

323888

An Intro to Getting Dynamo to talk with RAM Structure Using the API

Marcello Sgambelluri S.E.
Director Of Advanced Technology

John A. Martin Structural Engineers

Learning Objectives

- Learn how to understand the RAM API
- Learn basic C# to create custom Dynamo nodes
- Learn how to create custom Dynamo nodes that tap the RAM SS database
- Learn how to move data from RAM to other databases

Description

RAM structural analysis 3D models are a part of most structural engineering offices' workflows. The models are typically created by structural engineers and used by structural engineers to perform structural engineering analysis on projects. There is a wealth of rich building information in the 3D analysis models that could be used for design, quality control, and many other tasks. The reality is that most data extracted from these structural analysis models is extracted manually. This process is tedious, time consuming, and costly. What if you could show structural engineers how to extract all that rich data using automation techniques such as Dynamo and the RAM API? Would you be their hero? This class will teach you how! You do NOT need to be a structural engineer, nor do you need to know RAM Structural Systems to take this class. This class will NOT teach you how to run any structural analysis or link Revit software to RAM.

About the Speaker



Marcello currently serves as the Building Information Modelling (BIM) director at John A. Martin & Associates Structural Engineers in Los Angeles, California. Marcello has worked on many BIM projects over the last 20 years including the Walt Disney Concert Hall in Los Angeles; the Ray and Maria Stata Centre at the Massachusetts Institute of Technology; and the Tom Bradley International Terminal Expansion at Los Angeles International Airport. Sgambelluri is internationally recognized as one of the top BIM leaders and contributors to the education and implementation of BIM technology in the building industry. He continually speaks at Autodesk University and the Revit Technology Conference, and he has received a record total of 15 1st place speaker awards between both conferences. Marcello Sgambelluri received his bachelor's and master's degrees in civil engineering, and he is a licensed civil and structural engineer.

Marcello also has been building media that includes the following:

Simply Complex Blog Site -

<http://therevitcomplex.blogspot.com/>

Simply Complex YouTube

<https://www.youtube.com/channel/UC7IkO1Bc4PhFKAHEArmQ0jw/videos>

Simply Complex Podcast -

<http://simplycomplex.sharedcoordinates.com/>

AEC Complex Comic -

<https://www.aeccomplexcomic.com/>

INTRODUCTION

This Handout uses the following versions

RAM 15*

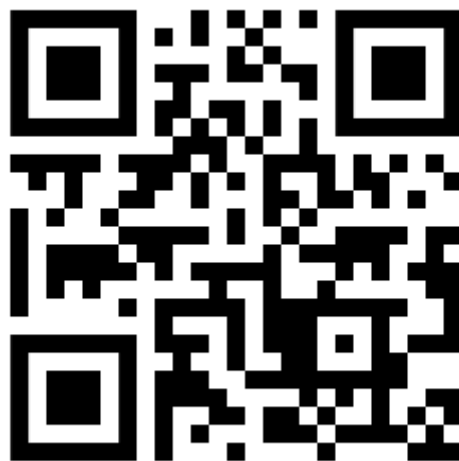
ETABS 17

Dynamo 1.3.3. OR Dynamo 2.X

Revit 2019 OR Revit 2020

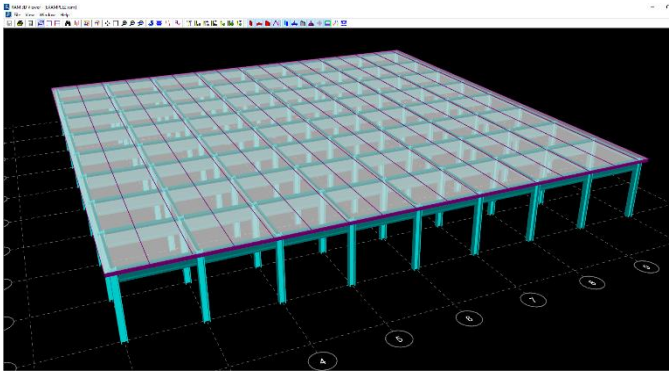
Dynamo Simplex Package 2019.X

Download entire dataset and extended handout here



<https://a360.co/2MQuFPO>

What is RAM?



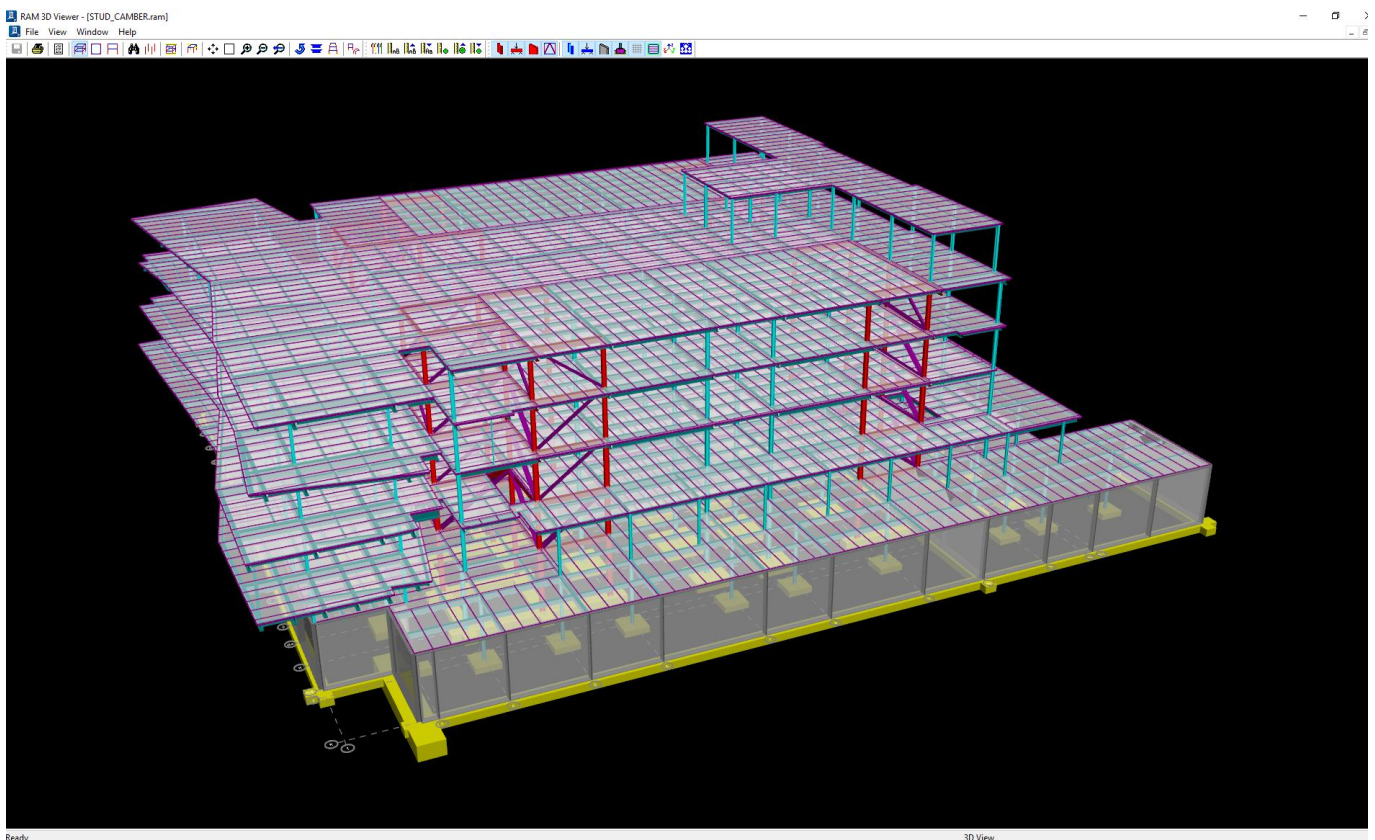
RAM Structural System is an integrated three-dimensional static and dynamic structural analysis and design program owned by Bentley. Due to its simple rules and great clear calculations that building departments enjoy, RAM is used by many structural engineers. It is made up of many modules including modeler, beam design, column design etc.

Also, the graphics and interface that are used

have not really change for the last 20 years so RAM could look a bit outdated.

The modelling interface is not very user friendly either. With all of RAM's "archaic" nature it remains a fan favourite amount structural engineers.

Structural engineers using RAM always need help "automating" tasks when using RAM and one way to do that is to use the RAM API. Even better, instead of making an add-in to RAM using the API that could be customized by engineers give them the power to make their own programs by using Dynamo. Why Dynamo? Dynamo is easy to learn and would allow engineers to interact with the RAM API without having to know how to code in C# or other text programming language. So..... lets look at Zero Touch and the RAM API.



Understanding ZERO TOUCH

Zero Touch is just a fancy word for saying “create a custom Dynamo Node using C#”

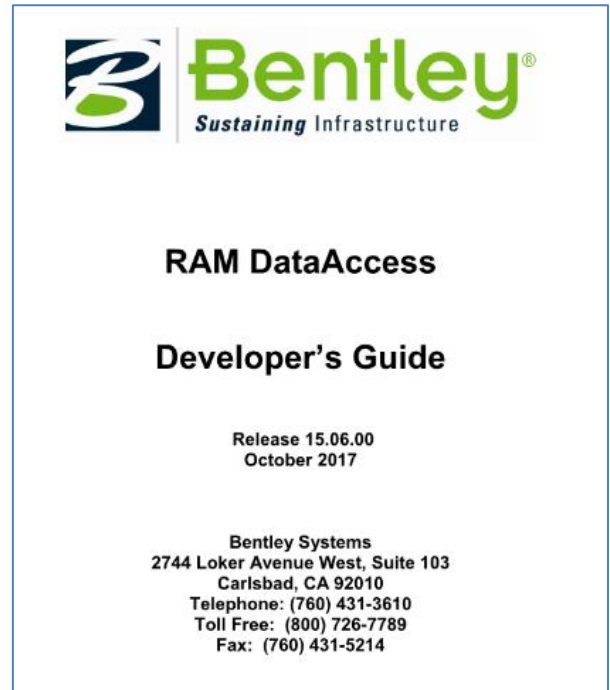
More information could be found here

<https://github.com/DynamoDS/Dynamo/wiki/Zero-Touch-Plugin-Development>

The next few examples will get you familiar with Zero Touch.

Understanding the RAM API

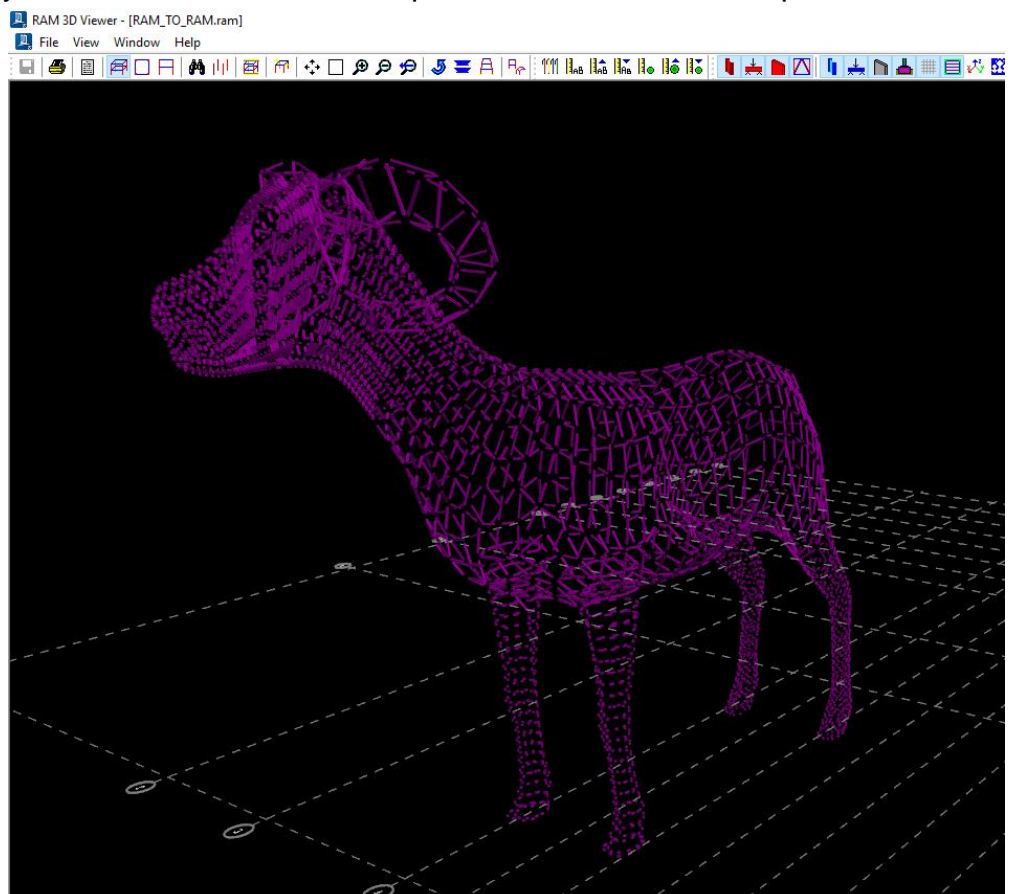
Understanding the RAM API is important to get Dynamo to talk with RAM. In this class we will create Dynamo nodes using Zero Touch and thus we need to use C#. Note that you could use Python to create custom Dynamo nodes that talk with the RAM API.



The RAM API is not easy to understand since it's only documentation was created with VB and C++ examples. Also the documentation is not complete and is inconsistent in many locations. The purpose of this class is to get you STARTED with the RAM API and connecting it to Dynamo. There is not way this class could cover ever possible method that is exposed in the RAM API. So we will start with some simple concepts.

First you will need to get the RAM API documentation. It is provided in your RAM install directory. Its called “RAM DataAccess Developers Guide.pdf” and it provided in the dataset.

It is possible to create some complex shapes using the RAM API such as a RAM in RAMSS.





ZERO TOUCH



ZERO TOUCH BASICS STARTING A VISUAL STUDIO PROJECT

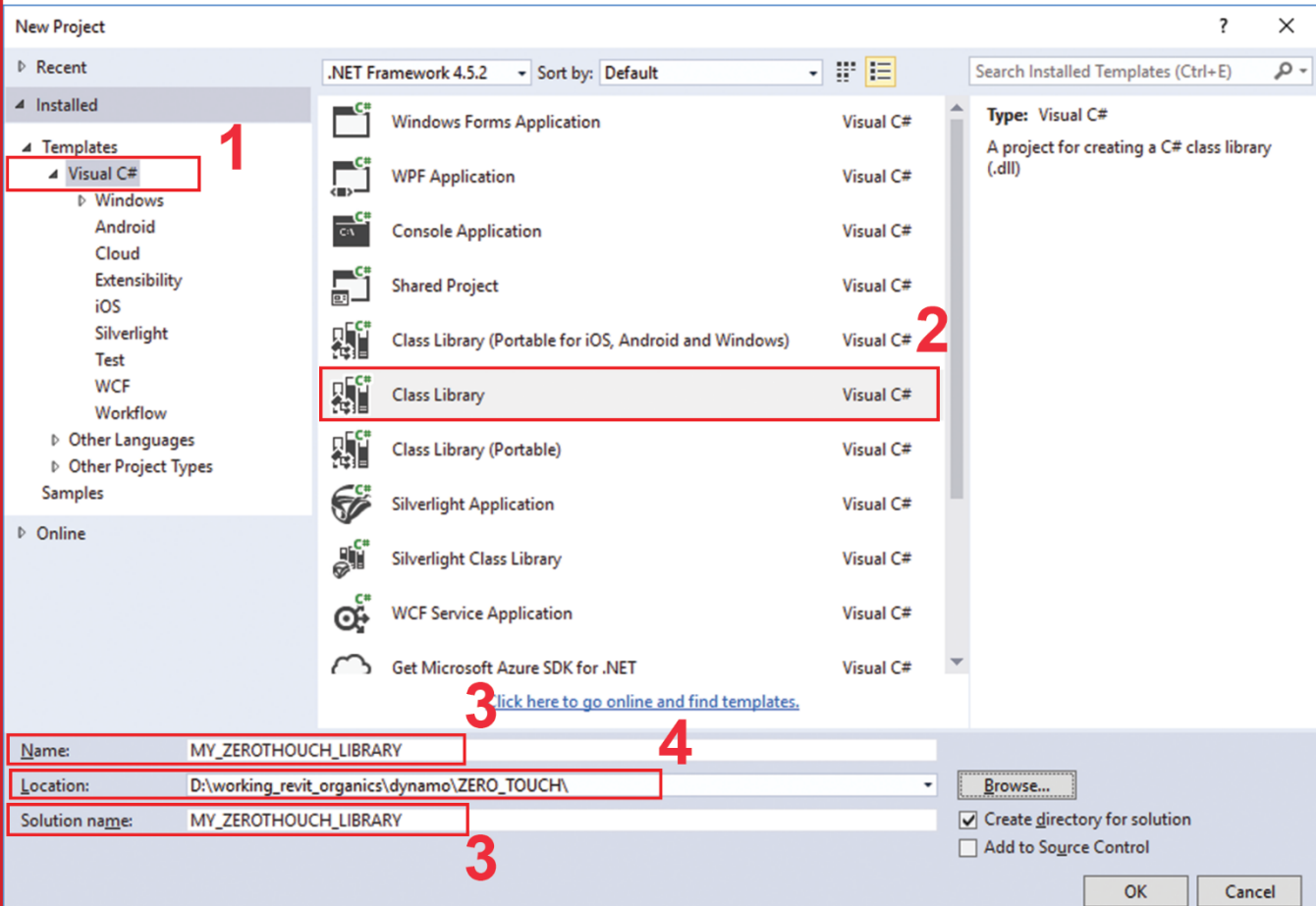
STEP 1:
OPEN VISUAL STUDIO AND START A NEW PROJECT
SELECT C#

STEP 2:
SELECT THE CLASS LIBRARY.
THE CLASS LIBRARY IS WHAT WILL "HOLD" THE DYNAMO NODES

STEP 3:
TYPE THE NAME OF THE PROJECT. THE PROJECT NAME WILL BE THE NAME OF THE .DLL. THIS IS ALSO KNOWN IN ZERO TOUCH AS YOUR "LIBRARY". THIS "DLL" WILL BE THE FILE THAT CONTAINS ALL THE NODES.

STEP 4:
SAVE THE PROJECT "DLL" TO A FOLDER. THIS FOLDER WILL BE THE LOCATION WHERE ALL THE SOLUTION FILES WILL BE STORED SO CHOOSE THIS CAREFULLY.

STEPS



VISUAL STUDIO START SCREEN

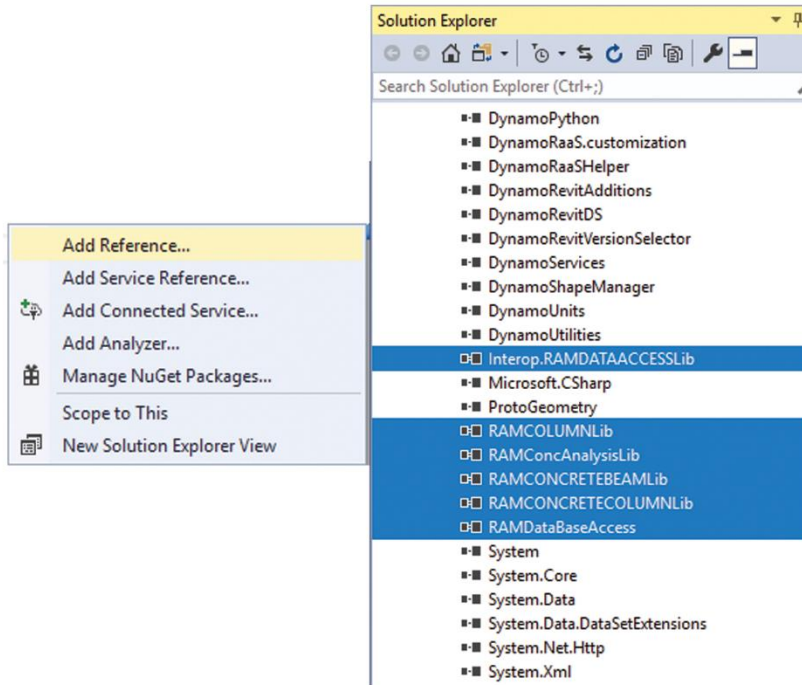
NOTES:
UNDERSTANDING THE STRUCTURE OF THE ZERO TOUCH PROJECT IS IMPORTANT AS IT WILL DETERMINE THE ORGANIZATION OF THE DYNAMO NODE LIBRARY. A LIBRARY IS A COLLECTION OF DYNAMO NODES.

NOTES

ZERO TOUCH BASICS LOADING IN REFERENCES

STEP 1:

NAVIGATE TO THE FOLDER THAT THE PROJECT IS STORED AND CLICK ON THE SOLUTION FILE “.SLN



STEP 2:

RIGHT CLICK OVER THE REFERENCES IN THE SOLUTION EXPLORER IN VISUAL STUDIO AND ADD THE PROPER REFERENCES THAT ARE “.DLL” FORMAT. REFERENCES ARE ANY PRESET LIBRARY THAT THE CURRENT PROGRAM NEEDS TO USE TO RUN PROPERLY. KNOWING WHICH REFERENCES TO ADD IS NOT ALWAYS EASY. IN THIS CASE GO TO THE RAM INSTALL DIRECTORY TO GET THESE DLLS OR YOU COULD GET THEM PER NOTES SHOWN BELOW

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using RAMDATAACCESSLib;  
using Autodesk.DesignScript.Runtime;  
using Autodesk.DesignScript.Geometry;
```

STEP 3:

ADD USING STATEMENTS BY SIMPLY TYPING THEM IN AT THE TOP OF THE SCREEN. TYPE THE ONES SHOWN.

NOTE:

USING STATEMENTS ARE NOT REQUIRED TO CREATE A DYNAMO NODE HOWEVER THEY ARE VERY HELPFUL BECAUSE USING STATEMENTS “SHORTEN” WHAT COMMANDS THAT NEEDED TO BE TYPE IN THE CODE. ALSO YOU MAY NEED MORE REFERENCES DEPENDING ON WHAT ELSE YOU ARE WORKING WITH.

VISUAL STUDIO START SCREEN STEPS

NOTES:

ADDING THE CORRECT REFERENCES AND FINDING THEM IS NOT ALWAYS VERY EASY. IT IS SOMETIMES EASIER TO GET THE REFERENCES AND USING STATEMENTS FROM A PREVIOUS PROJECT. OPEN THE SAMPLE FILE TO GET THESE REFERENCES IF NEEDED

NOTES

ANATOMY OF A SIMPLE ZERO TOUCH NODE NO INPUT

1:
THE NAME OF THE ZERO TOUCH LIBRARY OR HIGHESET LEVEL FOLDER NAME (SHOWN IN THE DYNAMO BROWSER) IS THE NAME OF THE DLL

2:
THE NAME OF THE ZERO TOUCH LIBRARY SUB-FOLDER IS THE NAME OF THE NAMESPACE IN THE ZERO TOUCH CODE

3:
THE NAME OF THE ZERO TOUCH LIBRARY SUB-SUB-FOLDER IS THE NAME OF THE CLASS IN THE ZERO TOUCH CODE

4:
THE NAME OF THE ZERO TOUCH NODE IS THE NAME OF THE METHOD IN THE ZERO TOUCH CODE

5:
THE NAME OF THE INPUT PORT IS THE NAME OF THE PARAMETER IN THE METHOD (BLANK IN THIS EXAMPLE)

6:
THE NAME OF THE OUTPUT PORT IS THE NAME OF THE METHOD TYPE RETURNED (INT IN THIS EXAMPLE) ALSO SEE NOTES BELOW.

DESCRIPTIONS

DYNAMO BROWSER

MY_ZEROTOUCH_LIBRARY > MY_ZEROTOUCH_LIBRARY > bin > Debug >

Name	Date modified	Type	Size
MY_ZEROTOUCH_LIBRARY.dll	10/8/2018 9:51 PM	Application extens...	
FriendsExample.dll	8/10/2018 8:28 PM	Application extens...	
DynamoRevitDS.dll	2/28/2017 10:13 PM	Application extens...	
DynamoRevitVersionSelector.dll	2/28/2017 10:13 PM	Application extens...	
RevitNodes.dll	2/28/2017 10:13 PM	Application extens...	
RevitServices.dll	2/28/2017 10:13 PM	Application extens...	
DynamoRevitAdditions.dll	2/28/2017 10:12 PM	Application extens...	

LIBRARY (DLL) LOC.

1: MY_ZEROTOUCH_LIBRARY
2: MY_LIBRARY_FOLDER
3: GETSIMPLEINFO
4: marcelloAgeNodeNoInput

```

namespace MY_LIBRARY_FOLDER
{
    public class GETSIMPLEINFO
    {
        private GETSIMPLEINFO()
        {
        }

        public static int marcelloAgeNodeNoInput()
        {
            int marcelloAge = Friends.Marcello.Age;
            return marcelloAge;
        }
    }
}
    
```

GETSIMPLEINFO.marcelloAgeNodeNoInput
int

VISUAL STUDIO CODE

NOTES:

UNDERSTANDING THE STRUCTURE OF THE ZERO TOUCH PROJECT IS IMPORTANT AS IT WILL DETERMINE THE ORGANIZATION OF THE DYNAMO NODE LIBRARY. ALSO IF MULTIPLE OUTPUT IS REQUIRED THEN THE METHOD WILL NEED TO OUTPUT A "DICTIONARY" OF MULTIPLE VALUES.

NOTES

ANATOMY OF A SIMPLE ZERO TOUCH NODE W/ INPUT

- 1: THE NAME OF THE ZERO TOUCH LIBRARY OR HIGHESET LEVEL FOLDER NAME (SHOWN IN THE DYNAMO BROWSER) IS THE NAME OF THE DLL
- 2: THE NAME OF THE ZERO TOUCH LIBRARY SUB-FOLDER IS THE NAME OF THE NAMESPACE IN THE ZERO TOUCH CODE
- 3: THE NAME OF THE ZERO TOUCH LIBRARY SUB-SUB-FOLDER IS THE NAME OF THE CLASS IN THE ZERO TOUCH CODE
- 4: THE NAME OF THE ZERO TOUCH NODE IS THE NAME OF THE METHOD IN THE ZERO TOUCH CODE
- 5: THE NAME OF THE INPUT PORT IS THE NAME OF THE PARAMETER. ALSO IF PARAMETER IN METHOD IS "=" TO A VALUE. THAT VALUE IS THE DEFAULT VALUE IN THE INPUT PORT.
- 6: THE NAME OF THE OUTPUT PORT IS THE NAME OF THE METHOD TYPE RETURNED (INT IN THIS EXAMPLE) ALSO SEE NOTES BELOW.

DESCRIPTIONS

LIBRARY (DLL) LOC.

VISUAL STUDIO CODE

NOTES

DYNAMO BROWSER

MY_ZEROTOUCH_LIBRARY > MY_ZEROTOUCH_LIBRARY > bin > Debug >

Name	Date modified	Type	Size
MY_ZEROTOUCH_LIBRARY.dll	10/8/2018 9:51 PM	Application extens...	
FriendsExample.dll	8/10/2018 8:28 PM	Application extens...	
DynamoRevitDS.dll	2/28/2017 10:13 PM	Application extens...	
DynamoRevitVersionSelector.dll	2/28/2017 10:13 PM	Application extens...	
RevitNodes.dll	2/28/2017 10:13 PM	Application extens...	
RevitServices.dll	2/28/2017 10:13 PM	Application extens...	
DynamoRevitAdditions.dll	2/28/2017 10:12 PM	Application extens...	

1: MY_ZEROTOUCH_LIBRARY
 2: MY_LIBRARY_FOLDER
 3: GETSIMPLEINFO
 4: marcelloAgeNodeNoInput

```

namespace MY_LIBRARY_FOLDER
{
    public class GETSIMPLEINFO
    {
        private GETSIMPLEINFO()
        {
        }

        public static int marcelloAgeNodeNoInput(int fudgefactor = 0)
        {
            int marcelloAge = Friends.Marcello.Age - fudgefactor;
            return marcelloAge;
        }
    }
}
    
```

5: marcelloAgeNodeNoInput
 6: int

GETSIMPLEINFO.marcelloAgeNodeNoInput
 fudgefactor (int, Default value: 0) > int

NOTES:
 UNDERSTANDING THE STRUCTURE OF THE ZERO TOUCH PROJECT IS IMPORTANT AS IT WILL DETERMINE THE ORGANIZATION OF THE DYNAMO NODE LIBRARY. ALSO IF MULTIPLE OUTPUT IS REQUIRED THEN THE METHOD WILL NEED TO OUTPUT A "DICTIONARY" OF MULTIPLE VALUES.

GET MARCELLO'S AGE ZERO TOUCH NODE W/ AND W/O INPUT

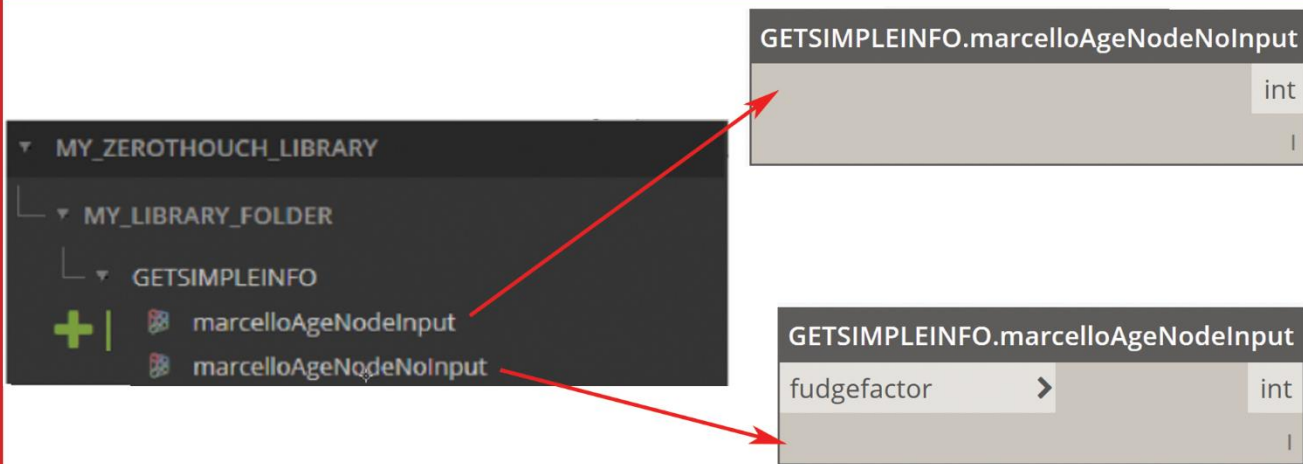
```
using FriendsExample;

namespace MY_LIBRARY_FOLDER
{
    public class GETSIMPLEINFO
    {
        private GETSIMPLEINFO()
        {
        }

        public static int marcelloAgeNodeNoInput()
        {
            int marcelloAge = Friends.Marcello.Age;
            return marcelloAge;
        }

        public static int marcelloAgeNodeInput(int fudgefactor = 0)
        {
            int marcelloAge = Friends.Marcello.Age - fudgefactor;
            return marcelloAge;
        }
    }
}
```

ZEROTOUCH CODE



DYNAMOLIBRARY + ZT NODES

NOTES:

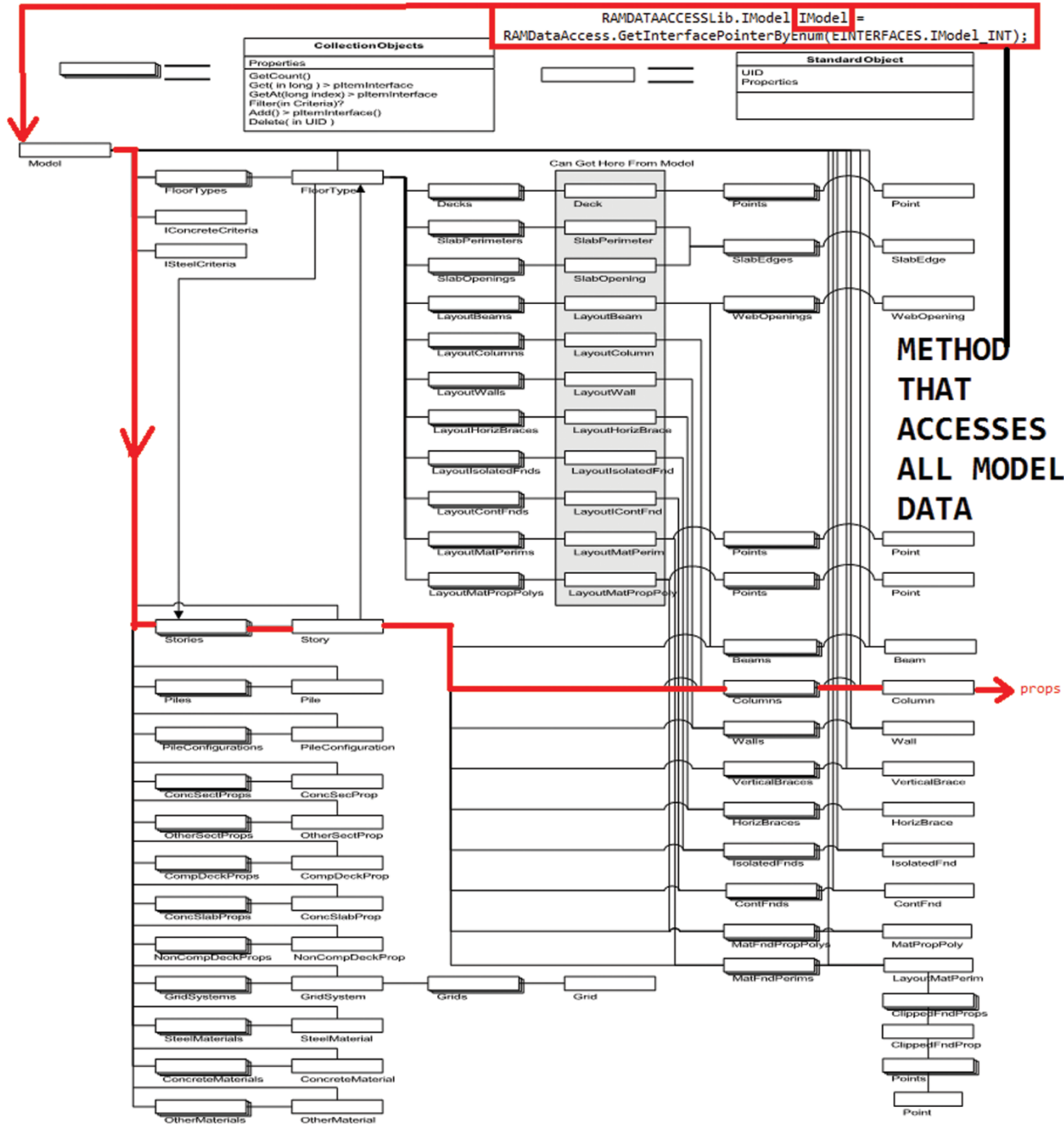
OPEN VISUAL STUDIO FOLDER "GET_MARCELLO_AGE_ZT_START" CLICK ON THE SLN FILE.
TYPE CODE AS SHOWN FOR BOTH WITH AND WITHOUT INPUT PORT CASE. BUILD THE SOLUTION.
OPEN DYNAMO VIA REVIT AND LOAD THE DLL FROM BIN DIR. OPEN "FINAL" FOLDER IF NEEDED.

STEPS



RAM SS API

USING RAM API ORGANIZATION CHART



RAM API PHYSICAL MODEL ORGANIZATION

NOTES: THIS CHART MAKES UP MOST OF THE ORGANIZATION OF HOW THE RAM API ORGANIZES THE PHYSICAL DATA OF THE RAM MODEL. THIS CHART COULD BE FOUND IN THE RAM API DOCS FOR EXAMPLE IF YOU WANTED COLUMN PROPERTIES YOU WOULD ACCESS THE MODEL OBJECT THEN STORIES OBJECT THEN STORY OBJECT THEN COLUMNS OBJECT THEN COLUMN.

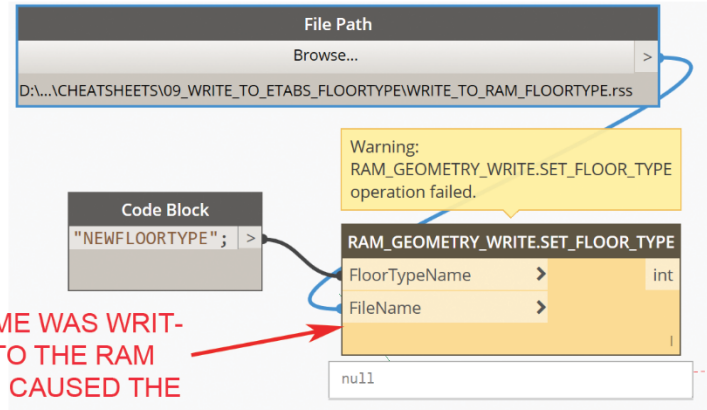
STEPS & NOTES

STEPS TO FIX "LOCKED" RAM FILE WHEN ERROR OCCURS IN IN DYNAMO

STEP 1

RECOGNIZE THE ERROR

ANY ERROR THAT OCCURS WHEN USING DYNAMO AND RAM WILL "LOCK" THE RAM MODEL FROM BEING USED WITH DYNAMO

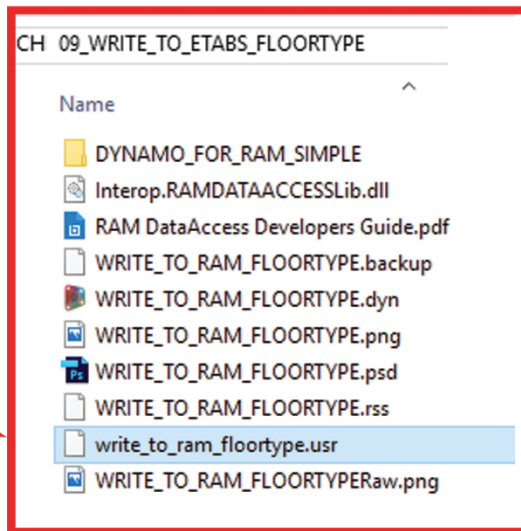


THE SAME NAME WAS WRITTEN TWICE INTO THE RAM MODEL WHICH CAUSED THE ERROR

STEP 2

REMOVE THE .USR FILE

ANY ERROR THAT OCCURS WILL CREATE A (SAME NAME AS RAM FILE).USR AND IT IS LOCATED NEXT TO THE ORIGINAL RAM FILE IN THIS CASE LOOK FOR THE FILE "WRITE_TO_RAM_FLOORTYPE.URS AND SIMPLY DELETE IT!



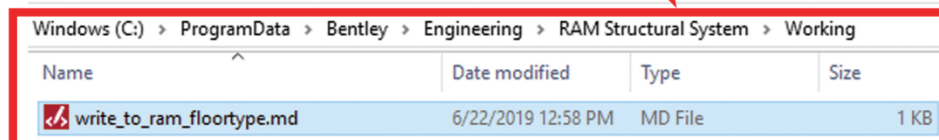
STEP 3

CLEAR THE WORKING DIRECTORY

ANY ERROR THAT OCCURS WILL CREATE A (SAME NAME AS RAM FILE).MD AND IT IS TYPICALLY LOCATED IN THIS WORKING DIRECTORY

C:\PROGRAMDATA\BENTLEY\ENGINEERING\RAM STRUCTURAL SYSTEM\WORKING
IF NOT NAVIGATE TO THE WORKING DIRECTORY AND DELETE EVERYTHING IN THAT FOLDER

FILE WILL THEN BE "UNLOCKED FOR USE WITH DYNAMO AGAIN!"



NOTES: ANY ERROR YOU GET WHEN USING DYNAMO ON THE RAM API WILL RESULT IN NOT ALLOWING THE SAME RAM FILE TO BE OPENED UNLESS THE .URS AND WORKING DIR IS CLEARED. IT IS NOT A DESIREABLE FIX HOWEVER BENTLEY STATED THIS IS THE ONLY WAY TO FIX THIS ERROR AND THIS ERROR FIXING PROCESS COULD BE AUTOMATED WITH ANOTHER DYNAMO NODE

GET NUMBER OF STORIES IN RAM USING DYNAMO USING RAM API AND ZERO TOUCH C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using RAMDATAACCESSLib;
using Autodesk.DesignScript.Runtime;
using Autodesk.DesignScript.Geometry;
```

```
namespace DYNAMO_FOR_RAM_SIMPLE
```

```
{
    public class RAM_GEOMETRY
```

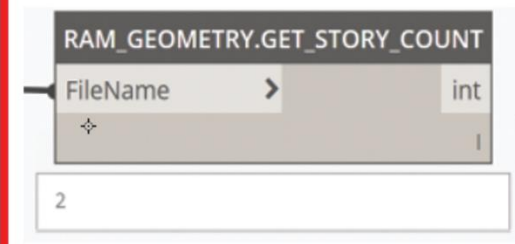
```
{
    private RAM_GEOMETRY()
```

```
{
    public static int GET_STORY_COUNT(string FileName)
```

```
{
    RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBIO1 IDBI = (RAMDATAACCESSLib.IDBIO1)
    RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBIO1 INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
    RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
```

```
IDBI.LoadDataBase2(FileName, "1");
    IStories My_stories = IModel.GetStories();
    int My_story_count = My_stories.GetCount();
    IDBI.CloseDatabase();
    return My_story_count;
```

```
}
}
```



DYNAMO NODE

IDBIO1
Philosophy: This interface is for performing functions on the database as a whole. This includes Loading and Saving. It also includes changing the database name. It does not include inquiries or specific data manipulation.

IModel
The IModel interface is the highest level in the hierarchy of interfaces. It represents the RAM Structural System model. Entities such as section definitions and floor types that are global to the model can be accessed through this interface. Other entities, such as decks or beams, which belong to a specific floor type, can be accessed through this interface using the entity's unique ID.

GetStories ([out, retval] IStories** pplStories)
Gets the collection of all stories in the model.
Parameters pplStories Pointer to an IStories collection interface that represents all stories in the model

GetCount ([out, retval] long* plCount)
Gets the number of stories in the collection.
Parameters plCount Number of stories in the collector

- VARIABLES THAT ACCESSES THE "IMODEL" OBJECT-THAT HAS ALL THE DATA NEEDED
- OPENS DATADASE VIA FILE
- GETS "ISTORY" OBJECT
- GETS NUMBER OF STORIES FROM "ISTORY"
- CLOSES DATADASE
- SENDS NUMBER OF STORIES TO DYNAMO OUTPUT PORT

ZERO TOUCH C# RAM API CODE

RAM API MANUAL

- STEP 1: OPEN VISUAL STUDIO FOLDER "GET_NUM_STORIES" OPEN SLN FILE - BUILD SLN
- STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE
- STEP 3: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
- NOTE: RAM DOES NOT NEED TO BE OPEN, AND NOTE RELATIONSHIP WITH API DOC AND CODE

STEPS & NOTES

GET ALL STORY NAMES USING RAM API AND ZERO TOUCH C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using RAMDATAACCESSLib;
using Autodesk.DesignScript.Runtime;
using Autodesk.DesignScript.Geometry;
namespace DYNAMO_FOR_RAM_SIMPLE
```

```
{
    public class RAM_GEOMETRY
    {
        private RAM_GEOMETRY()
        {
        }
        public static List<string> GET_STORY_NAMES(string FileName)
        {
            RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
            RAMDATAACCESSLib.IDBIO1 IDBI = (RAMDATAACCESSLib.IDBIO1)
            RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBIO1_INT);
            RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
            RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
            //OPEN
            IDBI.LoadDataBase2(FileName, "1");
            IStories My_stories = IModel.GetStories();
            int Story_Count = My_stories.GetCount();
            List<string> ListLine = new List<string>();
            for (int i = 0; i < Story_Count; i = i + 1)
            {
                string My_Story_Names = My_stories.GetAt(i).strLabel;
                ListLine.Add(My_Story_Names);
            }
            //CLOSE
            IDBI.CloseDatabase();
            return ListLine;
        }
    }
}
```

CREATES METHOD/DYNAMO NODE LIST STRING OUT AND FILE NAME IN

GETS IMODEL OBJECT

OPEN RAM MODEL

GETS IStories OBJECT

GETS NUMBER OF STORIES FROM IStories

SETS UP LIST OF NAMES FOR OUTPUT

SETS COUNTER FOR LOOPS

GETS NAMES OF EACH STORY

CLOSES RAM MODEL

ZERO TOUCH C# RAM API CODE

Level	Story Label	Floor Type	Height	Splice
2	LEVEL2	FLOOR01	10.0000	Yes
1	LEVEL1	FLOOR01	10.0000	Yes

RAM MODEL

DYNAMO NODES

- STEP 1: OPEN VISUAL STUDIO FOLDER "GET_STORY_NAMES" OPEN SLN FILE - BUILD SLN
- STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE
- STEP 3: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
- NOTE: RAM DOES NOT NEED TO BE OPEN, ALSO SEE SIMPLEX PACKAGE AND RAM API DOCS

STEPS & NOTES

GET ALL STORY IDS USING RAM API AND ZERO TOUCH C#

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using RAMDATAACCESSLib;
using Autodesk.DesignScript.Runtime;
using Autodesk.DesignScript.Geometry;

namespace DYNAMO_FOR_RAM_SIMPLE
{
    public class RAM_GEOMETRY
    {
        private RAM_GEOMETRY()
        {
        }

        public static List<int> GET_STORY_IDS(string FileName)
        {
            RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
            RAMDATAACCESSLib.IDBIO1 IDBI = (RAMDATAACCESSLib.IDBIO1)
            RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBIO1_INT);
            RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
            RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
            //OPEN
            IDBI.LoadDataBase2(FileName, "1");
            IStories My_stories = IModel.GetStories();
            int Story_Count = My_stories.GetCount();
            List<int> ListLine = new List<int>();
            for (int i = 0; i < Story_Count; i = i + 1)
            {
                int My_Story_Id = My_stories.GetAt(i).LUID;
                ListLine.Add(My_Story_Id);
            }
            //CLOSE
            IDBI.CloseDatabase();
            return ListLine;
        }
    }
}

```

CREATES METHOD/DYNAMO NODE LIST STRING OUT AND FILE NAME IN

GETS IMODEL OBJECT

OPEN RAM MODEL

GETS ISTORIES OBJECT

GETS NUMBER OF STORIES FROM ISTORIES

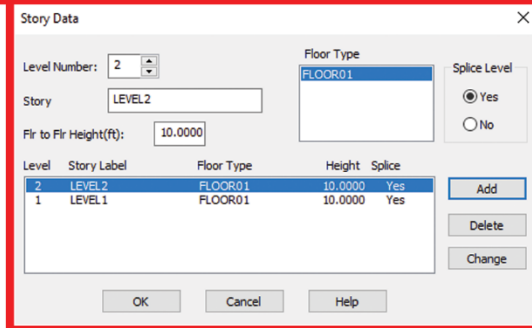
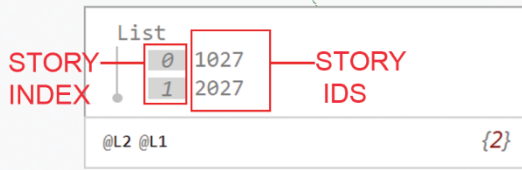
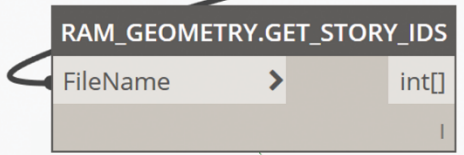
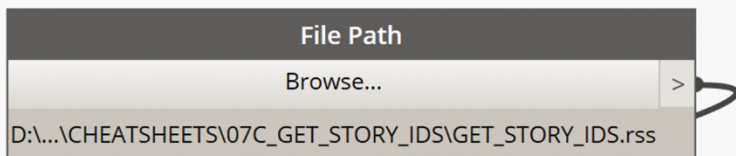
SETS UP LIST OF NAMES FOR OUTPUT

SETS COUNTER FOR LOOPS

GETS IDS OF EACH STORY

CLOSES RAM MODEL

ZERO TOUCH C# RAM API CODE



RAM MODEL

DYNAMO NODES

- STEP 1: OPEN VISUAL STUDIO FOLDER "GET_STORY_IDS" OPEN SLN FILE - BUILD SLN
- STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE
- STEP 3: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
- NOTE: RAM DOES NOT NEED TO BE OPEN, ALSO SEE SIMPLEX PACKAGE AND RAM API DOCS

STEPS & NOTES

GET NUMBER OF COLUMNS/STORY IN RAM VIA DYNAMO USING RAM API AND ZERO TOUCH C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using RAMDATAACCESSLib;
using Autodesk.DesignScript.Runtime;
using Autodesk.DesignScript.Geometry;
```

```
namespace DYNAMO_FOR_RAM_SIMPLE
```

```
{
    public class RAM_GEOMETRY
```

```
{
    private RAM_GEOMETRY()
    {
    }
}
```

```
public static int GET_STORY_COUNT(string FileName)
```

```
{
    RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBI01 IDBI = (RAMDATAACCESSLib.IDBI01)
    RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBI01 INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
    RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
```

```
IDBI.LoadDataBase2(FileName, "1");
```

```
IStories My_stories = IModel.GetStories();
```

```
int My_story_count = My_stories.GetCount();
```

```
IStory My_Story = My_stories.GetAt(In_Story_Count);
```

```
IColumns My_Columns = My_Story.GetColumns();
```

```
int Column_Count = My_Columns.GetCount();
```

GETS ICOLUMNS OBJECT AT EACH ISTORY OBJECT

```
IDBI.CloseDatabase();
```

```
return Column_Count;
```

VARIABLES THAT ACCESSES THE "IMODEL" OBJECT-THAT HAS ALL THE DATA NEEDED

OPENS DATABASE VIA FILE

GETS "ISTORY" OBJECT

GETS NUMBER OF STORIES FROM "ISTORY"

GETS "ISTORY" OBJECT FROM ISTORIES AT INDEX

CLOSES DATABASE

SENDS DYNAMO OUTPUT PORT

IDBI01

Philosophy: This interface is for performing functions on the database as a whole. This includes Loading and Saving. It also includes changing the database name. It does not include inquiries or specific data manipulation.

IModel

The IModel interface is the highest level in the hierarchy of interfaces. It represents the RAM Structural System model. Entities such as section definitions and floor types that are global to the model can be accessed through this interface. Other entities, such as decks or beams, which belong to a specific floor type, can be accessed through this interface using the entity's unique ID.

GetStories ([out, retval] IStories** pplStories)

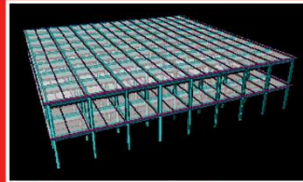
Gets the collection of all stories in the model.

Parameters
pplStories Pointer to an IStories collection interface that represents all stories in the model

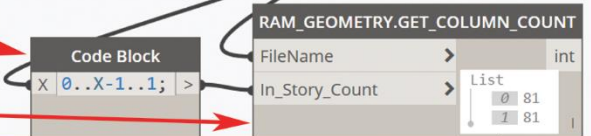
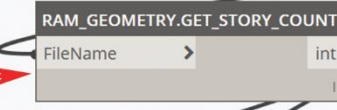
GetCount ([out, retval] long* plCount)

Gets the number of stories in the collection.

Parameters
plCount Number of stories in the collector



GETS STORY COUNT
MAKE LIST OF STORIES
GETS ALL COL PER STORY



DYNAMO NODES & RAM MODEL

ZERO TOUCH C# RAM API CODE

RAM API MANUAL

- STEP 1: OPEN VISUAL STUDIO FOLDER "GET_NUM_COLUMNS" OPEN SLN FILE - BUILD SLN
 - STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODES
 - STEP 3: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
- NOTE: RAM DOES NOT NEED TO BE OPEN, AND NOTE RELATIONSHIP WITH API DOC AND CODE

STEPS & NOTES

CREATES A NEW FLOOR IN RAM USING DYNAMO USING RAM API AND ZERO TOUCH C#

OUTPUT PORT TYPE

INPUT PORTS

```

public static int SET_FLOOR_TYPE(string FloorTypeName, string FileName)
{
    RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBIO1 IDBI = (RAMDATAACCESSLib.IDBIO1)
    RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBIO1_INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
    RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);

    //OPEN
    IDBI.LoadDataBase2(FileName, "1");

    IFloorTypes My_floortypes = IModel.GetFloorTypes();
    IFloorType My_New_floortype = My_floortypes.Add(FloorTypeName);
    int MyFlrTypeID = My_New_floortype.LUID;

    //CLOSE
    IDBI.SaveDatabase();
    IDBI.CloseDatabase();

    return MyFlrTypeID;
}
    
```

VARIABLES THAT ACCESS THE "IMODEL" OBJECT - THAT HAS ALL THE DATA NEEDED

OPENS RAM FILE

GETS ALL FLOOR TYPES

CREATES A NEW FLOOR TYPE

GETS THE NEW FLOOR TYPE ID

SAVES RAM FILE

CLOSES RAM FILE

SENDS FLOOR ID TO OUTPUT PORT

ZERO TOUCH C# RAM API CODE

File Path

Browse... >

D:\...\CHEATSHEETS\09_WRITE_TO_ETABS_FLOORTYPE\WRITE_TO_ETABS_FLOORTYPE.rss

Code Block

"MyFirstWrite"; >

RAM_GEOMETRY_WRITE.SET_FLOOR_TYPE

FloorTypeName > int

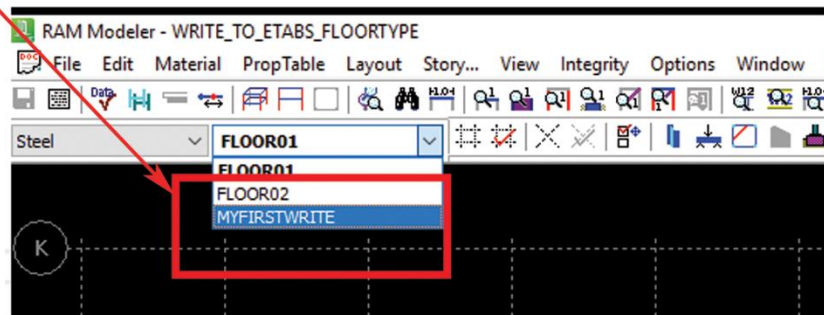
FileName >

SELECTS RAM FILE TO BE OPENED

FLOOR NAME INPUT AS STRING

CREATES A NEW FLOOR TYPE IN RAM. WILL CREATE MULTIPLE FLOORS AT ONCE

DYNAMO NODE



RAM MODEL

- STEP 1: OPEN VISUAL STUDIO FOLDER "WRITE_TO_RAM_FLOORTYPE" OPEN SLN FILE - BUILD SLN**
- STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE**
- STEP 3: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS**
- NOTE: RAM DOES NOT NEED TO BE OPEN, ERROR OPENING FILE? DELETE .USR AND WORKING DIR**

STEPS & NOTES

CONVERT BEAMS AND COLUMNS FROM RAM TO DYNAMO LINES USING RAM API AND C#

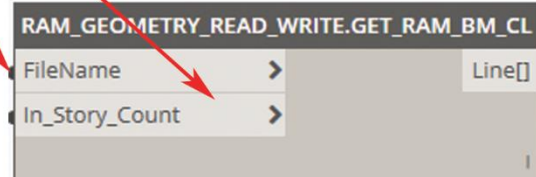
```
public static List<Autodesk.DesignScript.Geometry.Line> GET_RAM_COL_CL(string FileName, int In_Story_Count)
{
    RamDataAccess1 RamDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBIO1 IDBI = (RAMDATAACCESSLib.IDBIO1)
    RamDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBIO1_INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
    RamDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
    //OPEN
    IDBI.LoadDataBase2(FileName, "1");
    IStories My_stories = IModel.GetStories();
    int My_story_count = My_stories.GetCount();
    IStory My_Story = My_stories.GetAt(In_Story_Count);
    IColumns My_Columns = My_Story.GetColumns();
    int Column_Count = My_Columns.GetCount();
    SCoordinate P1 = new SCoordinate();
    SCoordinate P2 = new SCoordinate();
    List<Autodesk.DesignScript.Geometry.Line> ListLine = new List<Autodesk.DesignScript.Geometry.Line>();
    //create loop herenthru all count
    //start..end..step
    for (int i = 0; i < Column_Count; i = i + 1)
    {
        My_Story.GetColumns().GetAt(i).GetEndCoordinates(ref P1, ref P2);
        double P1x = P1.dXLoc;
        double P1y = P1.dYLoc;
        double P1z = P1.dZLoc;
        double P2x = P2.dXLoc;
        double P2y = P2.dYLoc;
        double P2z = P2.dZLoc;
        Autodesk.DesignScript.Geometry.Point PD1 =
            Autodesk.DesignScript.Geometry.Point.ByCoordinates(P1x, P1y, P1z);
        Autodesk.DesignScript.Geometry.Point PD2 =
            Autodesk.DesignScript.Geometry.Point.ByCoordinates(P2x, P2y, P2z);
        Autodesk.DesignScript.Geometry.Line Dline =
            Autodesk.DesignScript.Geometry.Line.ByStartPointEndPoint(PD1, PD2);
        ListLine.Add(Dline);
    }
    //CLOSE
    IDBI.CloseDatabase();
    //return list
    return ListLine;
}
```



CODE EXTRACTS ALL COLUMNS PER STORY AND CONVERTS RAM GEO TO DYNAMO LINES
ONLY WORKS ON VERTICAL COLUMNS

C# RAM API CODE AND DYNAMO NODE

```
public static List<Autodesk.DesignScript.Geometry.Line> GET_RAM_BM_CL(string FileName, int In_Story_Count)
{
    RamDataAccess1 RamDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBIO1 IDBI = (RAMDATAACCESSLib.IDBIO1)
    RamDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBIO1_INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
    RamDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
    //OPEN
    IDBI.LoadDataBase2(FileName, "1");
    IStories My_stories = IModel.GetStories();
    int My_story_count = My_stories.GetCount();
    IStory My_Story = My_stories.GetAt(In_Story_Count);
    IBeams My_Beams = My_Story.GetBeams();
    int Beam_Count = My_Beams.GetCount();
    SCoordinate P1 = new SCoordinate();
    SCoordinate P2 = new SCoordinate();
    List<Autodesk.DesignScript.Geometry.Line> ListLine = new List<Autodesk.DesignScript.Geometry.Line>();
    //create loop herenthru all count
    //start..end..step
    for (int i = 0; i < Beam_Count; i = i + 1)
    {
        My_Story.GetBeams().GetAt(i).GetCoordinates(EBeamCoordLoc.eBeamEnds, ref P1, ref P2);
        double P1x = P1.dXLoc;
        double P1y = P1.dYLoc;
        double P1z = P1.dZLoc;
        double P2x = P2.dXLoc;
        double P2y = P2.dYLoc;
        double P2z = P2.dZLoc;
        Autodesk.DesignScript.Geometry.Point PD1 =
            Autodesk.DesignScript.Geometry.Point.ByCoordinates(P1x, P1y, P1z);
        Autodesk.DesignScript.Geometry.Point PD2 =
            Autodesk.DesignScript.Geometry.Point.ByCoordinates(P2x, P2y, P2z);
        Autodesk.DesignScript.Geometry.Line Dline =
            Autodesk.DesignScript.Geometry.Line.ByStartPointEndPoint(PD1, PD2);
        ListLine.Add(Dline);
    }
    //CLOSE
    IDBI.CloseDatabase();
    return ListLine;
}
```



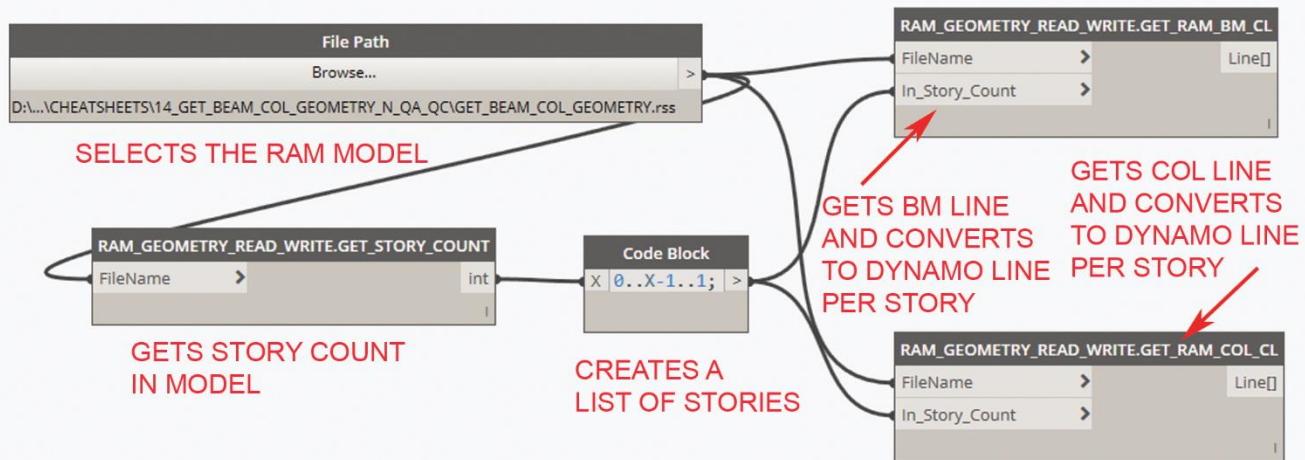
CODE EXTRACTS ALL BEAMS PER STORY AND CONVERTS RAM GEO TO DYNAMO LINES
ONLY WORKS ON VERTICAL COLUMNS

C# RAM API CODE AND DYNAMO NODE

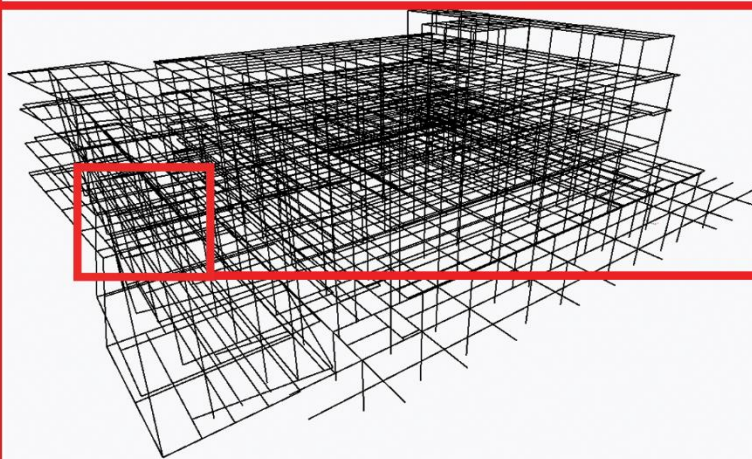
- STEP 1: OPEN VISUAL STUDIO FOLDER "EXTRACT_GEO_BEAM_COLUMNS_CODE" OPEN SLN FILE
STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
STEP 3: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL

STEPS & NOTES

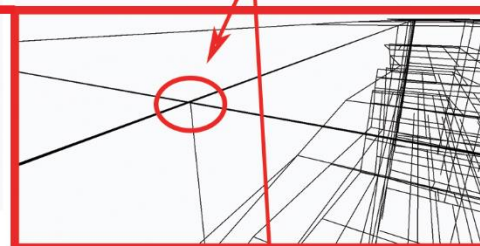
GET COLUMN AND BEAM CENTERLINES FROM RAM AND CONVERT TO DYNAMO LINES



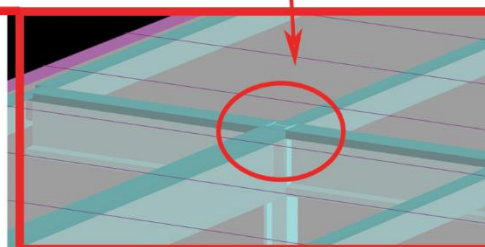
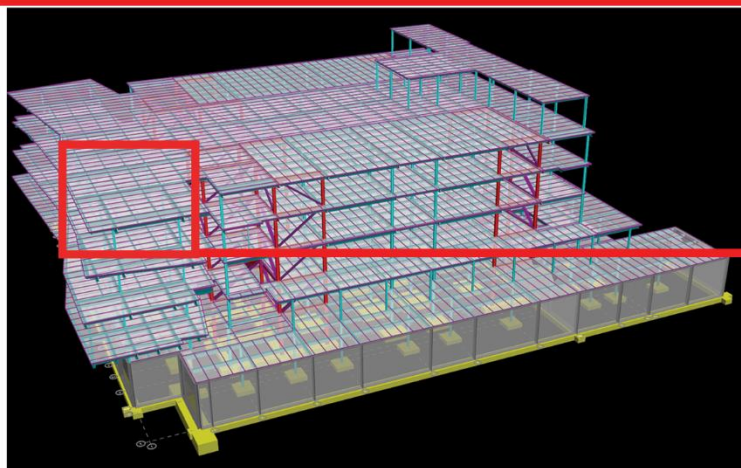
DYNAMO NODES



VIEWING BEAM AND COLUMN CENTERLINES IN DYNAMO ALLOWS FOR QUALITY CONTROL OF JOINTS BECAUSE RAM DOES NOT SHOWN CENTERLINE GEOMETRY IN 3D VIEW IN RAM



DYNAMO GEOMETRY



RAM MODEL

- STEP 1: OPEN VISUAL STUDIO FOLDER "GET_BEAM_COL_GEOMETRY" OPEN SLN FILE
 STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
 STEP 3: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
 NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL

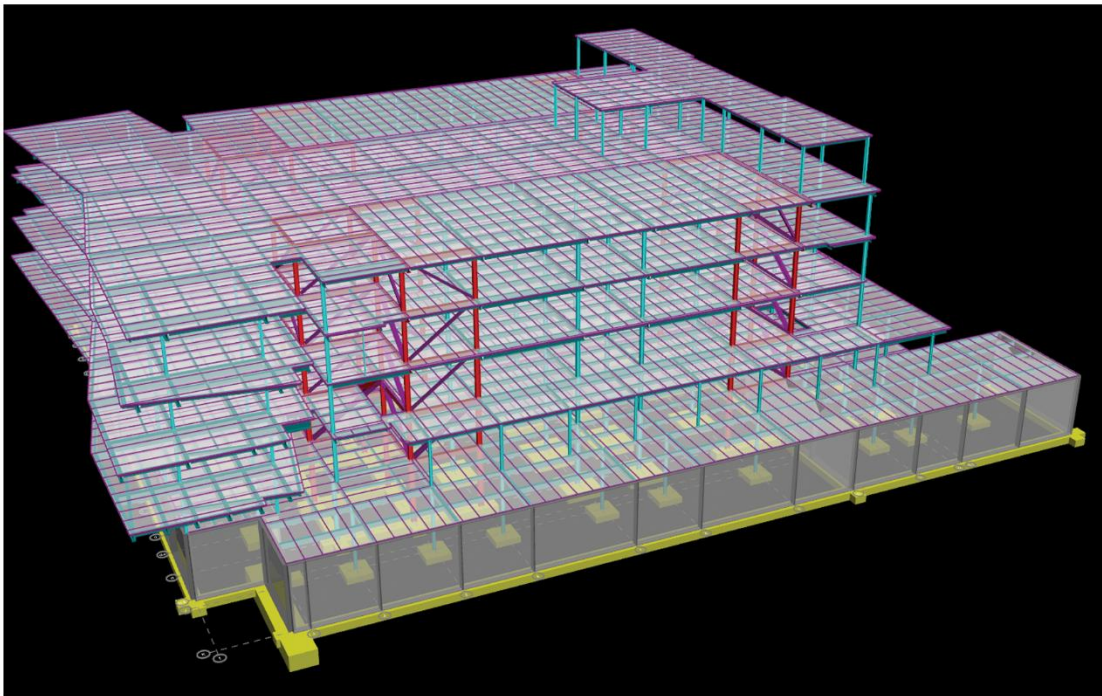
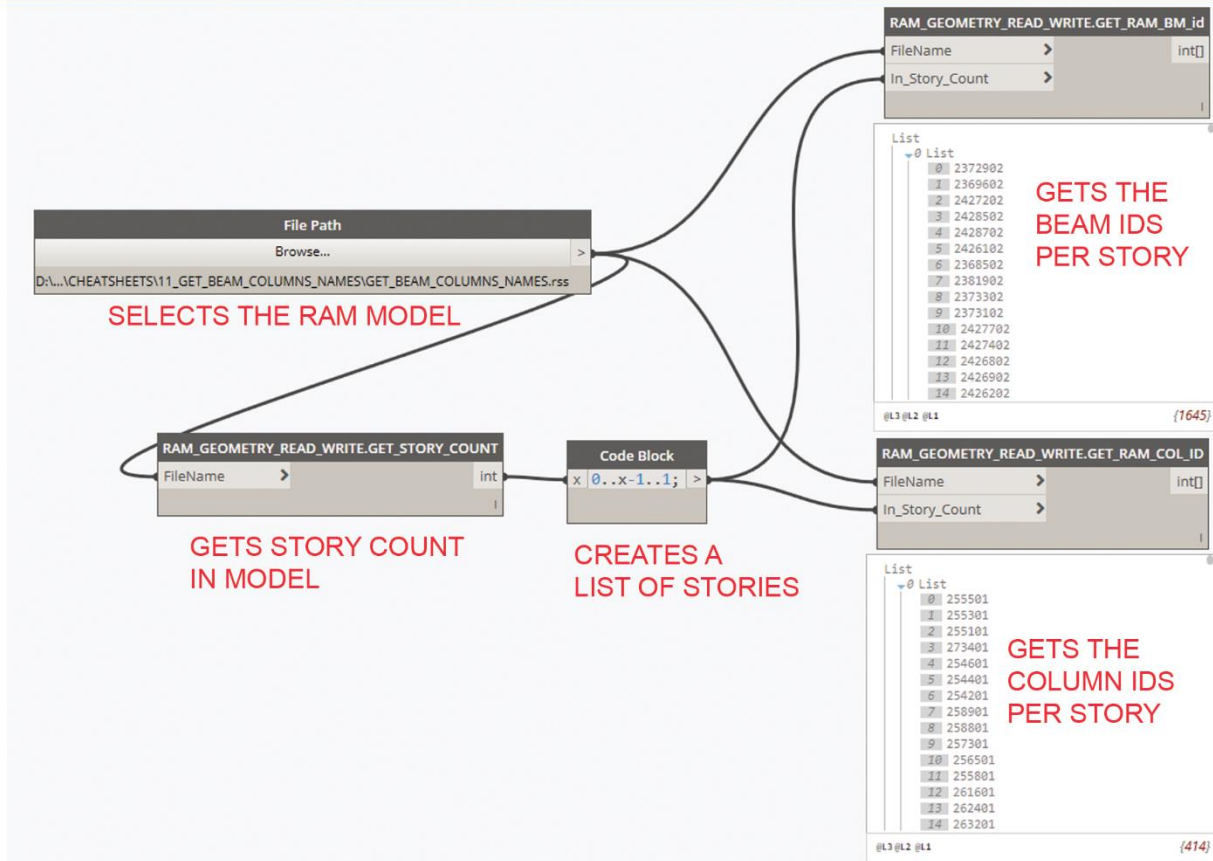
STEPS & NOTES

GET BEAM AND COLUMN IDS FROM RAM USING DYNAMO VIA RAM API AND C#

DYNAMO NODES

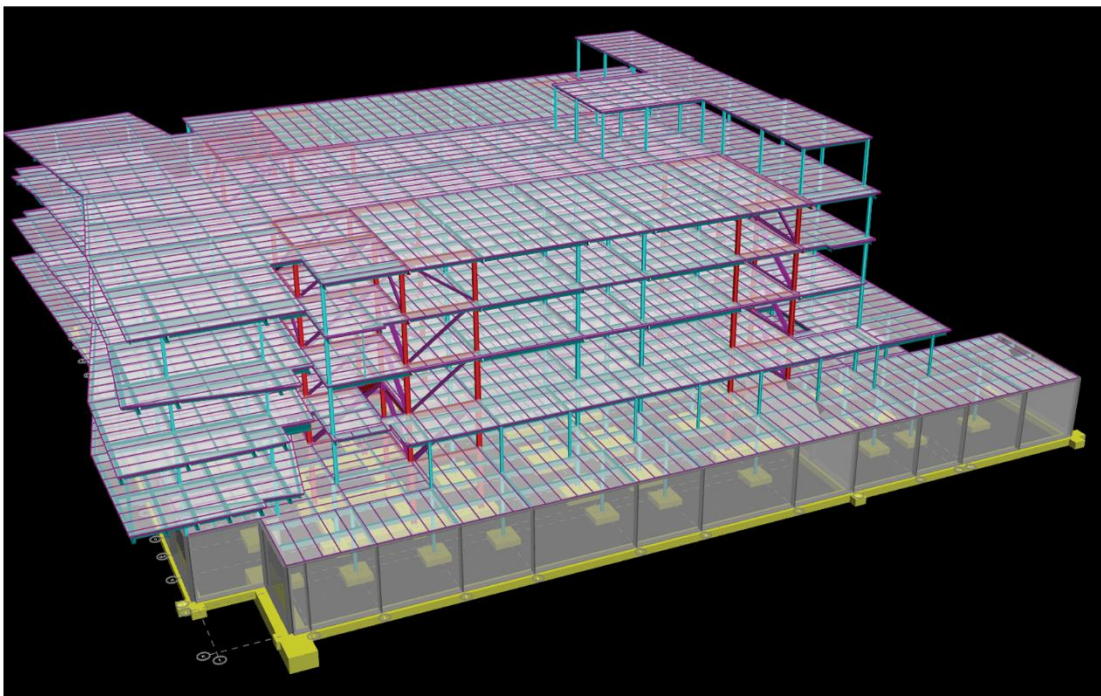
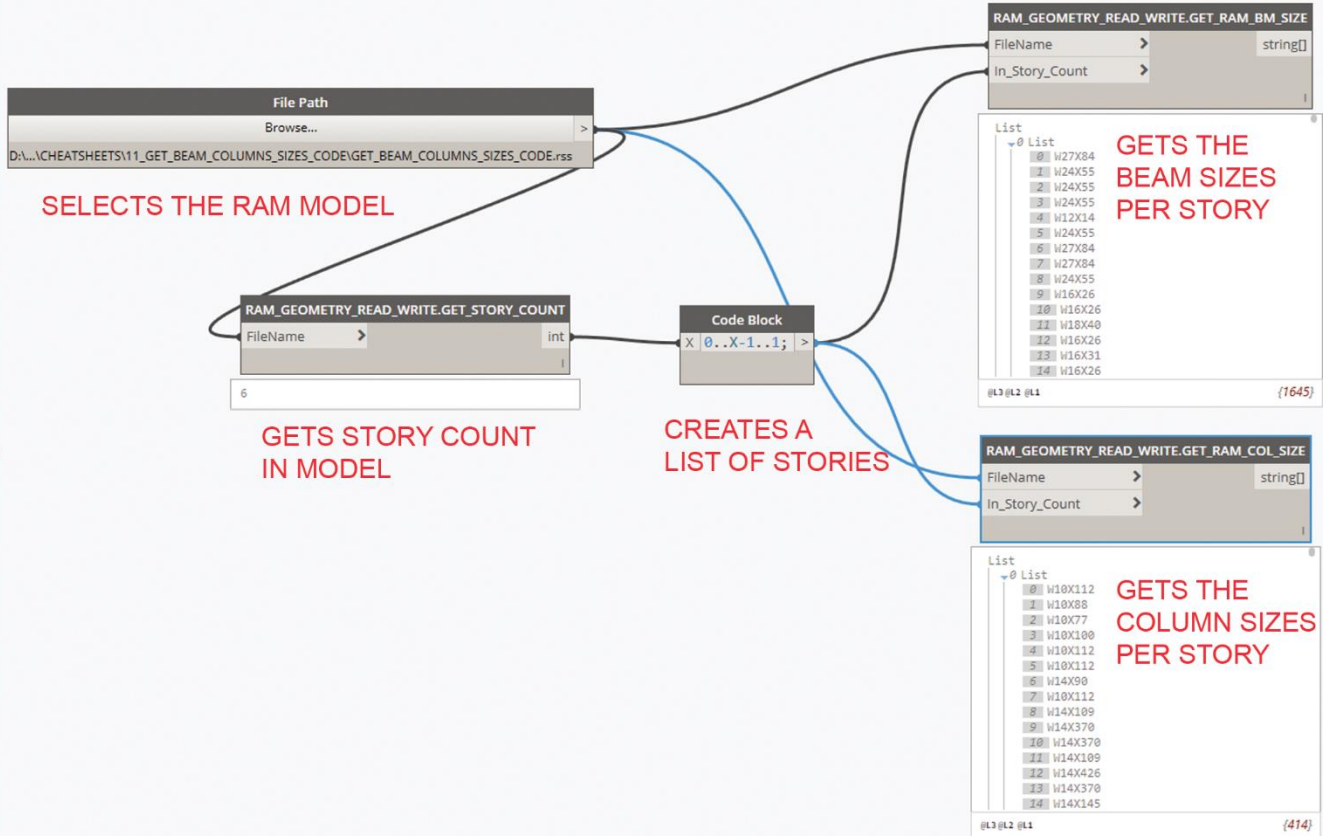
RAM MODEL

STEPS & NOTES



STEP 1: OPEN VISUAL STUDIO FOLDER "GET_BEAM_COLUMN_IDS" OPEN SLN FILE
 STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
 STEP 3: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
 NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL

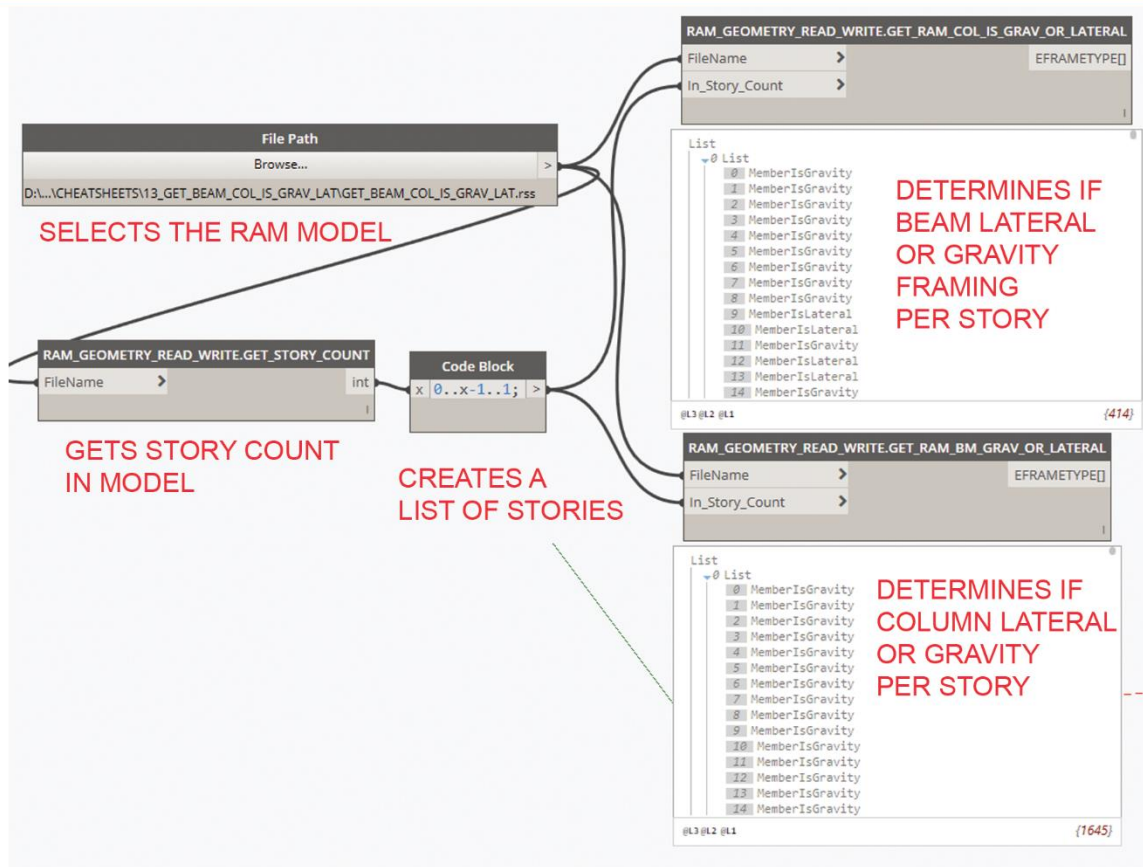
GET BEAM AND COLUMN SIZES FROM RAM USING DYNAMO VIA RAM API AND C#



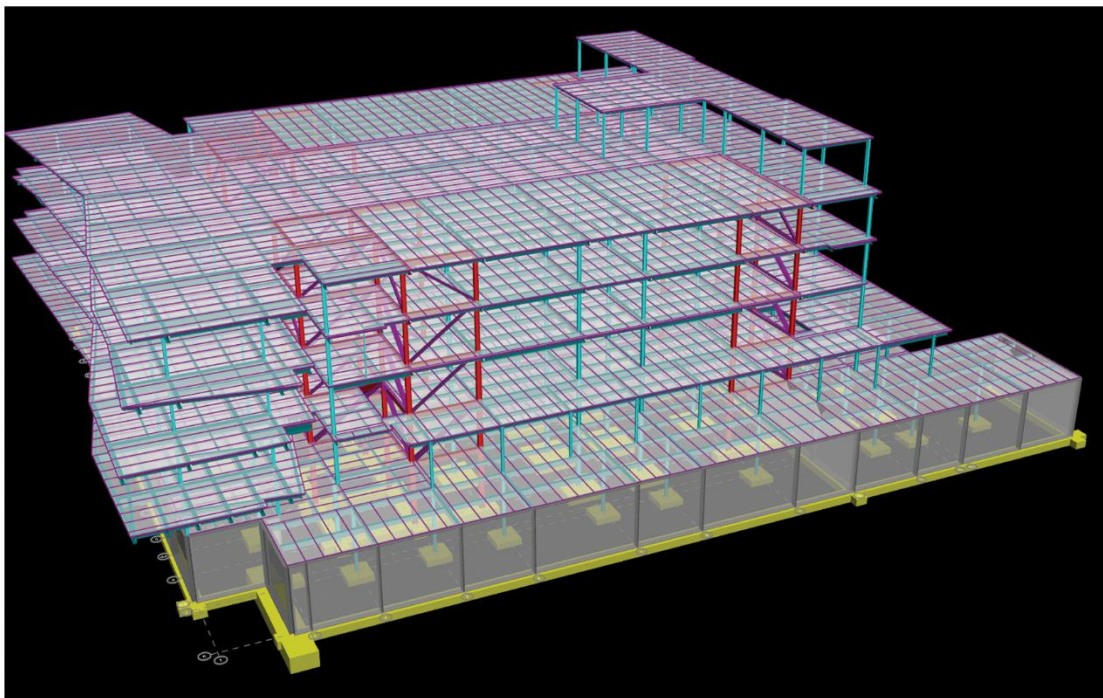
STEP 1: OPEN VISUAL STUDIO FOLDER "GET_BEAM_COLUMN_SIZES" OPEN SLN FILE
 STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
 STEP 3: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
 NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL

STEPS & NOTES

DETERMINE IF BEAM AND COLUMN FROM RAM ARE GRAVIT OR LATERAL FRAMING



DYNAMO NODES



RAM MODEL

- STEP 1: OPEN VISUAL STUDIO FOLDER "GET_BEAM_COL_IS_GRAV_LAT" OPEN SLN FILE
- STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
- STEP 3: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
- NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL

STEPS & NOTES

WRITES COLUMNS TO RAM VIA DYNAMO USING RAM API AND C#

```

public static int CREATE_RAM_STEEL_COL(int FloorIndex,string FileName,double XX,
double YY,double ZTop, double ZBot)
{
    RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBIO1 IDBI = (RAMDATAACCESSLib.IDBIO1)
        RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBIO1_INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
        RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
    //OPEN
    IDBI.LoadDataBase2(FileName, "1");

    IFloorTypes My_floortypes = IModel.GetFloorTypes();
    IFloorType My_floortype = My_floortypes.GetAt(FloorIndex);
    EMATERIALTYPES My_ColMaterial = EMATERIALTYPES.ESteelMat;

    ILayoutColumn My_LayoutColumn = My_floortype.GetLayoutColumns()
        .Add(My_ColMaterial, XX, YY, ZTop, ZBot);

    //CLOSE
    IDBI.SaveDatabase();
    IDBI.CloseDatabase();
    int My_New_Col_ID = My_LayoutColumn.LUID;
    return My_New_Col_ID;
}
    
```

GETS IMODEL OBJECT FROM RAM

GETS FLOORTYPES OBJECT FROM IMODEL

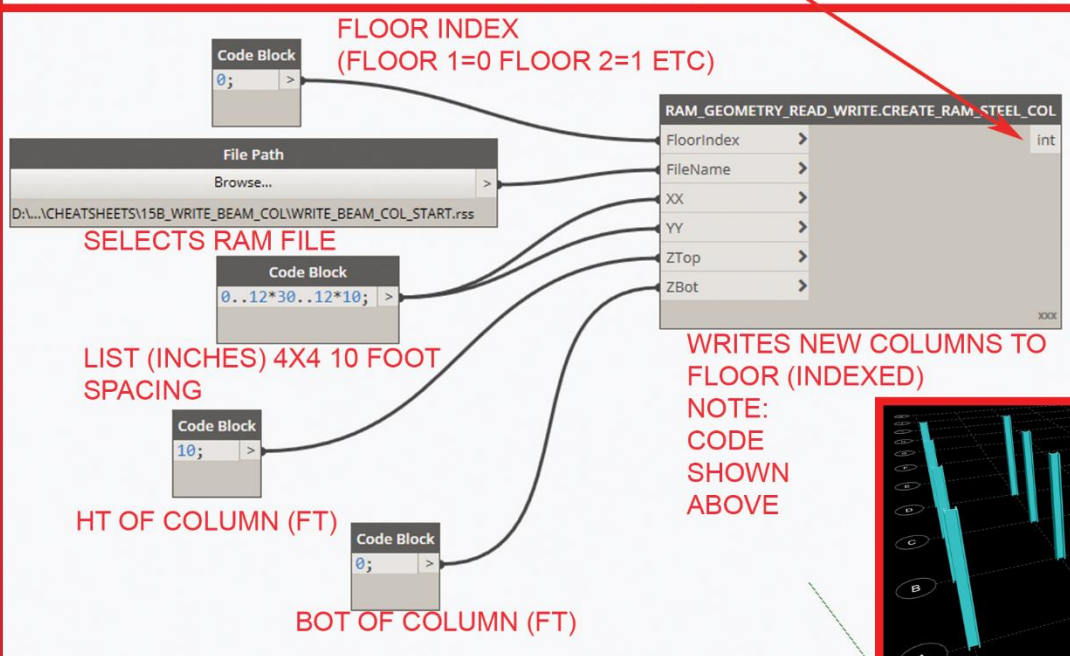
GETS SINGLE FLOORTYPE AT INDEX FROM INPUT PORT

GETS STEEL MATERIAL OBJECT

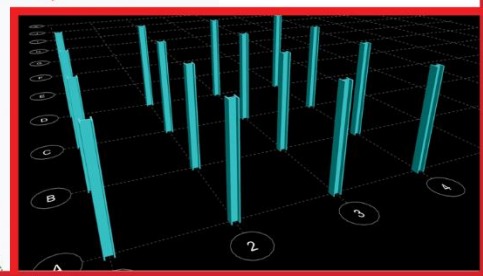
GETS ILayoutColumns OBJECT FROM SINGLE FLOORTYPE AND CREATE A SINGLE ILayoutColumn OBJECT THAT DEFINES A SINGLE COLUMN

GETS NEW COLUMN AND RETURNS ITS ID

ZERO TOUCH C# RAM API CODE



DYNAMO NODES



RAM MODEL

- STEP 1: OPEN FOLDER "WRITE_COL" OPEN SLN FILE
 - STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
 - STEP 3: SELECT THE RAM FILE IN FOLDER VIA FILE PATH NODE AND WATCH THE RESULTS
- NOTE: RAM DOES NOT NEED TO BE OPEN. ALSO SEE SIMPLEX PACKAGE AND RAM API MANUAL

STEPS & NOTES

WRITES BEAMS TO RAM VIA DYNAMO USING RAM API AND C#

```

public static int CREATE_RAM_STEEL_BM(int FloorIndex, string FileName,
double StartSupportX, double StartSupportY,
double StartSupportZ, double EndSupportX, double EndSupportY, double EndSupportZ)
{
    RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBI01 IDBI = (RAMDATAACCESSLib.IDBI01)
        RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBI01_INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
        RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
    //OPEN
    IDBI.LoadDataBase2(FileName, "1");

    IFloorTypes My_floortypes = IModel.GetFloorTypes();
    IFloorType My_floortype = My_floortypes.GetAt(FloorIndex);
    EMATERIALTYPES My_BmMaterial = EMATERIALTYPES.ESteelMat;

    ILayoutBeam My_LayoutBeam = My_floortype.GetLayoutBeams()
        .Add(My_BmMaterial, StartSupportX, StartSupportY,
            StartSupportZ, EndSupportX, EndSupportY, EndSupportZ);

    //CLOSE
    IDBI.SaveDatabase();
    IDBI.CloseDatabase();
    int My_New_Beam_ID = My_LayoutBeam.LUID;
    return My_New_Beam_ID;
}

```

GETS IMODEL OBJECT FROM RAM

GETS FLOORTYPES OBJECT FROM IMODEL

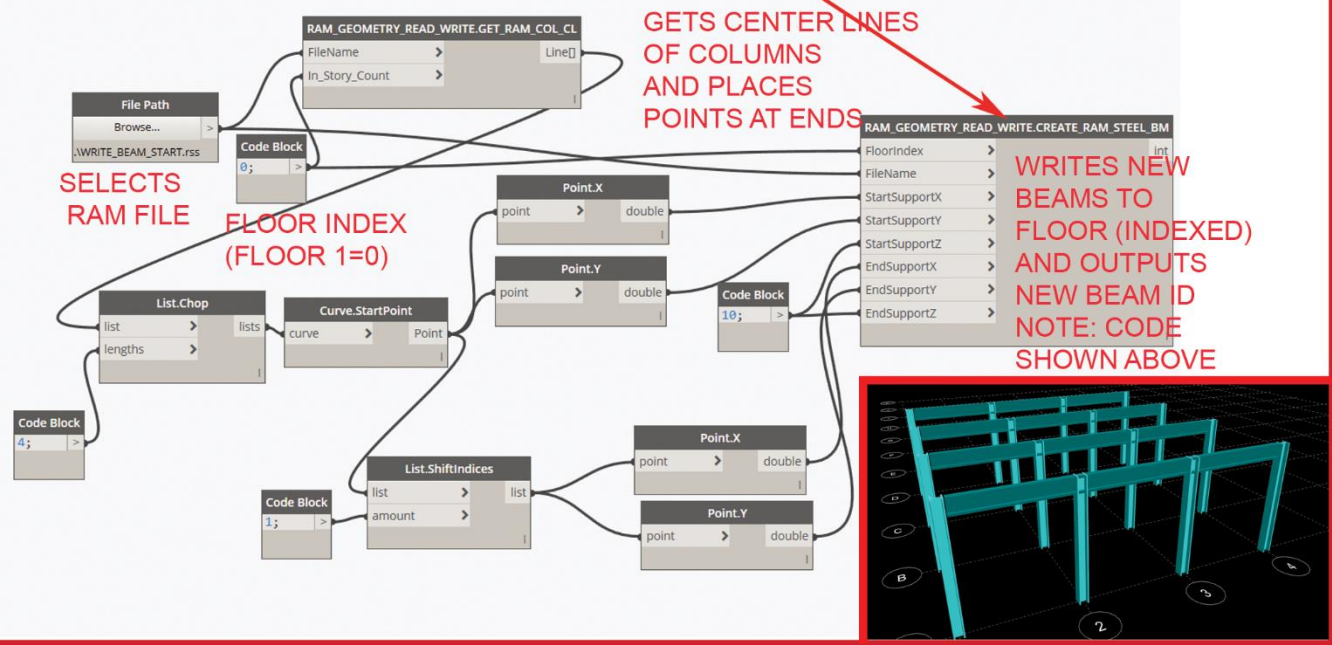
GETS SINGLE FLOORTYPE AT INDEX FROM INPUT PORT

GETS STEEL MATERIAL OBJECT

GETS I LAYOUT BEAMS OBJECT FROM SINGLE FLOORTYPE AND CREATE A SINGLE I LAYOUT BEAM OBJECT THAT DEFINES A SINGLE BEAM

GETS NEW BEAM AND RETURNS ITS ID

ZERO TOUCH C# RAM API CODE



DYNAMO NODES

RAM MODEL

STEP 1: OPEN FOLDER "WRITE_BEAM" OPEN SLN FILE
 STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
 STEP 3: SELECT THE RAM FILE IN FOLDER VIA FILE PATH NODE AND WATCH THE RESULTS
 NOTE: RAM DOES NOT NEED TO BE OPEN. ALSO SEE SIMPLEX PACKAGE AND RAM API MANUAL

STEPS & NOTES

WRITES BEAMS TO RAM VIA DYNAMO USING RAM API AND C#

```

public static int CREATE_RAM_STEEL_BM(int FloorIndex, string FileName,
double StartSupportX, double StartSupportY,
double StartSupportZ, double EndSupportX, double EndSupportY, double EndSupportZ)
{
    RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBI01 IDBI = (RAMDATAACCESSLib.IDBI01)
        RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBI01_INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
        RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
    //OPEN
    IDBI.LoadDataBase2(FileName, "1");

    IFloorTypes My_floortypes = IModel.GetFloorTypes();
    IFloorType My_floortype = My_floortypes.GetAt(FloorIndex);
    EMATERIALTYPES My_BmMaterial = EMATERIALTYPES.ESteelMat;

    ILayoutBeam My_LayoutBeam = My_floortype.GetLayoutBeams()
        .Add(My_BmMaterial, StartSupportX, StartSupportY,
            StartSupportZ, EndSupportX, EndSupportY, EndSupportZ);

    //CLOSE
    IDBI.SaveDatabase();
    IDBI.CloseDatabase();
    int My_New_Beam_ID = My_LayoutBeam.LUID;
    return My_New_Beam_ID;
}
    
```

GETS IMODEL OBJECT FROM RAM

GETS FLOORTYPES OBJECT FROM IMODEL

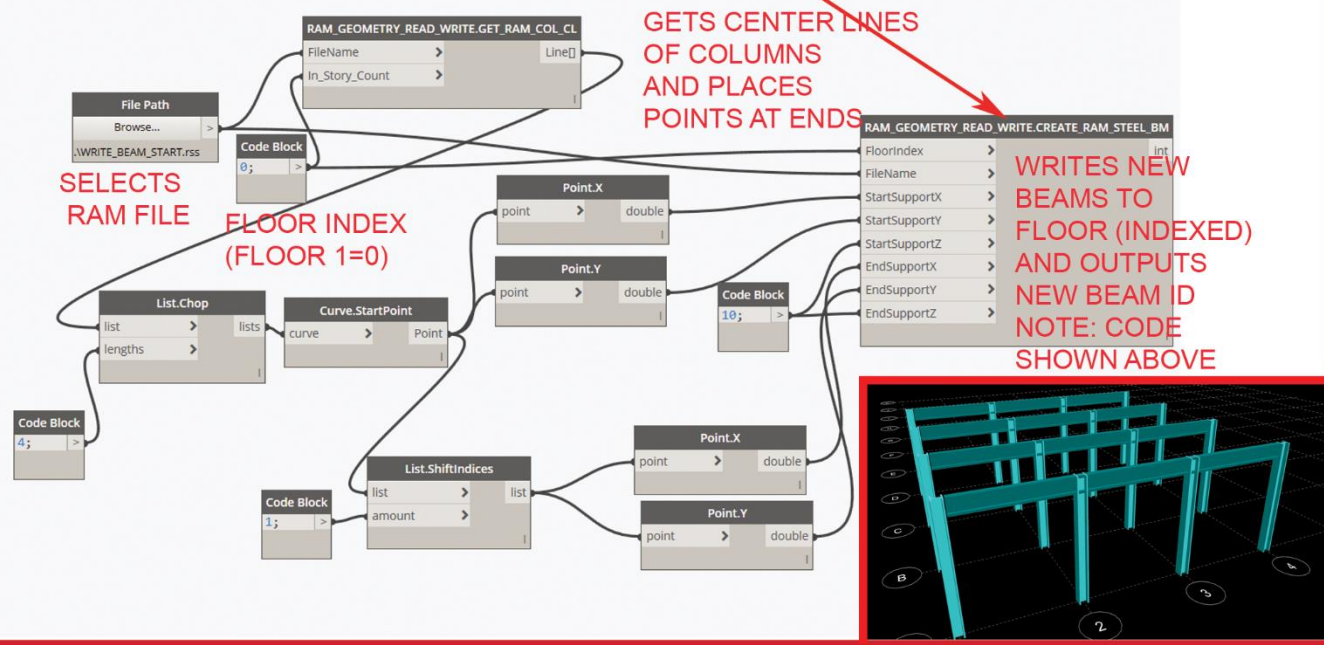
GETS SINGLE FLOORTYPE AT INDEX FROM INPUT PORT

GETS STEEL MATERIAL OBJECT

GETS I LAYOUT BEAMS OBJECT FROM SINGLE FLOORTYPE AND CREATE A SINGLE I LAYOUT BEAM OBJECT THAT DEFINES A SINGLE BEAM

GETS NEW BEAM AND RETURNS ITS ID

ZERO TOUCH C# RAM API CODE



DYNAMO NODES

RAM MODEL

STEP 1: OPEN FOLDER "WRITE_BEAM" OPEN SLN FILE
 STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
 STEP 3: SELECT THE RAM FILE IN FOLDER VIA FILE PATH NODE AND WATCH THE RESULTS
 NOTE: RAM DOES NOT NEED TO BE OPEN. ALSO SEE SIMPLEX PACKAGE AND RAM API MANUAL

STEPS & NOTES

WRITES HORIZONTAL BRACES TO RAM VIA DYNAMO USING RAM API AND C#

```

public static int CREATE_RAM_STEEL_BRACE(int FloorIndex, string FileName,
double StartSupportX, double StartSupportY,
double StartSupportZ, double EndSupportX, double EndSupportY,
double EndSupportZ)
{
    RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBIO1 IDBI = (RAMDATAACCESSLib.IDBIO1)
        RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBIO1_INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
        RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
    //OPEN
    IDBI.LoadDataBase2(FileName, "1");

    IFloorTypes My_floortypes = IModel.GetFloorTypes();
    IFloorType My_floortype = My_floortypes.GetAt(FloorIndex);
    EMATERIALTYPES My_BmMaterial = EMATERIALTYPES.ESteelMat;

    ILayoutHorizBrace My_LayoutBrace = My_floortype.
        GetLayoutHorizBraces().Add(My_BmMaterial,
        StartSupportX, StartSupportY,
        StartSupportZ, EndSupportX, EndSupportY, EndSupportZ);
    //CLOSE
    IDBI.SaveDatabase();
    IDBI.CloseDatabase();
    int My_New_Brace_ID = My_LayoutBrace.LUID;
    return My_New_Brace_ID;
}

```

GETS IMODEL OBJECT FROM RAM

GETS FLOORTYPES OBJECT FROM IMODEL

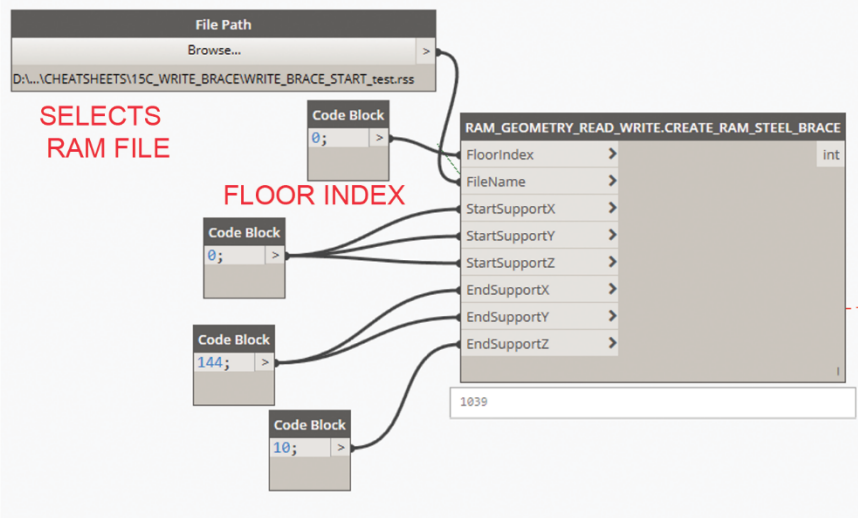
GETS SINGLE FLOORTYPE AT INDEX FROM INPUT PORT

GETS STEEL MATERIAL OBJECT

GETS ILAYOUTBRACE OBJECT FROM SINGLE FLOORTYPE AND CREATE A SINGLE ILAYOUTBRACE OBJECT THAT DEFINES A SINGLE BRACE

GETS NEW BEAM AND RETURNS ITS ID

ZERO TOUCH C# RAM API CODE

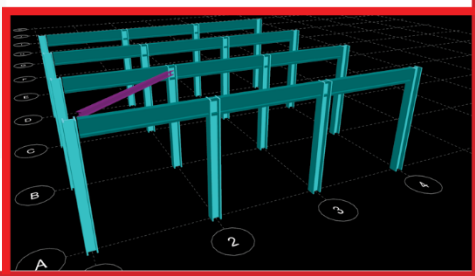


SELECTS RAM FILE

FLOOR INDEX

WRITES NEW BRACE TO FLOOR (INDEXED) AND OUTPUTS NEW BRACE ID
NOTE: CODE SHOWN ABOVE

DYNAMO NODES

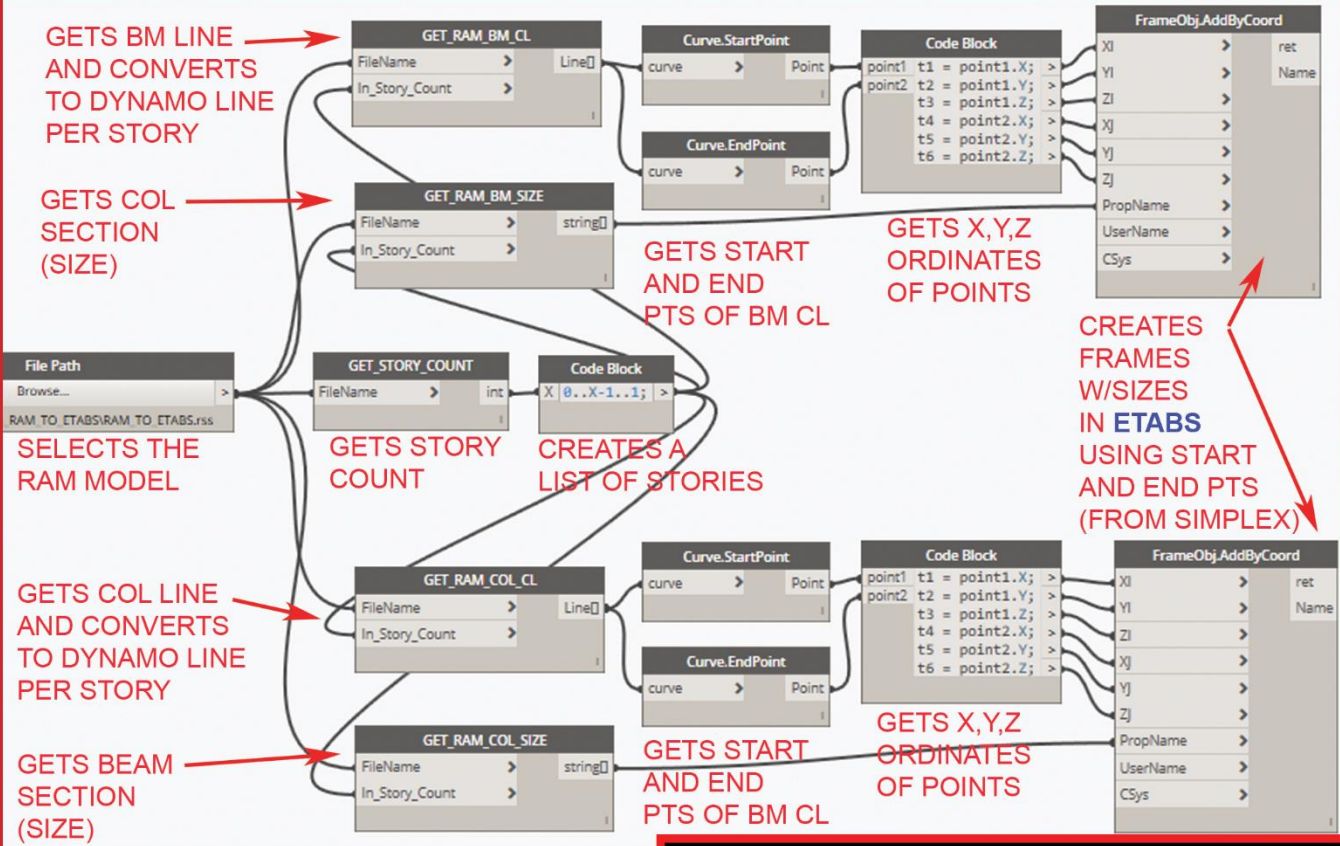


RAM MODEL

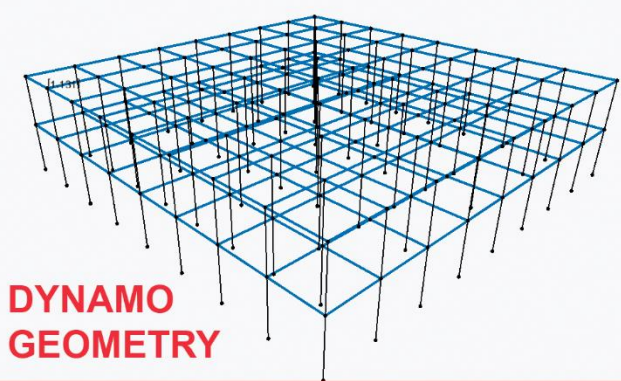
- STEP 1: OPEN FOLDER "WRITE_BRACE" OPEN SLN FILE
 - STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
 - STEP 3: SELECT THE RAM FILE IN FOLDER VIA FILE PATH NODE AND WATCH THE RESULTS
- NOTE: RAM DOES NOT NEED TO BE OPEN. ALSO SEE SIMPLEX PACKAGE AND RAM API MANUAL

STEPS & NOTES

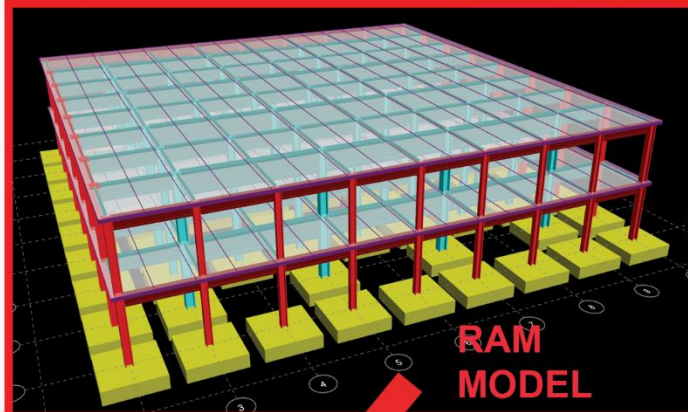
WRITE RAM BEAMS AND COLUMNS TO ETABS USING DYNAMO!



DYNAMO NODES AND GEOMETRY

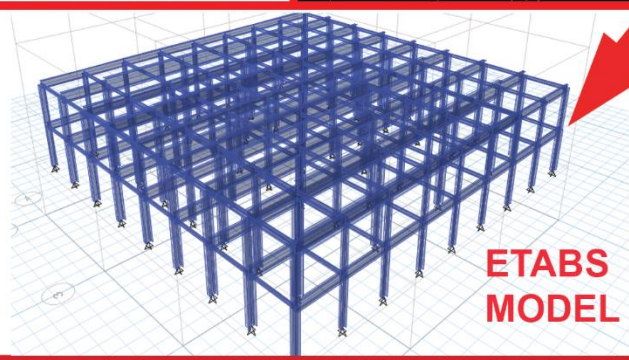


DYNAMO GEOMETRY



RAM MODEL

RAM MODEL



ETABS MODEL

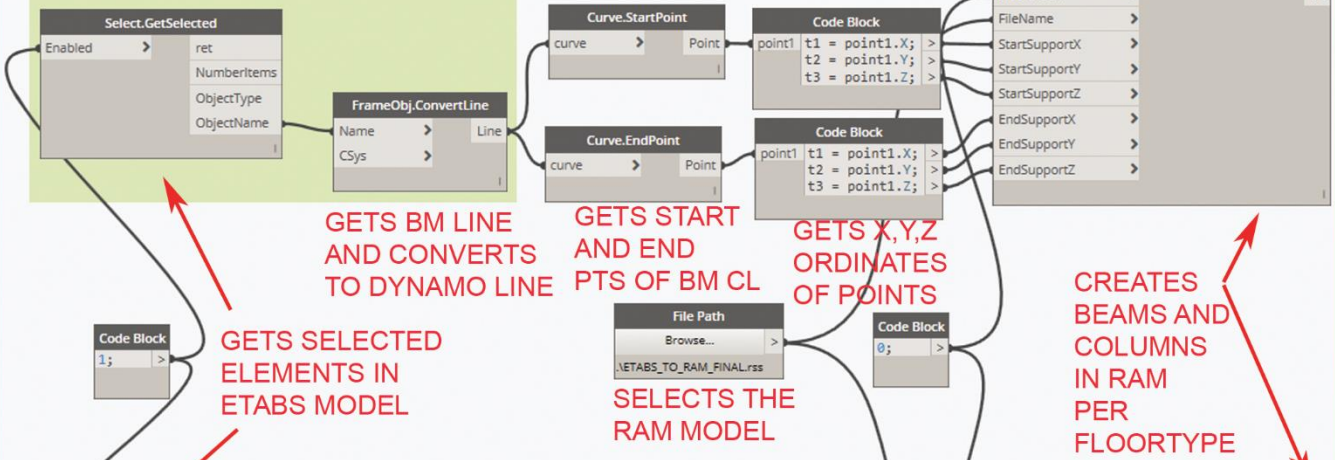
ETABS MODEL

- STEP 1:** OPEN VISUAL STUDIO FOLDER "RAM_TO_ETABS" OPEN SLN FILE, BUILD SOLUTION
STEP 2: OPEN ETABS FILE "RAM_TO_ETABS_START.EDB"
STEP 3: OPEN DYNAMO, LOAD RECENTLY BUILT DLL, PLACE NODES AS SHOWN (MAKE SURE TO HAVE SIMPLEX INSTALLED)
STEP 4: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL FOR REFERENCE

STEPS & NOTES

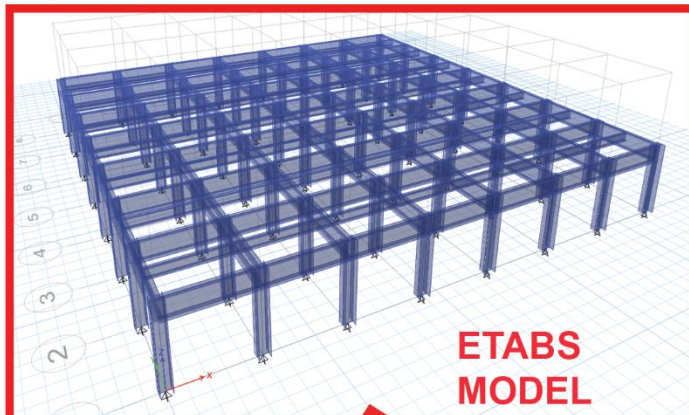
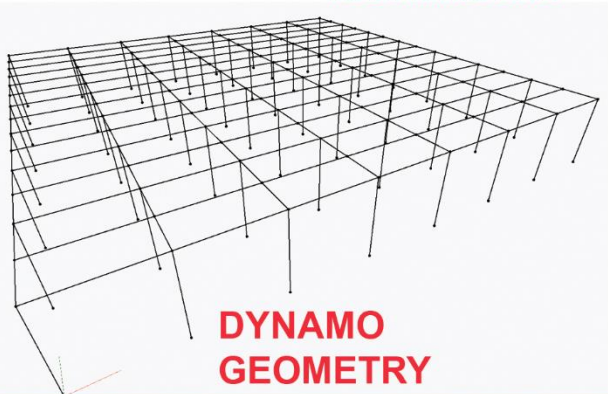
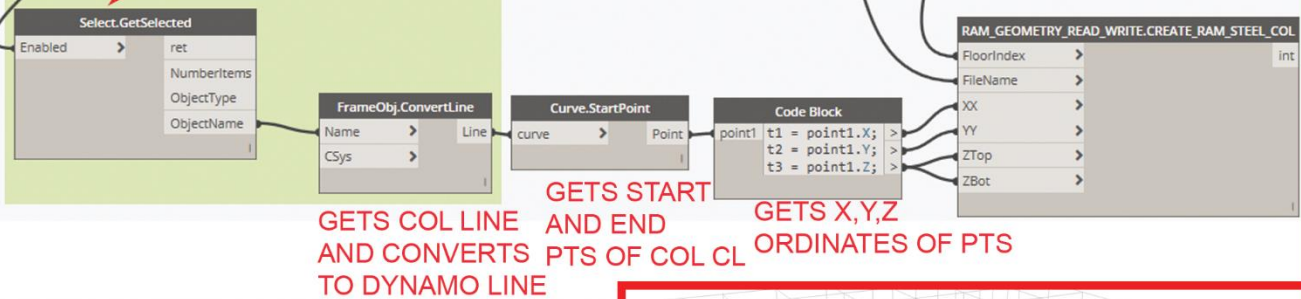
WRITE ETABS BEAMS AND COLUMNS TO RAM USING DYNAMO!

BEAMS NODES FROM SIMPLEX

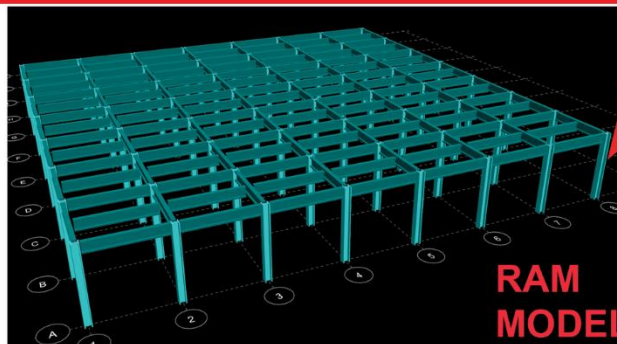


DYNAMO NODES AND GEOMETRY

COLUMNS NODES FROM SIMPLEX



ETABS MODEL



RAM MODEL

- STEP 1: OPEN FOLDER "ETABS_TO_RAM" OPEN SLN FILE, BUILD SOLUTION
 STEP 2: OPEN ETABS FILE "ETABS_TO_RAM_START.EDB"
 STEP 3: OPEN DYNAMO, LOAD RECENTLY BUILT DLL, PLACE NODES AS SHOWN (MAKE SURE TO HAVE SIMPLEX INSTALLED)
 STEP 4: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
 NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL FOR REFERENCE
 MORE THAN TWO STORIES BUILT EACH FLOOR SEPARATELY

STEPS & NOTES

GET GRID INFORMATION USING DYNAMO VIA RAM API AND C#

```
[MultiReturn(new[] { "Grid_Name", "Grid_Ordinates", "Grid_Axis" })]
public static Dictionary<string, object> GET_GRID_INFO(string FileName)
{
    RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBIO1 IDBI = (RAMDATAACCESSLib.IDBIO1)
    RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBIO1_INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
    RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
    Dictionary<string, object> OutPutPorts = new Dictionary<string, object>();
    //OPEN
    IDBI.LoadDataBase2(FileName, "1");
    IModelGrids My_Model_Grids = IModel.GetGridSystems().GetAt(0).GetGrids();
    int My_Grid_Count = My_Model_Grids.GetCount();
    List<double> Grid_Ordinates = new List<double>();
    List<string> Grid_Name = new List<string>();
    List<string> Grid_Axis = new List<string>();

    for (int i = 0; i < My_Grid_Count; i = i + 1)
    {
        //round up and convert grids from inches to feet
        double My_Grid_ORD = Math.Ceiling(My_Model_Grids.GetAt(i).dCoordinate_Angle/12);
        string My_Model_Grid_Names = My_Model_Grids.GetAt(i).strLabel;
        string My_Model_Grid_Axis = My_Model_Grids.GetAt(i).eAxis.ToString();
        string My_String_Cleanup1 = My_Model_Grid_Axis.Remove(0,5);
        string My_String_Cleanup2 = My_String_Cleanup1.Remove(1);
        Grid_Ordinates.Add(My_Grid_ORD);
        Grid_Name.Add(My_Model_Grid_Names);
        Grid_Axis.Add(My_String_Cleanup2);
    }
    OutPutPorts.Add("Grid_Name", Grid_Name);
    OutPutPorts.Add("Grid_Ordinates", Grid_Ordinates);
    OutPutPorts.Add("Grid_Axis", Grid_Axis);
    //CLOSE
    IDBI.CloseDatabase();
    return OutPutPorts;
}
```

SETS THE OUTPUT PORTS
CREATES NODE/METHOD

GETS IMODEL OBJECT

OPENS THE RAM MODEL
GETS THE MODEL GRID OBJECT
GETS NUMBER OF X AND Y GRIDS

CREATES LISTS FOR OUTPUT

GETS X-Y GRID ORDINATES (FT)
GETS X-Y GRID LABELS

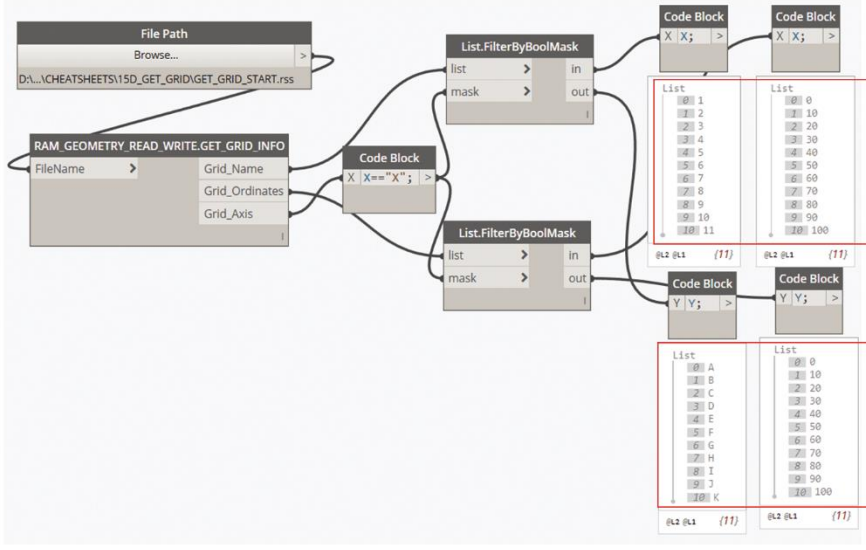
GETS THE AXIS
IF "X" OR "Y" GRID

CREATES LIST FOR OUTPUT

SENDS LISTS TO OUTPUT PORTS

CLOSE RAM MODEL

C# ZERO TOUCH CODE



DYNAMO NODES

X Grid	Y Grid	Label	ft	Min	Max	Label I	Label J
1			0.0000			Y	N
2			10.0000			Y	N
3			20.0000			Y	N
4			30.0000			Y	N
5			40.0000			Y	N
6			50.0000			Y	N
7			60.0000			Y	N
8			70.0000			Y	N
9			80.0000			Y	N
10			90.0000			Y	N
11			100.0000			Y	N

X Grid	Y Grid	Label	ft	Min	Max	Label I	Label J
	A		0.0000			Y	N
	B		10.0000			Y	N
	C		20.0000			Y	N
	D		30.0000			Y	N
	E		40.0000			Y	N
	F		50.0000			Y	N
	G		60.0000			Y	N
	H		70.0000			Y	N
	I		80.0000			Y	N
	J		90.0000			Y	N
	K		100.0000			Y	N

RAM GRID DIALOG BOXES

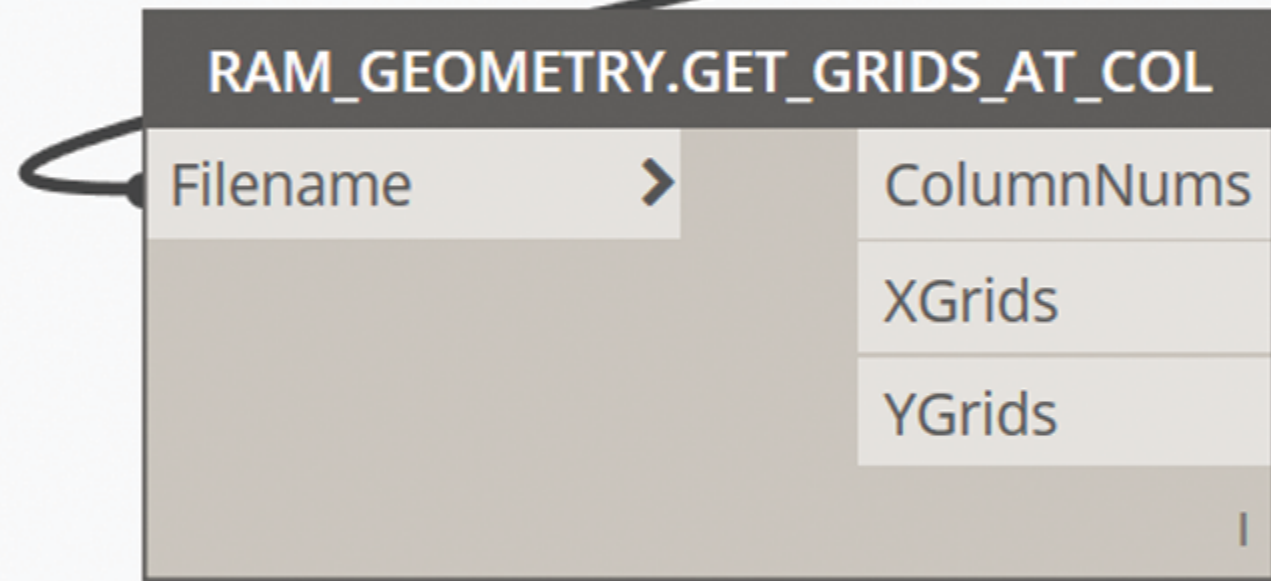
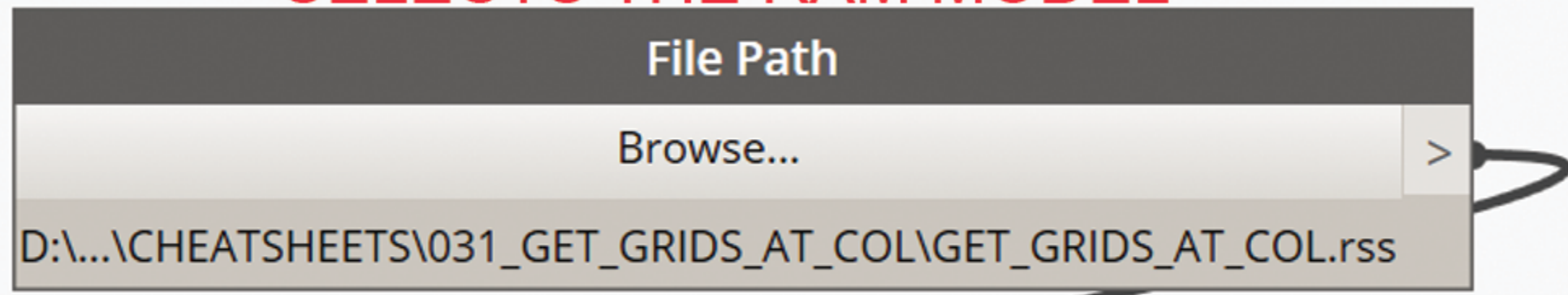
DYNAMO NODES AND RAM MODEL

- STEP 1: OPEN VISUAL STUDIO FOLDER "GET_GRIDS" OPEN SLN FILE
- STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
- STEP 3: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
- NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL

STEPS & NOTES

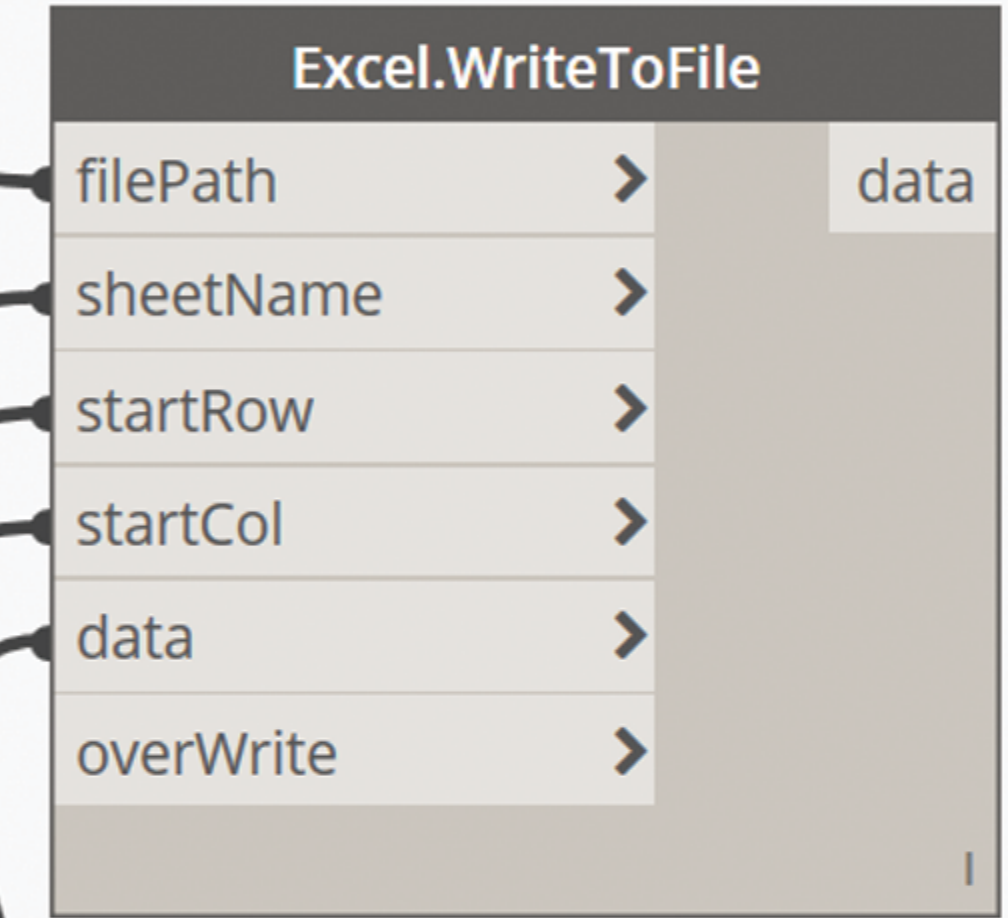
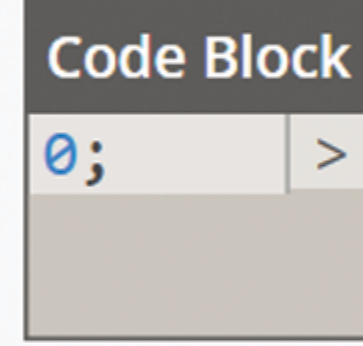
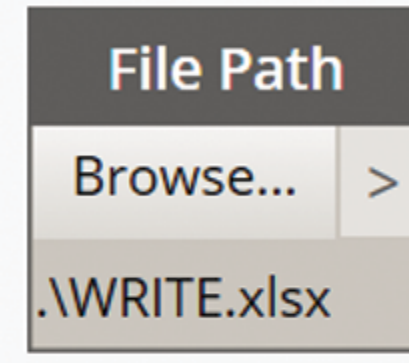
GET GRIDS AT COLUMNS FROM RAM USING DYNAMO AND WRITE TO EXCEL

SELECTS THE RAM MODEL

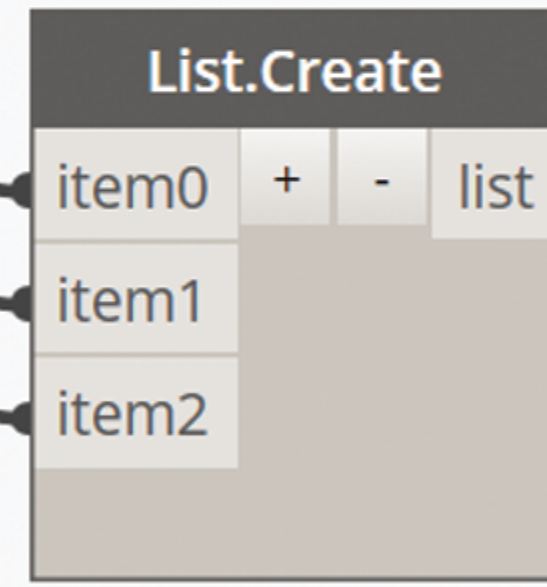


GETS X AND Y GRIDS AT EACH COLUMN

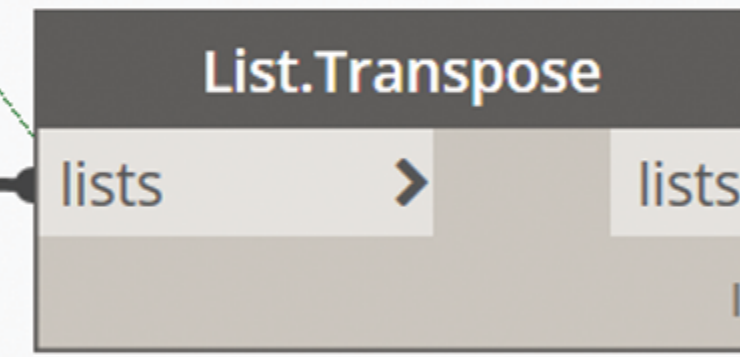
EXCEL FILE TO WRITE TO



WRITES DATA TO EXISTING EXCEL FILE



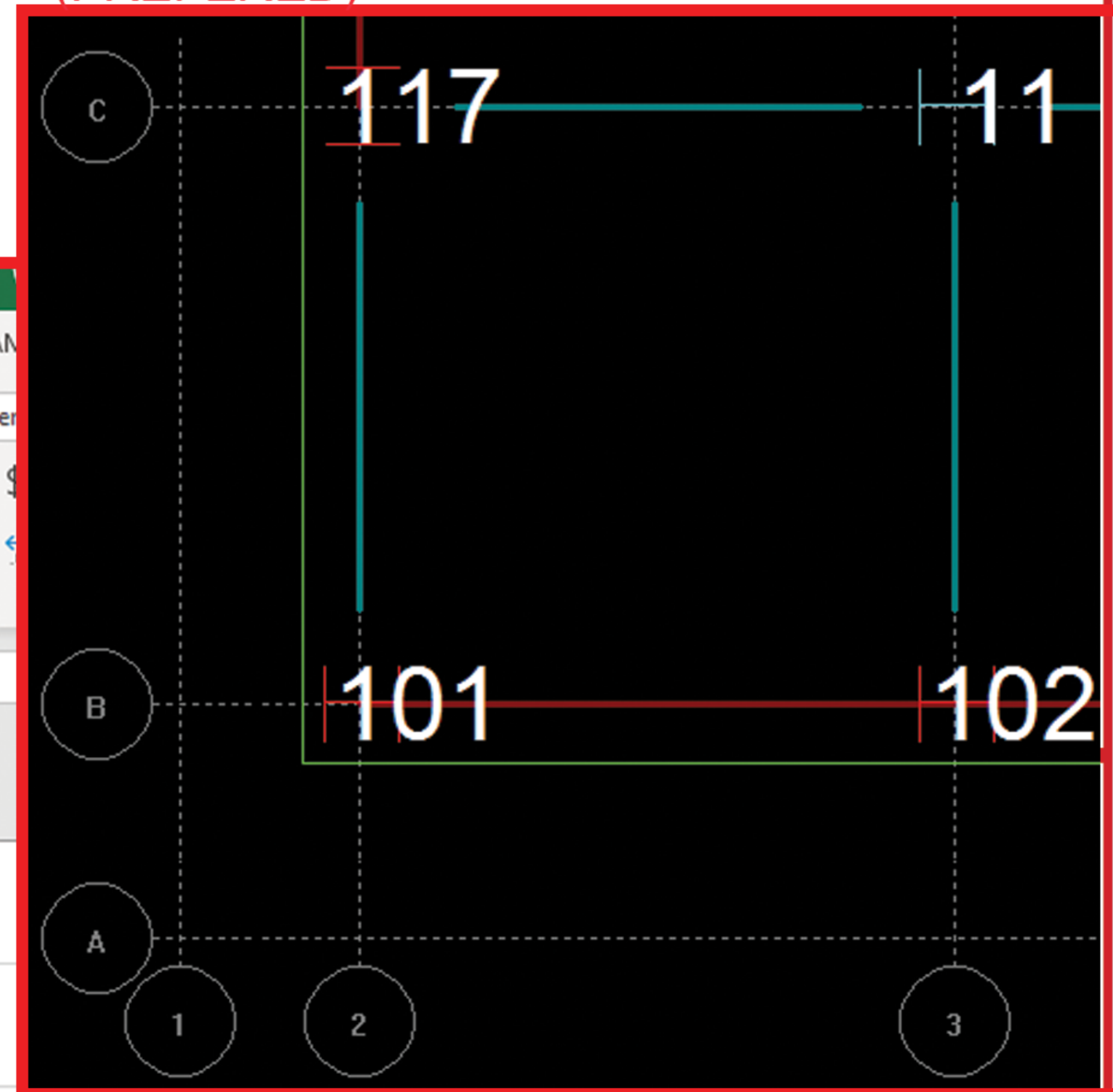
CREATES A NESTED LIST WITH OUTPUT



TRANSPoses LIST TO WRITE TO EXCEL (PREFERRED)

DYNAMO NODES

	A	B	C
11	11	3	C
12	12	3	D
13	13	3	E
14	14	3	F
15	15	3	G
16	16	3	H
17	17	3	I



RAM COLUMN #11 AT GRID (C-3)

EXCEL FILE

- STEP 1: OPEN FOLDER "GET_GRIDS_AT_COL_TO_EXCEL" OPEN SLN FILE, BUILD SOLUTION
- STEP 2: OPEN DYNAMO, LOAD RECENTLY BUILT DLL, PLACE NODES AS SHOWN (MAKE SURE TO HAVE SIMPLEX INSTALLED)
- STEP 4: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
- NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL FOR REFERENCE
- ALTERNATIVELY USE SIMPLEX DYNAMO NODES

STEPS & NOTES

CREATE X Y GRIDS IN RAM USING DYNAMO VIA RAM API AND C#

```
[MultiReturn(new[] { "NewXGrid(ID)", "NewYGrid(ID)", })]
public static Dictionary<string, object> CREATE_GRIDS(string FileName,
string XGridLabel, double XGridCoord, string YGridLabel, double YGridCoord)
{
    RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBI01 IDBI = (RAMDATAACCESSLib.IDBI01)
    RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBI01_INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
    RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
    Dictionary<string, object> OutPutPorts = new Dictionary<string, object>();

    //OPEN
    IDBI.LoadDataBase2(FileName, "1");
    IModelGrids My_Model_Grids = IModel.GetGridSystems().GetAt(0).GetGrids();

    //CONVERT TO FEET BY *12 ON INPUT GRID COORDINATE
    IModelGrid MyXIModelGrid = My_Model_Grids.Add(XGridLabel,
    EGridAxis.eGridXorRadialAxis, XGridCoord*12);
    IModelGrid MyYIModelGrid = My_Model_Grids.Add(YGridLabel,
    EGridAxis.eGridYorCircularAxis, YGridCoord * 12);

    int My_NewXIModelGridID = MyXIModelGrid.LUID;
    int My_NewYIModelGridID = MyYIModelGrid.LUID;

    OutPutPorts.Add("NewXGrid(ID)", My_NewXIModelGridID);
    OutPutPorts.Add("NewYGrid(ID)", My_NewYIModelGridID);

    //CLOSE
    IDBI.SaveDatabase();
    IDBI.CloseDatabase();
    return OutPutPorts;
}
```

SETS THE OUTPUT PORTS
CREATES NODE/METHOD

GETS IMODEL OBJECT

OPENS THE RAM MODEL
GETS THE MODEL GRID OBJECT

CREATES X GRIDS

CREATES Y GRIDS

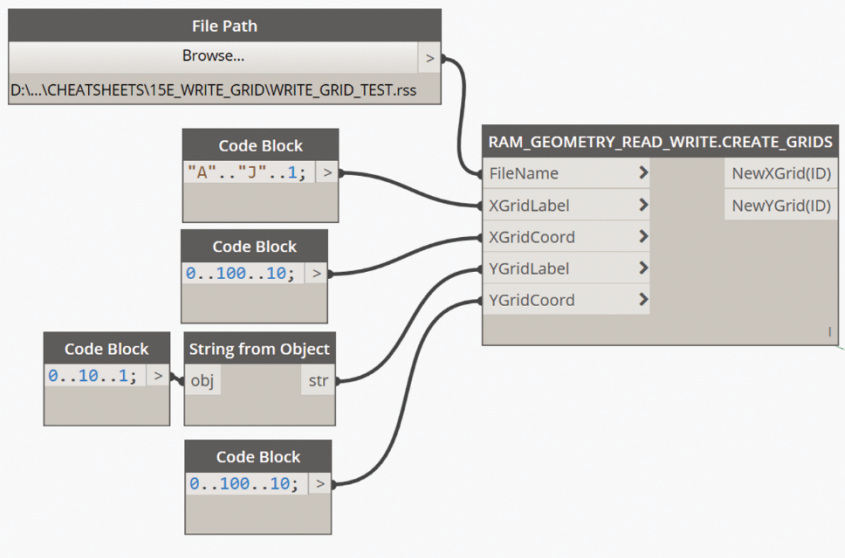
GETS THE ID OF THE NEWLY
CREATED "X" AND "Y" GRID

ADDS TO LISTS FOR OUTPUT

SAVES AND CLOSES RAM MODEL

SENDS LISTS TO OUTPUT PORTS

C# ZERO TOUCH CODE



DYNAMO NODES

X Grid	Y Grid				
Label	ft	Min	Max	Label I	Label J
1	0.0000			Y	N
2	10.0000			Y	N
3	20.0000			Y	N
4	30.0000			Y	N
5	40.0000			Y	N
6	50.0000			Y	N
7	60.0000			Y	N
8	70.0000			Y	N
9	80.0000			Y	N
10	90.0000			Y	N
11	100.0000			Y	N

X Grid	Y Grid				
Label	ft	Min	Max	Label I	Label J
A	0.0000			Y	N
B	10.0000			Y	N
C	20.0000			Y	N
D	30.0000			Y	N
E	40.0000			Y	N
F	50.0000			Y	N
G	60.0000			Y	N
H	70.0000			Y	N
I	80.0000			Y	N
J	90.0000			Y	N
K	100.0000			Y	N

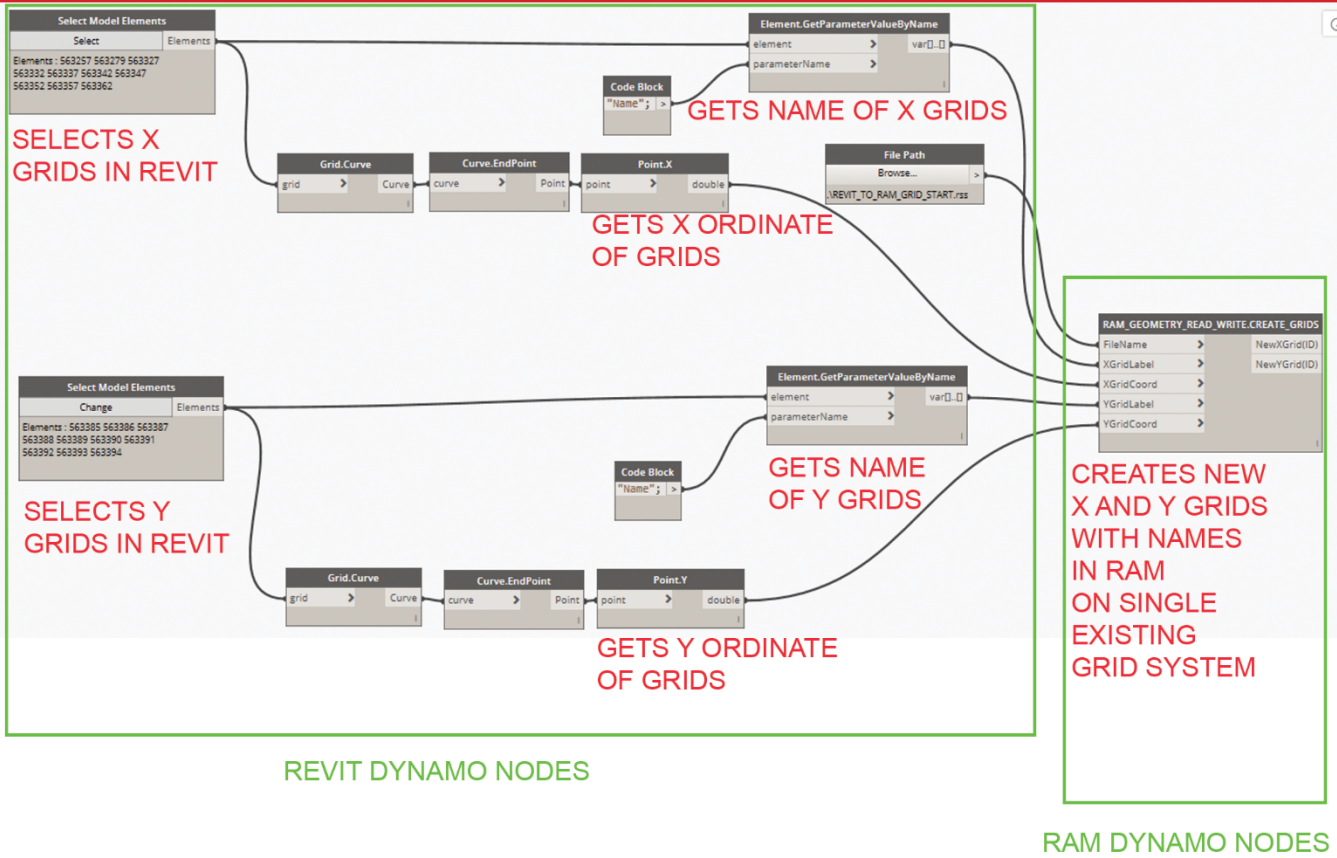
RESULTING
GRID DIALOG BOXES
FROM RAM MODEL

DYNAMO NODES AND RAM MODEL

- STEP 1: OPEN VISUAL STUDIO FOLDER "WRITE_GRIDS" OPEN SLN FILE
 STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
 STEP 3: SELECT THE RAM FILE "WRITE_GRIDS.rss" VIA FILE PATH NODE AND WATCH THE RESULTS
 NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL

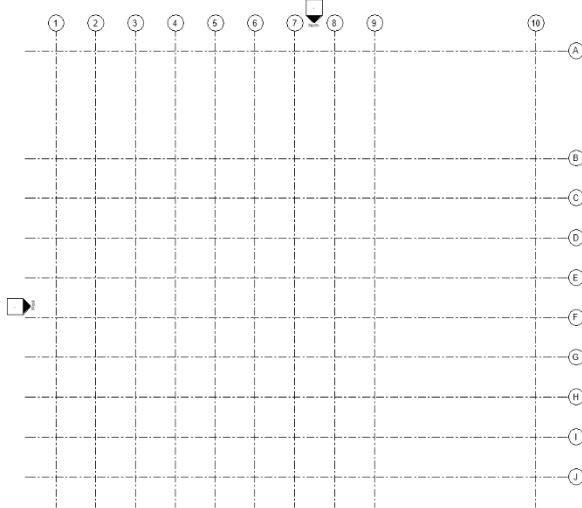
STEPS &
NOTES

CREATE GRIDS IN RAM FROM REVIT! USING DYNAMO VIA RAM API AND C#

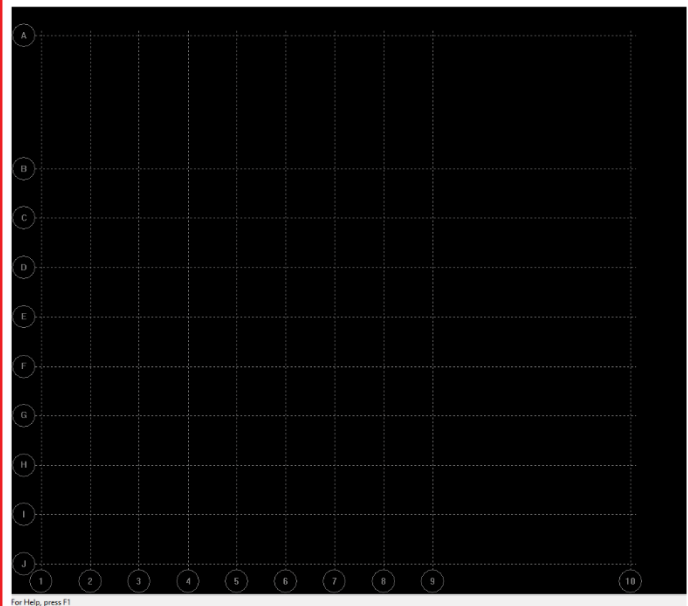


DYNAMO NODES

RAM DYNAMO NODES



REVIT MODEL GRID



RAM MODEL GRID

REVIT MODE AND RAM MODEL

- STEP 1: OPEN VISUAL STUDIO FOLDER "REVIT_TO_RAM_GRID" OPEN SLN FILE
 STEP 2: OPEN REVIT FILE "REVIT_TO_RAM_GRID_START.RVT" AND OPEN DYNAMO FOR REVIT
 STEP 3: LOAD DLL, SELECT THE RAM FILE "REVIT_TO_RAM_GRID_START.rss"
 NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL

STEPS & NOTES

GET FORCES ON GRAVITY COLUMNS FROM RAM USING DYNAMO AND C#

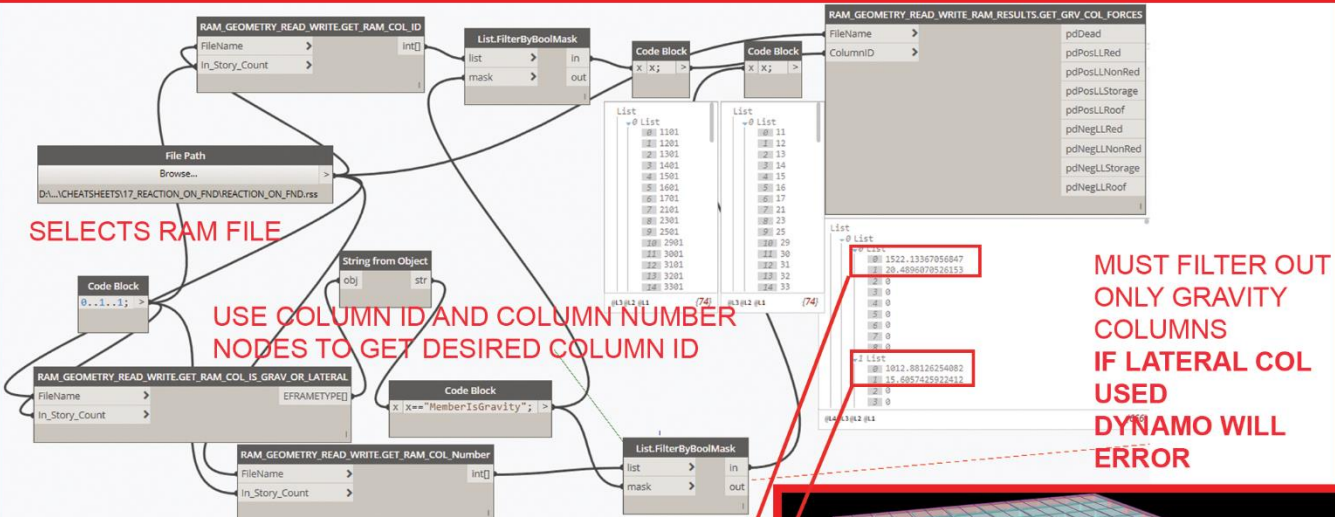
```
[MultiReturn(new[] { "pdDead", "pdPosLLRed", "pdPosLLNonRed", "pdPosLLStorage", "pdPosLLRoof", "pdNegLLRed",
    "pdNegLLNonRed", "pdNegLLStorage", "pdNegLLRoof" })]
public static Dictionary<string, object> GET_GRP_COL_FORCES(string FileName, int ColumnID)
{
    RamDataAccess1 RAMDataAccess = new RAMDATAACCESSLib.RamDataAccess1();
    RAMDATAACCESSLib.IDBI01 IDBI = (RAMDATAACCESSLib.IDBI01)
        RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IDBI01_INT);
    RAMDATAACCESSLib.IModel IModel = (RAMDATAACCESSLib.IModel)
        RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IModel_INT);
    RAMDATAACCESSLib.IForces1 IForces1 = (RAMDATAACCESSLib.IForces1)
        RAMDataAccess.GetInterfacePointerByEnum(EINTERFACES.IForces1_INT);
    Dictionary<string, object> OutPutPorts = new Dictionary<string, object>();
    //OPEN
    IDBI.LoadDataBase2(FileName, "1");
    double pdDead = 0;    double pdPosLLNonRed = 0;
    double pdPosLLRed = 0; double pdPosLLStorage = 0;
    double pdPosLLRoof = 0; double pdNegLLNonRed = 0;
    double pdNegLLRed = 0; double pdNegLLStorage = 0; double pdNegLLRoof = 0;
    IForces1.GetGrvColForcesForLCase(ColumnID, ref pdDead, ref pdPosLLRed,
        ref pdPosLLNonRed, ref pdPosLLStorage, ref pdPosLLRoof, ref pdNegLLRed,
        ref pdNegLLNonRed, ref pdNegLLStorage, ref pdNegLLRoof);
    IDBI.CloseDatabase();
    OutPutPorts.Add("pdDead", pdDead); OutPutPorts.Add("pdPosLLRed", pdPosLLRed);
    OutPutPorts.Add("pdPosLLNonRed", pdPosLLNonRed); OutPutPorts.Add("pdPosLLStorage", pdPosLLStorage);
    OutPutPorts.Add("pdPosLLRoof", pdPosLLRoof); OutPutPorts.Add("pdNegLLRed", pdNegLLRed);
    OutPutPorts.Add("pdNegLLNonRed", pdNegLLNonRed); OutPutPorts.Add("pdNegLLStorage", pdNegLLStorage);
    OutPutPorts.Add("pdNegLLRoof", pdNegLLRoof);
    return OutPutPorts;
}
```

SETS UP MULTIRETURN (OUTPUT PORT RETURN)

GETS IFORCES1 OBJECT FROM RAM

GETS AXIAL FORCES BY GRAVITY COLUMN ID

ZERO TOUCH C# RAM API CODE



DYNAMO NODES AND RAM COLUMN OUTPUT

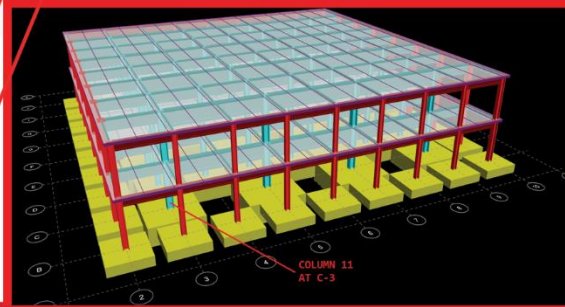
Column Load Summary

RAM Steel 15 08 00 37 Page 2/10

Building Code: IBC Steel Code: AISC 360-10 LRFD

Column Line 3-C	Level	Col#	Length	Dead	Self	+Live	-Live	MinTot	MaxTot
LEVEL1	11	10.00	1518.9	3.3	20.5	0.0	1522.1	1542.6	

Column Line 3-D	Level	Col#	Length	Dead	Self	+Live	-Live	MinTot	MaxTot
LEVEL1	12	10.00	1011.0	1.9	15.6	0.0	1012.9	1028.5	

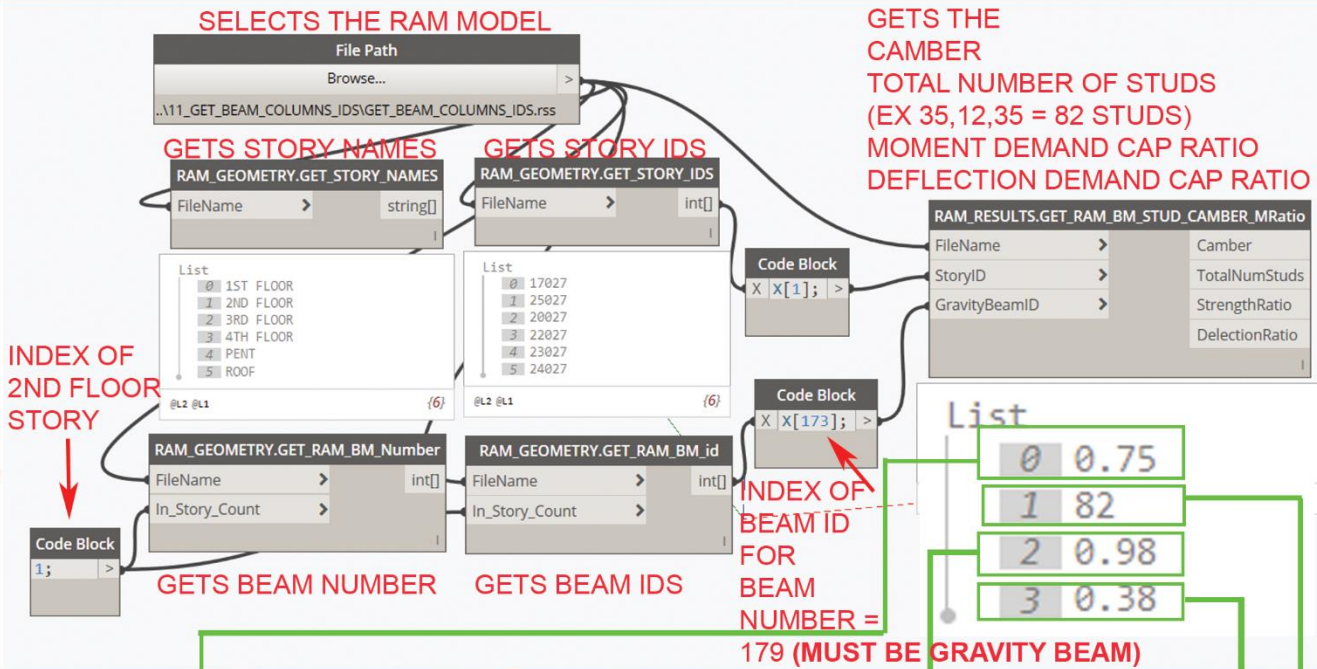


RAM MODEL

- STEP 1: OPEN FOLDER "COLUMN_FORCES" OPEN SLN FILE
- STEP 2: OPEN DYNAMO AND LOAD DLL AND PLACE NODE ADD STORIES AS A LIST
- STEP 3: SELECT THE RAM FILE IN FOLDER VIA FILE PATH NODE AND WATCH THE RESULTS
- NOTE: RAM DOES NOT NEED TO BE OPEN. ALSO SEE SIMPLEX PACKAGE AND RAM API MANUAL ONLY WORKS ON GRAVITY COLUMNS! IF LATERAL COLUMNS ARE USE IT WILL ERROR

STEPS & NOTES

GET BEAM DESIGN INFO: STUDS,CAMBER, ETC USING DYNAMO



DYNAMO NODES

View/Update Beam

Floor Type: 2ND NEW FLOOR2 Beam Number = 179
 Building Code: UBC1 Steel Code: ASD 9th Ed.
 Span information (ft): Length = 33.14 I-End [-159.82,77.43] J-End [-126.82,80.43]
 Decking Orientation: Left = parallel Right = parallel
 User Size = W24X68 MinDepth(in)=11.00

Beam Size: Sx Fy (ksj): 50.00

Beam Size	Sx
W18X65	117.0
W12X65	87.9
W16X67	117.0
W8X67	60.4
W24X68	154.0

I Section

Composite
 Noncomposite

Demand/Capacity Ratio

Strength: 0.98
 Deflection: 0.38

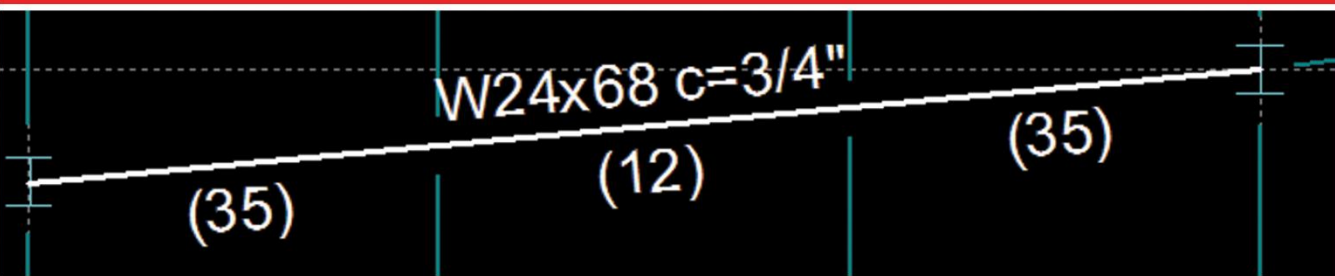
Optimize
 Analyze
 View Results
 View Loads
 View Diagrams
 Update Database
 Cancel
 Help

Camber = 3/4 in

Stud Configuration

	Uniform Spacing	Segmented		
		1	2	3
<input checked="" type="radio"/> Segmented				
<input type="radio"/> Uniform	Full: 152	51	2	51
	Partial: 106	35	12	35
	Actual:	35	+12	+35
		=82		

RAM BEAM #179 DESIGN INFO



RAM BEAM # 179

- STEP 1: OPEN FOLDER "BEAM DESIGN_OUTPUT" OPEN SLN FILE, BUILD SOLUTION
- STEP 2: OPEN ETABS FILE "ETABS_TO_RAM_START.EDB"
- STEP 3: OPEN DYNAMO, LOAD RECENTLY BUILT DLL, PLACE NODES AS SHOWN (MAKE SURE TO HAVE SIMPLEX INSTALLED)
- STEP 4: SELECT THE RAM FILE VIA FILE PATH NODE AND WATCH THE RESULTS
- NOTE: RAM DOES NOT NEED TO BE OPEN. SEE SIMPLEX PACKAGE AND RAM API MANUAL FOR REFERENCE
- COMPARE STORIE NAMES TO GET STORY IDS, SOMETIMES WILL TAKE A MOMENT TO RUN RESULTS

STEPS & NOTES