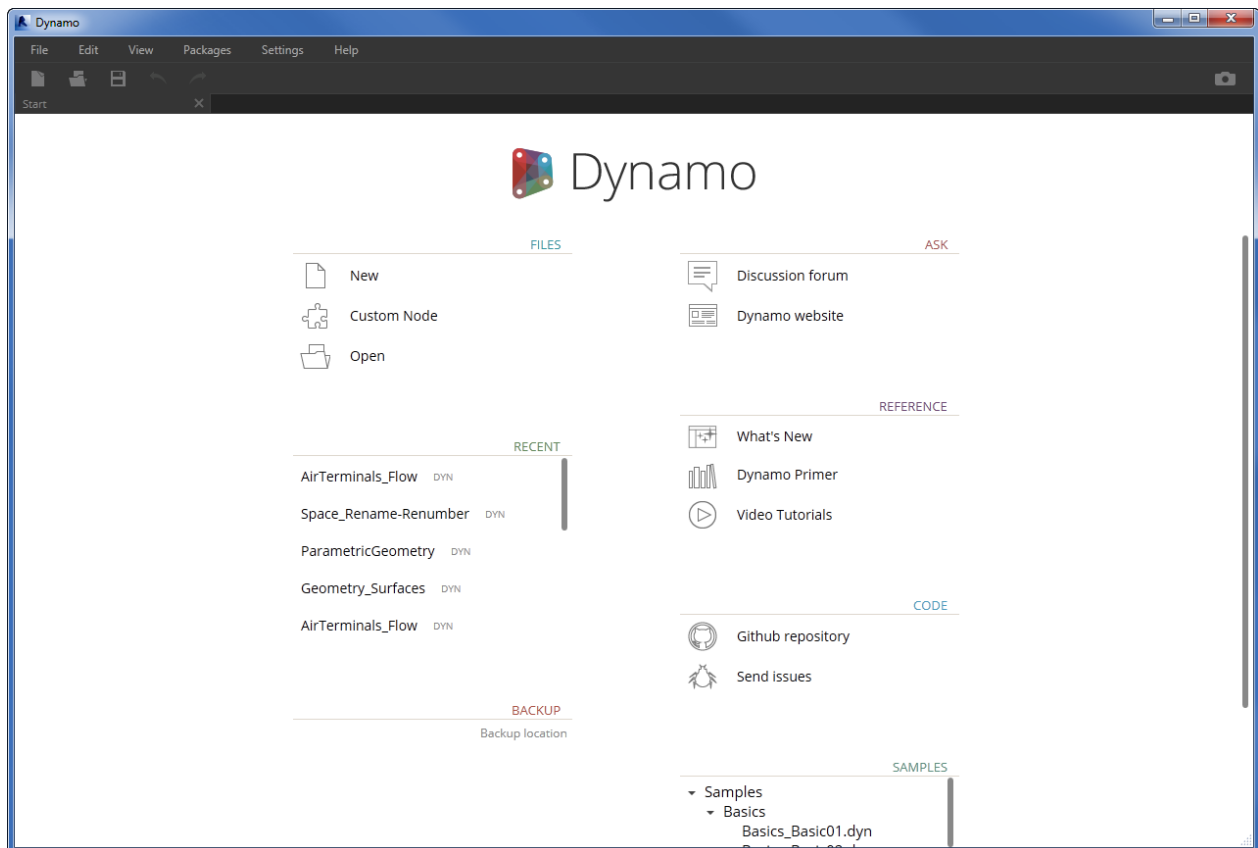


FIGURE 1: THE DYNAMO START PAGE THAT APPEARS WHEN YOU OPEN THE PROGRAM.



What is Visual Programming?

Dynamo is a visual programming language. As with any type of programming, there must be some type of source code or commands. The code then creates a program that will automate a specific task or solve a given problem. In Dynamo, the code is created by placing nodes. Nodes are the building blocks of a Dynamo graph or program.

With other programming languages, the source code is text-based. However, you must understand the specific language. You cannot just enter any text. For this reason, it takes time to learn and understand the programming language. With visual programming in Dynamo, the learning curve is not as steep as with other languages.

Take a look at a simple example. To create a program that adds two numbers, you can use an addition (+) node and two **Number** nodes. To place a node, first find the node listing in the **Library**. For this example, click **Operators**. When you do, the category expands and you can see all the available nodes.

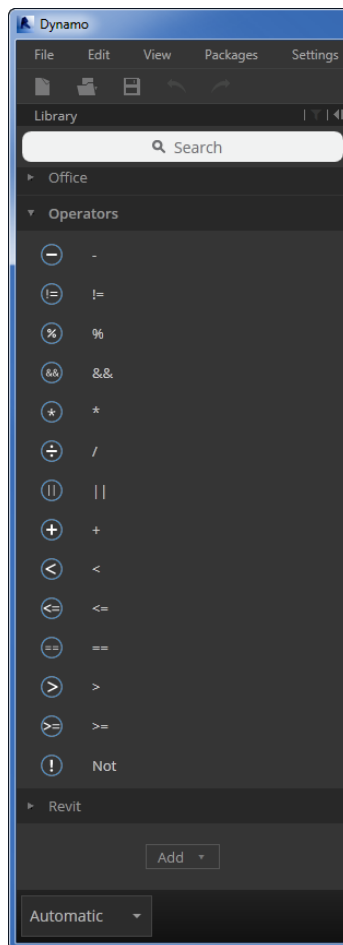


FIGURE 2: ALL DYNAMO NODES CAN BE ACCESSED FROM THE NODE LIBRARY.

To place a node onto the canvas, simply click the node listing in the **Library**. To place the addition node, click the listing with the plus icon (+). You can also use the **Search** field to search for nodes. Enter "number" to search for the **Number** node. Once the **Number** node listing appears, click it to place the node on the canvas. Click it again so that two **Number** nodes are on the canvas.



Most nodes have both input and output ports. However, some nodes, such as **Number** and **String** nodes, only have output ports. This is because the information is input directly into the node. Arrange the nodes so that the **Number** nodes are to the left of the addition node. Next, enter **4** into one of the **Number** nodes and **2** into the other. With information specified in the nodes, you must connect them in order for the information to be processed.

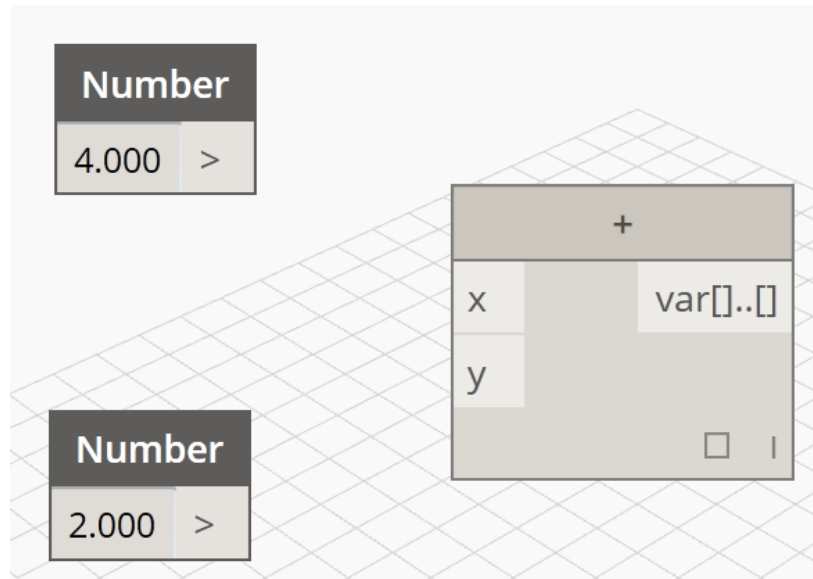


FIGURE 3: NODES PLACED ON THE CANVAS WITH INFORMATION.

To connect nodes, click the output port and then click the input port. This creates a connector (also referred to as a wire) between the nodes. Be aware that you can place multiple connectors from an output port, but only one connector can be input into an input port.

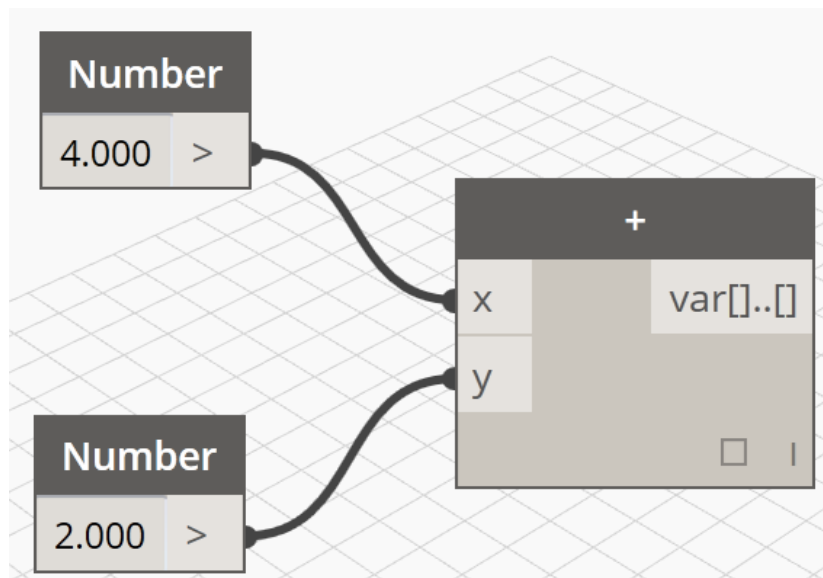


FIGURE 4: NODES CONNECTED TO SPECIFY THE FLOW OF INFORMATION.

Once nodes are placed, information is specified, and the nodes are connected, a graph is created. You must then run the graph in order to process the information and create an output. There is a **Run** drop-down in the lower-left corner of the Dynamo application.

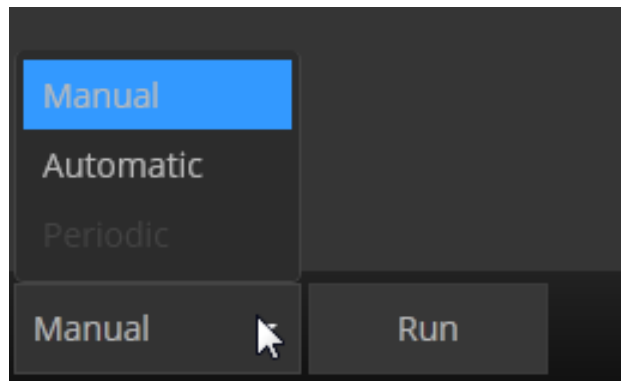


FIGURE 5: A RUN DROP-DOWN IS AVAILABLE IN THE LOWER-LEFT CORNER OF THE APPLICATION.

There are three options, but you will probably just work with two. When set to **Manual**, there is a **Run** button. You can click **Run** to run the graph. When set to **Automatic**, the graph is continually running. When working with Revit, it is always a good idea to have the **Manual** option selected. Sometimes you can perform an unintended action while you are still creating your graph.

Back to the simple addition example, when the graph is run, there is an output. You can hover your cursor over the output indicator to see the output in a notification. When you select the output indicator, the notification will remain visible as you continue to work.

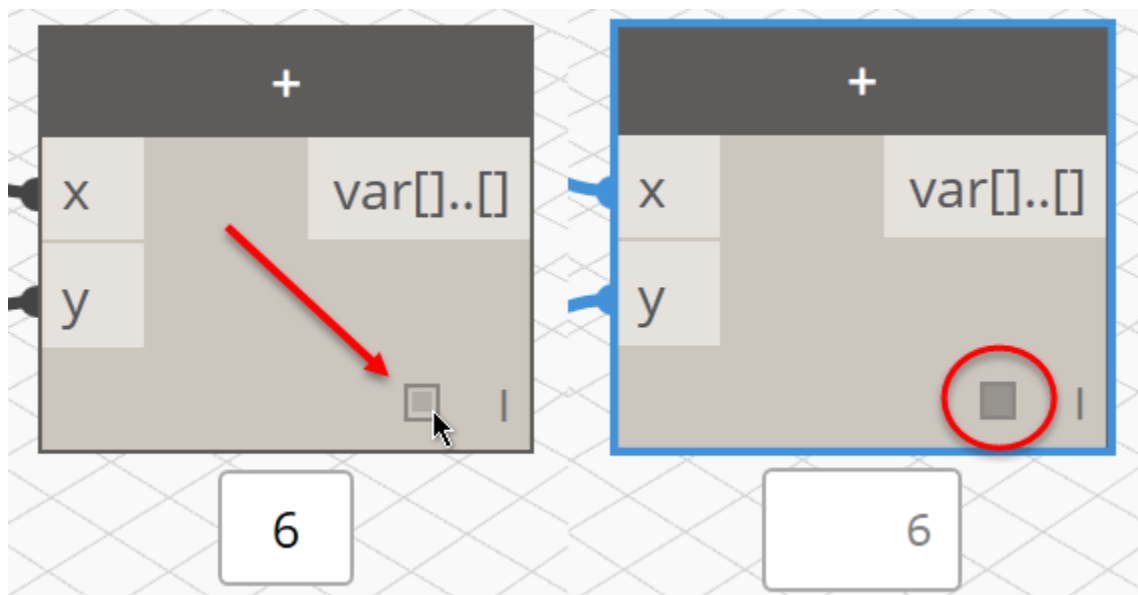


FIGURE 6: THE OUTPUT INDICATOR SHOWS THE INFORMATION THAT IS OUTPUT FROM THE NODE.

This is obviously a simple example, but it shows you the basics of visual programming. Now it is time to see the power of Dynamo and what you can do by simply placing and connecting nodes.

Using Dynamo with Revit

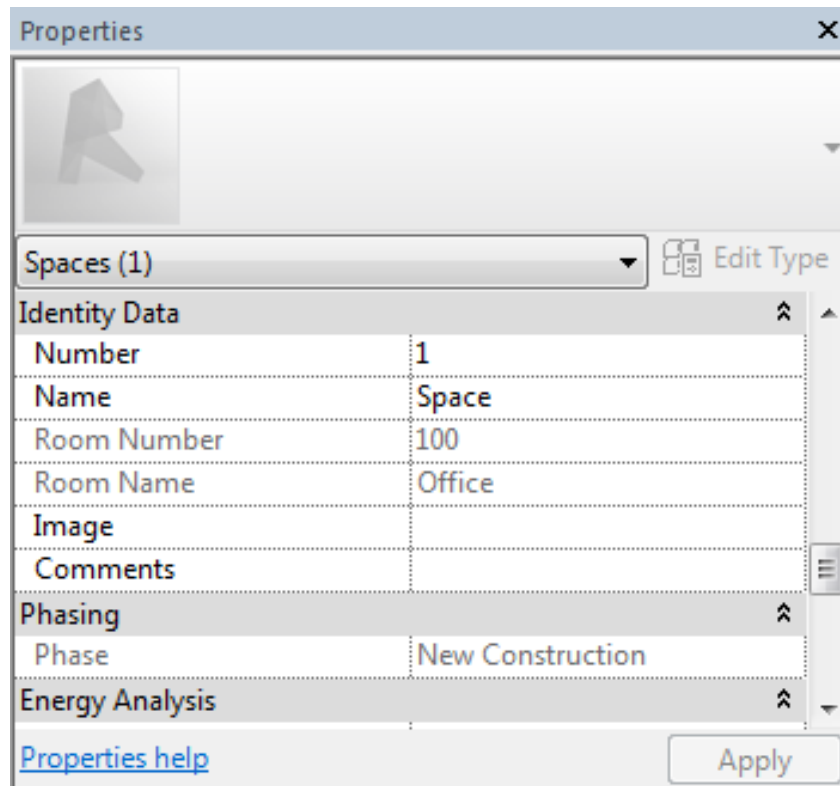
As I've already stated, Dynamo is a very powerful tool. Revit is also a very powerful tool. When you combine the two, the possibilities seem endless. Dynamo is still being developed at an extremely fast pace, so there is still more to come!

I will focus on what Dynamo can do in Revit—more specifically, on what Dynamo can do with MEP elements. This paper just scratches the surface of what you can do with Dynamo in Revit on the MEP side. Additionally, I will just focus on the database side of Dynamo. In other words, I won't go over using Dynamo to create geometry in Revit. I will stick to using Dynamo to get and set parameters in Revit. That's the main focus of MEP, right?

Rename and Renumber Space Example

To get a glimpse of what Dynamo can do in Revit, take a look at an example where you use Dynamo to rename and renumber spaces to match the architectural room names and room numbers.

In Revit, space elements have a **Number** and a **Name** parameter. They also have **Room Number** and **Room Name** parameters that are read-only. They are read-only because they reference the corresponding room element from the same model or a linked architectural model.



The screenshot shows the Revit Properties window for a Space element. The 'Spaces (1)' dropdown is selected, and the 'Edit Type' button is visible. The 'Identity Data' section is expanded, showing the following parameters:

Identity Data	
Number	1
Name	Space
Room Number	100
Room Name	Office
Image	
Comments	
Phasing	
Phase	New Construction
Energy Analysis	

At the bottom of the window, there is a 'Properties help' link and an 'Apply' button.

FIGURE 7: IDENTITY DATA PARAMETERS FOR A SPACE ELEMENT.



You can use Dynamo to get the value of the **Room Name** parameter and then set the **Name** parameter to that value. Similarly, you can use the same process to set the **Number** to the **Room Number**.

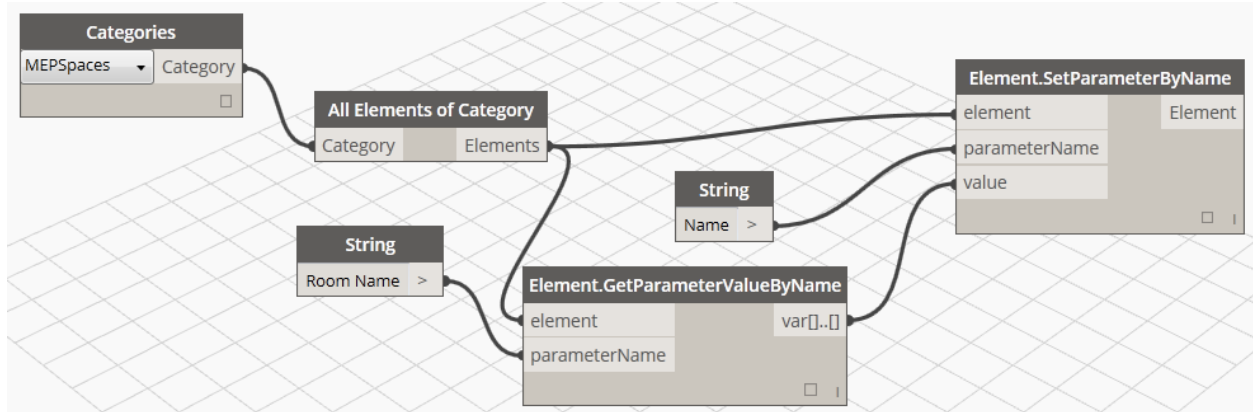


FIGURE 8: DYNAMO GRAPH THAT CHANGES THE SPACE NAME TO MATCH THE ROOM NAME.

First, you need to select all the space elements in the Revit model. Next, you need to get the **Room Name** parameter for each space element. After that, you can use those values to set the **Name** parameter to match. Figure 6 shows the graph that accomplishes this. The two nodes that you will use often are **Element.GetParameterValueByName** and **Element.SetParameterByName**.

To renumber the spaces, you can either create a new graph and then change the parameter names, or you can copy the get and set nodes and then change the parameter names. Typically, there are multiple ways to accomplish something in Dynamo.

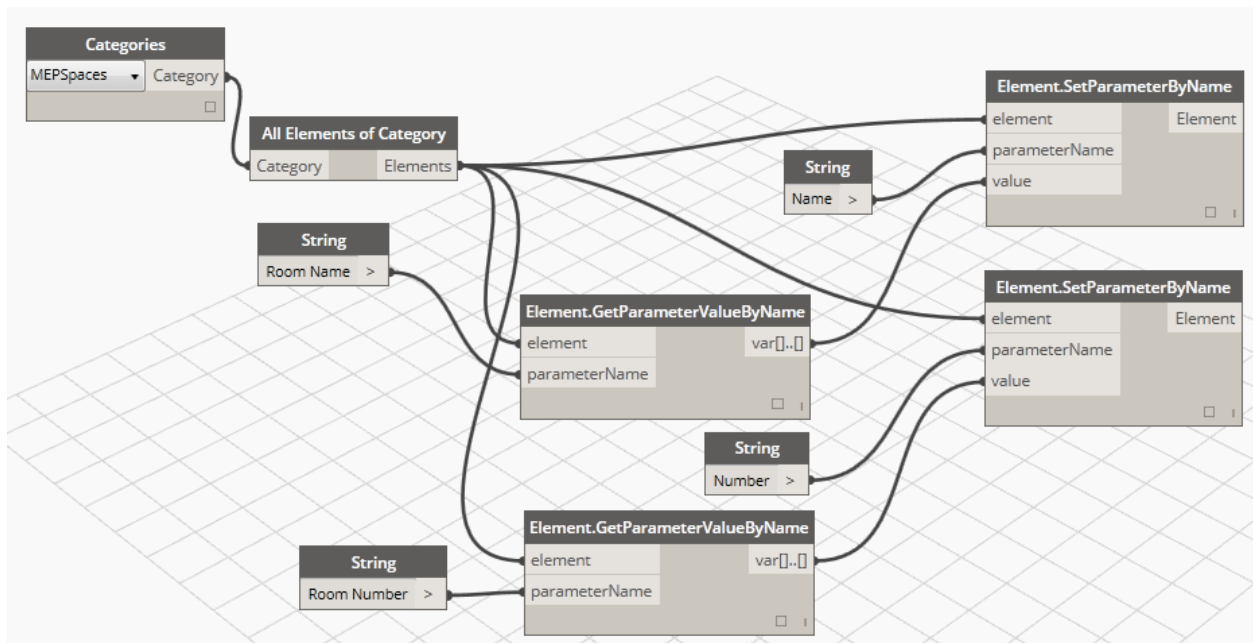


FIGURE 9: DYNAMO GRAPH THAT CHANGES THE SPACE NAME AND NUMBER TO MATCH THE ROOM NAME AND ROOM NUMBER, RESPECTIVELY.



When you have this graph saved in a Dynamo workspace, you can simply run it once it is connected to the Revit project that you want. Once you do, the space names will be changed to match the room names and the space numbers will be changed to match the room numbers.

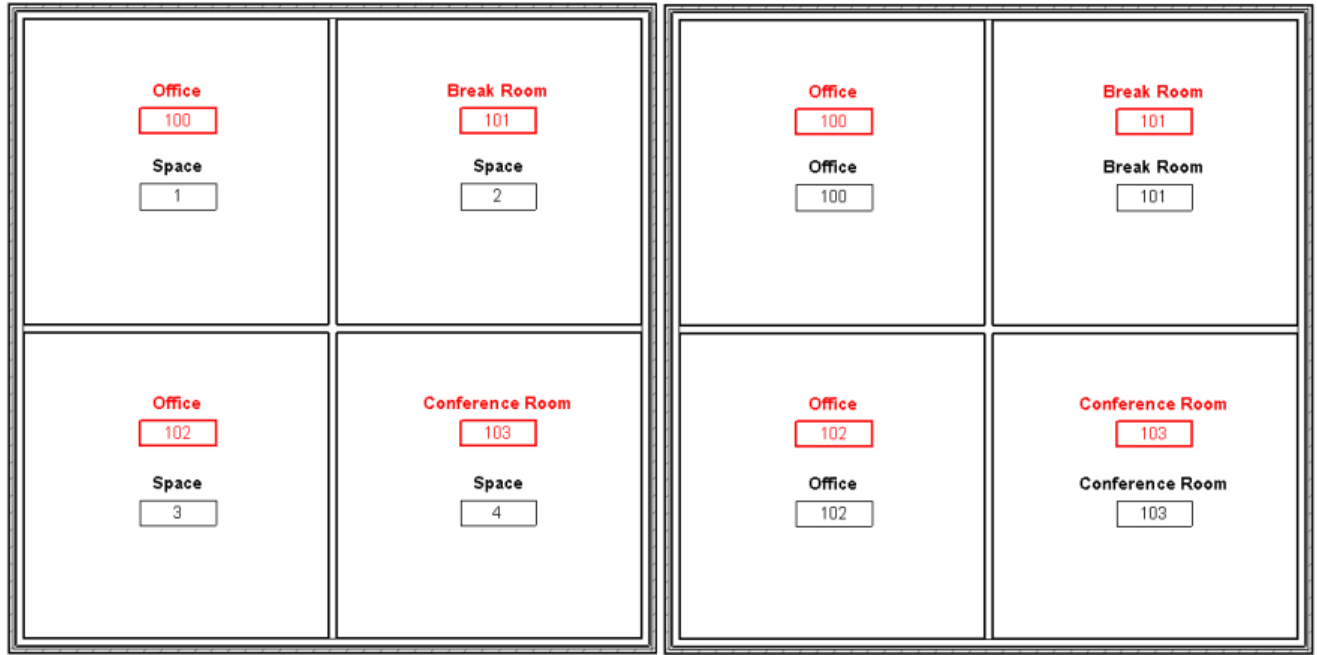


FIGURE 10: SPACE NAMES AND NUMBERS BEING UPDATED USING DYNAMO.

In this example, three things happened. First, Dynamo was used to select elements in Revit. Next, parameter values from those elements were extracted. Finally, the parameter values were changed in Revit. When working with MEP elements, you will typically either get parameter values or set parameter values, or in some cases, you will do both. But before you do any of this, you must select the elements you are going to work with.

Selecting Elements

In order to perform some type of action on a Revit element or elements, you must first select the single element or multiple elements. There are several ways to use Dynamo to select elements in Revit. The **Selection** nodes can be found in the node **Library**, under the **Revit** category. Simply click **Selection** to expand the subcategory. Here, there are multiple nodes that you can use to select elements in Revit.

Take a look at the main selection methods for MEP elements. You can select a single model element or multiple model elements. You can also select elements by category. Additionally, you can select all the elements belonging to a system family or all the elements belonging to a loadable family type. Some of the methods require two nodes. Figure 11 shows the **Selection** nodes in the node **Library**. They are circled and color-coded to indicate which nodes work together.

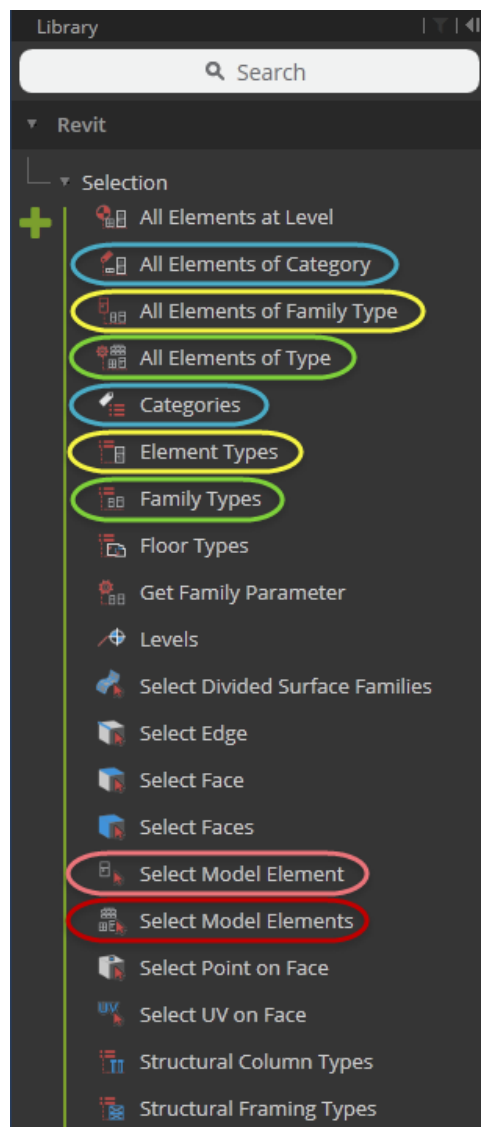


FIGURE 11: REVIT SELECTION NODES IN THE NODE LIBRARY.



Selecting Model Elements

You can select a single model element with the **Select Model Element** node, or you can select multiple model elements with the **Select Model Elements** node. Pay special attention to which node you place on the canvas.

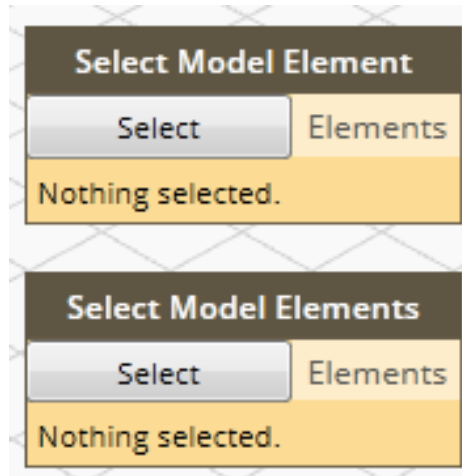


FIGURE 12: THE SELECT NODES HAVE A SELECT BUTTON SO YOU CAN MANUALLY SELECT ELEMENTS IN REVIT.

When using the select nodes, you must first click **Select** in the node in Dynamo. Then you can manually select elements in Revit. With the singular select node, you must click to select a single element. With the plural select node, you must use a window to select multiple elements. For that reason, it is best to prepare the view in Revit. For example, you can isolate the elements you want to select in a view.

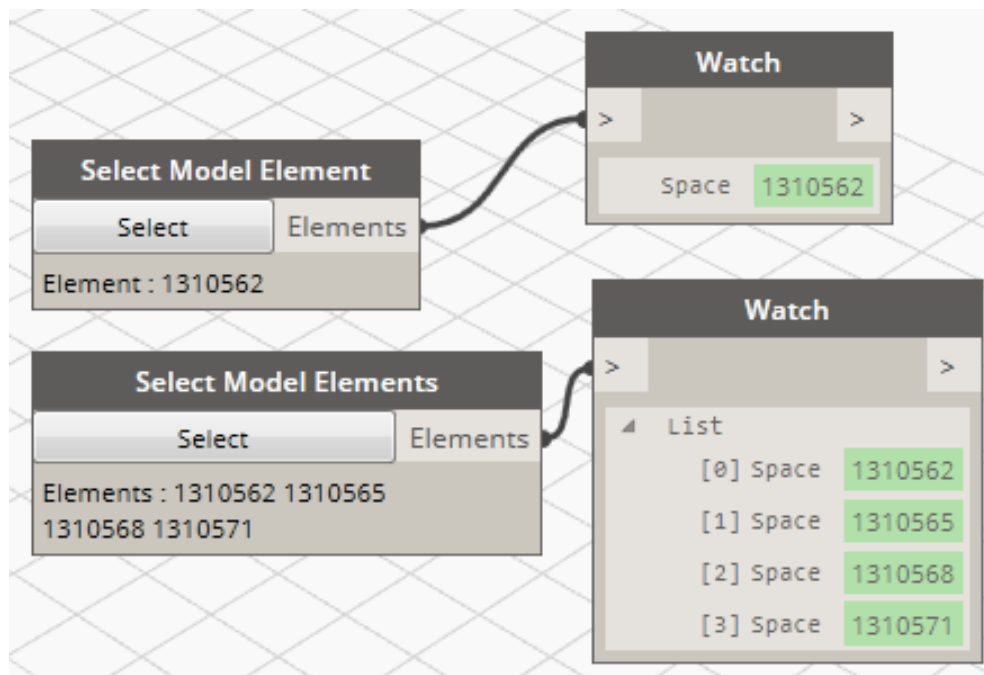


FIGURE 13: ELEMENT IDS APPEAR IN THE SELECT NODES ONCE ELEMENTS HAVE BEEN SELECTED.

Once an element or elements are selected, the element ID or IDs will appear in the node. You can use a **Watch** node to see the category of the element if the element belongs to a system family, or the type name if the element belongs to a loadable family.

Select Elements by Category

To select elements by category, use the **Categories** node along with the **All Elements of Category** node. You can then select the category from the drop-down in the **Categories** node. Take note that some of the category names may not be exactly the same as they are in Revit. For example, in Revit, spaces belong to the **Spaces** category. In Dynamo, the category is called **MEPSpaces**.

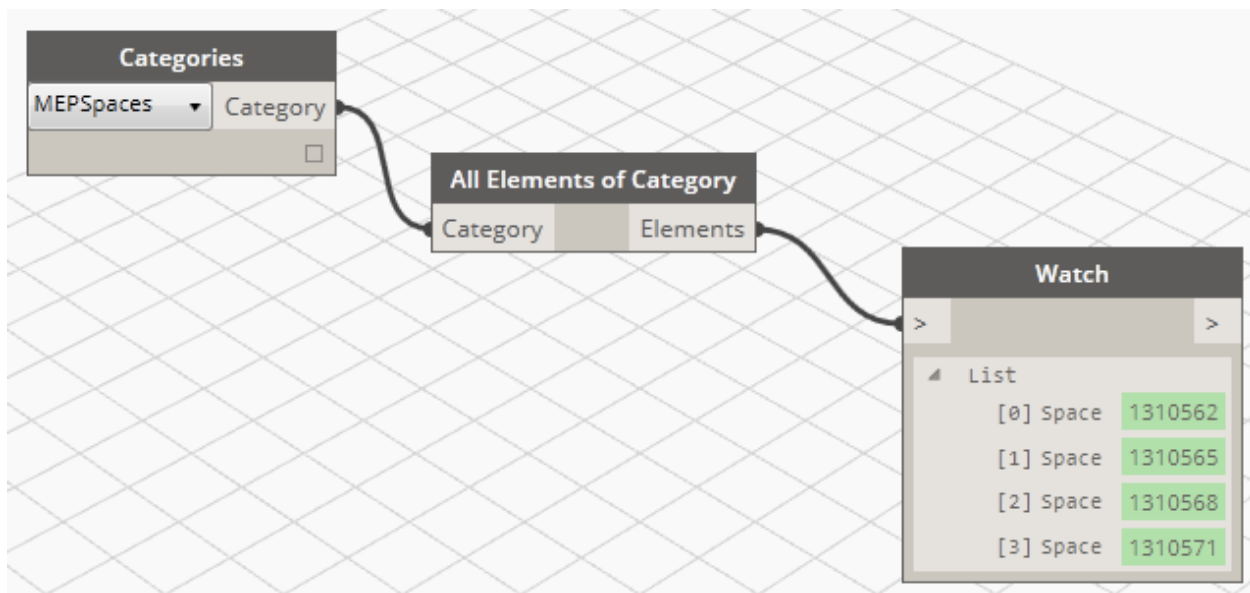


FIGURE 14: USE THE CATEGORIES AND ALL ELEMENTS OF CATEGORY NODES TO SELECT ALL THE ELEMENTS BELONGING TO A CATEGORY.

Once again, you can use a **Watch** node to better understand the output of the node. Also, be aware that the output is multiple elements. When that is the case, a list is created.

Selecting System Families and Elements in a System Family

To select all the elements belonging to a system family, use the **Element Types** and **All Elements of Type** nodes. You can select a system family from the drop-down in the **Element Types** node. However, be aware that you can also select all the types belonging to a system family.

For example, you can select **Duct** or **DuctType**, **Pipe** or **PipeType**, **Conduit** or **ConduitType**, and so on. When you select a system family in the **Element Types** node, you can then connect it to the **All Elements of Type** node to select all the elements belonging to that system family. When you select the listing with **Type**, then all the types belonging to that system family will be selected in the **All Elements of Type** node.

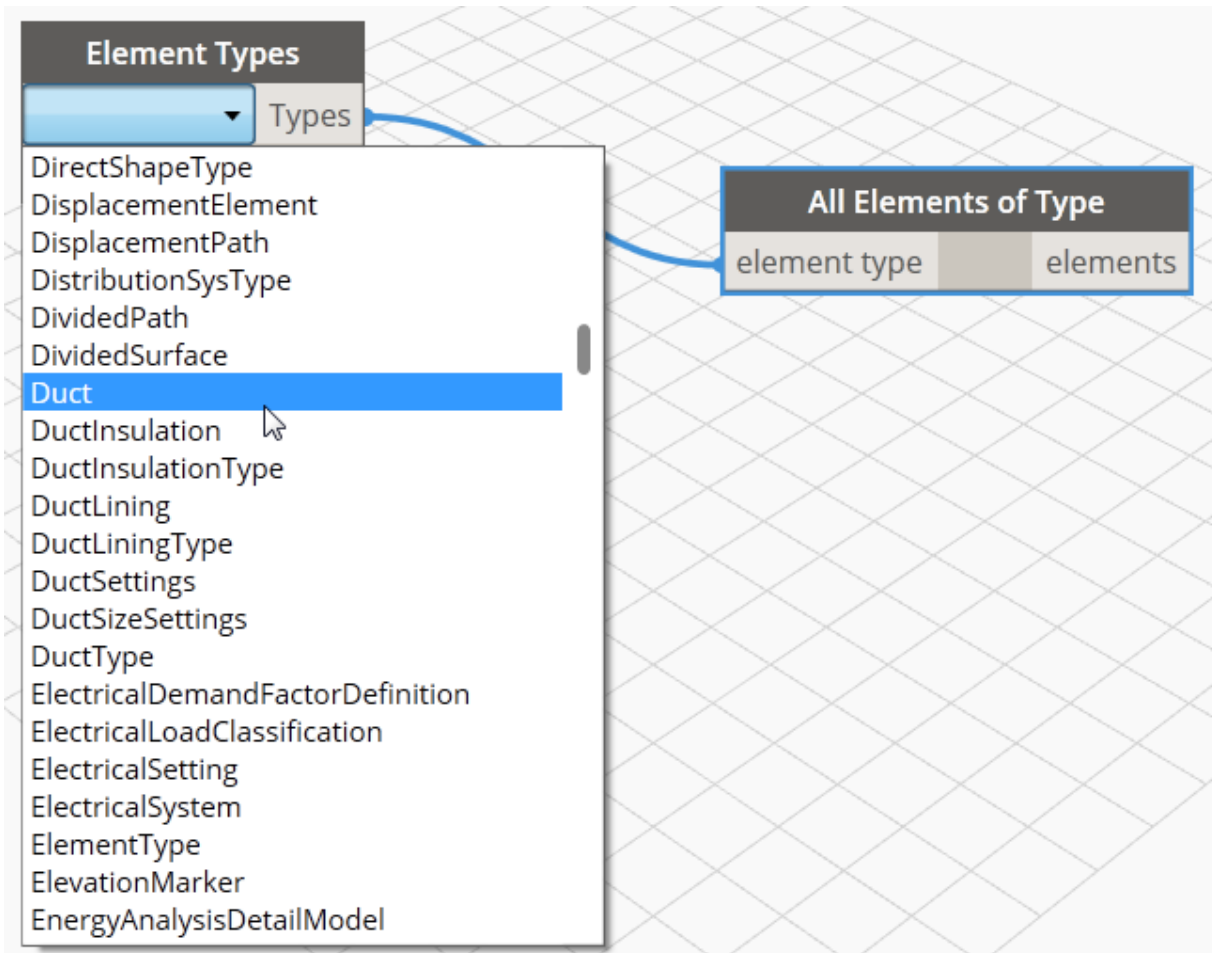


FIGURE 15: USE THE ELEMENT TYPES AND ALL ELEMENTS OF TYPE NODES TO SELECT ALL THE ELEMENTS BELONGING TO A SYSTEM FAMILY.

To summarize, the system family you choose in the drop-down in the **Element Types** node is selected in that node. If you need a system family output for another action, you can use that output. When you connect it to the **All Elements of Type** node, all the elements belonging to that system family will be selected in the **All Elements of Type** node. And, if the **Type** listing is selected, all the types belonging to the system family will be selected. Once again, the output will be a list of elements or a list of element types.



Selecting Loadable Family Types and Elements in a Loadable Family Type

You can use the **Family Types** node to select a loadable family type. A drop-down is available that lists all the loadable family types in the connected Revit project. You can then connect it to the **All Elements of Family Type** node to select all the elements belonging to that loadable family type.

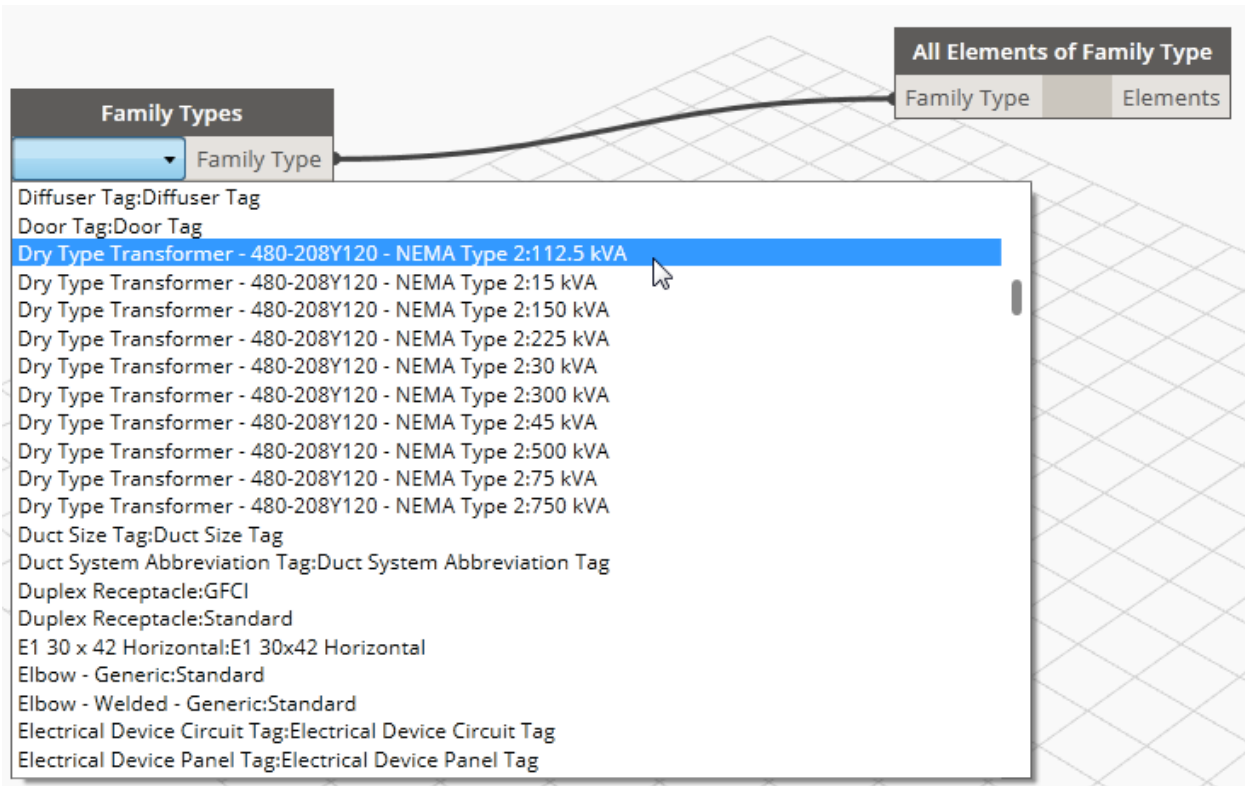


FIGURE 16: USE THE FAMILY TYPES AND ALL ELEMENTS OF FAMILY TYPE NODES TO SELECT ALL THE ELEMENTS BELONGING TO A LOADABLE FAMILY TYPE.

Understanding Lists

In order to really harness the power of Dynamo, you must understand how to use and work with lists. Lists in Dynamo allow you to have multiple inputs. When working with Revit, lists allow you to work with multiple elements or multiple parameters at once – or multiple elements and multiple parameters! Whenever you select multiple elements, regardless of which method you use, the output will be a list of elements.

One of the biggest things to wrap your head around is the list numbering system. It's really not that hard, but you must be aware of it at all times. The list numbering system is by index and starts at **index0**. Therefore, the first item in a list is **index0** or **[0]**.

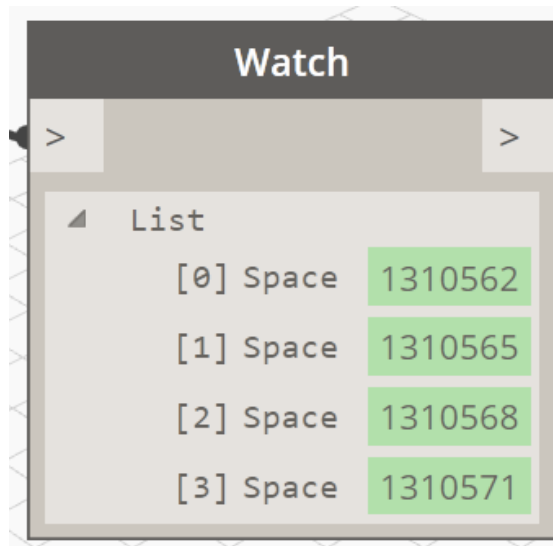


FIGURE 17: THE LIST NUMBERING SYSTEM STARTS AT INDEX0.

Getting Parameters

Once you understand how to select elements, then you can get parameters and parameter values from those elements. This paper will focus on two nodes that can be used to get parameter values and parameter names.

First, the **Element.GetParameterValueByName** node is very flexible and useful. You can input a single element or a list of elements in the element input. Then you can enter a parameter name in the **parameterName** input. The output will then be that parameter value.

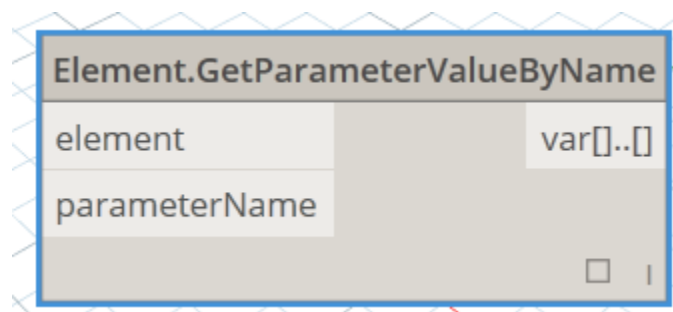


FIGURE 18: THE GETPARAMETERVALUEBYNAME NODE WILL EXTRACT A PARAMETER VALUE.

The **Element.GetParameterValueByName** node can be found in the node **Library** under **Revit** > **Elements** > **Element**. If you input a list of elements and a list of parameter names, be sure to change the **Lacing** to **Cross Product**. That way you can extract each parameter from each element. You can change the **Lacing** by right-clicking a node.

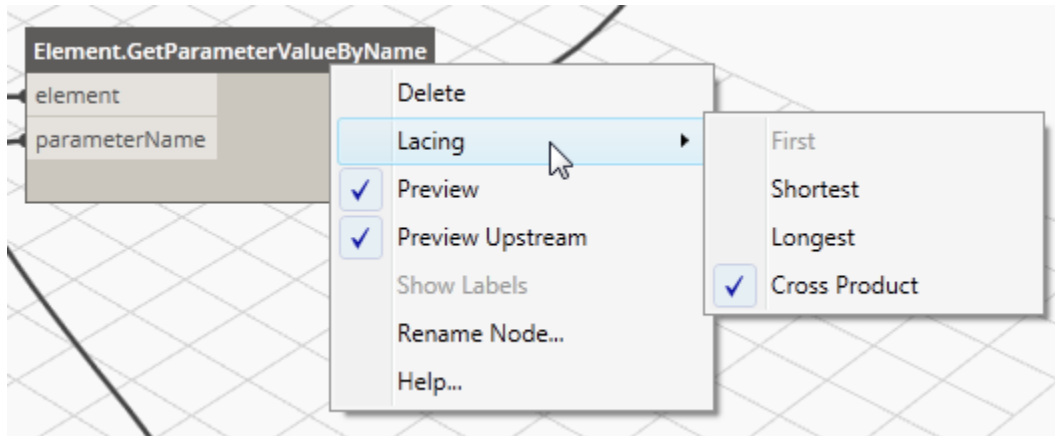


FIGURE 19: YOU CAN CHANGE THE LACING BY RIGHT-CLICKING A NODE.

The other node that you can use to get parameters is the **Element.Parameters** node. This node simply requires an element input. The output is then a list of the element's parameters along with the parameter values.

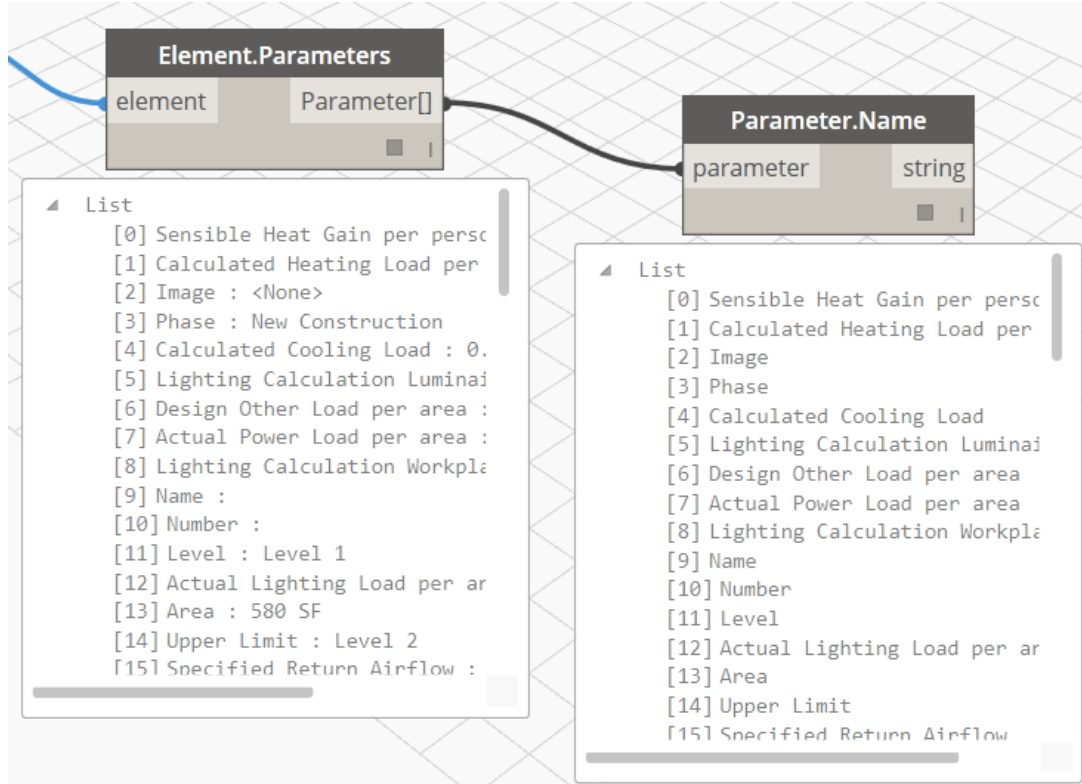


FIGURE 20: THE ELEMENT.PARAMETERS NODE WILL EXTRACT ALL PARAMETERS AND PARAMETER VALUES.

The **Element.Parameters** node is in the node **Library** under **Revit > Elements > Element > Query**. Since the output is both the parameters and parameter values, you may want to use the **Parameter.Name** node to then extract just the parameter names. Then you can use that output for another node. The **Parameter.Name** node is in the node **Library** under **Revit > Elements > Parameter > Query**. Figure 20 shows the graph for using both nodes.

The output for the **Element.Parameters** node contains both the parameter name and value, so you really can't use that as an input for another node. However, when you use the **Parameter.Name** node to extract just the parameter names, you can use the list of parameter names as an input somewhere else.

Wall Parameters Example

When beginning an MEP project, one of the first steps is to calculate the heating and cooling loads. However, you need to know if the architectural model contains the correct information. You can use Dynamo to quickly help you. Open the architectural model and get all the wall type parameters with the **Element.Parameters** node.

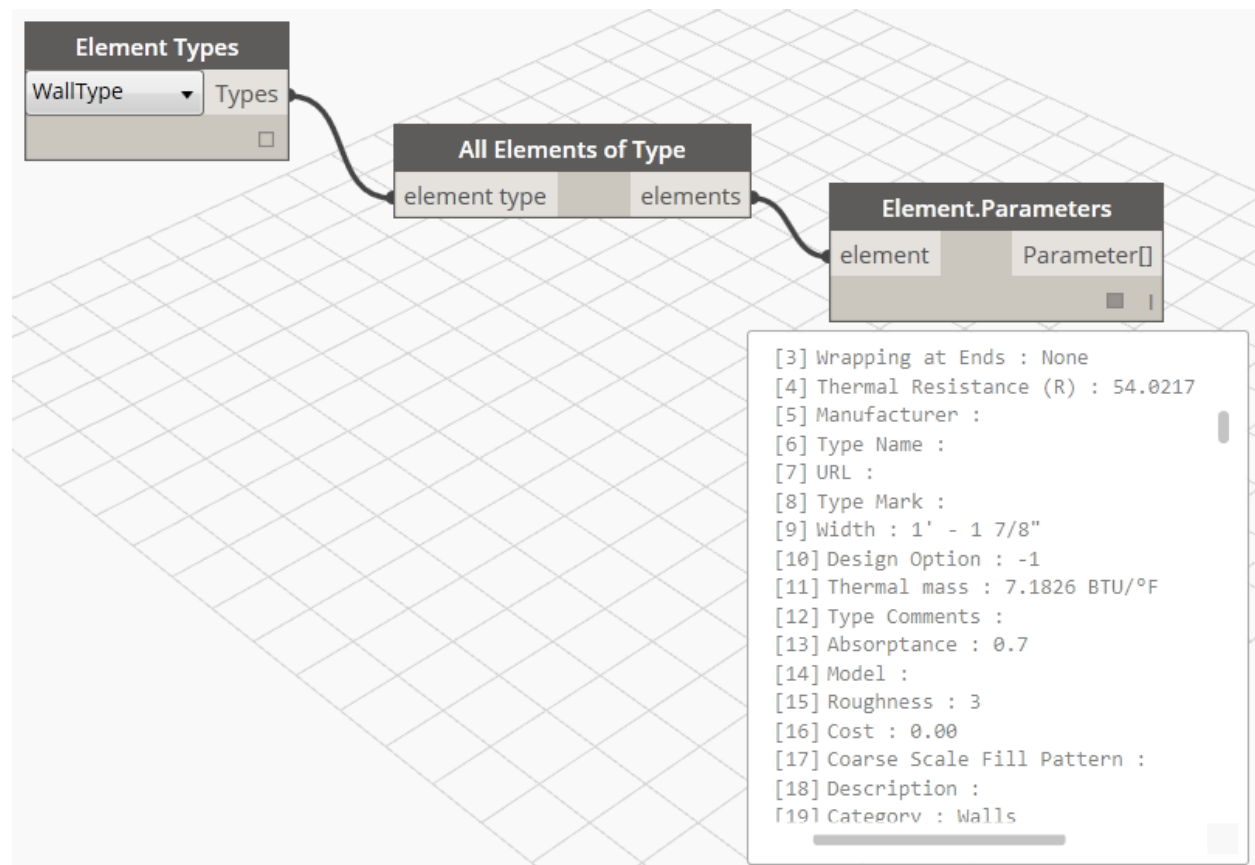


FIGURE 21: A GRAPH THAT CAN BE USED TO GET ALL THE WALL TYPE PARAMETERS.

You can also use the **Element.GetParameterValueByName** node to get a specific parameter or a list of parameters that you want to see. This can quickly help you determine the construction of the building envelope and whether or not all the information has been input into the architectural model. You can use these same methods for floors, roofs, windows, doors, etc.

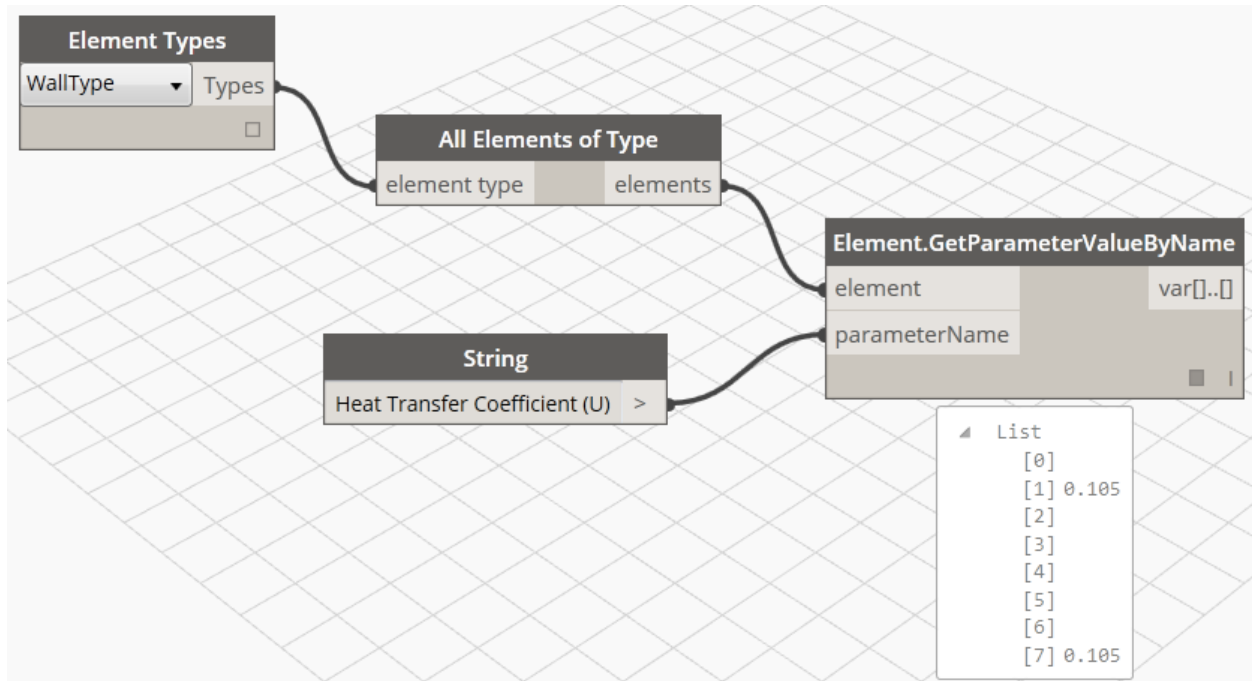


FIGURE 22: A GRAPH THAT CAN BE USED TO GET A SINGLE WALL TYPE PARAMETER.

Space Parameters Example

Spaces are very important when it comes to MEP design. Spaces are essentially containers for MEP information. You can use Dynamo to extract parameters from spaces. First, you can use the **Element.Parameters** node to extract all parameter names and parameter values from the spaces.

When you do this, you will have a list for each space of all the parameter names and parameter values. All this information can be overwhelming. A good practice is to create a spreadsheet containing all the parameters that you want to extract from a space. You can then import that list of parameters from Excel to Dynamo. After that, you can use the list of parameters in the **Element.GetParameterValueByName** node. That way, you get just the information you need.

When working with space parameters, there are several parameters that depend on another parameter. Essentially, some parameters can be user-specified, or based on an actual value. For example, the **Return Airflow** parameter can be set to **Specified**, **Specified Supply Airflow**, **Calculated Supply Airflow**, or **Actual Supply Airflow**. When it is set to **Specified**, then you can use the **Specified Return Airflow** parameter. When set to one of the other options, the **Specified Return Airflow** parameter is unavailable. So pay attention to these types of space parameters when getting and setting these parameters. Parameters that are unavailable do not work well with Dynamo.



Back to the space parameter example, you can start with the **Element.Parameters** node to extract all the parameters from the spaces in a model. Then you can evaluate the list to see which parameters are most important to you. Then you can create a list of parameters in Excel.

One final note for getting parameters. You may not be able to get every parameter that you want. Some parameters just do not get extracted when using Dynamo. You may see some type of warning in Dynamo. Just be aware that it may be a parameter that you cannot get and not anything more serious than that.

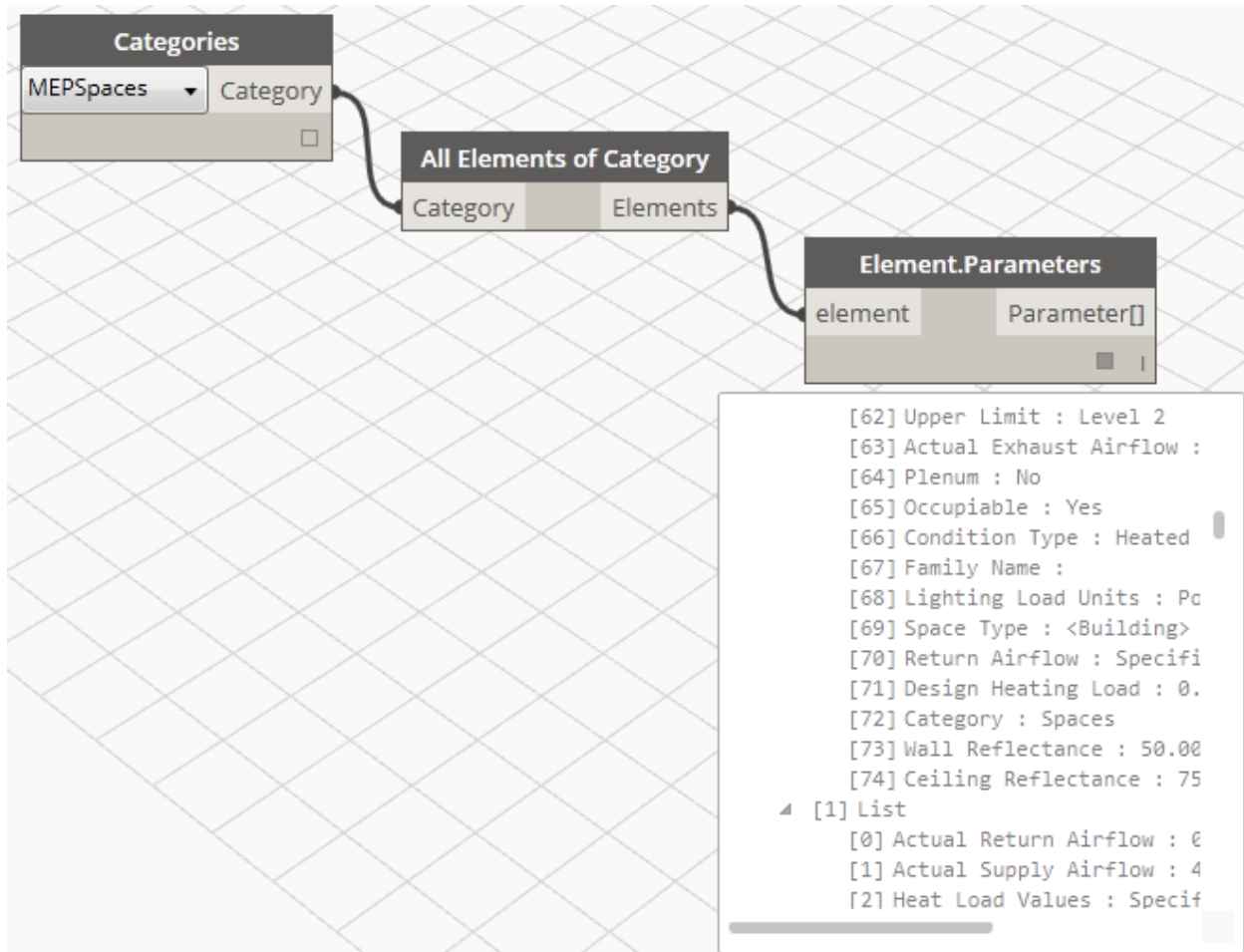


FIGURE 23: A GRAPH THAT CAN BE USED TO GET ALL THE SPACE PARAMETERS.

Exporting to Excel

You can export information from Dynamo to Excel. This is typically beneficial for exporting lists of information. For example, you can export lists of space parameter values to easily sort through the information. The Excel nodes are located in the node **Library** under the **Office** category.

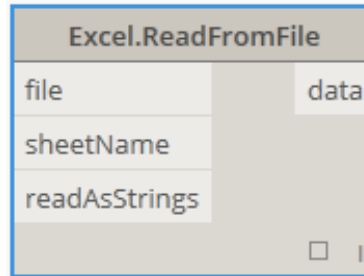


FIGURE 24: THE EXCEL.WRITEToFile NODE CAN EXPORT INFORMATION FROM DYNAMO TO EXCEL.

When exporting information from Dynamo to Excel, you need an existing Excel file to write to. You can use a **File Path** node to specify the file path for the existing file. Then you can use a **String** node to input the **sheetName**. The **startRow** and **startCol** inputs then need the index number (programming info) of the row and column to specify where to start writing information. This works similar to lists. If you want to start writing at cell **A1**, then both inputs are **0**. The data is simply the information in Dynamo to export to Excel.

Importing From Excel

You can also import information from Excel. A good scenario is if you have a list of parameter names in Excel. You can then import the information and use the parameter names to get or set parameter values. The Excel nodes are located in the node **Library** under the **Office** category.

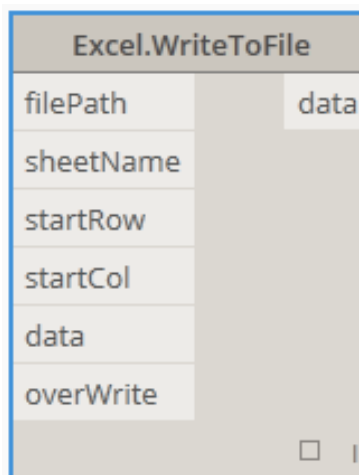


FIGURE 25: THE EXCEL.READFROMFILE NODE CAN IMPORT INFORMATION FROM EXCEL TO DYNAMO.

The file input must have the actual file, so you can use a **File Path** node along with a **File.FromPath** node. Then you can use a **String** node to input the **sheetName**.



Space Parameters from Excel Example

When you have a list of parameters in Excel, you can import the list to Dynamo and then use the **Element.GetParameterValueByName** node to extract those specific parameters.

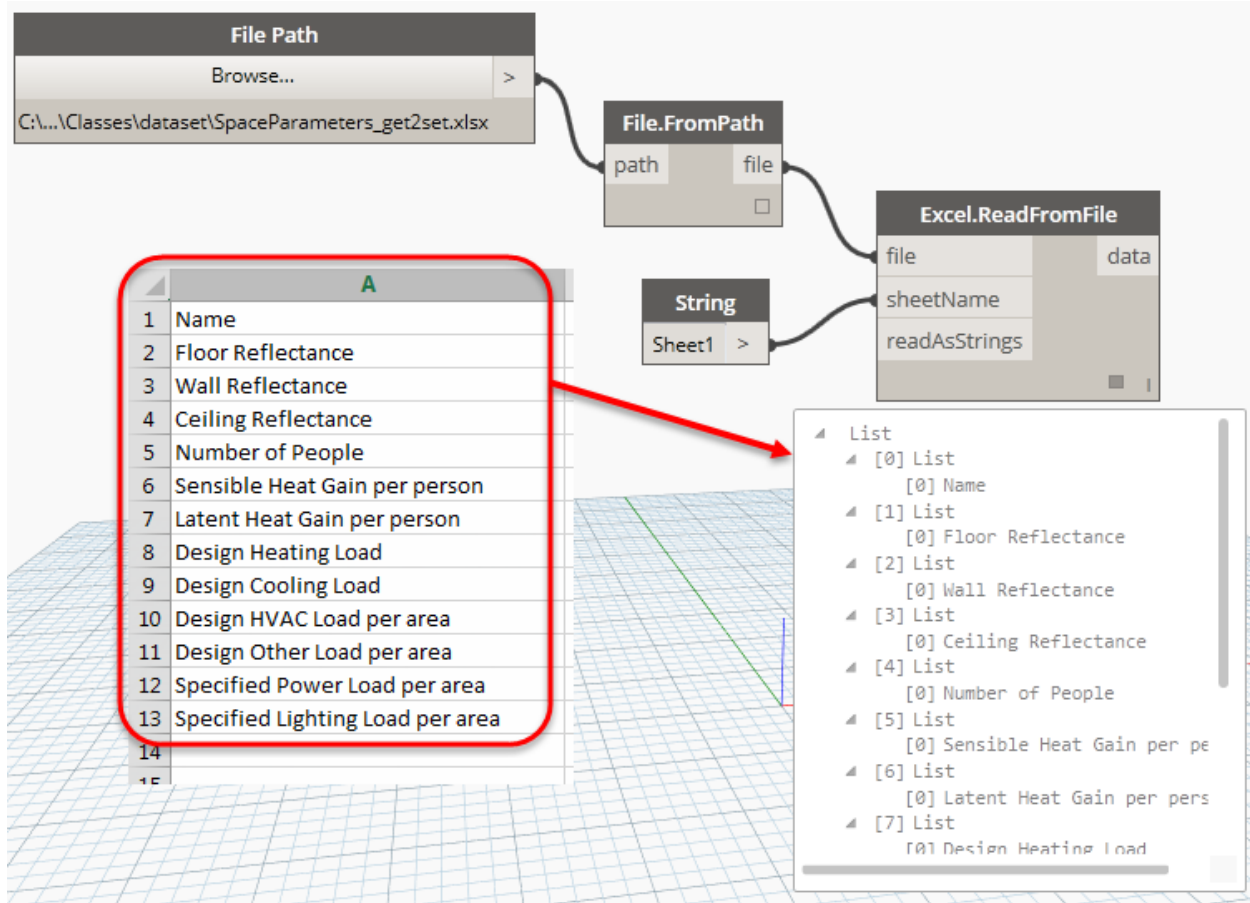


FIGURE 26. EACH ROW OF INFORMATION IN EXCEL IS ADDED TO A LIST IN DYNAMO.

After importing information from Excel, you may need to clean up the lists that are imported before you can use them. For example, Figure 26 shows a list of parameters in Excel that is imported into Dynamo. As you can see, each row of information is added to a list. If the parameters are listed in row 1 in the various columns, then you would probably be fine. However, a typical approach is to list parameters in the first column in the various rows.

In order to create a single list, you can transpose the list with the **List.Transpose** node or use the **List.Flatten** node. These nodes can be found in the node **Library** under the **Core** category. Expand the **List** subcategory and you will find both under **Action**. You can use either node to create a single list from the list of nested lists, which each contain a single item. That way, you can have a list of parameters in Dynamo. After that, you can use the list in the **Element.GetParameterValueByName** node.

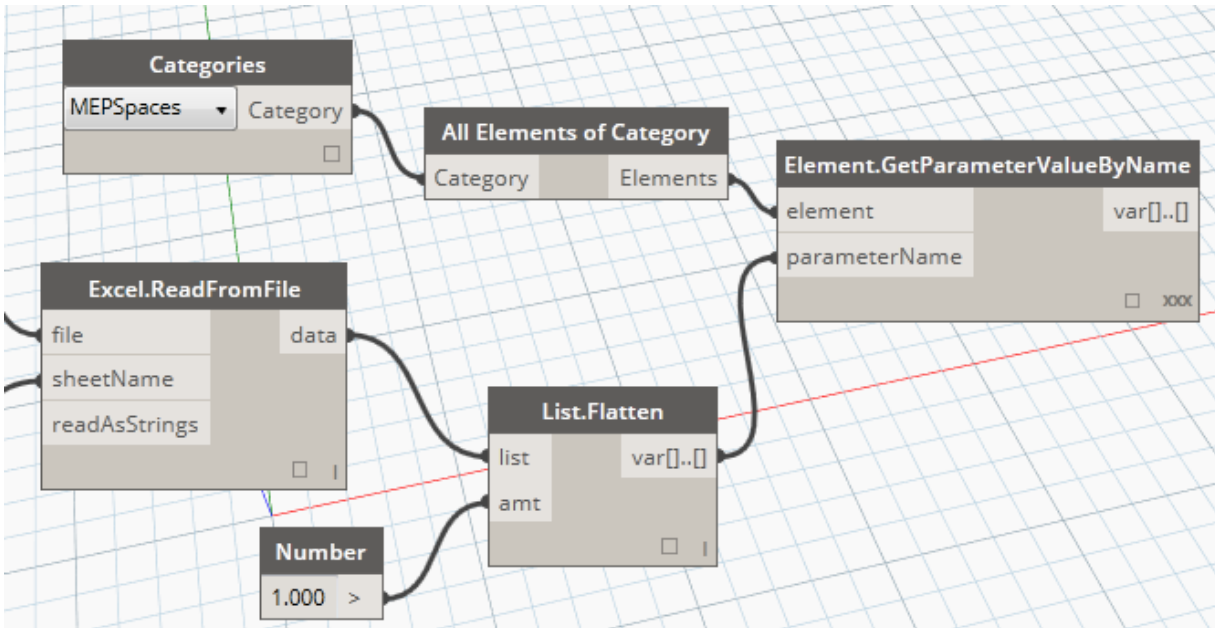


FIGURE 27: A GRAPH TO GET ALL THE PARAMETERS THAT ARE INPUT FOR ALL THE INPUT SPACES.

One more thing to be aware of: When you input a list of elements and a list of parameters, you need to change the node **Lacing**. To get all the parameter values for all the elements, change the **Lacing** to **Cross Product**. Dynamo reads lists starting at **index0** and then moves down. For this example of getting parameter values, if the **Lacing** was set to **Shortest**, Dynamo would simply go down each list one by one. So the first parameter would be extracted from the first element, the second parameter extracted from the second element, and so on, until the shortest list was completed. If it was set to **Longest**, then the longest list would be completed. This is not what you want when getting parameters.

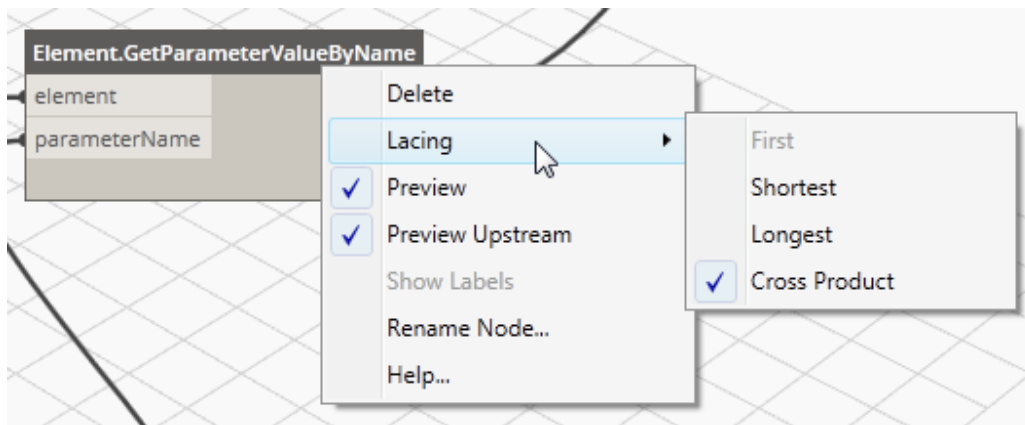


FIGURE 28: YOU CAN CHANGE THE LACING BY RIGHT-CLICKING A NODE.

Setting Parameters

Once you select elements, you can then set parameter values. The **Element.SetParameterByName** node can be used to set a parameter to a specified value. Be aware that the value can only be a string or number. In other words, it cannot be a parameter whose value is set by selecting from a list.

The **Element.SetParameterByName** node can be found in the node **Library** under **Revit > Elements > Element**. The **element** and **parameterName** inputs work exactly the same as the identical inputs on the **Element.GetParameterValueByName** node.

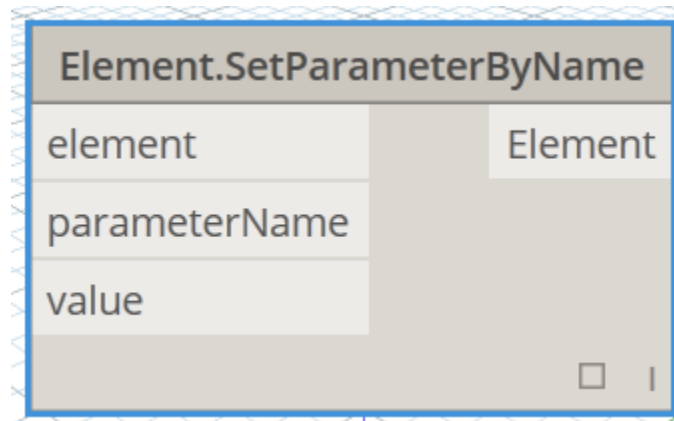


FIGURE 29: THE SETPARAMETERBYNAME NODE CAN BE USED TO SET A PARAMETER.

The **value** input is then the value that you want to set the parameter to for the selected element. However, remember that the value can only be a string or number. You are somewhat limited in the parameters that you can set in Revit. The graph shown in Figure 9 is an ideal situation. The get parameter and set parameter nodes work together really well.

You can actually get several more parameters than you can set. For example, for spaces, you can get almost every parameter – including parameters that are specified with a checkbox and parameters that are read-only. But when it comes to setting parameters, you cannot set parameters that are read-only. You also cannot set parameters that are controlled with a drop-down, checkbox, or anything of that nature. Not to make things more confusing, but there are a few that can be set that are controlled with a drop-down, but that is beyond the scope of this paper.

Take a look at a few examples. However, realize that there are so many possibilities when it comes to quickly changing parameters!



Duct Example

To quickly change the **Offset** of the ductwork in your project, you can use the graph in Figure 30. However, realize that if you have multiple stories, you will need to select the ductwork on each story separately.

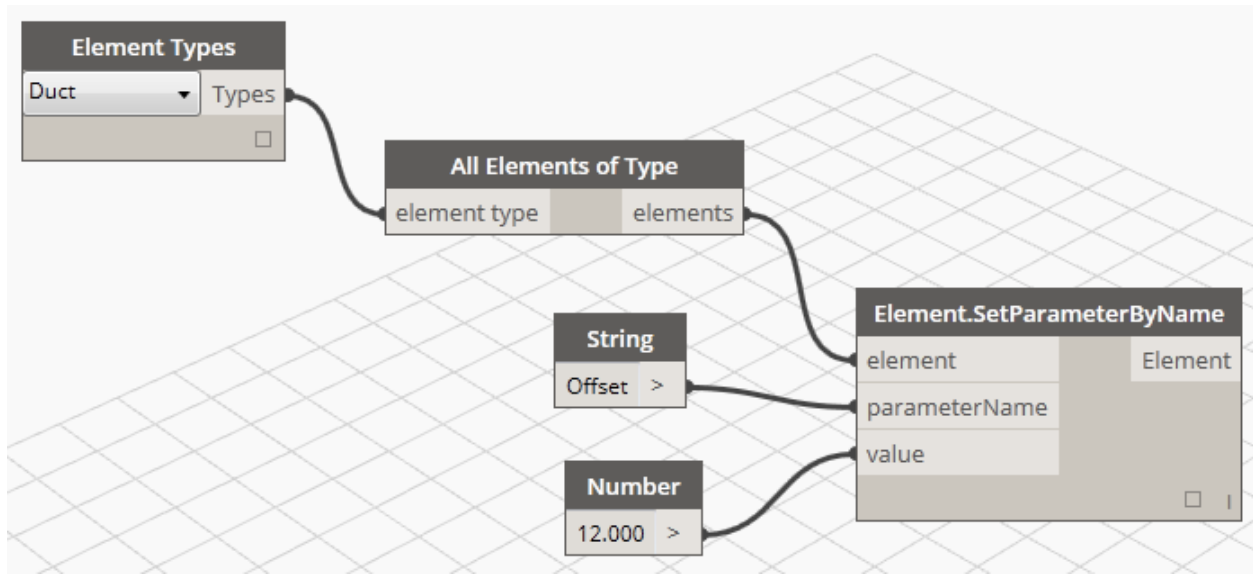


FIGURE 30. A GRAPH TO QUICKLY CHANGE THE OFFSET OF DUCTWORK.

Lighting Fixture Example

To quickly change the **Enclosure** for all panelboards of a specified type, you can use the graph in Figure 31. Take note that the parameter name must match exactly.

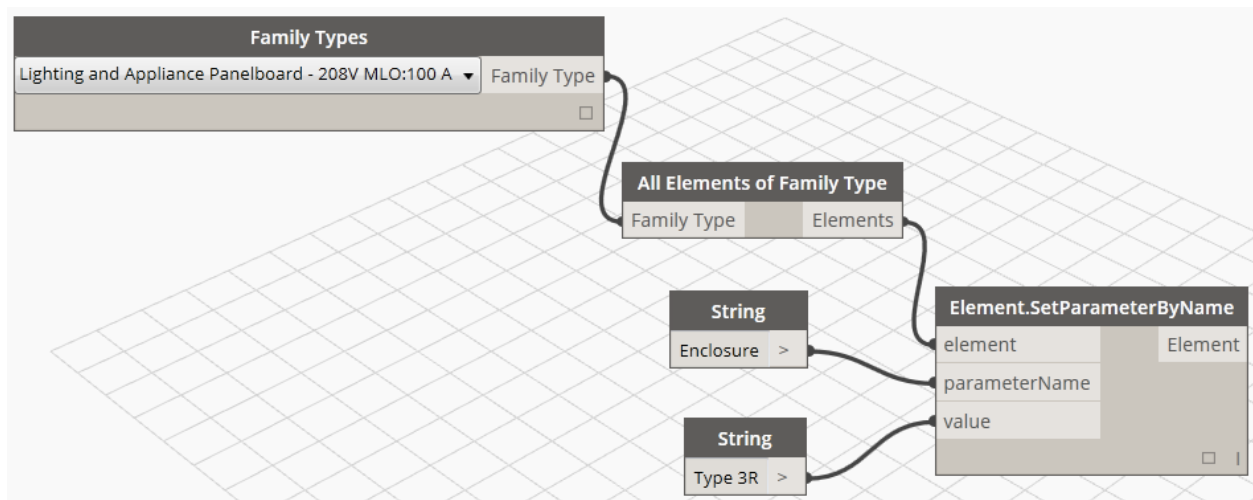


FIGURE 31: A GRAPH TO QUICKLY UPDATE THE ENCLOSURE FOR ALL PANELBOARDS OF THE SELECTED TYPE.



Plumbing Fixture Example

Although changing a type parameter in Revit is very simple, you can use Dynamo to update type properties as well. Take a look at the graph in Figure 32. It updates the **Manufacturer** type parameter.

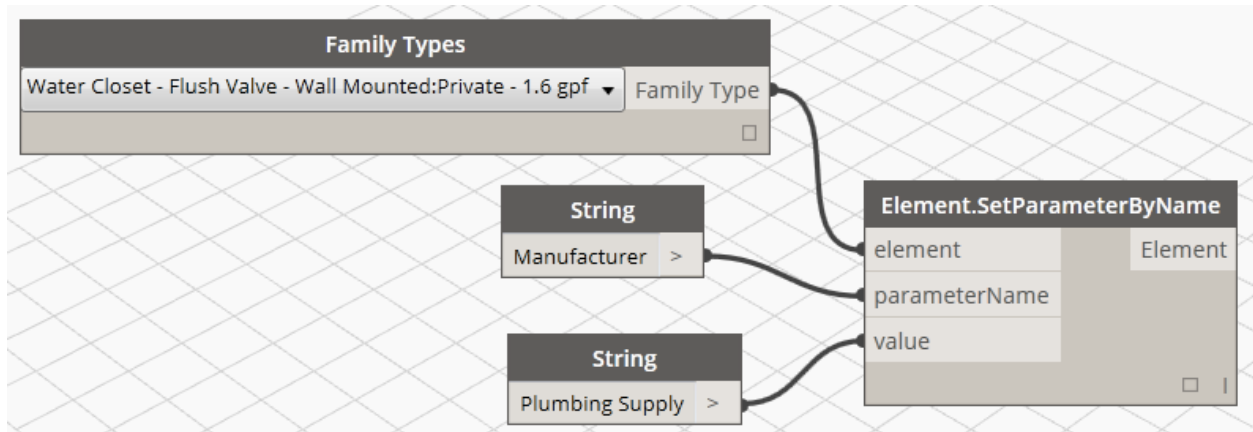


FIGURE 32: A GRAPH TO UPDATE MANUFACTURER TYPE PARAMETER FOR THE PLUMBING FIXTURE TYPE.

Units

Dynamo uses the units of the connected Revit project. However, that does not mean everything is going to flow perfectly between the two programs. If the **Length** units are **Feet and fractional inches** in Revit, then some parameters use feet and some use inches. For example, when working with ductwork and piping, the **Offset** is in feet, whereas the **Width**, **Height**, and **Diameter** parameters are in inches.

In those situations, when using Dynamo to change the **Offset**, you can simply use the **Number** node with the exact number of feet. But when setting a parameter that is specified in inches, you need to input the value in Dynamo in feet. So to get 6 inches in Revit, use 0.5 in Dynamo. Just be aware of how the units are specified in Revit, and understand that Dynamo uses the main unit settings and not the specific parameter's setting.

If needed, you can use the **Convert Between Units** node. This node can be found in the node **Library** under the **Core** category. Expand **Units** to see this node.

Advanced Examples

Setting Multiple Parameters for All Spaces

It is cool to get several parameters from multiple elements or from a type, and it is awesome to be able to set a parameter for elements. However, there is so much more to Dynamo! You can set multiple parameters for multiple elements. In this example, look at setting multiple parameters for all of the spaces in a model. It really boils down to working with lists and using lists properly.

In this paper, there is a section on getting all the parameters for all the spaces in a model. After that, wouldn't it be great to change those parameters in Excel and send them back to Revit? Well you can! Just remember that you cannot set every parameter that you can get. Therefore, you must be careful with the parameters that you input to the **Element.SetParameterByName** node.

A good idea is to have a list of parameters that you can get and set in Excel. That way, you can get all the parameters, modify them in Excel, and then send them back to Revit through Dynamo.



Remember that the **Element.SetParameterByName** node has three inputs: the elements, the parameters, and the parameter values. For that reason, you cannot simply input the list of spaces, the list of parameters, and multiple lists of parameter values for each space and then simply change the **Lacing to Cross Product**. It just does not work like that. The simple answer is building lists. There are other ways, but to begin, you can use lists.

Figure 33 shows a sample table listing multiple parameters for multiple spaces. The parameters are listed in the first row in the various columns. The spaces are listed in the first column in the various rows. The values are then entered appropriately. There are multiple ways you can push this information back to Revit through Dynamo, but one way is to build lists. You need a list of spaces to match the parameter values and a list of parameters to match the parameter values. That way, everything is one-to-one in the **Element.SetParameterByName** node.

	A	B	C	D	E	F
1	Name	Floor Reflectance	Wall Reflectance	Ceiling Reflectance	Number of People	Sensible Heat Gain
2	Office	0.2	0.5	0.75	1	
3	Break Room	0.2	0.5	0.75	4	
4	Office	0.2	0.5	0.75	1	
5	Conference Room	0.2	0.5	0.75	10	

FIGURE 33: SAMPLE TABLE FOR SPACE PARAMETERS.

Figure 34 shows the graph that imports the Excel file shown in Figure 33 and extracts the parameters and then the parameter values. It then builds a list of spaces and a list of parameters to match the list of values.

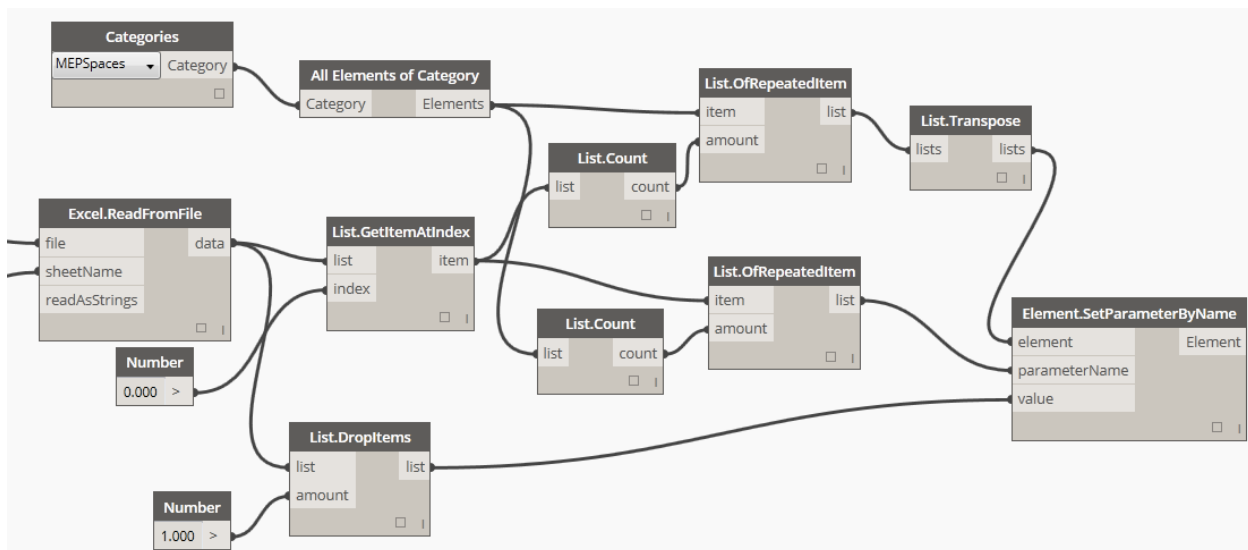


FIGURE 34: A GRAPH THAT IMPORTS SPACE PARAMETERS AND SPACE VALUES AND SETS THE VALUES IN REVIT.

Figure 35 gives you an idea of the lists that Dynamo builds in this example. Here is a brief overview of what the graph is doing:

1. Import table from Excel. In the table, parameter names are listed in the first row. Space names are listed in the first column. The values for each space are then specified for each parameter.
2. Building a list of spaces. All the space elements are selected in the model with the **All Elements of Category** node. The graph then uses the **List.OfRepeatedItem** node to create a list based on the number of parameters. The **List.Count** node gets the number of parameters based on the number of parameters that are imported. Lastly, the **List.Transpose** node transposes the list. This is needed because the output from the **List.OfRepeatedItem** node is several lists of all the spaces. You need a list of the same spaces to match the number of parameters.
3. Building a list of parameters. The **List.GetItemAtIndex** node extracts the first nested list in the list of all the parameters and parameter values. The first list is extracted since the first row in the Excel file is the parameter names. Then a list is built by duplicating the parameters to match the number of spaces.
4. Once the list of parameters and parameter values are imported, you just need to remove the parameter names from the list. This is done with the **List.DropItems** node.



2

	A	B	C	D	E	
1	Space 1	Space 1	Space 1	Space 1	Space 1	Space 1
2	Space 2	Space 2	Space 2	Space 2	Space 2	Space 2
3	Space 3	Space 3	Space 3	Space 3	Space 3	Space 3
4	Space 4	Space 4	Space 4	Space 4	Space 4	Space 4

3

	A	B	C	D	E	
1	Name	Floor Reflectance	Wall Reflectance	Ceiling Reflectance	Number of People	Sensible Heat Gain
2	Name	Floor Reflectance	Wall Reflectance	Ceiling Reflectance	Number of People	Sensible Heat Gain
3	Name	Floor Reflectance	Wall Reflectance	Ceiling Reflectance	Number of People	Sensible Heat Gain
4	Name	Floor Reflectance	Wall Reflectance	Ceiling Reflectance	Number of People	Sensible Heat Gain

4

	A	B	C	D	E	F
1	Name	Floor Reflectance	Wall Reflectance	Ceiling Reflectance	Number of People	Sensible Heat Gain
2	Office	0.2	0.5	0.75	1	
3	Break Room	0.2	0.5	0.75	4	
4	Office	0.2	0.5	0.75	1	
5	Conference Room	0.2	0.5	0.75	10	

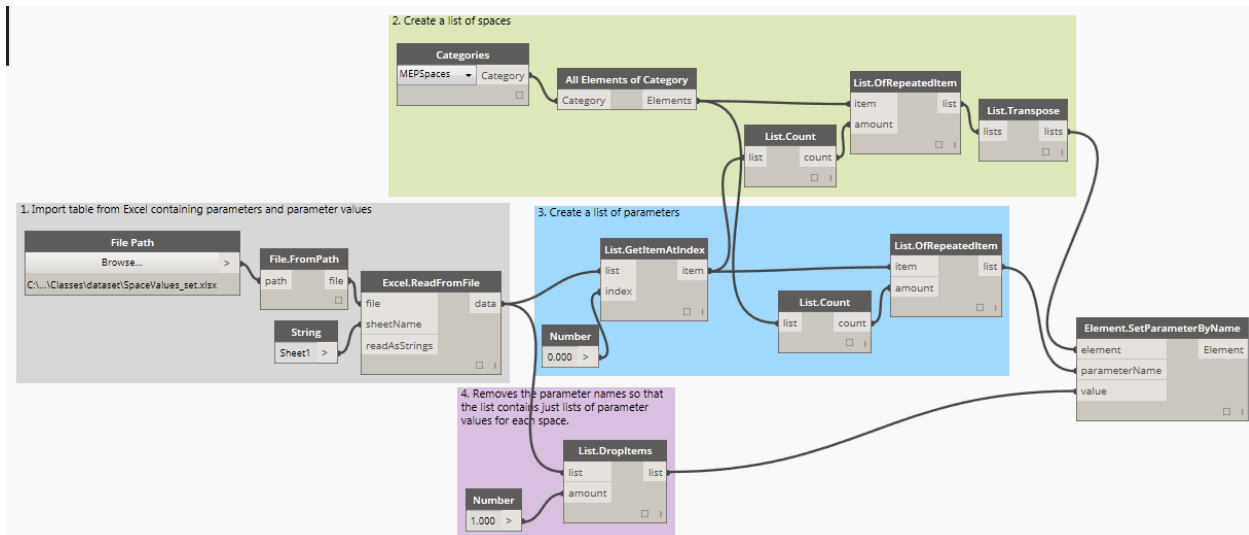


FIGURE 35: GRAPHICAL EXPLANATION OF THE LISTS THE DYNAMO GRAPH IS CREATING.

Setting Air Terminal Flow Based on Space Requirements

As you get more and more experienced with Dynamo, you will find that there are several ways to accomplish certain tasks. Additionally, you will find the certain nodes behave differently in certain situations. For that reason, examples like this one are hard to explain, and hard to definitively say that it will work exactly the same way in every similar type of situation. Not that it won't, but there is no guarantee.

Figure 36 shows a graph that will set the **Flow** parameter for air terminals based on the **Specified Supply Airflow** parameter for a space. But it just does not set the **Flow** equal to the required airflow. It takes into account the number of air terminals in the space and then divides the **Specified Supply Airflow** for the space by the number of air terminals. This is very powerful, as it eliminates the need to perform a lot of manual calculations!

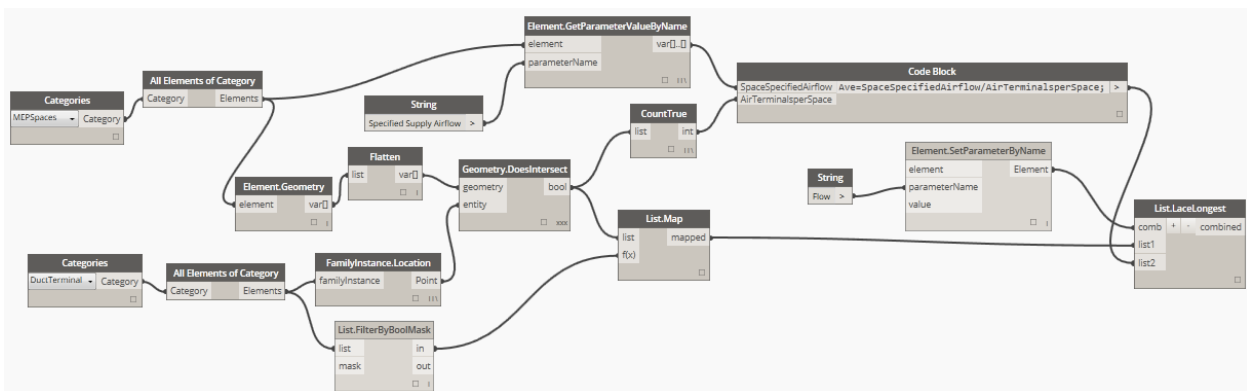


FIGURE 36: A GRAPH THAT SETS THE FLOW IN AIR TERMINALS BASED ON THE REQUIRED AIRFLOW IN A SPACE

Much of what is going on has already been covered in other examples. There are just some more complex nodes being used. The **Element.GetParameterValueByName** node is being used to get the **Specified Supply Airflow** from the spaces. Take note that the **Lacing is Longest**. The air terminals are selected with the **All Elements of Category** node. The category is **DuctTerminal**.

The next step in the process is to see which air terminals are in which space. This is done by creating the space geometry with the **Element.Geometry** node and finding the air terminal locations with the **FamilyInstance.Location** node (**Lacing=Longest**). The **Geometry.DoesIntersect** node (**Lacing=Cross Product**) is then used to see which air terminals intersect which spaces.

From there, the ones that do intersect a space are used to calculate the average airflow per air terminal. A **Code Block** is used to create a simple formula: **Ave=SpaceSpecifiedAirflow/AirTerminalsperSpace**. Double-click the canvas to create a Code Block. Then simply enter the formula and the inputs are created for you. **Ave** is used as an abbreviation for average – nothing more than that. The inputs for the **Code Block** are the parameters in the formula. The **SpaceSpecifiedAirflow** should come from the output of the **Element.GetParameterValueByName** node, which is the **Specified Supply Airflow** from each space. The **AirTerminalsperSpace** should come from the **CountTrue** node (**Lacing=Longest**). This node is counting the true values from which air terminal points intersect the space geometry (which is done in the **Geometry.DoesIntersect** node, as described earlier).



To begin putting all this together, focus on the final node: **List.LaceLongest**. Basically, Dynamo needs to set the **Flow** parameter for all the air terminals. The value needs to be based on the average for the air terminals in the space. Remember, you don't just want to set the **Flow** to the **Specified Supply Airflow**.

For the **List.LaceLongest** node, there are initially two list inputs and a combinator input. You can add additional list inputs if needed. For this example, only two are needed. The combinator (**comb** input) in this example is the **Flow** for the air terminals. The input comes from the output of the **Element.SetParameterByName** node with just a **String** input to the **parameterName** node. That input is **Flow**. Due to the nature of this graph, Dynamo will understand that you want to set the **Flow** parameter based on the two lists that are input.

For the **list1** and **list2** inputs, be aware that which list goes into which input is very important! The **list1** input needs to be the air terminals, and the **list2** input needs to be the **Flow** values. The **Flow** values can come from the average airflow that is calculated in the **Code Block**.

Now to wrap this up, you just need the air terminals. This is really the hardest part to understand! The **list1** input needs to come from the **List.Map** node. This is a very versatile node in Dynamo. It tells Dynamo how you want to perform certain actions. So in this case, you need to tell Dynamo the specific **Flow** values for each air terminal. And that is the average flow for the space for the air terminals that are located in that space.

This can all be done with the **List.Map** node. There are two inputs into this node: a **list** and a procedure **f(x)**. The **list** input should come from the **Geometry.DoesIntersect** node. The procedure input should come from the **in** output of the **List.FilterByBoolMask** node. The list input for this node is simply the selected air terminals. This tells the **List.Map** node to only use the air terminals that are placed in a specific space. Therefore, when everything is processing in the **List.LaceLongest** node, the **Flow** parameter will be set for the air terminals based on the space where the air terminal exists.

This example really shows the power of Dynamo. This graph can be used with building models that contain air terminals and spaces with a **Specified Supply Airflow** parameter. Think about all the time calculating the airflow for each air terminal and then manually specifying the **Flow**. Once you have this graph in Dynamo, the process can be done in less than a minute. Okay, maybe a few minutes for extremely large buildings!

Conclusion

As you can see, there are several things you can do with Dynamo when it comes to getting and setting parameters. You can set parameters equal to each other, you can specify new values, or you can simply extract parameters and parameter values to get a better understanding of the model. Just focus on selecting the correct elements or types, and then you can get or set parameters!

