

BES319401

# Design Automation for Structural Engineering

Tomasz Fudala, MSc in Structural Engineering  
Autodesk

## Learning Objectives

- Learn how to apply design automation techniques on your designs
- Learn more about the structural design automation packages in Dynamo
- Learn how to automate the placement of steel connections and rebar detailing in Revit
- Learn how to create automation scripts for analytical models

## Description

In the structural engineering industry, one of the challenges is to reduce manual and tedious design tasks. Computational design tools such as Dynamo give structural designers, engineers, and detailers the possibility to automate the creation of their deliverables to build structures with minimal energy. That way, they spend less time on the repetitive tasks and more energy on the important parts of the design. In this class, you'll learn how Dynamo can support design automation and computational modeling workflows for structural analysis and design of concrete and steel structures. You'll learn about the appropriate packages to perform automation in your own daily workflows, and you'll receive a set of teaching examples.

## Speaker(s)

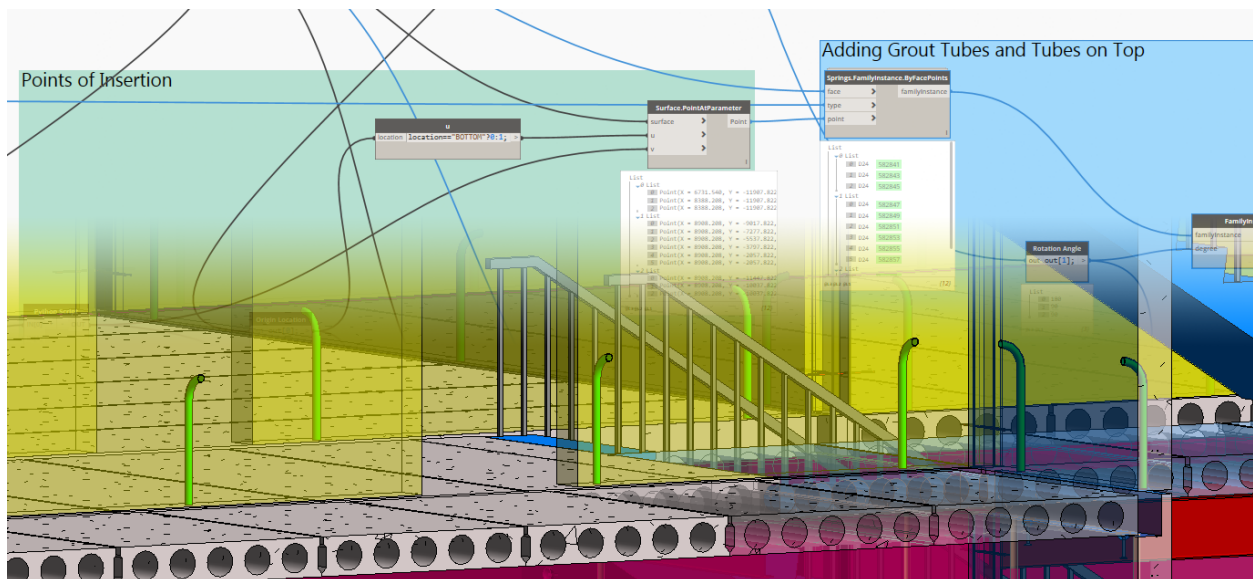
Tomasz Fudala is the Technical Marketing Manager for Structure at Autodesk. He has over 16 years of experience in the software industry and a comprehensive background and vast knowledge of structural solutions in the Autodesk portfolio. He achieved a Master of Science degree in Structural Engineering from the Cracow University of Technology, Poland. Find him on Twitter [@tomekf](https://twitter.com/tomekf)

<b>Design Automation for Structural Engineering .....</b>	<b>1</b>
<b>Learning Objectives .....</b>	<b>1</b>
<b>Description.....</b>	<b>1</b>
<b>Speaker(s).....</b>	<b>1</b>
Automated process of adding mounting parts to precast elements using Dynamo .....	3
Structural Design package .....	3
Input Parameters .....	6
Check the Orientation of Faces .....	8
Wall Part Length .....	10
Points of Insertion .....	12
Adding Grout Tubes and Tubes on Top .....	15
<b>Portal Frame Geometry in Dynamo .....</b>	<b>18</b>
<b>Automated process of reinforcement detailing around wall openings using Dynamo ..</b>	<b>23</b>
Input Parameters .....	24
Get Window/Door Opening Characteristics .....	27
Straight Rebars .....	29
Diagonal Rebars .....	33
Rebar Visibility .....	35
Boundary Stirrups .....	36
<b>Portal Frame with Structural Analysis for Dynamo Package .....</b>	<b>43</b>
Structural Analysis for Dynamo Package .....	43
Analytical Model.....	46
Section Shapes.....	47
Supports and bar end releases .....	50
Structural Materials .....	53
Creation of load cases and loads .....	53
Analysis .....	54
Results.....	56
<b>Summary .....</b>	<b>58</b>

## Automated process of adding mounting parts to precast elements using Dynamo

In the first exercise I would like to show you how we can easily automate the process of adding different mounting parts to precast elements in Revit. In this example we will add grout tubes and tubes on top to precast walls which have been split already.

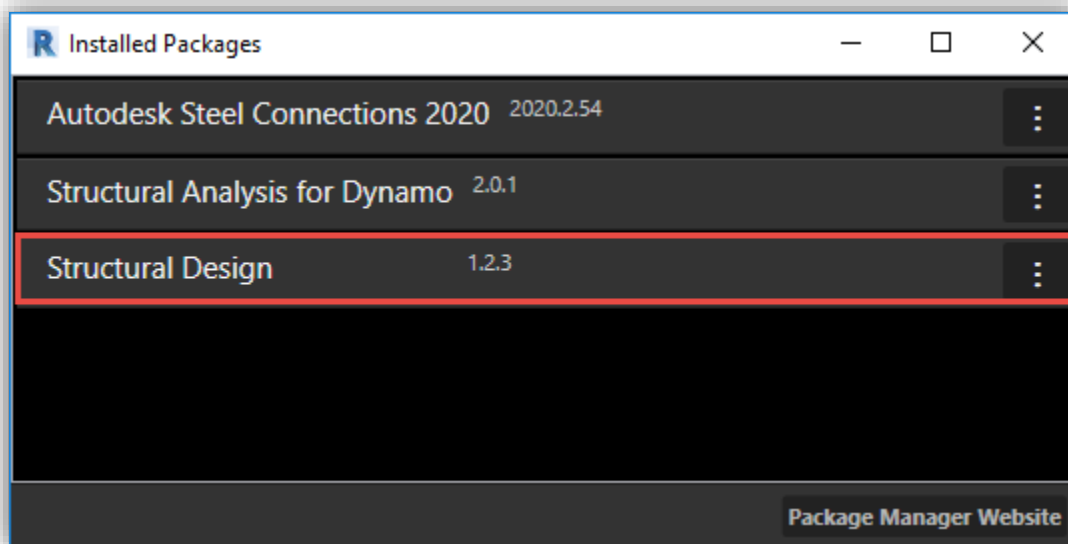
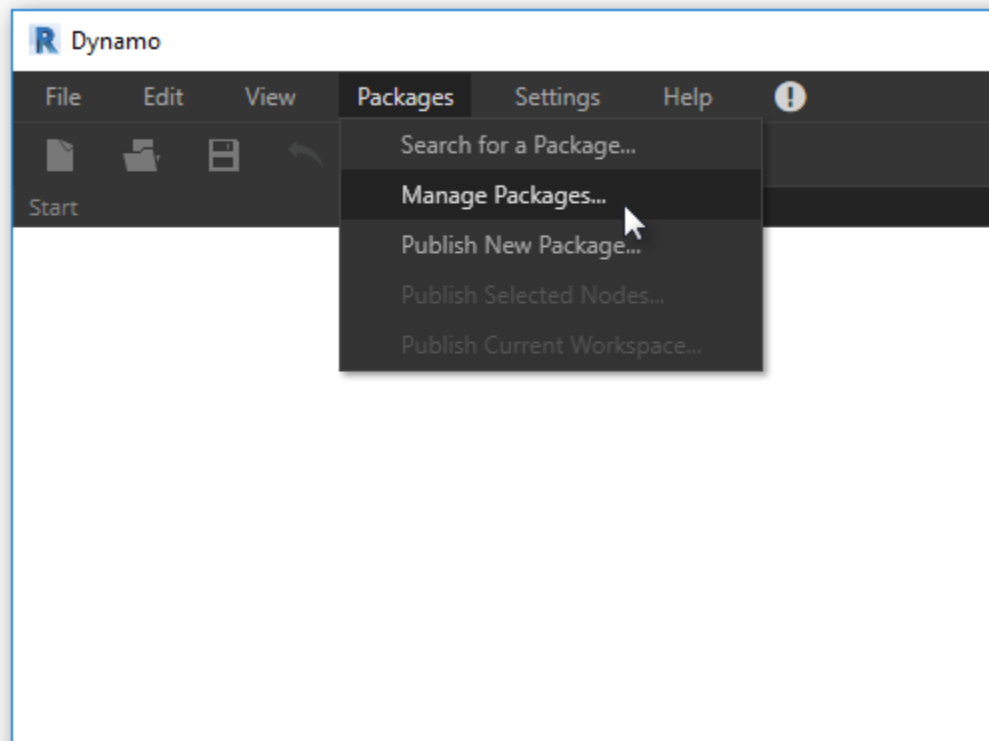
Grout tube and tube on top are families that come with the installation of Structural Precast Extension for Revit. They are both face-based families. When modeling you can use them or your own families. Currently Structural Precast Extension for Revit 2019 does not offer any dedicated tool to automate/support the process of adding grout tubes. This is a situation where Dynamo for Revit comes in handy.



## Structural Design package

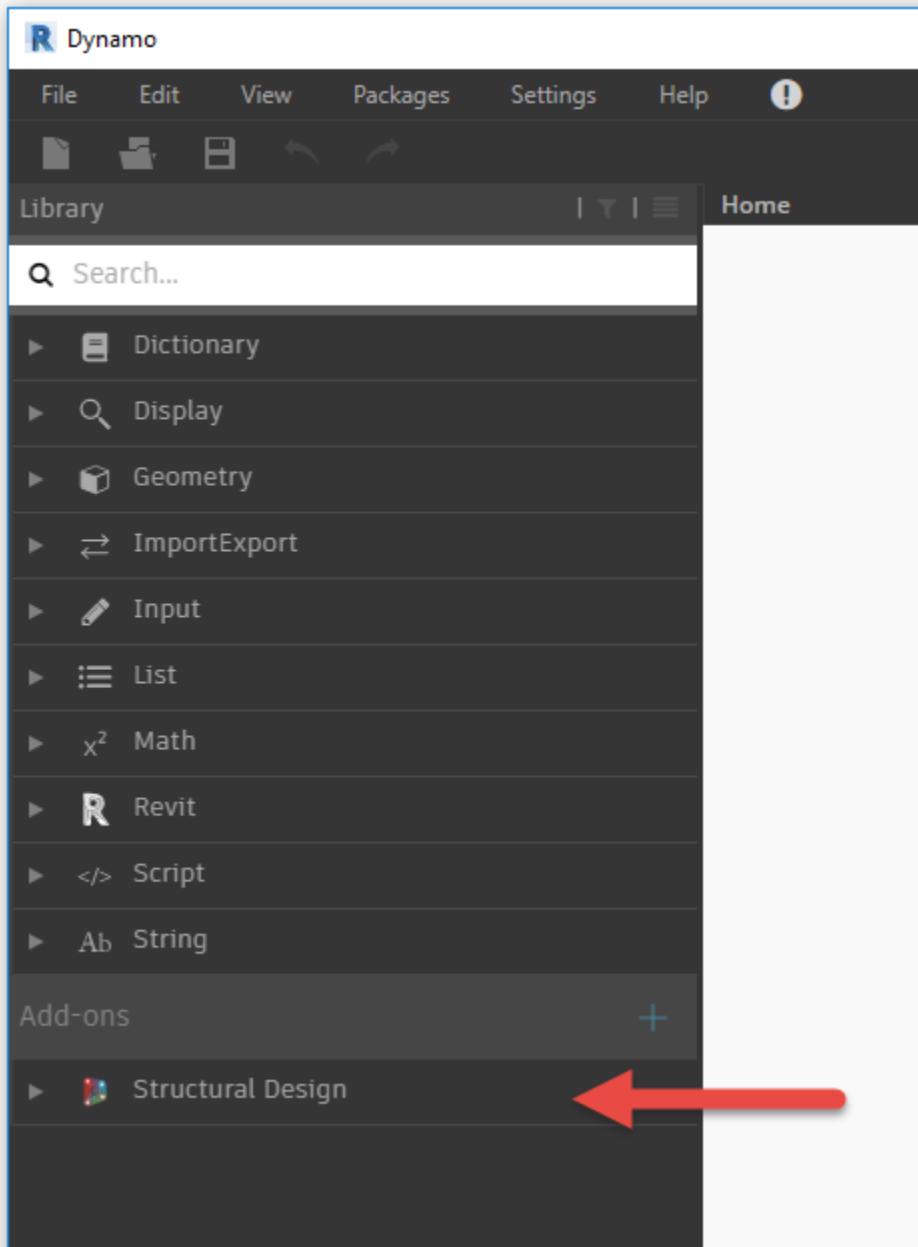
The Structural Design Dynamo package supports various structural workflows in Dynamo and Revit.

Before getting started please make sure and verify the package is installed on your machine.



You can use the *Search for a Package* option if the package is not installed on your computer yet.

If the package is installed properly you should see it under the *Add-ons*.



You may have noticed that the Dynamo 2 library looks different than previous versions (1.X). Aside from the library being web-compatible, several significant updates were made to the user interface and the organization of nodes to make it easier for both new and existing users to browse for nodes.

## Input Parameters

First, we need to define input parameters. Many Dynamo scripts require input before you can run them. In such situations we can use Dynamo Player to provide this input to adjust the script to meet our immediate needs with the current model without opening our script in the Dynamo environment.

Let's get started:

- 1) First, define the following input nodes and group them all together.

### Input Parameters

Select Faces

Select
Surfaces

Nothing selected.

Grout Tube

Solid Wall - Grout tubes:D24 ▾
Family Type

Tube on Top

Solid Wall - Tube On Top:D24 ▾
Family Type

Distance from Edge [mm]

500.000
>

Max Distance Between Elements [mm]

2000.000
>

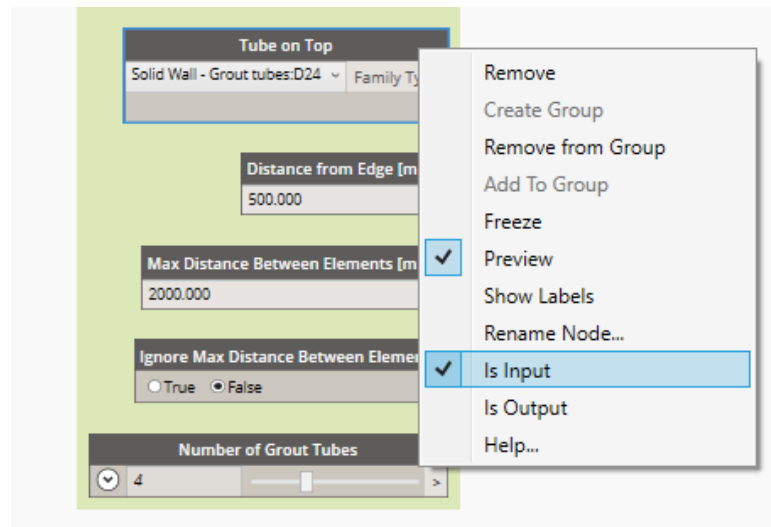
Ignore Max Distance Between Elements

☐ True ☒ False
>

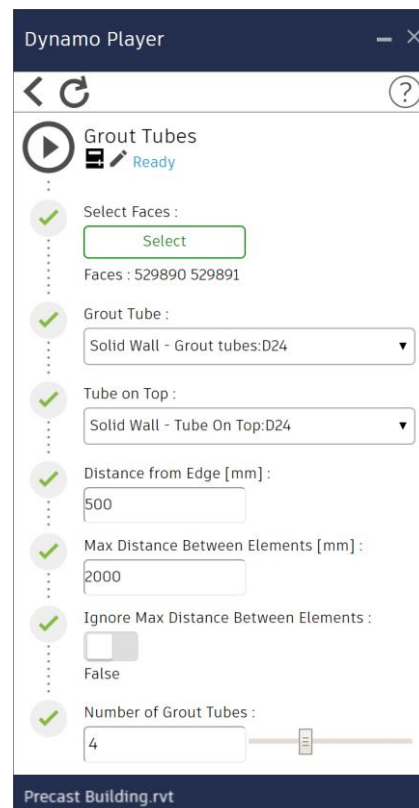
Number of Grout Tubes

▾ 4 ▸
>

- 2) As we would like to run this script via Dynamo Player, we should make sure all input nodes have the *Is Input* option checked.



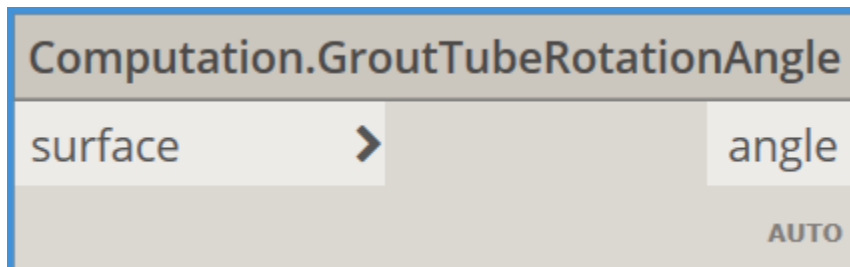
- 3) Verify how the input nodes look in Dynamo Player.



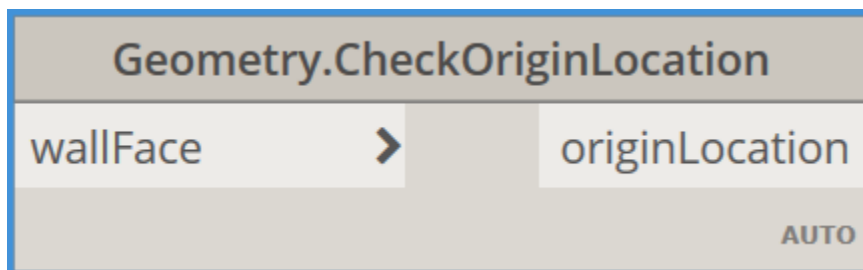




**Note:** *Computation.GroutTubeRotationAngle* & *Geometry.CheckOriginLocation* nodes are part of the Structural Design package.



Returns a grout tube rotation angle based of the geometry of the wall surface.

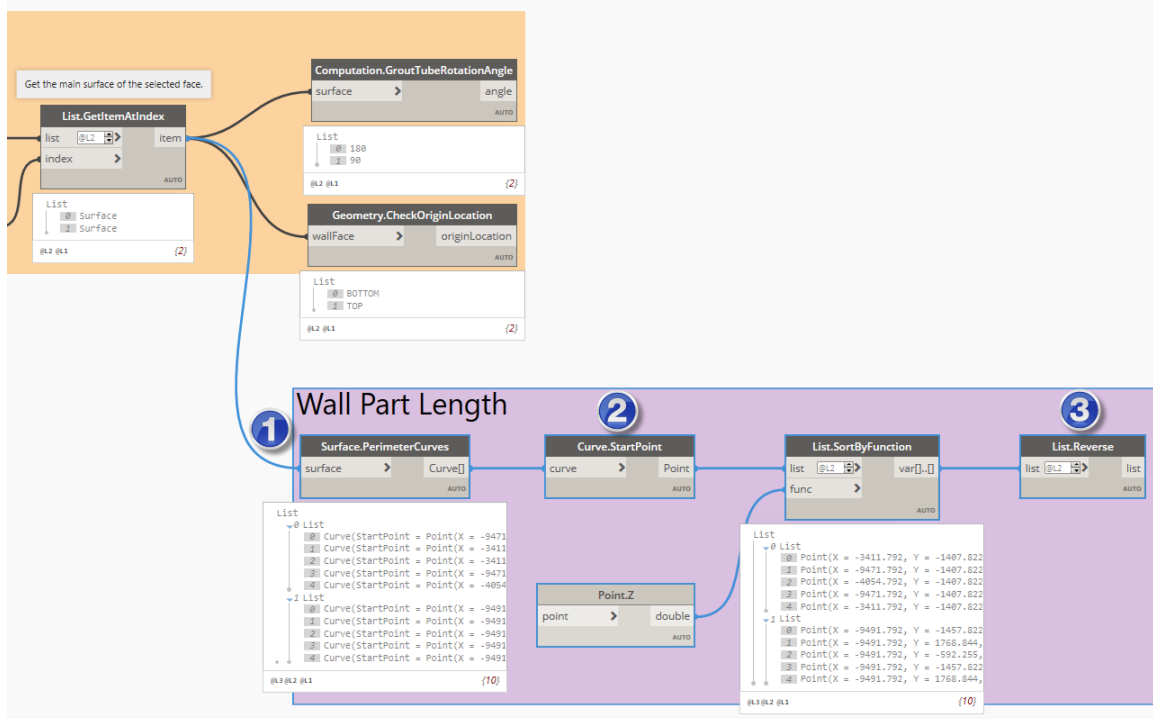


Checks if a location of the origin is at the top or at the bottom of the wall.

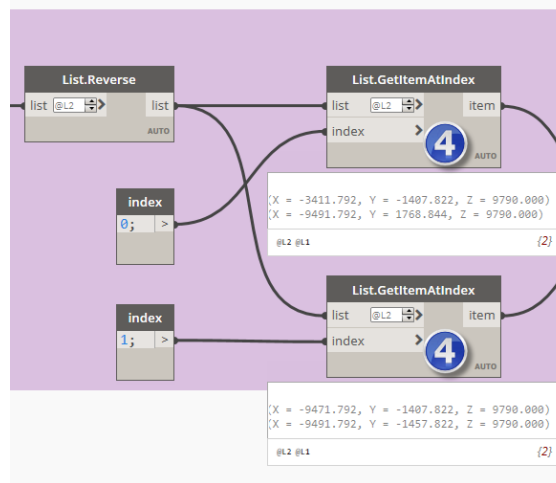
## Wall Part Length

Next, we need to determine a length of the selected wall part face.

- 1) In the first step we need to find the boundary curves of the surface.
- 2) Next let's get a starting point of each boundary curve.
- 3) Then let's sort these points by the highest Z coordinate.

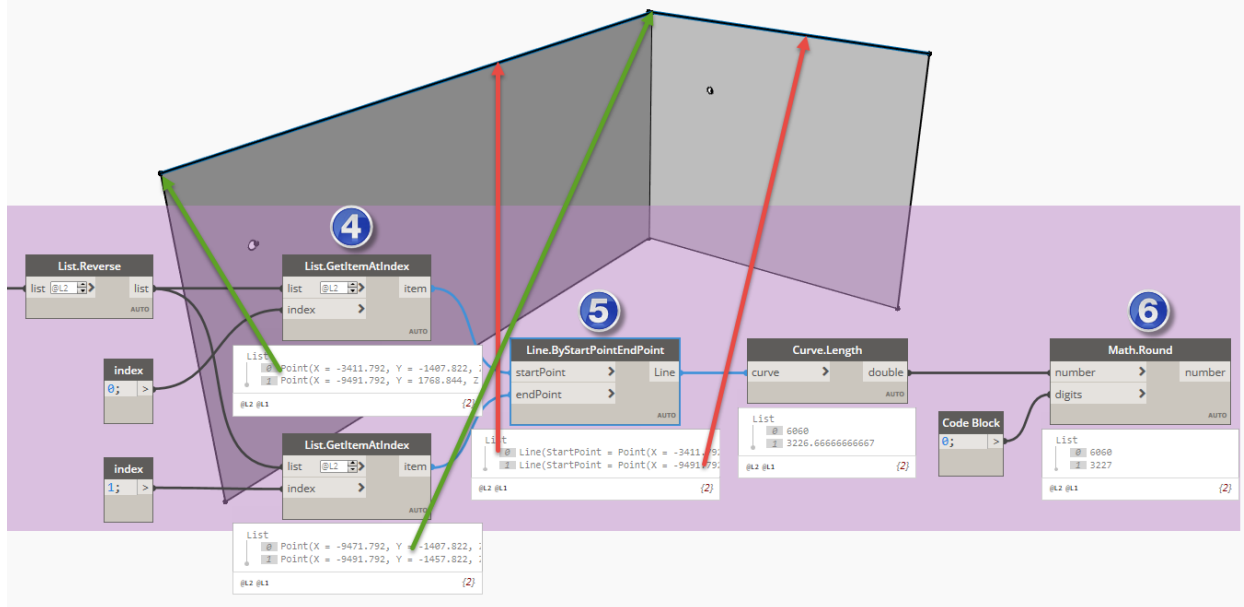


- 4) The first two items of the reversed list are the points of the top curve of the wall part surface.



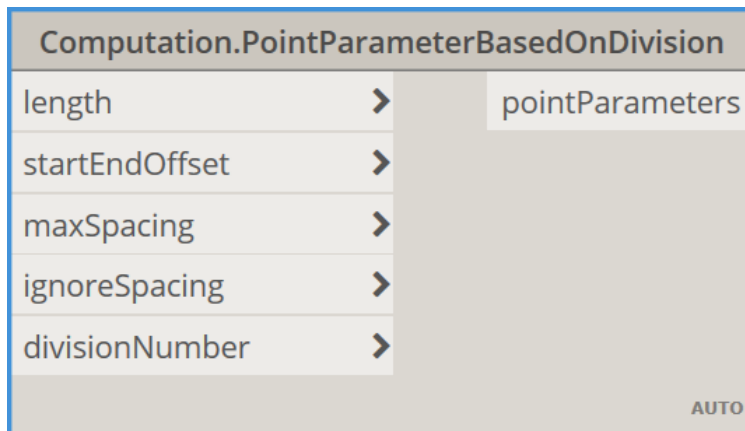
5) Having start and end points we can create a line (top edge).

6) The last step is to get a length of the line. This value is a wall part length.



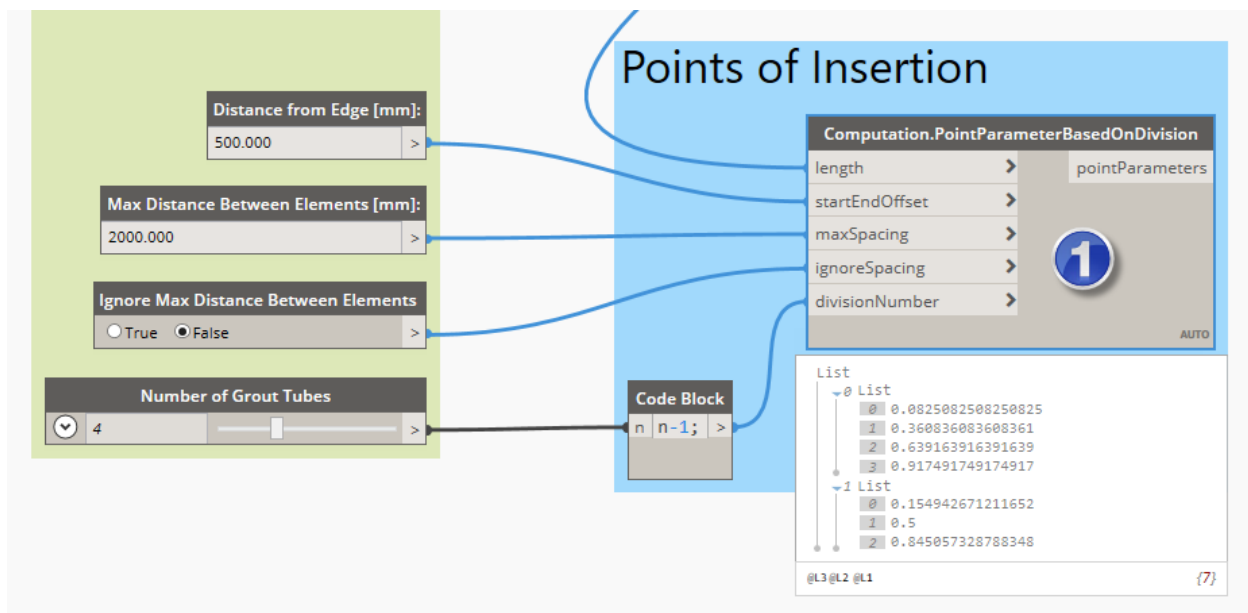
## Points of Insertion

Next, we should determine points where grout tubes and tubes on top will be inserted based on the input parameters. The **Structural Design** package consists of a dedicated node (**Computation.PointParameterBasedOnDivision**) to help you with this.

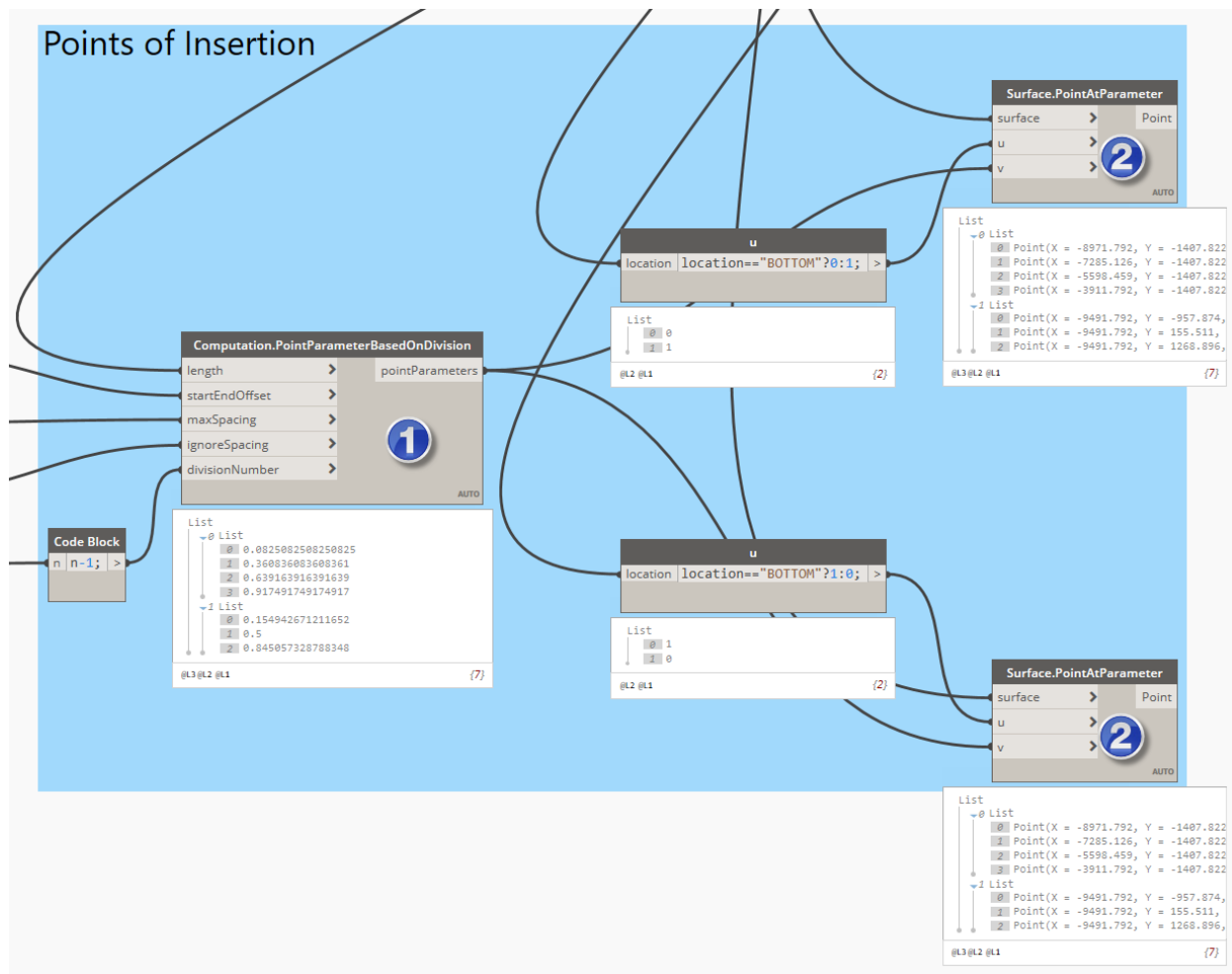


Returns point parameters based on specified division.

- 1) First, let's connect all input parameters and let's see what we get as an output. In this case the *length* parameter is the wall part length we calculated in the previous step.

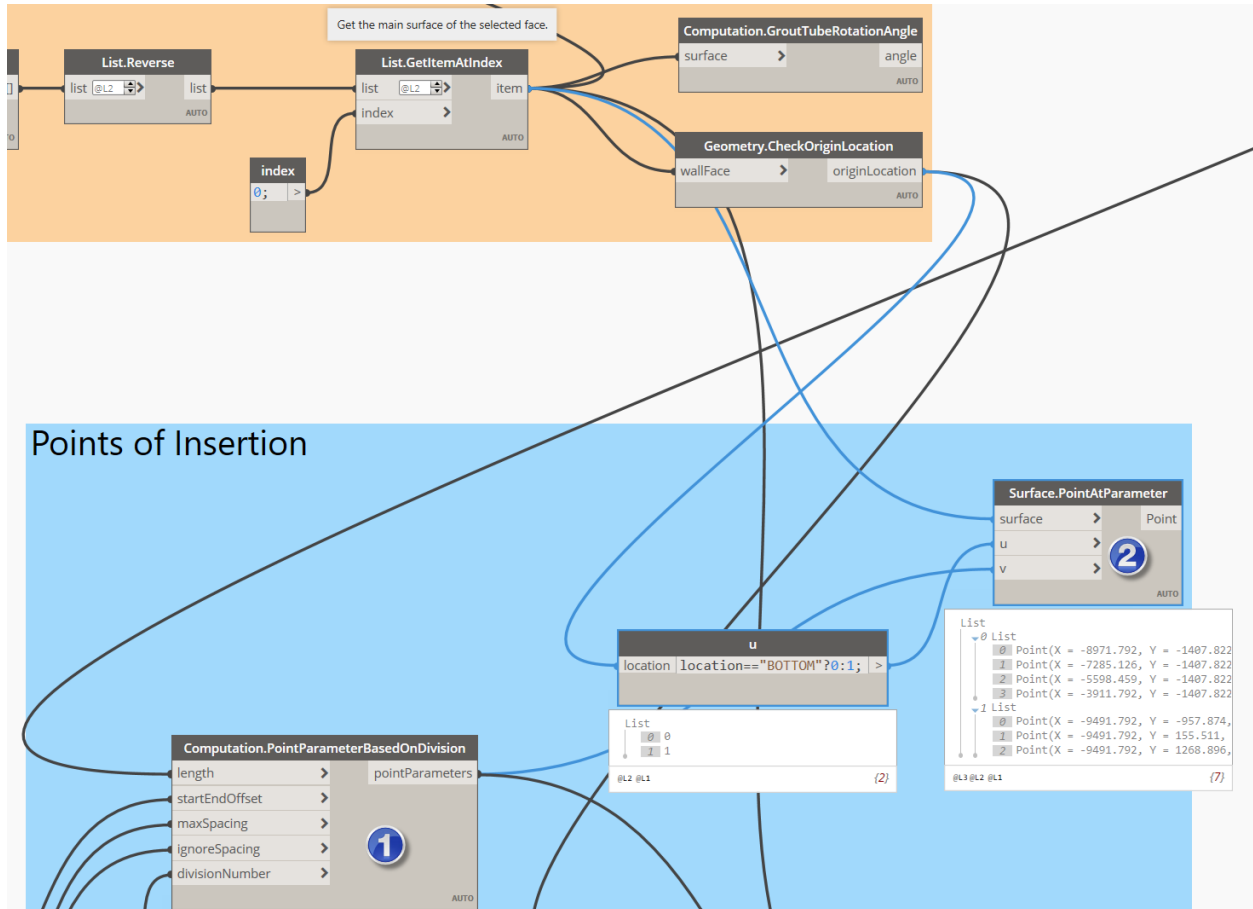


- 2) So far, we have point parameters, but we need to get real insertion points for grout tubes (at the bottom of a precast wall) and for tubes on top (at the top of a precast wall).



To get “u” value we can add the **Code Block** node. In short, code blocks are a text-scripting interface within a visual-scripting environment. They can be used as numbers, strings, formulas, and other data types. There are a few basic shorthand methods in the code block which, simply put, make data management a lot easier. In our example we are using conditional statement formulas: `Location == “BOTTOM” ? 0:1;` and `Location == “BOTTOM” ? 1:0;`

If you are struggling with wire connectivity the below screenshot might be helpful for you.



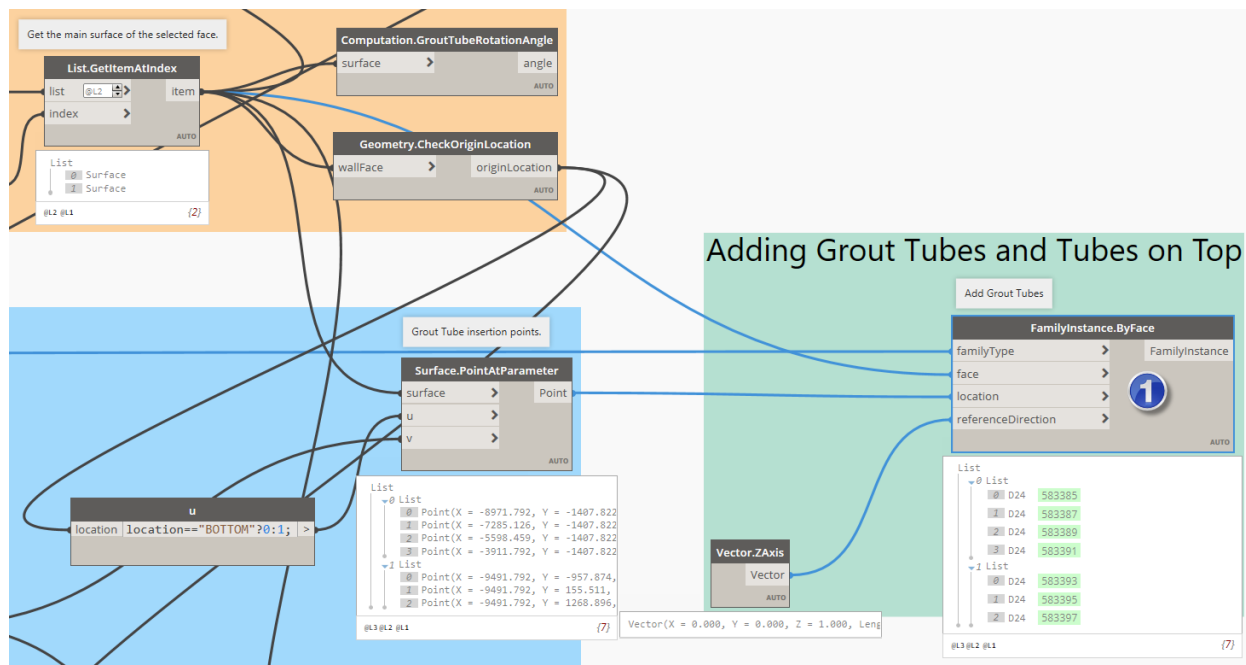
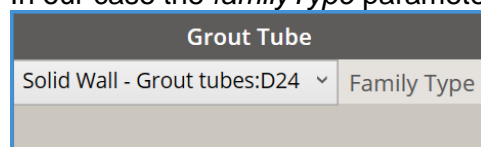
## Adding Grout Tubes and Tubes on Top

Now it's time to finally add grout tubes to our precast walls.

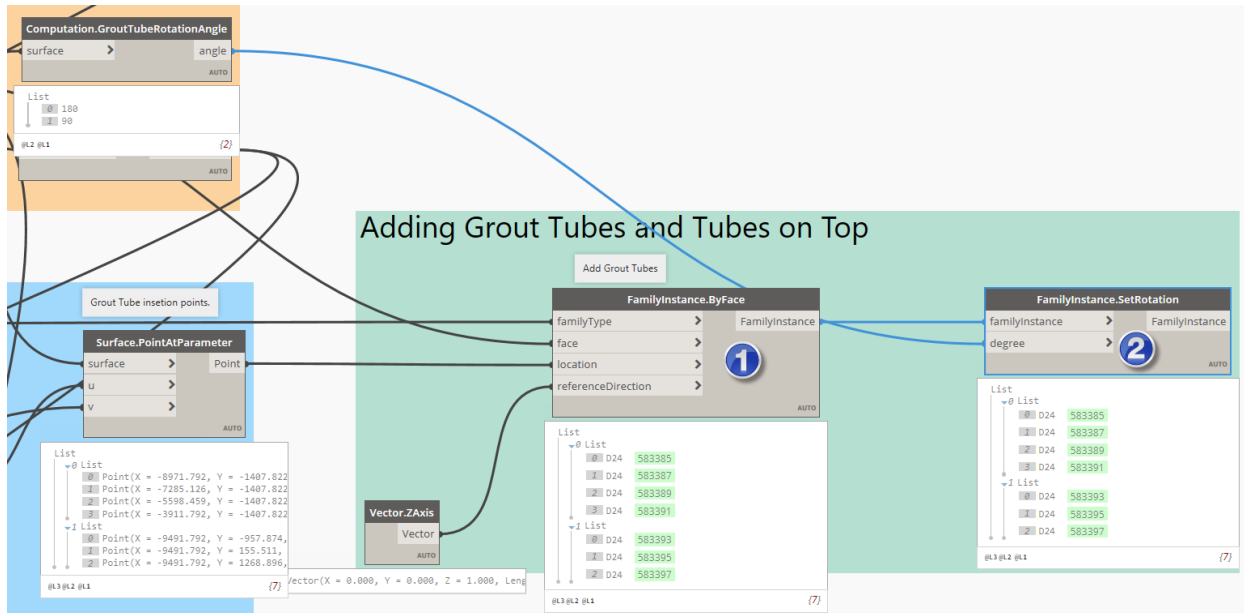
- 1) Let's use the **FamilyInstance.ByFace** node to do this. This node places a Revit family instance given the FamilyType (also known as the FamilySymbol in the Revit API) on a surface derived from a backing Revit face as reference, a reference direction, and a point location where to place the family.

*Note:* The FamilyType should be workplane based and the input surface must be created from a Revit Face. The reference direction defines the rotation of the instance on the reference, and thus cannot be perpendicular to the face.

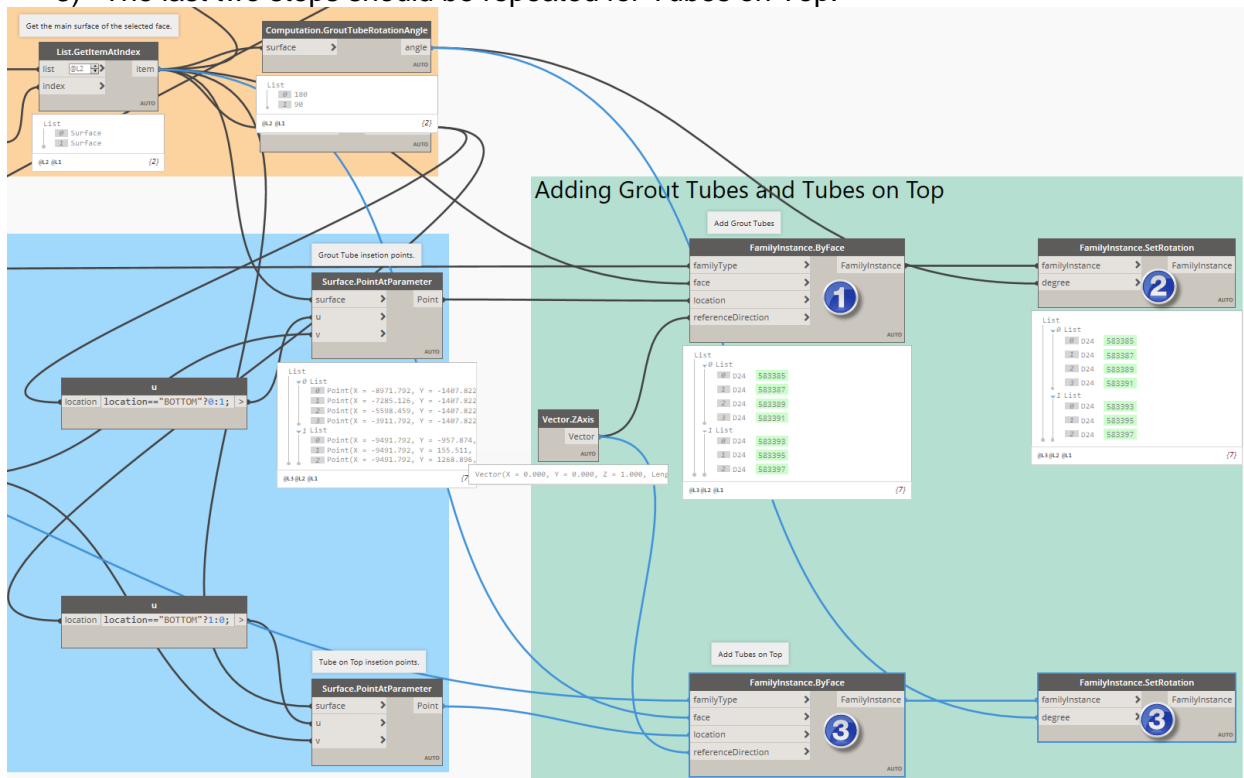
In our case the *familyType* parameter is connected with the *Grout Tube* node.



- 2) We should not forget that we need to orient our grout tubes properly. Let's add the **FamilyInstance.SetRotation** node and connect it with the list of family instances and the list of rotation angles.



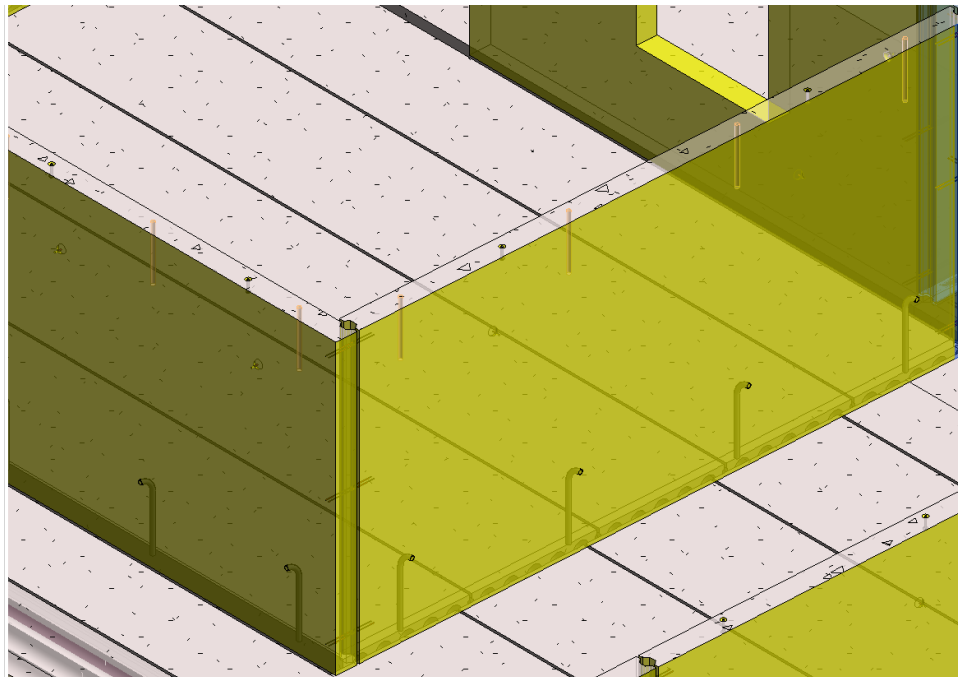
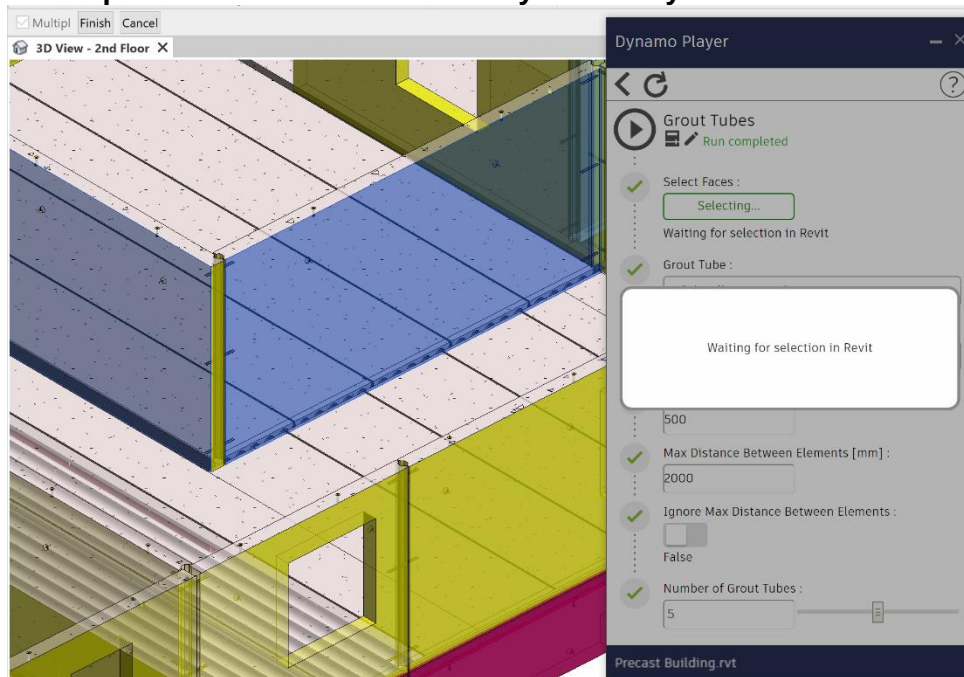
- 3) The last two steps should be repeated for Tubes on Top.



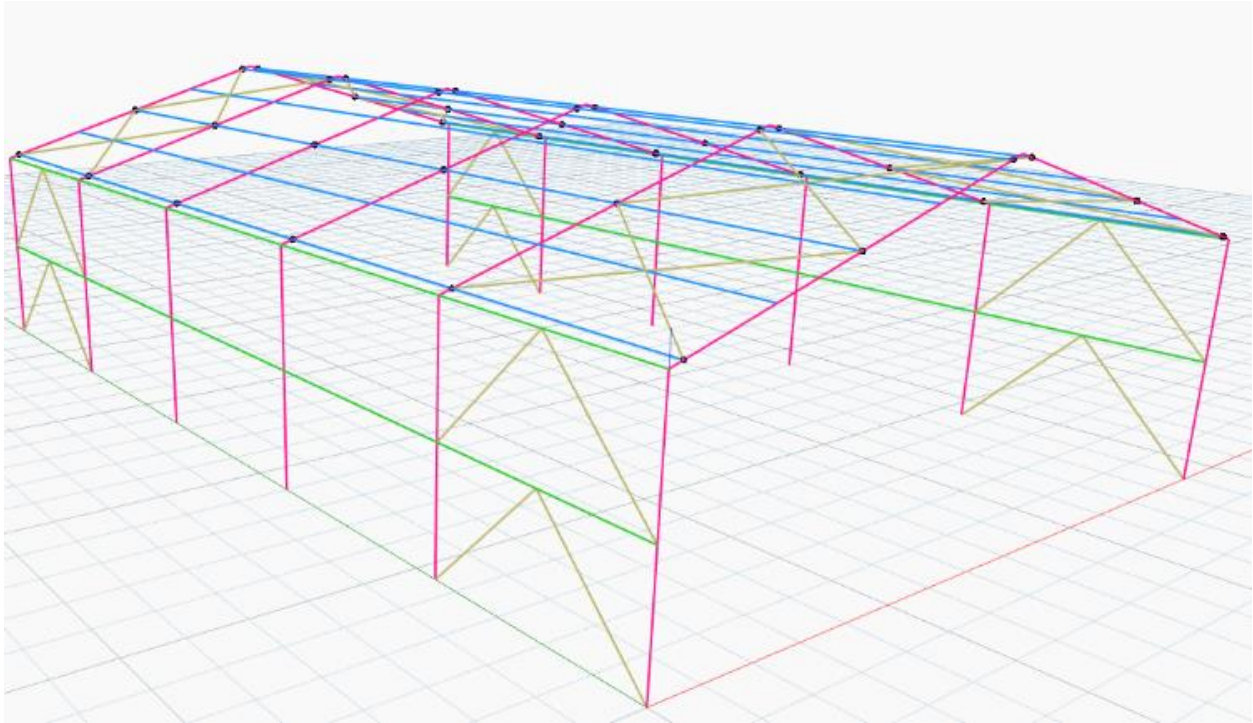


Now you can save the script, close Dynamo and give a try with this solution using Dynamo Player.

**Take a look at the “extra” folder in the Structural Design package installation folder to find the final script that can be used with the Dynamo Player.**



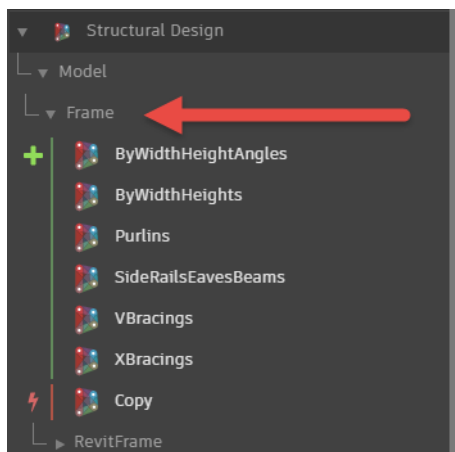
## Portal Frame Geometry in Dynamo



In this chapter I would like to show you how to create a fully parametrized portal frame geometry in the Dynamo environment.

Generally portal frames are used for single-story buildings, but they can be used for low-rise buildings with several floors where they can be economic if the floors do not span right across the building.

Recently I added a few Dynamo nodes to the Structural Design package to help you create such structural geometry.



Let me walk you through details on these nodes and how to put together a sample script using them.

The first two nodes (**Frame.ByWidthHeights**, **Frame.ByWidthHeightAngles**) I would like to talk about create a 2D frame geometry. They do the same and they differ input parameters only.

Frame.ByWidthHeights		
insertionPoint	>	2DFrame
vector	>	Columns
width	>	Beams
height1	>	
height2	>	
AUTO		

*insertionPoint* - Insertion Point. Default value: (0, 0, 0)

*vector* - Vector in the 2D frame plane. Default value: (1, 0, 0)

*width* - Frame width

*height1* - Column height

*height2* - Roof height

Frame.ByWidthHeightAngles		
insertionPoint	>	2DFrame
vector	>	Columns
width	>	Beams
height	>	
angle	>	
AUTO		

*angle* - Beam angle. The angle has to be greater than or equal 0 and less or equal than 60 degrees.

The **Frame.Copy** node helps you duplicate your 2D input frame geometry with a specified spacing between frames. The spacing can be equal or varying.

Frame.Copy		
frame2DGeometry	>	2DFrames
withEqualSpacing	>	Columns
numberOfTimes	>	Beams
spacing	>	
varyingSpacing	>	
flip	>	
AUTO		

*frame2DGeometry* - 2D Frame geometry. Input value should have the following data structure: Curve[Column[], Beam[]]

*withEqualSpacing* - Spacing type. True = equal spacing, False = varying spacing.

*numberOfTimes* - Number of times a 2D frame should be copied when spacing between frames is equal.

*spacing* - Spacing value between frame when distance between them is equal.

*varyingSpacing* - Varying spacing. Default value: [6, 5, 4, 5, 6]

*flip* - Toggles the frame orientation.

The **Frame.Purlins** node creates purlins elements between rafters.

Frame.Purlins		
rafters	>	Purlins
number	>	Points
dp1	>	
dp2	>	
AUTO		

*rafters* - Rafters.

*number* - Number of purlins.

*dp1* - Start offset.

*dp2* - End offset.

The **SideRailsEavesBeams** node creates side rails and eaves beams between columns.

Frame.SideRailsEavesBeams		
columns	>	Beams
heights	>	Points
AUTO		

*columns* - Columns.

*heights* - Z coordinate of beam ends.

The last two nodes I would like to talk about create X and V bracings.

Frame.XBracings		
points	>	XBracings
zones	>	
AUTO		

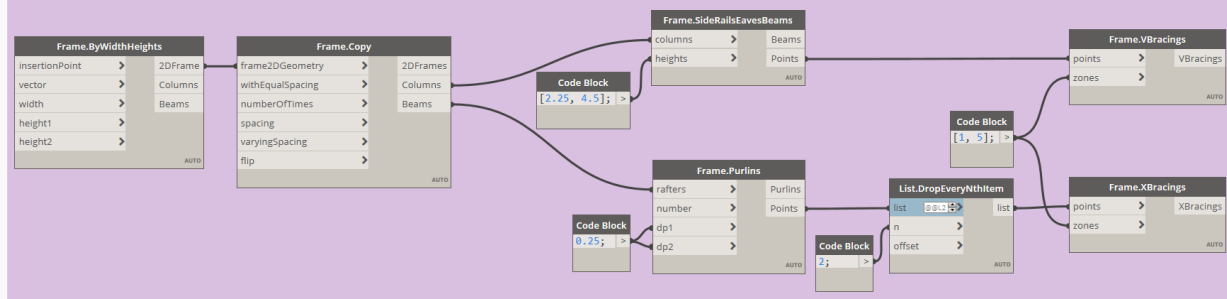
Frame.VBracings		
points	>	VBracings
zones	>	
AUTO		

*points* - Element end points.

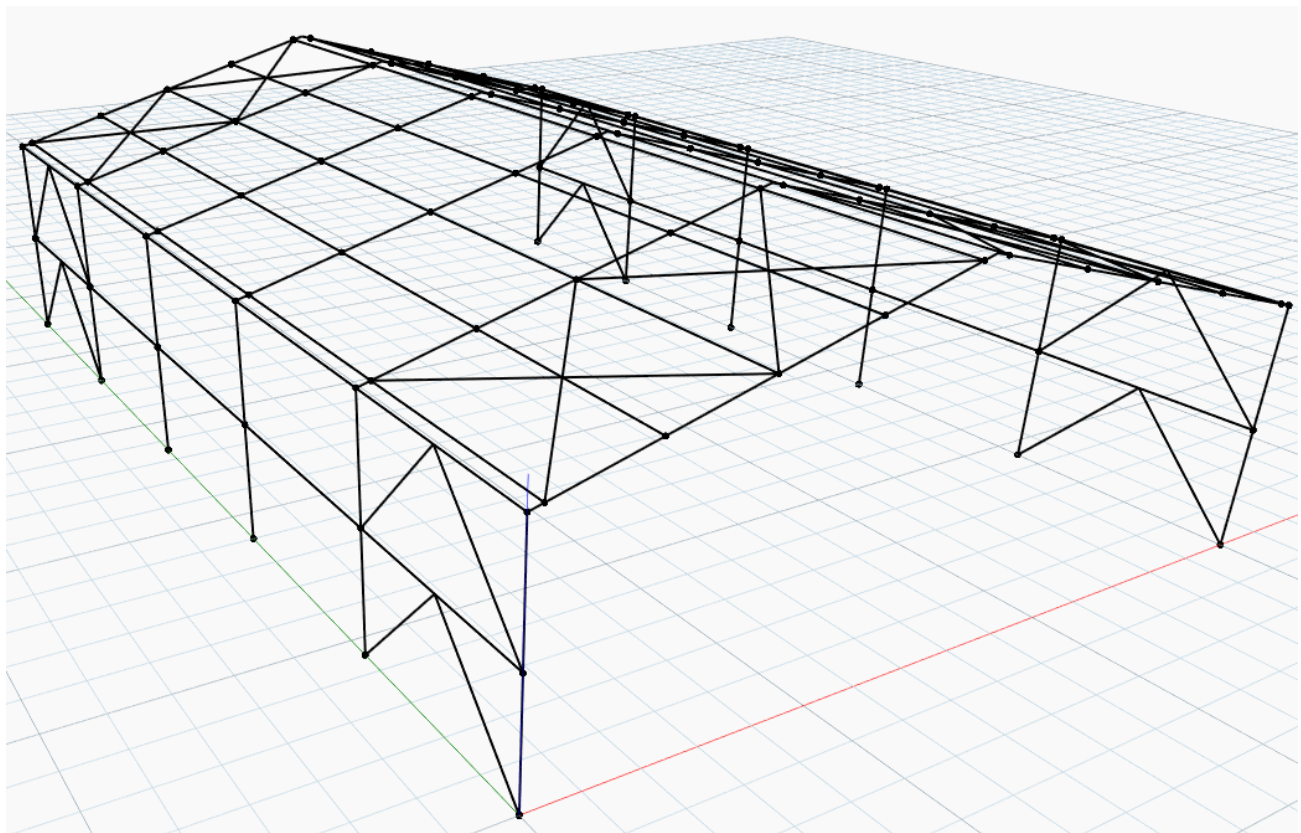
*zones* - Frame zones. A zone is a space between 2D frames, e.g. between frame #1 and frame #2 we have zone #1, between frame #2 and frame #3 we have zone #2 etc...

Building a sample script using the nodes I covered is more than easy. Simply connect these nodes in this way...

### Portal Frame Geometry



...to get the following output:

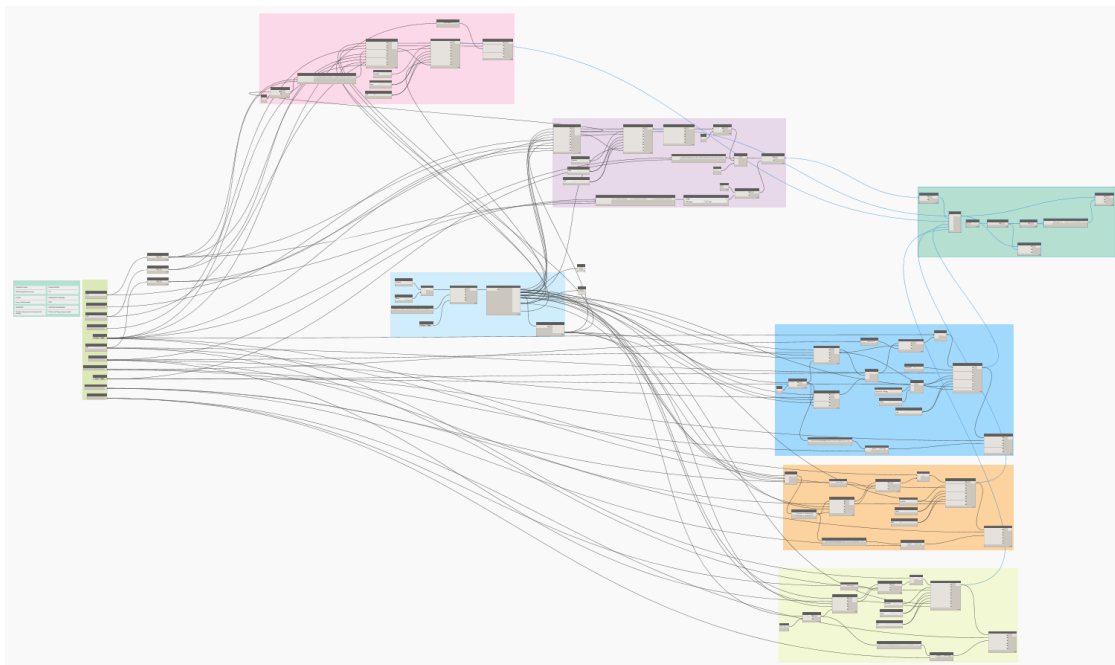
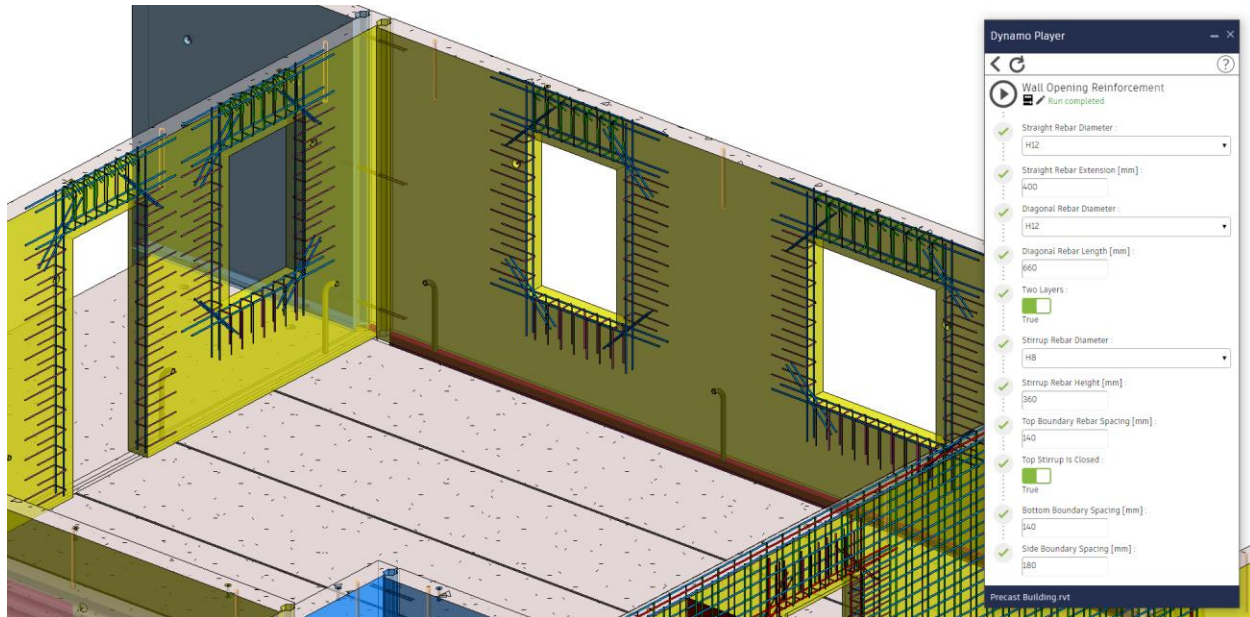


Take a look at the “extra” folder in the Structural Design package installation folder to find the example script.



## Automated process of reinforcement detailing around wall openings using Dynamo

The second example script will show you how quickly you can generate detailed reinforcement designs for window and door openings.



...let's get started...

## Input Parameters

Similarly to what we have done in the first example, let's define some input parameters.

### Input Parameters

Straight Rebar Diameter

H12

Rebar Bar Type

Straight Rebar Extension [mm]

400.000

>

Diagonal Rebar Diameter

H12

Rebar Bar Type

Diagonal Rebar Length [mm]

660.000

>

Two Layers

☒ True
 ☐ False

>

Stirrup Rebar Diameter

H8

Rebar Bar Type

Stirrup Rebar Height [mm]

360.000

>

Top Boundary Rebar Spacing [mm]

140.000

>

Top Stirrup is Closed

☒ True
 ☐ False

>

Bottom Boundary Spacing [mm]

140.000

>

Side Boundary Spacing [mm]

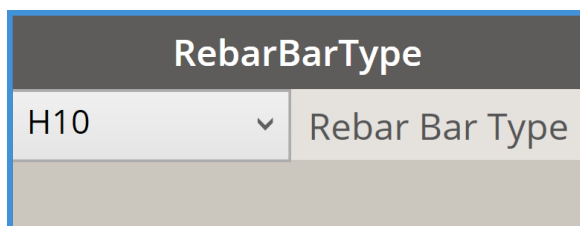
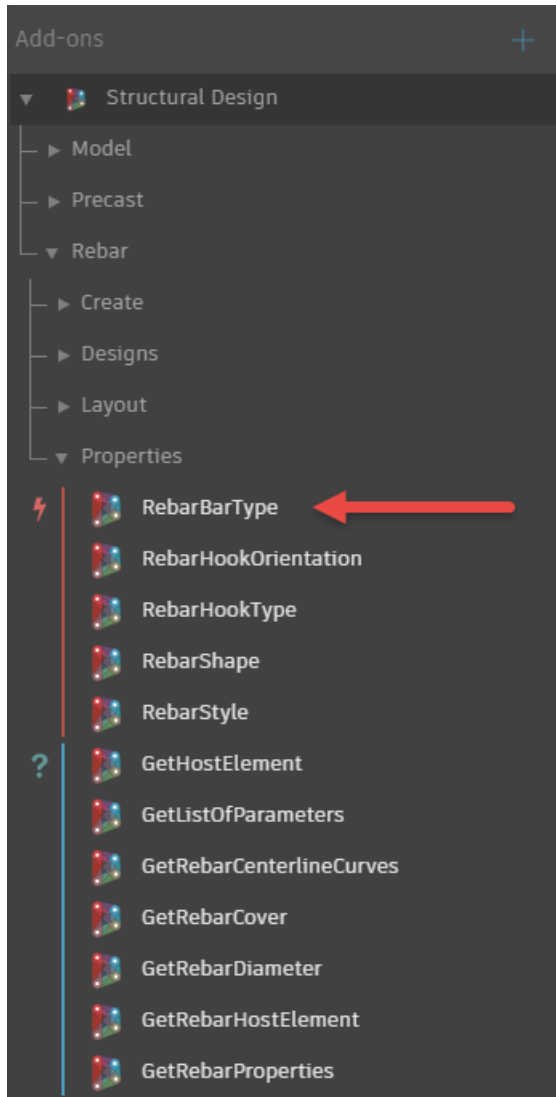
180.000

>

Page 24

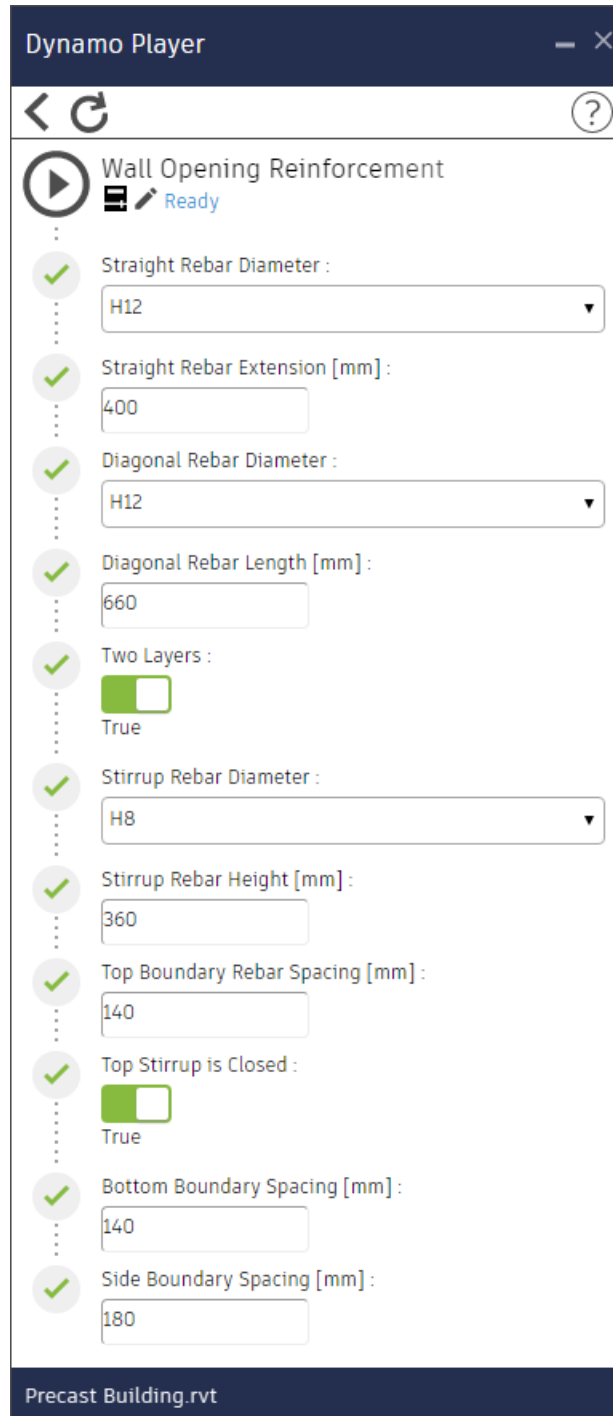


**Note:** To define **Straight Rebar Diameter** and **Diagonal Rebar Diameter** and **Stirrup Rebar Diameter** input nodes you should use the **RebarBarType** node from the **Structural Design** package. It can be found under *Structural Design->Rebar->Properties*.



Double click the node title bar to rename it accordingly.

This is how the input parameters should look in Dynamo Player:



The screenshot shows the 'Dynamo Player' window with the 'Wall Opening Reinforcement' node selected. The node is in a 'Ready' state, indicated by a green checkmark and the word 'Ready' in blue. The input parameters are listed below the node name, each with a green checkmark icon to its left:

- Straight Rebar Diameter :** H12 (dropdown menu)
- Straight Rebar Extension [mm] :** 400 (text input)
- Diagonal Rebar Diameter :** H12 (dropdown menu)
- Diagonal Rebar Length [mm] :** 660 (text input)
- Two Layers :** ☒ True (toggle switch)
- Stirrup Rebar Diameter :** H8 (dropdown menu)
- Stirrup Rebar Height [mm] :** 360 (text input)
- Top Boundary Rebar Spacing [mm] :** 140 (text input)
- Top Stirrup is Closed :** ☒ True (toggle switch)
- Bottom Boundary Spacing [mm] :** 140 (text input)
- Side Boundary Spacing [mm] :** 180 (text input)

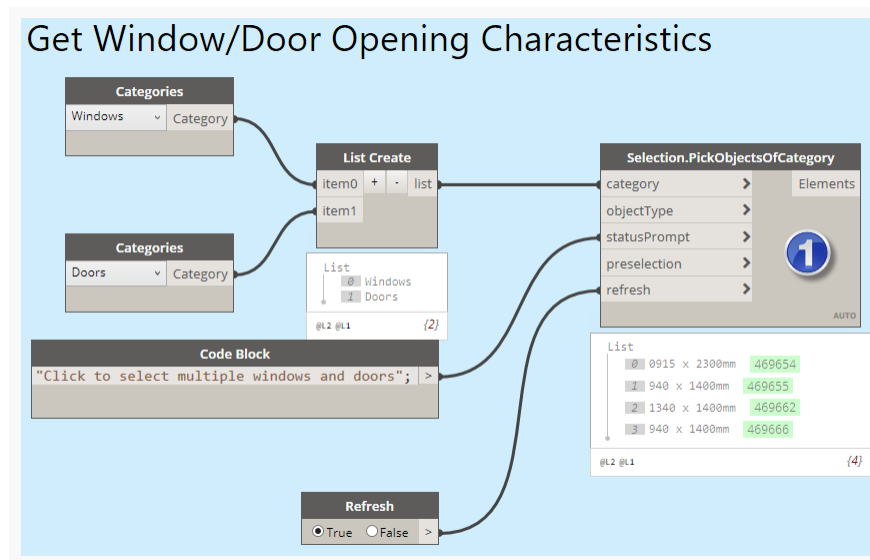
The bottom of the window shows the file name 'Precast Building.rvt'.

## Get Window/Door Opening Characteristics

In the previous example (*Adding Grout Tubes*) we had the *Select* button as a part of the script UI. We needed to click it, select objects/faces and then hit the *Run script* button to execute our script.

In this case we will create a different approach. Instead of clicking the *Select* button, we set up input parameters and click the *Run script* button to execute the script, and then we are asked to make a selection. Once the selection is made the script proceeds its execution.

- 1) We will use the ***Selection.PickObjectsOfCategory*** node to define the selection functionality of our script. The node can be found under *Structural Design->Model->Selection*.



This node prompts the user to select multiple objects of the defined categories while showing a custom status prompt string.

- ***category***: Single category or a list of categories.
- ***objectType* (int)**: Specifies the type of object to be selected 0-Nothing, 1-Element, 2-PointOnElement, 3-Edge, 4-Face, 5-LinkedElement, 6-Subelement. Default value: 1
- ***statusPrompt* (string)**: The message shown on the status bar. *Default value*: "Click to select multiple elements, TAB for alternates, ESC quit"
- ***preselection* (bool)**: Takes into account the initially selected set of objects. *Default value*: false
- ***refresh* (bool)**: Refresh selection. *Default value*: true

In this case the user will be able to pick only elements of the *Window* and the *Door* categories.

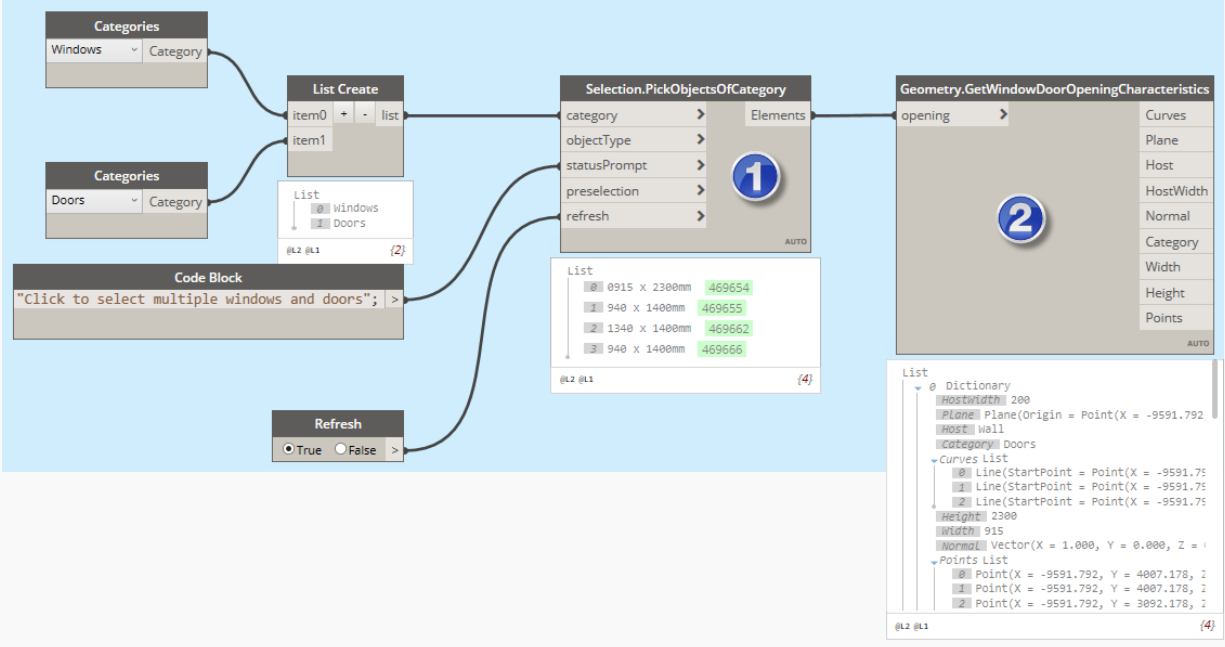
2) Next, we need to retrieve window and door opening characteristics using a dedicated node from the **Structural Design** package.

The **Geometry.WindowDoorOpeningCharacteristics** node can be found under *Structural Design->Model->Geometry*.

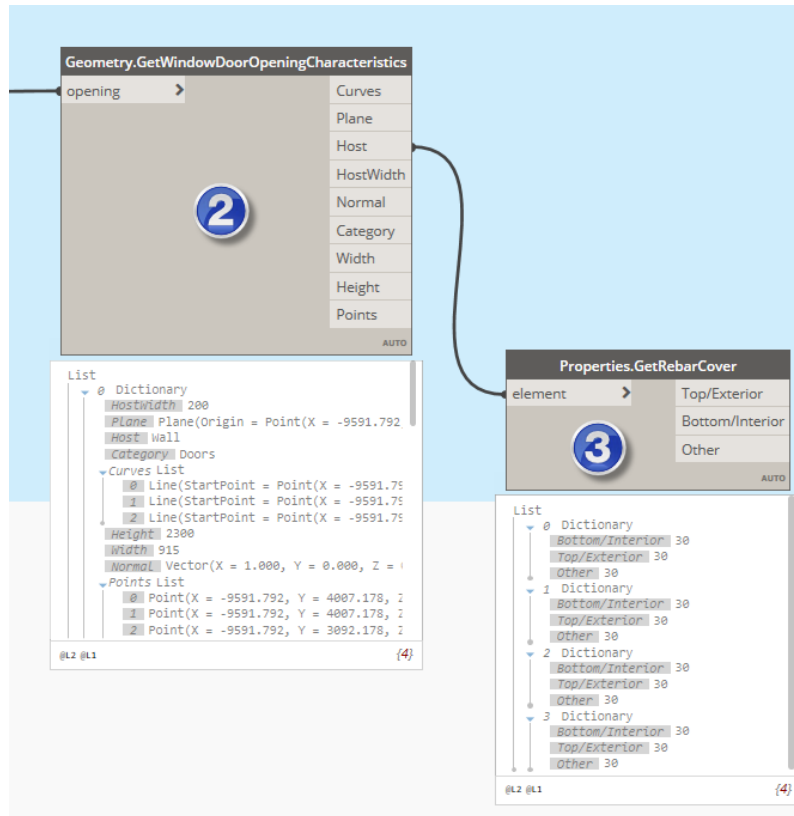
Thanks to this node we can get:

- Curves from which an opening is created.
- Opening plane.
- Opening host.
- Opening host width.
- Vector normal to opening plane.
- Opening category.
- Opening width.
- Opening height.
- Opening corner points.

## Get Window/Door Opening Characteristics



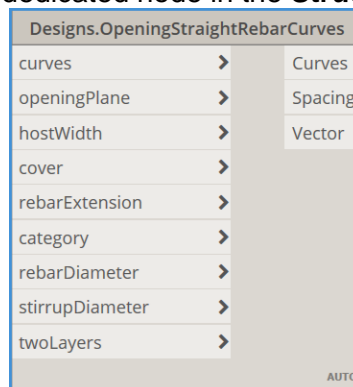
3) Let's get the rebar cover dimensions for the host of the selected opening element.



## Straight Rebars

Now, based on the selected opening characteristics, we can create straight rebars around the opening.

- 1) First, we need to generate Dynamo curves which will be used during the creation of straight rebars. There is a dedicated node in the **Structural Design** package to do this.



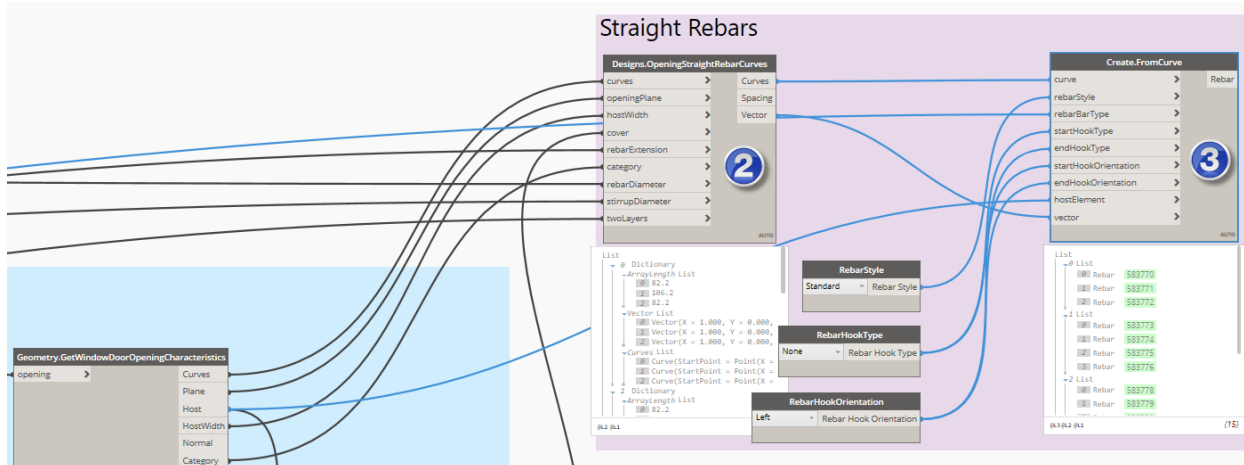
ia



ia



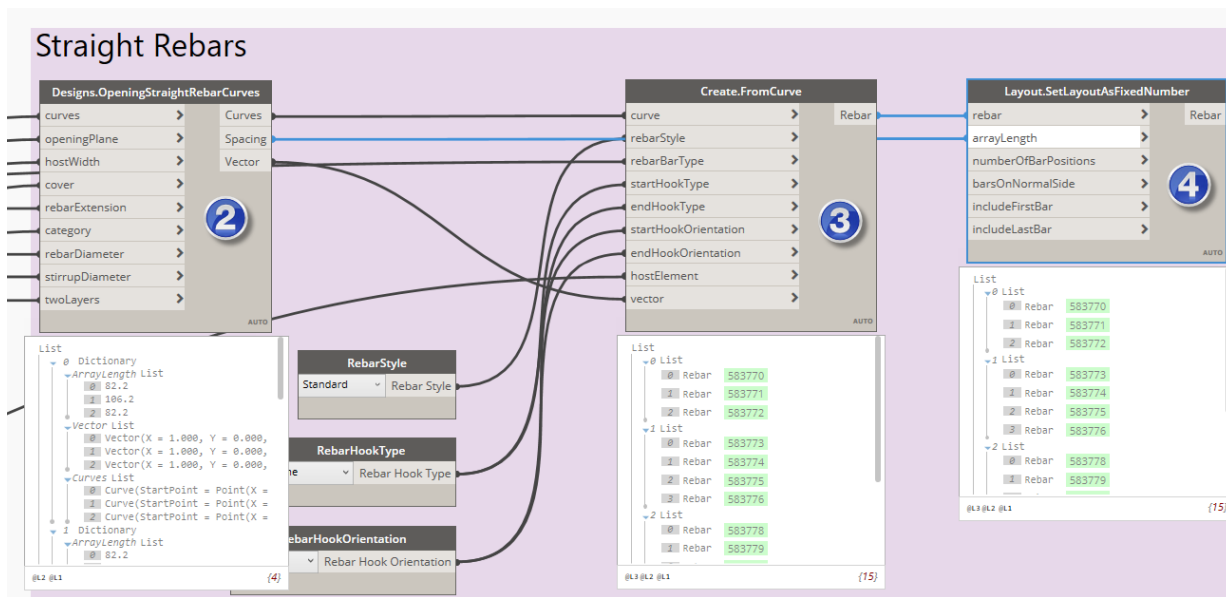
- 3) Next, we can create Revit rebars using the **Create.FromCurve** node (Structural Design->Rebar->Create).



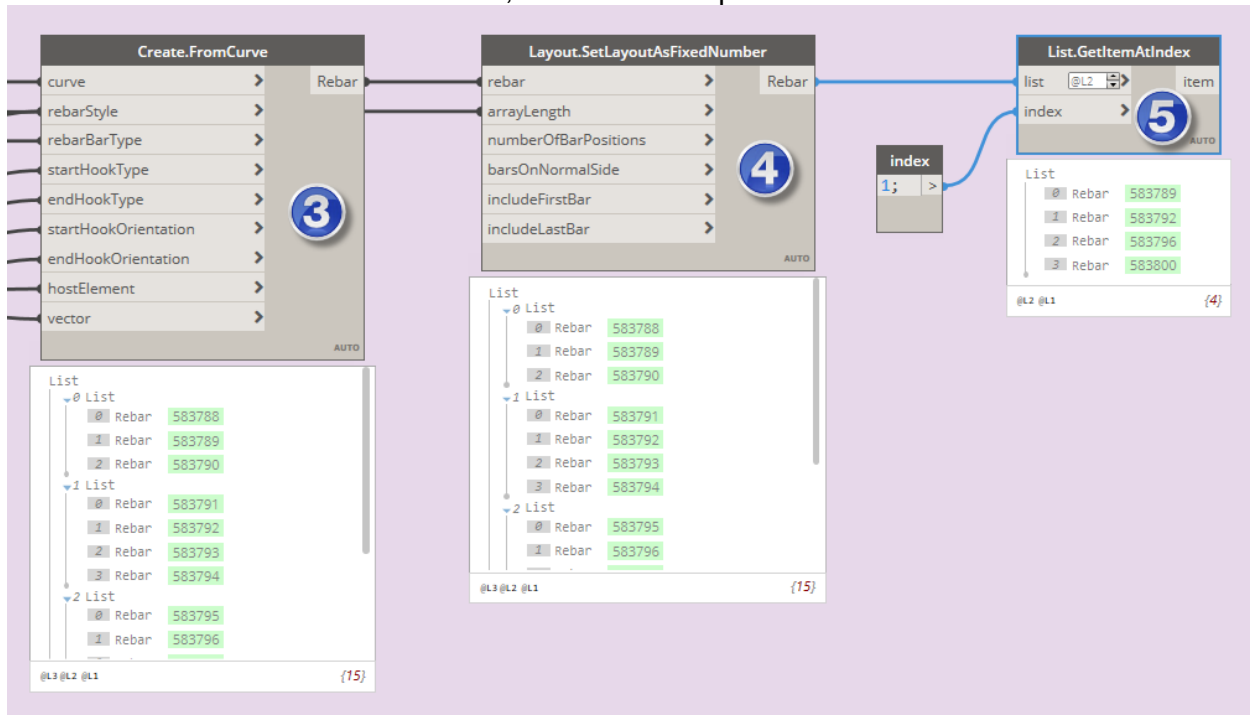
**Note:** The respective vectors are taken from the **Vector** output of the **Designs.OpeningStraightRebarCurves** node. **Vector** is the normal to the plane that the rebar curve lies on.

- 4) Next, let's set the Layout Rule property of newly created rebars to Fixed Number. The default number of bar positions in rebar set is 2, so no need to make a connect in this case as this is the number of rebars we want to set up.

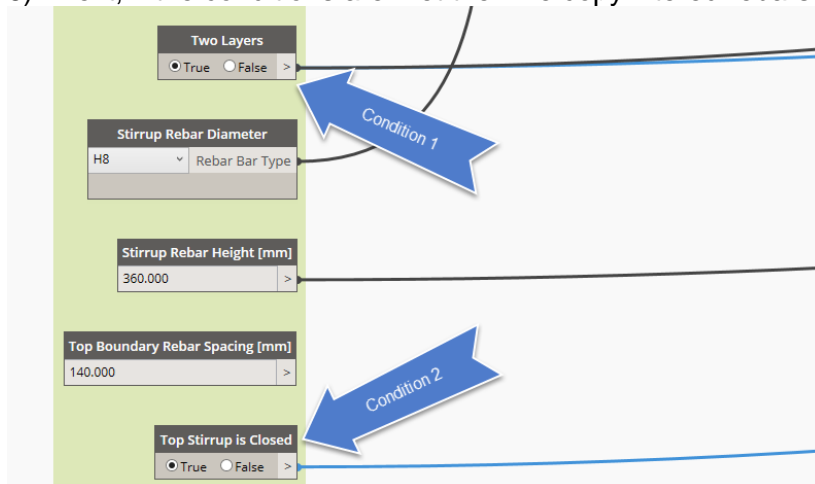
**Note:** The respective distribution length of rebar sets (**arrayLength**) is taken from the **Spacing** output of the **Designs.OpeningStraightRebarCurves** node.



- 5) There is one more thing we are going to include in the logic of creation of straight rebars. When the **Top Stirrup is Closed** input parameter is *True*, and the **Two Layers** parameter is *True* too, then we would like to have the top straight rebars copied to create reinforcement of a lintel. To do this, let's filter out top bars first.

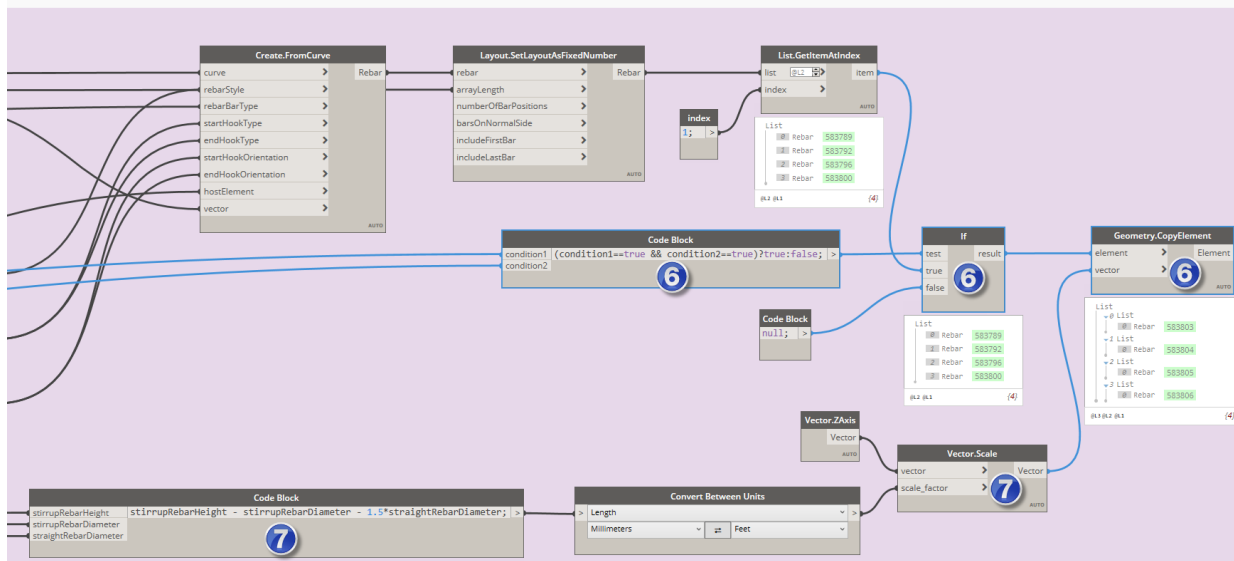


- 6) Next, if the conditions are met then we copy filtered rebars.





7) Of course, we need to determine a vector that is used when we copy elements.

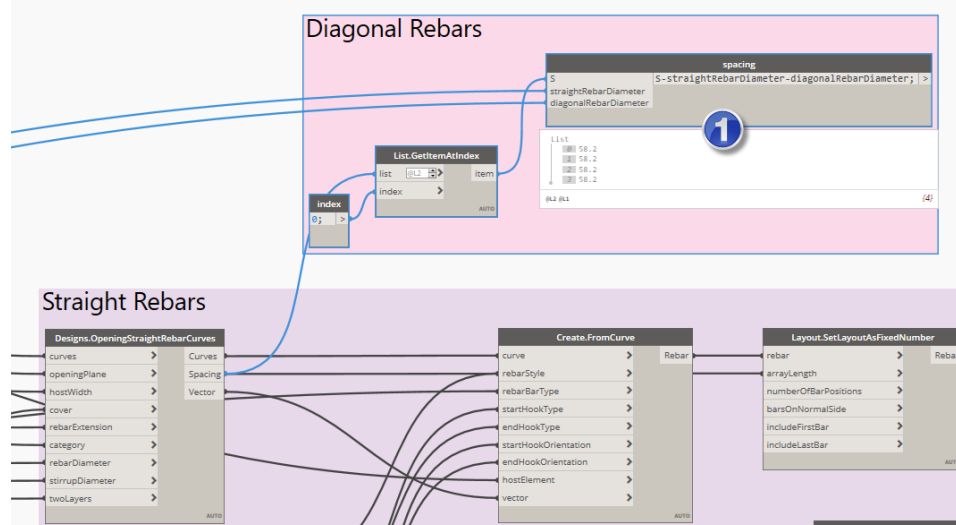


**Note:** In this case we don't perform an operation on a Dynamo geometry when we copy elements, but instead when we copy Revit objects. The **Geometry.CopyElement** node copies an element and places the copy at a location indicated by a given transformation (*Structural Design->Model->Geometry*).

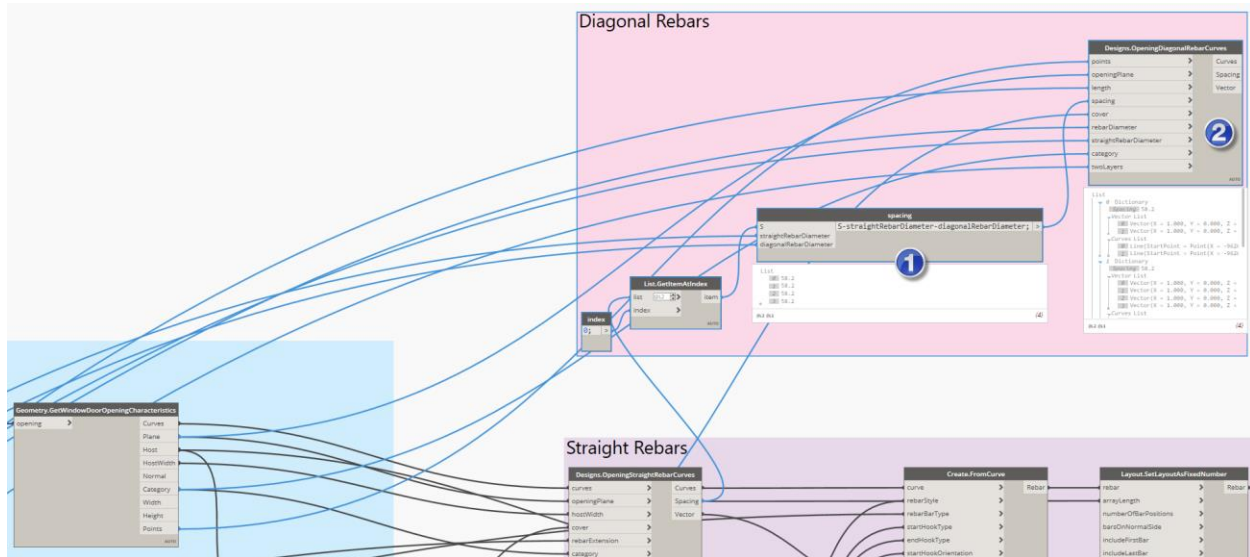
## Diagonal Rebars

The creation of diagonal rebars is very similar to what we have done for straight ones.

- 1) Before we wire the **Designs.OpeningDiagonalRebarCurves** node to the right data, we need to calculate a spacing between diagonal rebars.

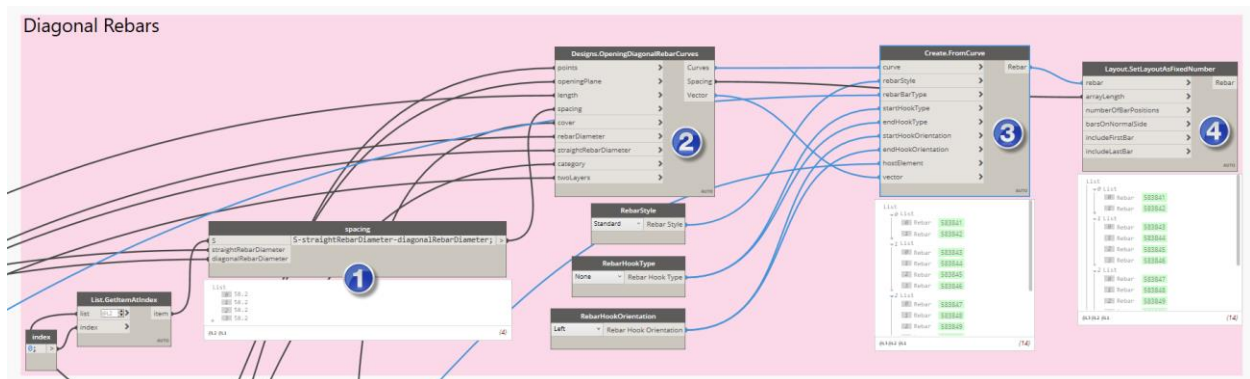


2) Now, we are ready to generate curves for diagonal rebars.



3) Now, it's time to create rebars in Revit.

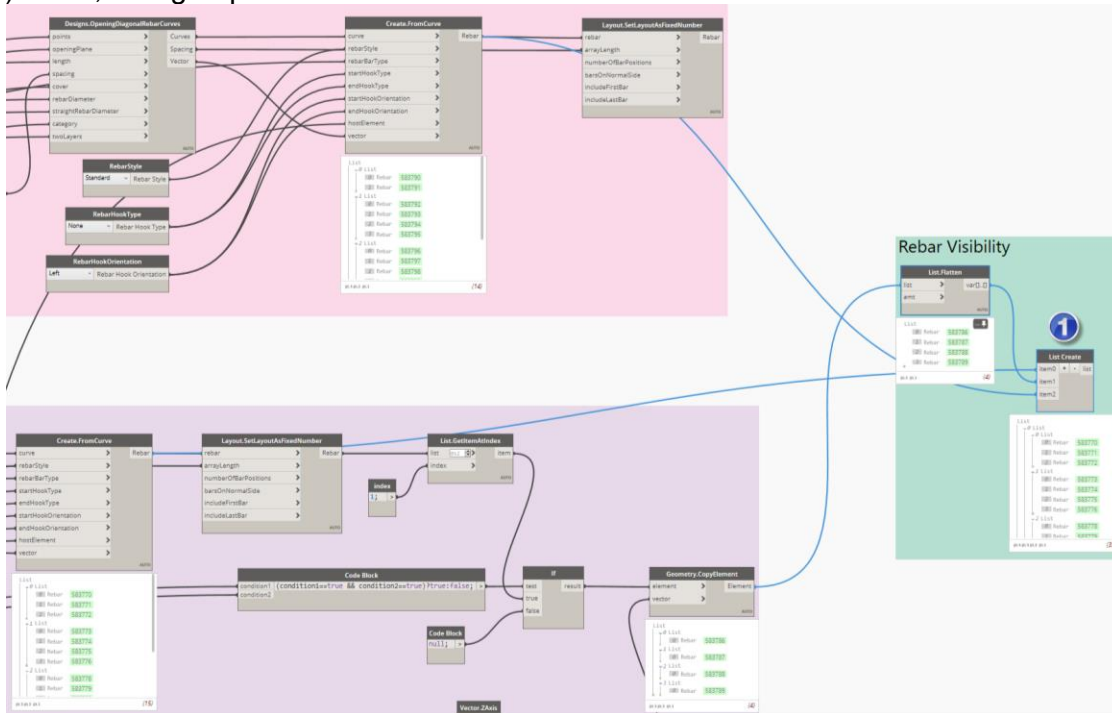
4) The final step is a definition of a rebar set if a spacing is greater than 0 (**Two Layers** is *true*).



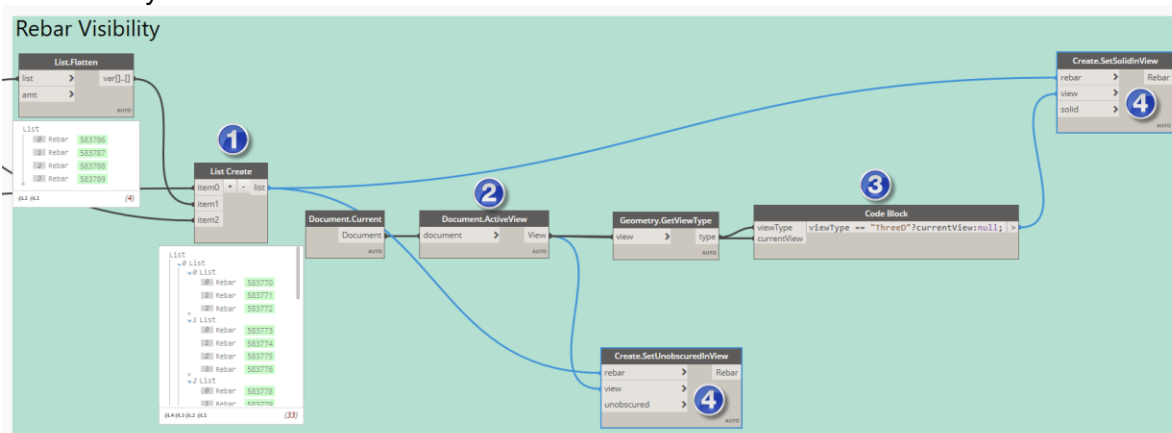
## Rebar Visibility

Before we move on, let's take care of rebar visibility as we would like to have the generated rebars presented as solid and unobscured in the active view.

- 1) First, let's group all rebars we created so far into a list.

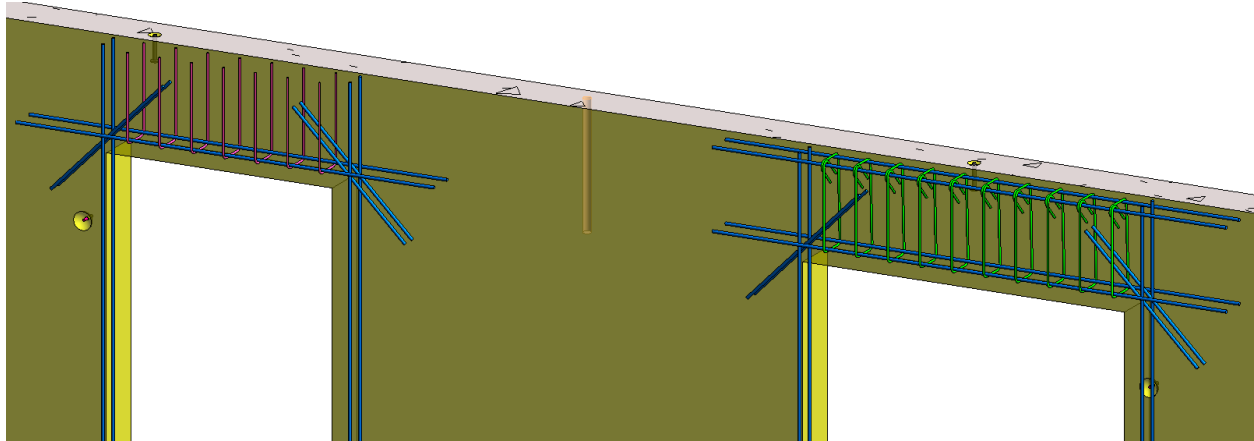


- 2) Next, let's retrieve the current active view.
- 3) To set up rebar visibility we will use the **Create.SetUnobscuredInView** and the **Create.SetSolidInView** nodes. The second node takes a 3D View as an input parameter, so we need to check if the current view is a 3D View.
- 4) Finally, we can set up rebar visibility. The **unobscured** and **solid** input parameters are *true* by default.



## Boundary Stirrups

Let's continue and create the remaining rebars. Now we are going to create top boundary stirrups.



As you can see on the above image depending on the **Top Stirrup is Closed** parameter value (*true/false*) we will be getting two different types/shapes of stirrups. In the **Structural Design** package, you can find two respective nodes that support creations of these two shapes (*Structural Design->Rebar->Designs*).

Designs.OpeningStirrupShapeCurves		
openingPlane	>	Curves
segmentA	>	Vector
hostWidth	>	
cover	>	
openingCurve	>	
AUTO		

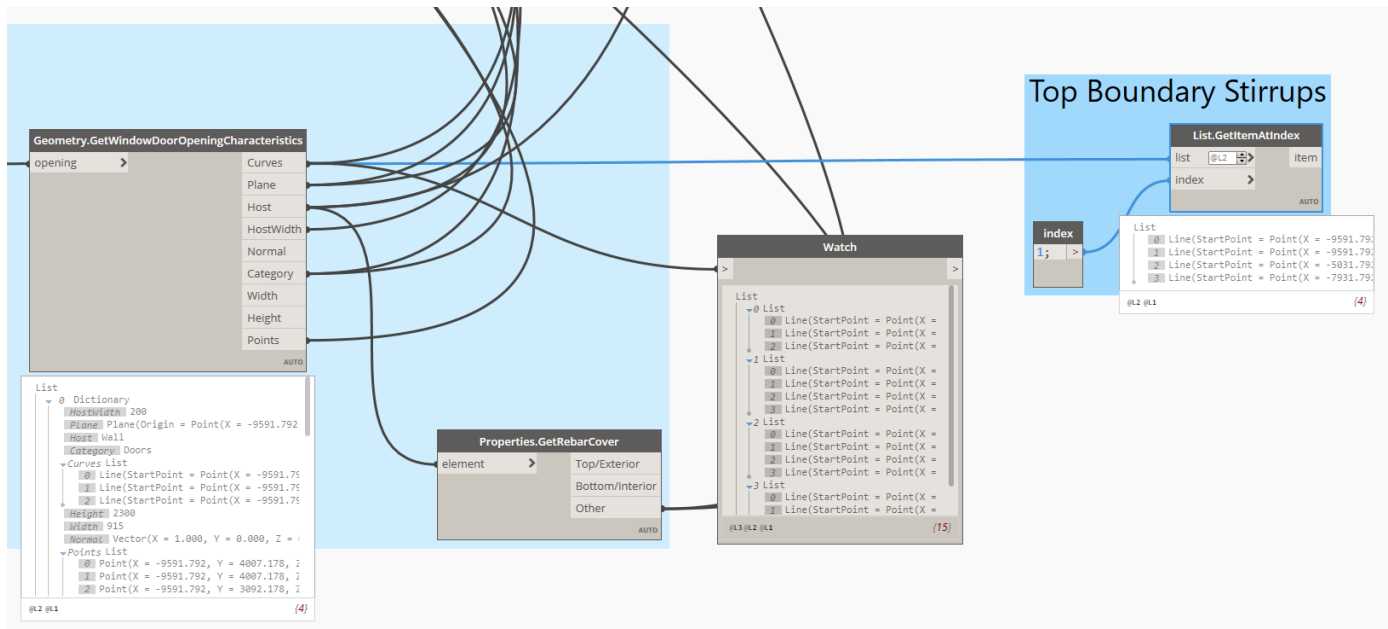
Returns closed stirrup shape rebar curves.

Designs.OpeningUShapeCurves		
openingPlane	>	Curves
segmentA	>	Vector
hostWidth	>	
cover	>	
openingCurve	>	
AUTO		

Returns U-shape rebar curves.

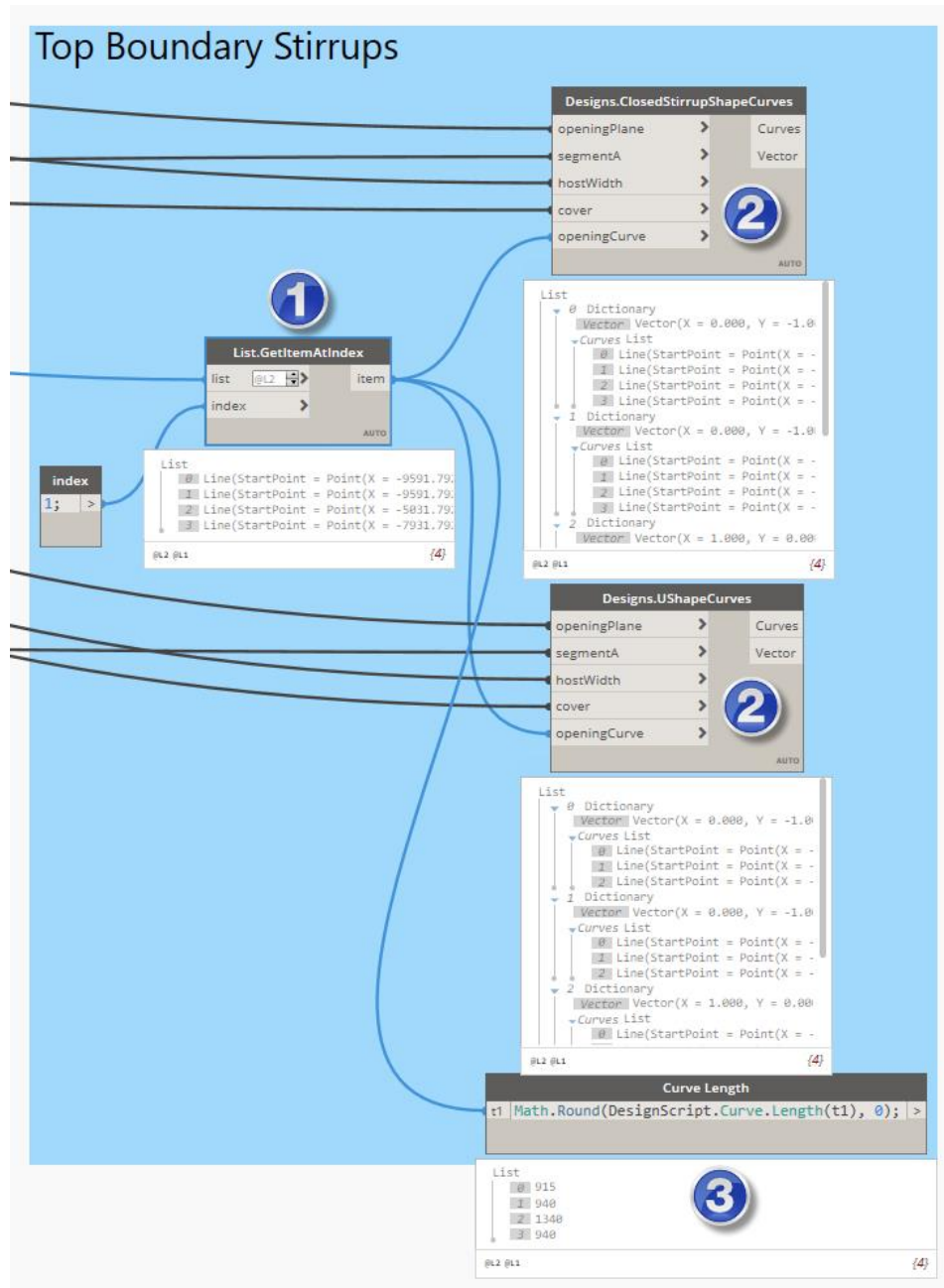
They both also return the normal vector to the plane that the rebar curve lies on.

- 1) First, we need to retrieve all top curves openings that were collected during the selection process. To get such list of top curves we need to collect all second items of the Curves[ ] list.



**Note:** That in this case we're getting 4 items as 4 openings were selected.

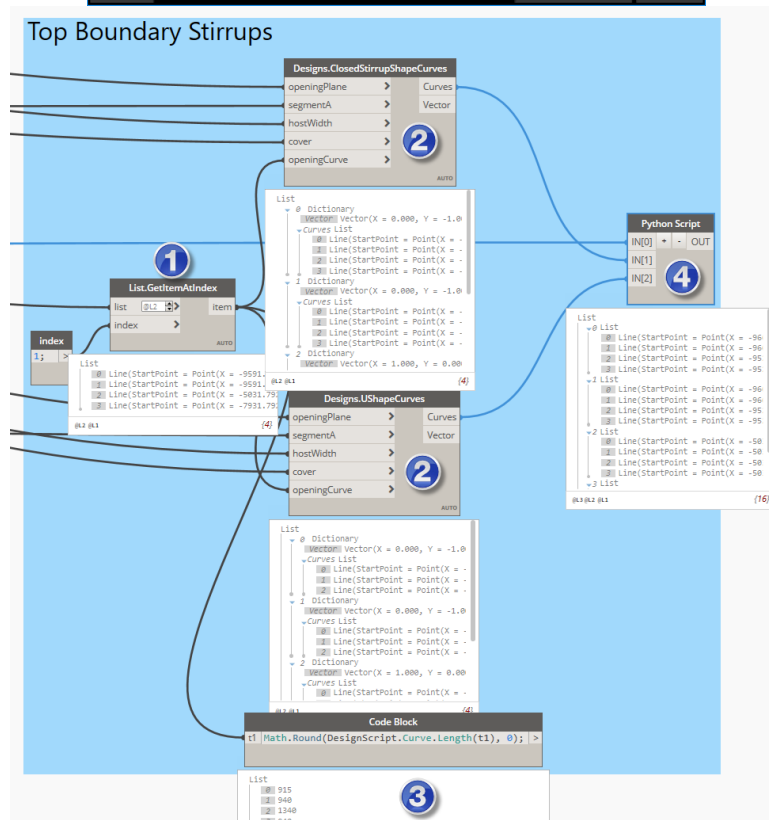
- 2) Next connect the right data with the **\*ShapeCurves** nodes.
- 3) Later we will need the lengths of the curves so let's get these values using the Design Script syntax.



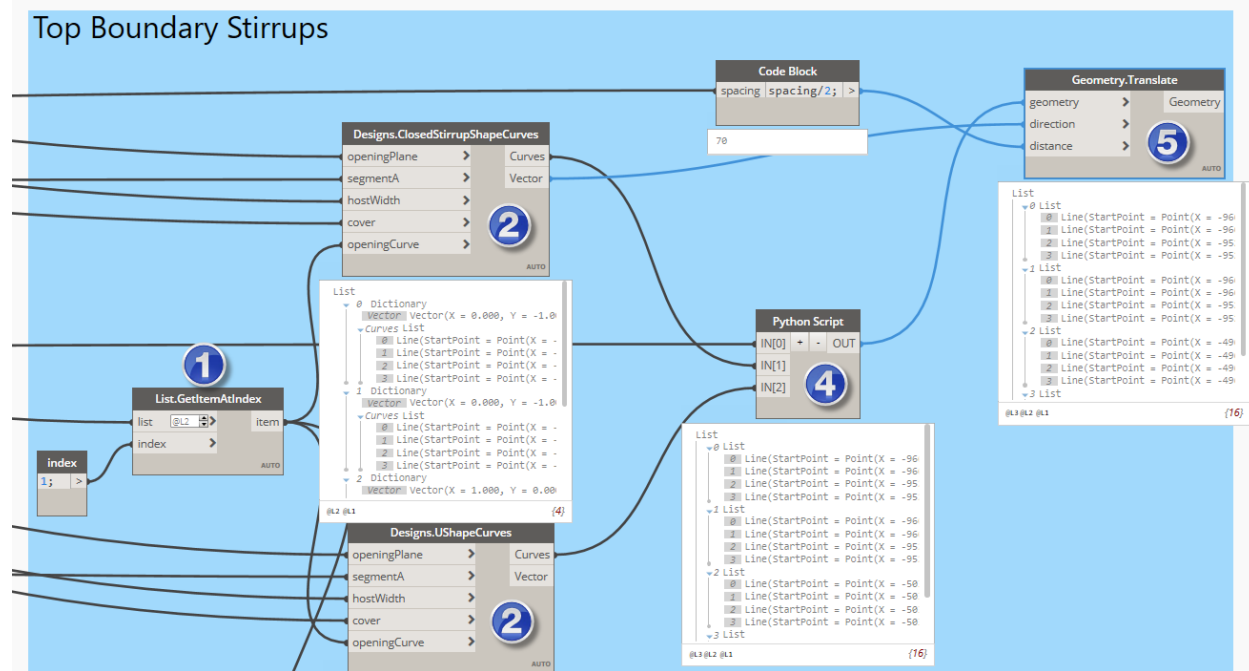
- 4) Having the stirrup geometry defined, it's time to add a logic that will manage what kind of stirrup is used. We will do this using the **Python Script** node. Our simple Python Script will return the result of the node's inputs passed through an embedded IronPython script. We will define 3 inputs:
- condition – a value of the **Top Stirrup is Closed** parameter
  - closedStirrups – a list of curves to create closed stirrups
  - uShapeStirrups – a list of curves to create U-shape stirrups

```

Python Script
1 # Enable Python support and load DesignScript library
2 import clr
3 clr.AddReference('ProtoGeometry')
4 from Autodesk.DesignScript.Geometry import *
5
6 # The inputs to this node will be stored as a list in the IN variables.
7 dataEnteringNode = IN
8
9 condition = IN[0]
10 closedStirrups = IN[1]
11 uShapeStirrups = IN[2]
12
13 if condition:
14     output = closedStirrups
15 else:
16     output = uShapeStirrups
17
18 # Place your code below this line
19
20 # Assign your output to the OUT variable.
21 OUT = output
  
```



- 5) Next, we need to translate a representative stirrup the half stirrup spacing distance along the top border of the opening.



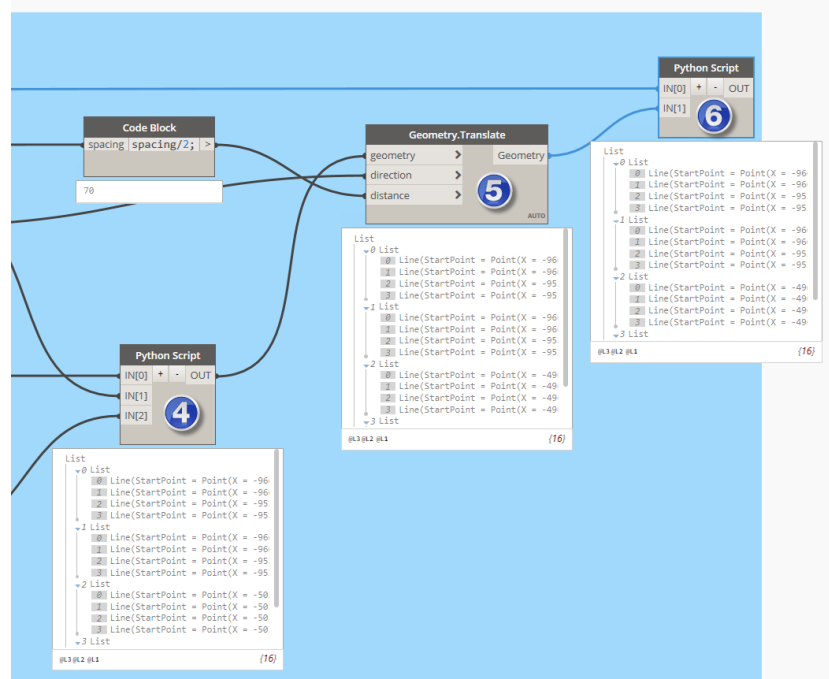
- 6) Now we can create rebars in Revit, but before we do this we need to check if the creation of stirrups makes sense (if the **Two Layers** parameter is **True**). Again, in this case we will use the **Python Script** node and write a simple script.

```

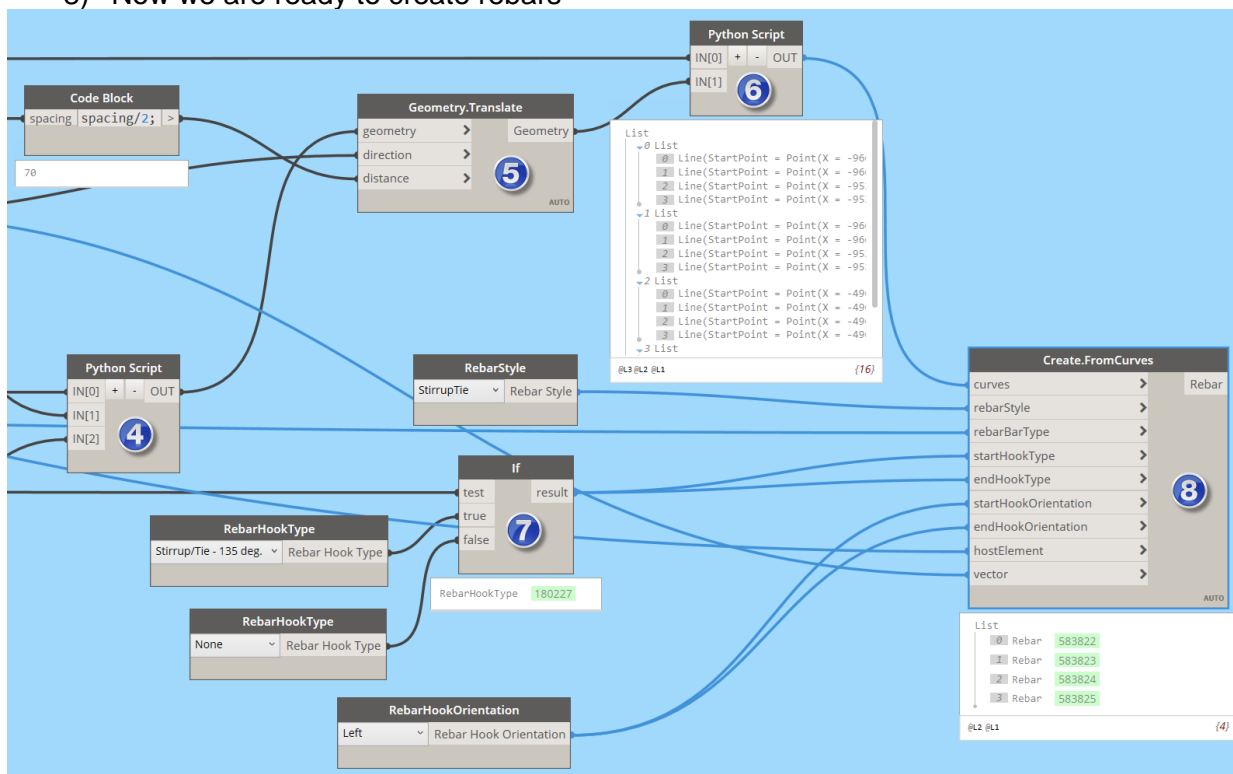
Python Script
1 # Enable Python support and load DesignScript library
2 import clr
3 clr.AddReference('ProtoGeometry')
4 from Autodesk.DesignScript.Geometry import *
5
6 # The inputs to this node will be stored as a list in the IN variables.
7 dataEnteringNode = IN
8
9 twoLayers = IN[0]
10 rebarCurves = IN[1]
11
12 if twoLayers:
13     rebarCurves = rebarCurves
14 else:
15     rebarCurves = []
16
17 # Assign your output to the OUT variable.
18 OUT = rebarCurves
  
```

Run Save Changes Revert

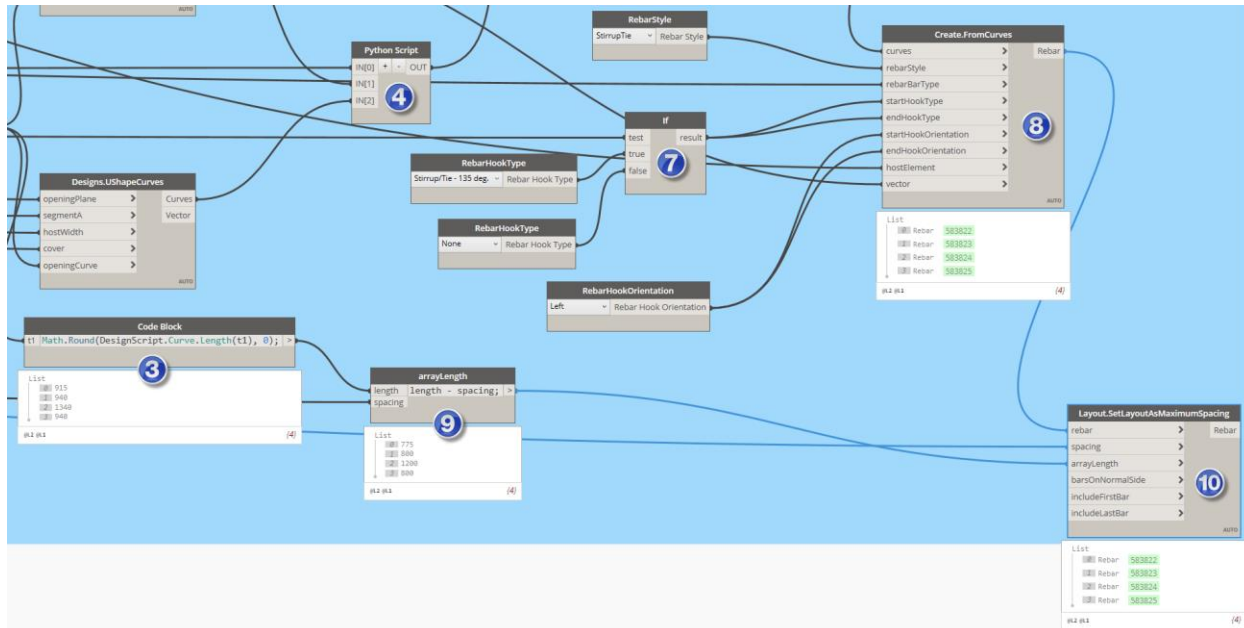




- 7) Because the *RebarHookType* depends on the **Top Stirrup is Closed** parameter, we need to check it first and then connect the right type to the **Create.FromCurves** node.
- 8) Now we are ready to create rebars



- 9) To define a rebar set from our stirrup we need to calculate *arrayLength* first. In this case again we can use the **Code Block** node to provide a formula and renaming the node we can call it respectively.
- 10) Finally, we have everything to create a set of stirrups.

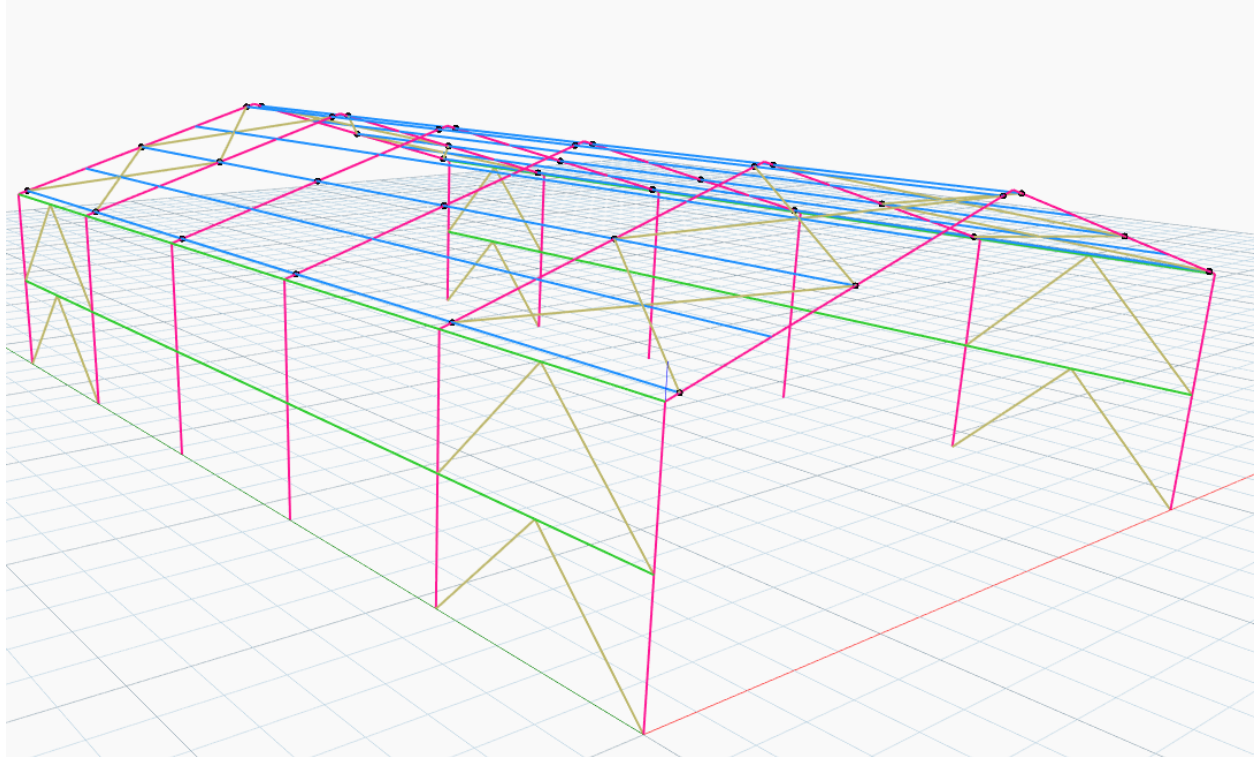


The process of creation of the bottom and side rebars is very similar.

Take a look at the “extra” folder in the Structural Design package installation folder to find the final script that can be used with the Dynamo Player.

## Portal Frame with Structural Analysis for Dynamo Package

In one of my previous chapters I showed you how to create a fully parametrized portal frame geometry in the Dynamo environment.

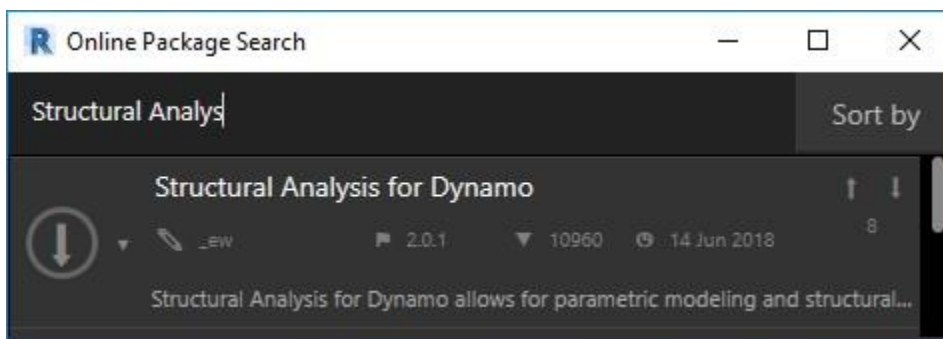


## Structural Analysis for Dynamo Package

After the Dynamo geometry is set, you can create the analytical model representation of this geometry in the [Robot Structural Analysis Professional](#) environment.

The Structural Analysis for Dynamo package enables parametric modeling and structural analysis workflows in Dynamo and Robot Structural Analysis Professional.

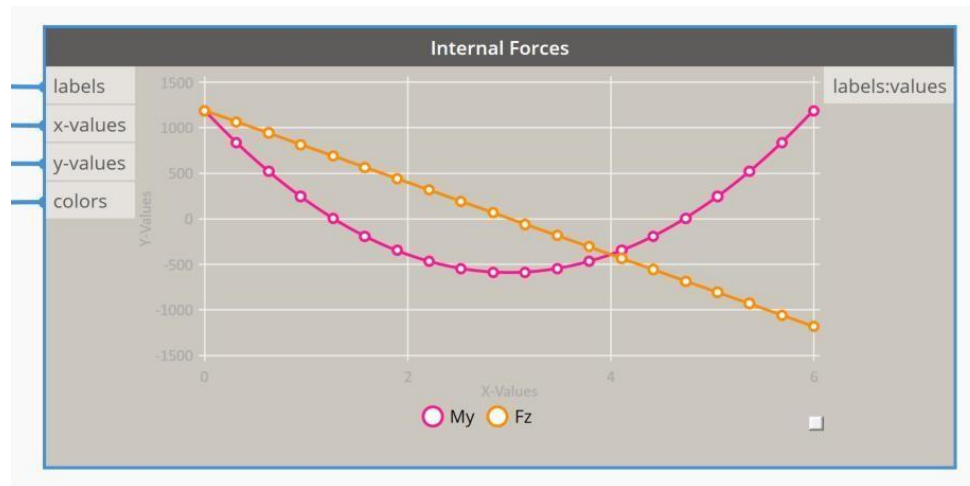
First, install the package. If you have never installed any package you may find this blog post useful.



Based on the Dynamo geometry, structural engineers can create an analytical model, and apply section shapes and boundary conditions such as supports and releases.

Engineers can also automatically apply structural loads which recalculate every time the structural geometry changes.

Structural engineers can review the results of this analysis in the Robot Structural Analysis environment or they can retrieve the results within the Dynamo environment.



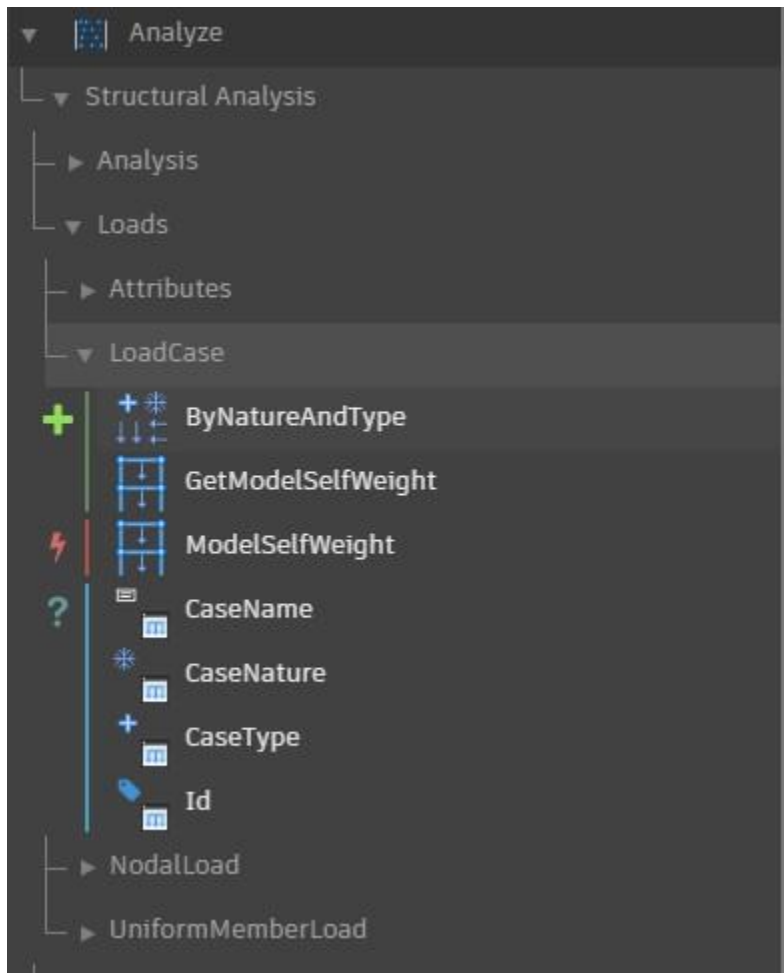
With the Structural Analysis package for Dynamo software, structural engineers can optimize their existing structural workflows or create new workflows that will improve their productivity.

Dynamo for Robot Structural Analysis helps users not only generate structural geometry but also:

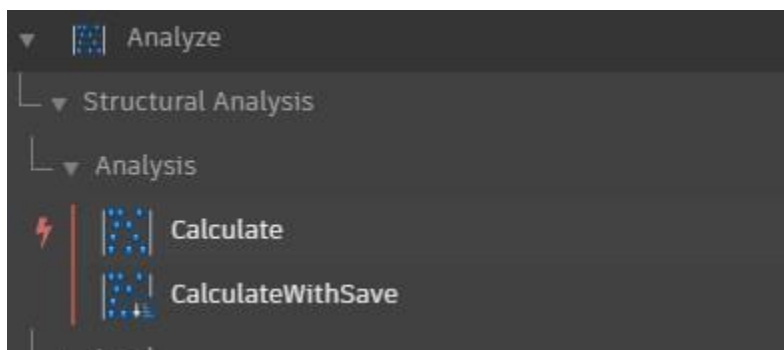
Assign structural properties such as: section shapes, materials, releases etc...



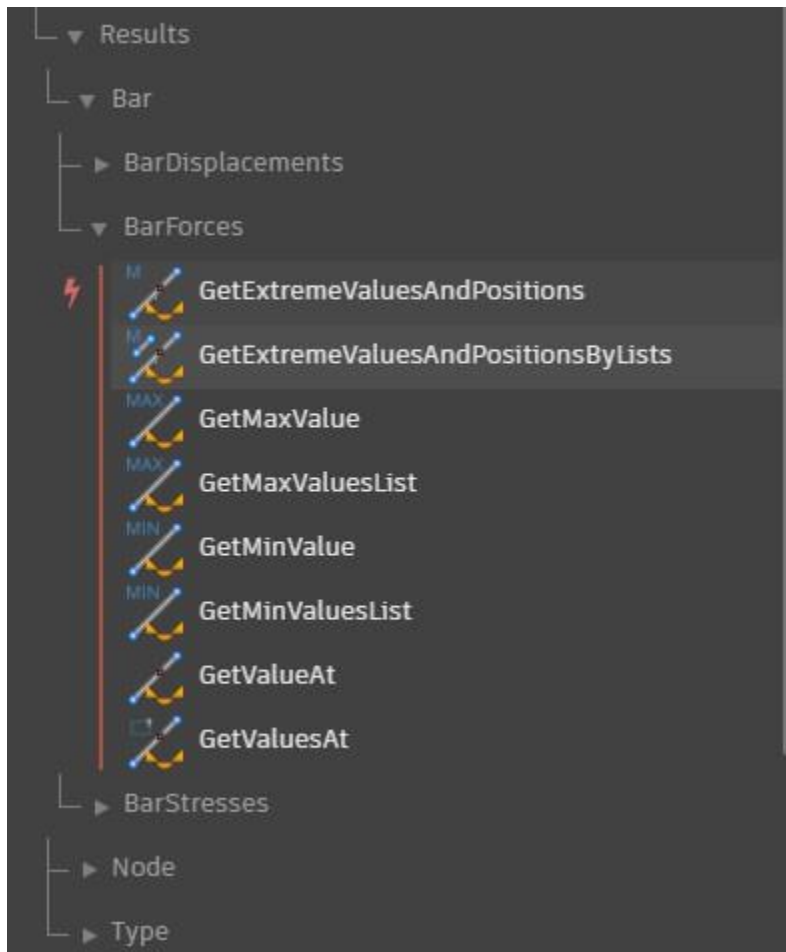
Create load cases and apply various types of loads:



Run calculations:



And finally, display and retrieve results of structural analysis.



I would like to show you the process in the following example of a parametric portal frame.

### Analytical Model

First, let's create the analytical model representation of Dynamo geometry in the Robot Structural Analysis Professional environment.

We need to connect the Dynamo geometry I covered in the previous post with the **AnalyticalBar.ByLines** nodes.

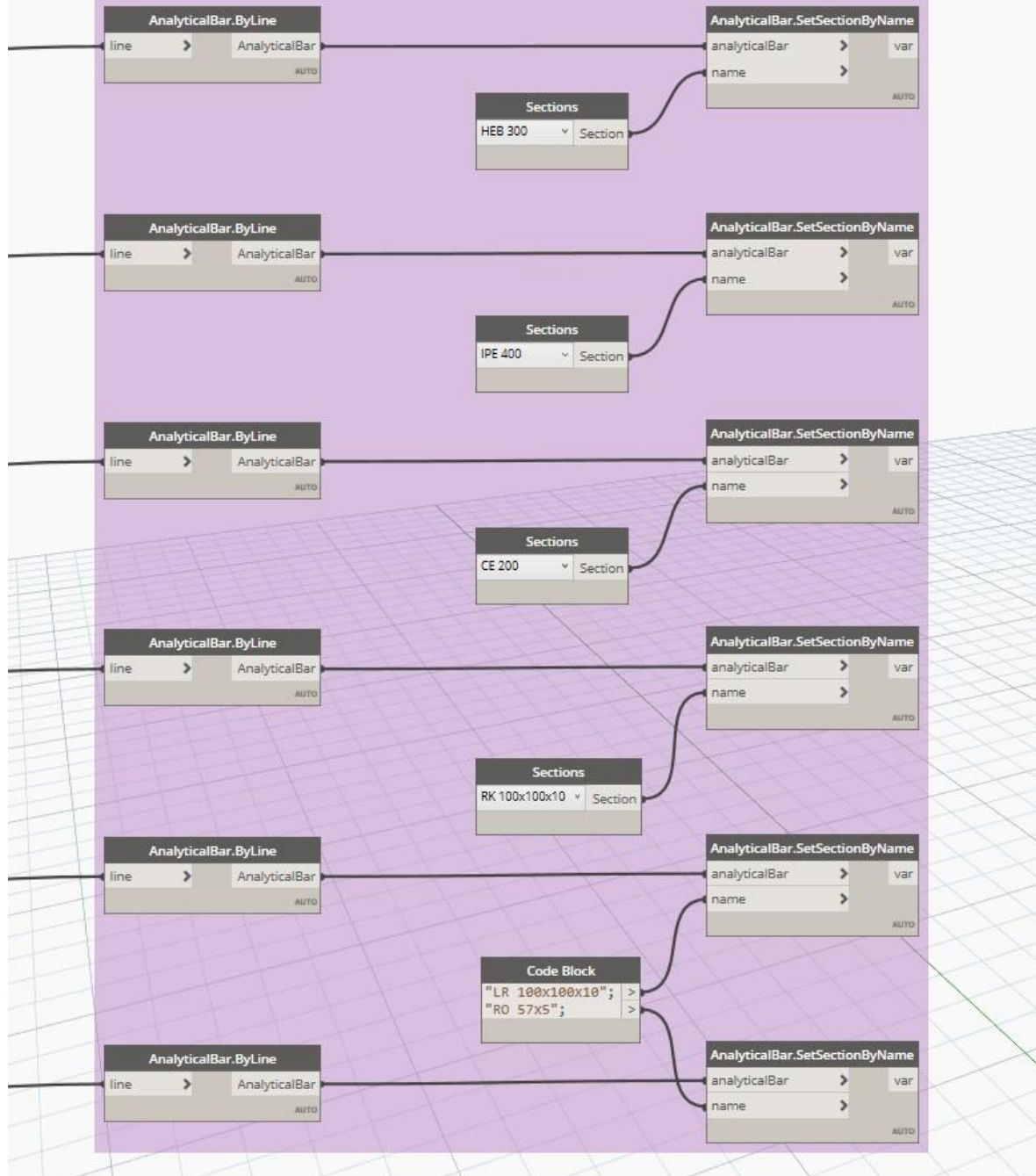
Before you assign cross-section properties to the analytical members, you need to make sure the sections are loaded in Robot. This can be done in the Robot project file you use, but this can also be performed in Dynamo with the ***Bars.LoadSections*** node for steel or timber sections.

Page 47



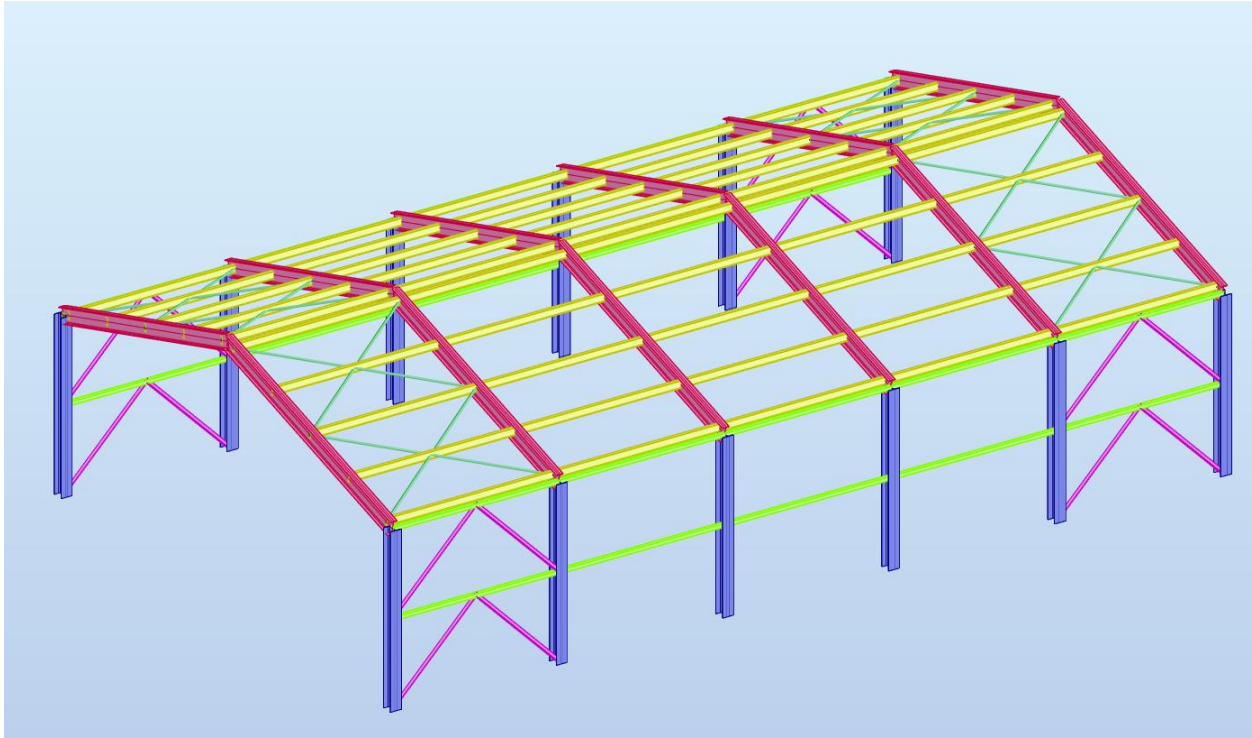


## Create Analytical Model and Assign Section Profiles



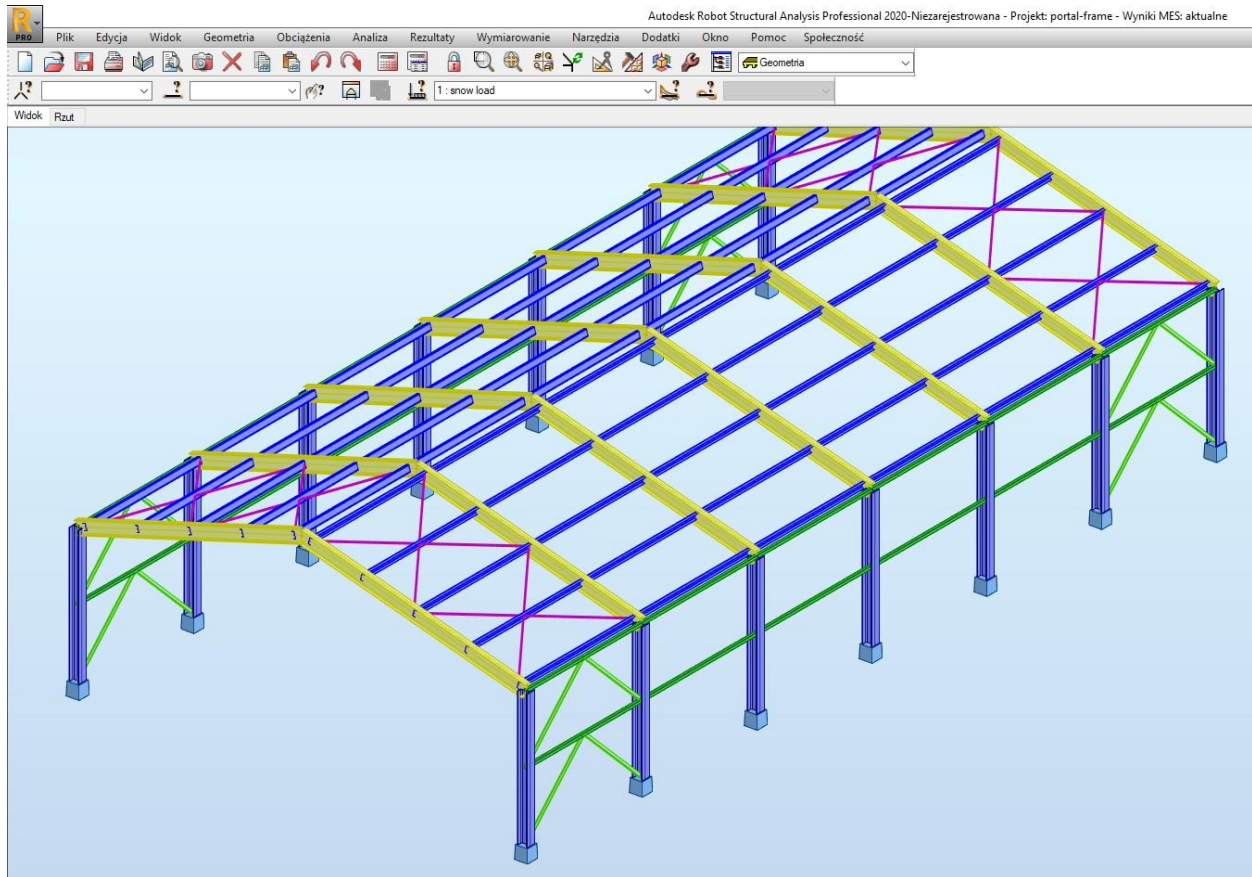


Let's test it and click the Run button to see what we've got in the Robot environment.



So far I demonstrated how to leverage the portal frame Dynamo geometry to create its analytical model representation in [Robot Structural Analysis Professional](#). I also introduced the Structural Analysis for Dynamo package that enables parametric modeling and structural analysis workflows in Dynamo and Robot Structural Analysis Professional.

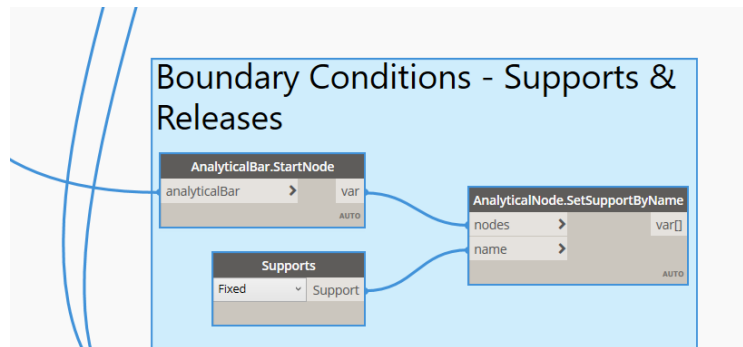
I ended the workflow at the point where the analytical model with sections assigned was generated in Robot environment. Now, I still need to set a few things up before I run analysis and retrieve results.



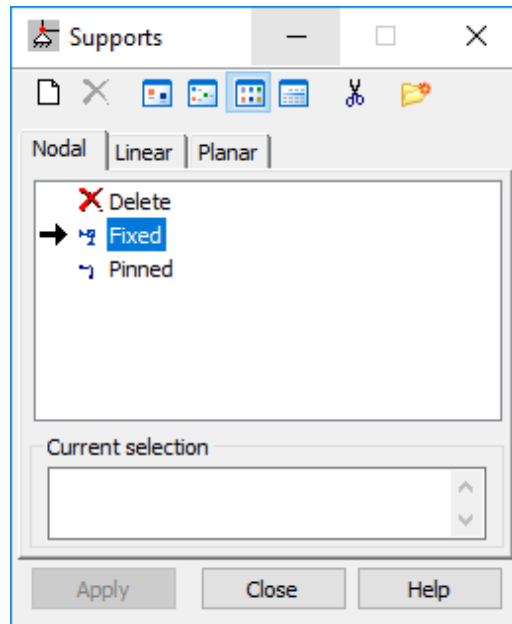
## Supports and bar end releases

The boundary conditions are the most important characteristics to focus on for the simulation of a structure. The boundary conditions are the places where the structure interacts with the environment either through the application of an external force or through some restraint that is imposing a displacement. The most common displacement boundary conditions in structural analysis are those that restrain the movement of the structure in one or more degrees of freedom at a point. These restraints are also called supports. They will define the behavior of a structure, and in Robot, it is necessary for them to be defined.

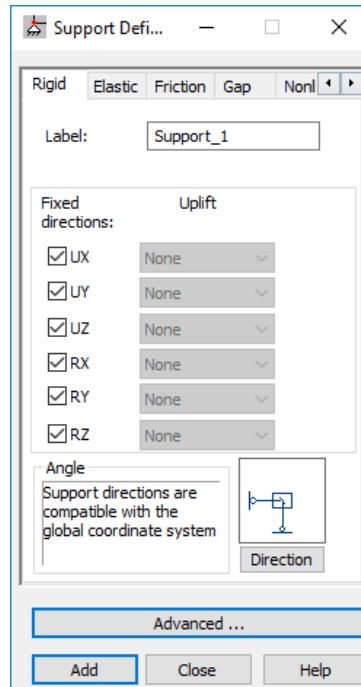
In the next step in my script, the supports are defined by the **AnalyticalNode.SetSupportByName** node. In this case the supports are set to analytical nodes. The nodes can be detected using the **AnalyticalBar.StartNode** or **AnalyticalBar.EndNode** nodes, depending on which side of the element needs support.



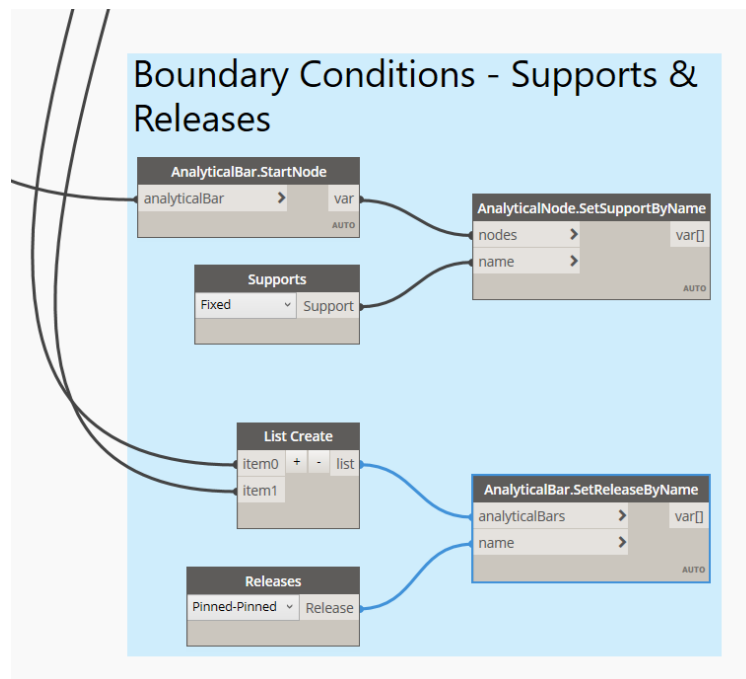
The support name is a string referring to the Support label available in Robot. Make sure the support label is available in Robot. This can be checked in **Supports** dialog (*Geometry -> Supports*) in Robot.



In the case support is not available, you can define one yourself using the **New** button in the dialog box. That is where you need to set the conditions of the support.



In the case no release conditions are defined, Robot assumes elements of the structure are perfectly fixed. The bar end releases are optional, but in the case of my portal frame bracings, they are indispensable. They can be set up in the same way as supports, but in this circumstance, you use the **AnalyticalBar.SetReleaseByName** node instead.

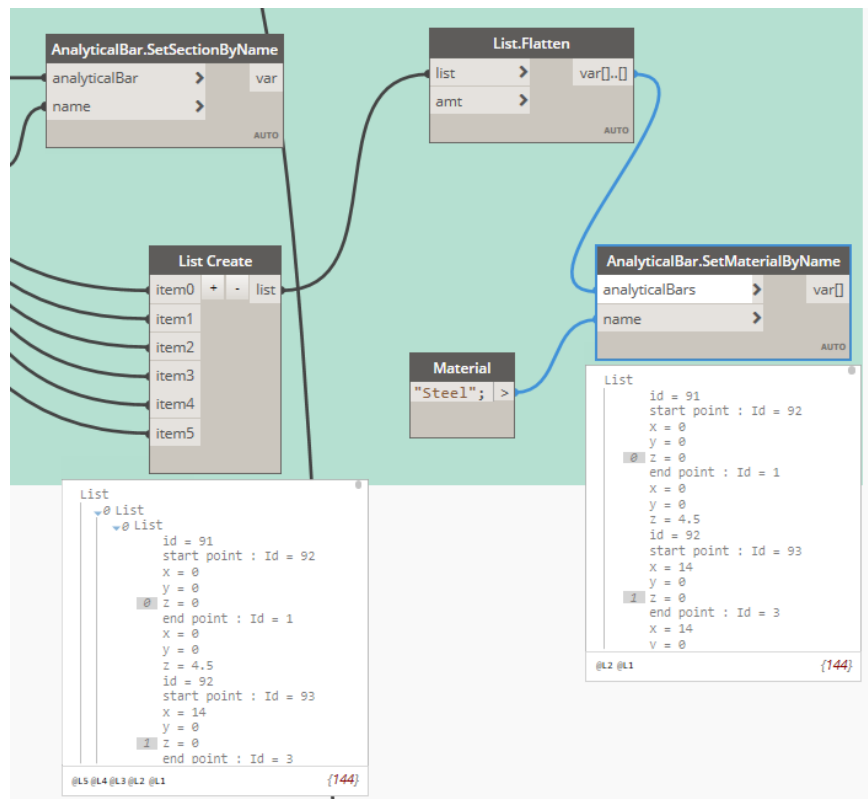


The input for this node is *AnalyticalBars*. The direction of an analytical bar is very important for the definition of releases. In the case of an asymmetric release (i.e. fixed at the start, pinned at the end), you need to make sure you are using the right order in Dynamo to create the geometry lines. The way they are created will define the local x-direction of the analytical bar (i.e. the choice of start and endpoints with the **LineByStartPointEndPoint** node).

Similarly in the supports case, the release name is a string referring to the Release label available in Robot so make sure the label is available in the Robot environment.

## Structural Materials

The package also allows you to set up the structural material. Make sure the material label is available in the Robot environment.



## Creation of load cases and loads

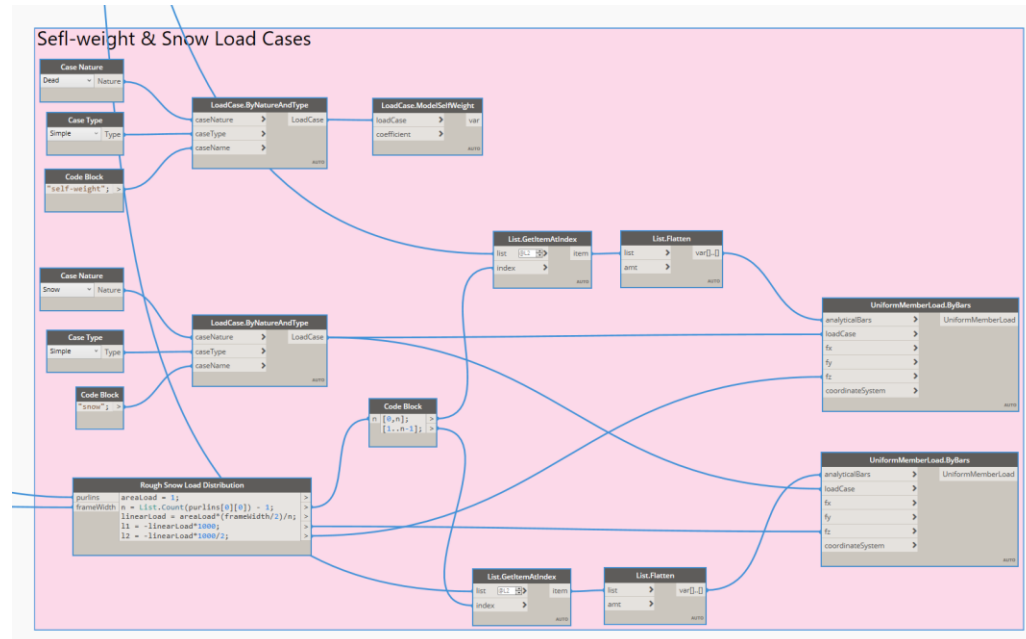
The next step is the definition of load cases in Structural Analysis Professional. These load cases contain the structural loads which will be applied to the structure later. Depending on the load duration and type, different natures of loads can be created (dead, live, wind, etc.) by structural engineers.

A load case can be created with the **LoadCase.ByNatureAndType** node. This node requires three inputs:

- *Case Nature*: dead load, live load, wind, snow etc...

- **Case Type:** At this moment there is only the “simple” option available.
- **Case Name:** This is a string representing the name of the load case in Robot Structural Analysis Professional.

Engineers can also automatically apply and recalculate structural loads every time the structural geometry changes.



## Analysis

The next step is the most important one, and also the step that needs a bit of your attention before you go to work with it. The **Analysis.Calculate** node is the step that starts the analysis in Robot. It's advised to only connect this node when your analysis model is set up completely (geometry, sections, loads and boundary conditions).

First create a list which contains the following elements:

- All analytical bars (and panels if they exist in the model) that are generated
- The analytical nodes with supports applied (take the output from the **AnalyticalNode.SetSupportByName** node)
- All defined load cases
- All load definitions (except for the **LoadCase.ModelSelfWeight** node, only superimposed loads).

The screenshot displays a software interface with three main components:

- List Create Window:** A table with columns 'item0', '+', '-', and 'list'. It contains rows for 'item0' through 'item6'. Blue arrows point from each item row to the 'list' column.
- Analysis.Calculate Window:** A table with columns 'elements' and 'analyticalBars', 'analyticalNodes', 'analyticalPanels', and 'loadCases'. The 'elements' column is selected.
- Dictionary Window:** A list of 'analyticalBars' with properties like id, start point, end point, x, y, z.

item0	+	-	list
item0			
item1			
item2			
item3			
item4			
item5			
item6			

elements	analyticalBars	analyticalNodes	analyticalPanels	loadCases

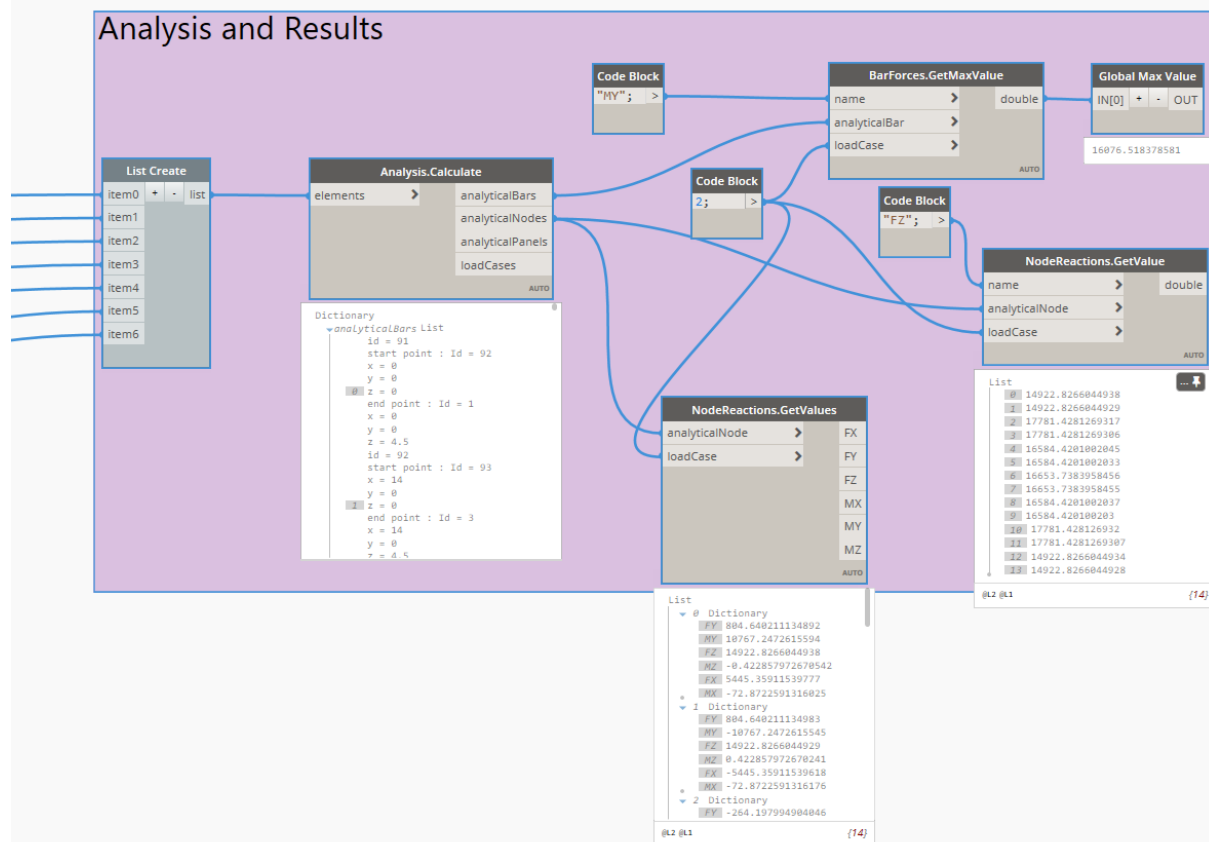
```

Dictionary
  analyticalBars List
    id = 91
    start point : Id = 92
    x = 0
    y = 0
    z = 0
    end point : Id = 1
    x = 0
    y = 0
    z = 4.5
    id = 92
    start point : Id = 93
    x = 14
    y = 0
    z = 0
    end point : Id = 3
    x = 14
    y = 0
    z = 4.5
  
```



## Results

The output of the **Analysis.Calculate** node contains 4 separated lists with elements classified by category (bars, nodes, panels and load cases). At the outputs of this node you can connect the Result nodes. Depending on which type of results you want to request, different inputs are needed. But all the result nodes have one thing in common, which is the *loadCase* input. This needs to be an integer representing the LoadCase.ID of a defined load case.



Dynamo for Robot Structural Analysis enables the analysis of more design options, more complex structures, and more resilient structures. Based on the Dynamo geometry, structural engineers can create an analytical model, apply section shapes and boundary conditions, such as supports and releases. Engineers can also create structural load cases and automatically apply loads, which can be recalculated every time the structural geometry is changed. Once the analysis is done, structural engineers can review the results of the analysis in Robot Structural Analysis or they can retrieve the results within Dynamo.

Using Dynamo, structural engineers produce optimized designs faster, with an integrated design and analysis workflow that allows them to iterate through design options faster and more fluidly, enabling them to win more work and stay competitive.





## Summary

The visual programming interface of Dynamo gives structural engineers the tools to build optimized structures with minimal energy, and subsequently make their own design tools. Based on the Revit Platform, we can use our creativity to develop optimized structural systems using computational logic in an advanced building information modeling environment.

The combination of the Revit information database and scripting with Dynamo opens a world of possibilities to working fast with complex and optimized structures. The exercises we have gone through provide just a few examples. Now, make your own design solutions and share them with the world!

