



Having a Blast with Dynamo and Revit

Michael Kero, Huston Dawson – Integrated Applications Specialist, Associate Principle – Weidlinger Associates

SE5619

Using a combination of the Dynamo extension and the Revit API (as well as in-house software) to analyze a structure subjected to blast loads, we can enable our engineers to visualize how a curtain wall will behave under the prescribed conditions. In this class we will provide an overview of blast analysis and the workflow we have developed within the Revit and Dynamo environment. You will learn how to use the Revit and Dynamo APIs to extract geometric data from a Revit model, parse the data for blast analysis, and then generate views to render the analysis visualizations. Along the way, attendees will learn proper Revit API coding practices, the Revit Analysis Visualization Framework, and the Zero Touch Plugin Development workflow for developing custom Dynamo nodes.

Learning Objectives

At the end of this class, you will be able to:

- Understand Blast Analysis Concepts and Methods
- Understand Extraction of Geometric Data Using the Revit API and Proper API Coding Practices
- Understand the Revit API Analysis Visualization Framework
- Understand the Dynamo Environment
- Understand Zero Touch Development to Extend Dynamo Functionality

About the Speaker

Michael Kero is the Integrated Applications Specialist at Weidlinger Associates. With a B.Arch and Computer Science background, he has worked on computational problems within the AEC Industry since 2008. In his position at WAI, he develops custom Interoperability, Analysis, and Production tool packages for the engineers. He has extensive experience developed tools using the Revit API and Visual Programming interfaces.

Huston Dawson is an Associate Principal at Weidlinger Associates. He is a structural engineer who specializes in extreme loads, which include bomb blast, vehicle impact and flood mitigation. He has worked on over a dozen US Embassy projects, multiple Federal Buildings, and dozens of private sector buildings. Many of these projects were related to the renovation of existing facilities. Mr. Dawson has published a paper on protective design of curtain walls and the installation of bollards in New York City. Mr. Dawson has been involved in developing Finite Element models of custom curtain walls and anti-ram systems and most recently developing blast design tools to integrate into Revit.

Understand Blast Analysis Concepts and Methods

Collateral Damage in Blast Events

Alfred Murrah Federal Building

- 253 Sustained Minor Damage
- 21 Sustained Structural Damage
- 10 Collapsed

World Trade Center

- 37 Sustained Moderate Damage
- 11 Sustained Major Damage
- 3 Sustained Partial Collapse
- 5 Sustained Full Collapse

Areas of Physical Security Vulnerability

- Perimeter Protection
- Interior Non-Secured Space
 - Lobby, Loading Dock, Mailroom
- Mission Critical Spaces
 - Utility, Egress, MEP, Generators, Special Area
- Façade Protection
 - Glazing and Non-Glazing
- Progressive Collapse
 - Perimeter columns

Areas of Design Conflict

- Program and Mission Compliance
- Project Cost \$
- Importance of Security Precautions
- Drawing Attention to Facility
- Open Facility
- Open Public Spaces
- Retrofitting Existing Facilities
- Large Glazing Surfaces
- Architectural Features
- Aesthetics

Security Design Criteria

U.S. Government Facilities

- Interagency Security Committee (ISC)
- “Security Design Criteria” for GSA

- Unified Facilities Criteria (UFC)
- "Minimum Anti-terrorism Standards for Buildings" for the DoD
- Department of State - International Code Supplement to the IBC

Commercial Property

- Currently NO Criteria or Guidelines
- Threat and Risk Analyses (TARA) suggested for each structure

Site Protection

Type I – Special Purpose Facilities (DoS and DoD)

- Campus Site with 100% Vehicle Vetting and local Bollard Protection
- Minimum 100' Standoff to 'Public' Space

Type II – Law Enforcement & Intelligence Community

- 100% Vehicle Vetting and Bollard Protection around Building
- Minimum 50' to 100' Standoff to 'Public' Space

Type III – Standard GSA

- 100% Vehicle Vetting and Bollard Protection around Building
- Minimum 20' to 100' Standoff to 'Public' Space

Type IV – Developer Building/Data Center

- Limited Vehicle Vetting and Bollard Protection
- No Minimum Standoff

Glazing Protection

Type I – Special Purpose Facilities (DoS and DoD)

- Blast and Ballistic Protection
- 2-1/2" glass clad polycarbonate with A36 mullion system

Type II – Law Enforcement & Intelligence Community

- Blast OR Blast and Ballistic Protection
- 2-1/2" glass clad polycarbonate with A36 mullion system

Type III – Standard GSA

- Blast Protection
- 1" to 2" Laminated Inner Lite, Balanced Glazing System

Type IV – Developer/Private Building/Data Center

- ...

Blast Engineering Methodologies

Empirical – 0th Order

- Response Tables, WINDAS
- Pressure & Impulse Diagrams

Single Degree of Freedom (SDOF) – 1st Order

- Structural Response: SBEDS, SDOF, CEDAW, etc.
- Glass Response: WINGARD, HAZL, WINLAC, etc.

Finite Element Method – nth Order

- SAP2000 (implicit)
- ANSYS (implicit)
- LS-DYNA (explicit)
- NLFlex (explicit)

Glazing Systems

Response to Blast Loads

- First Line of Defense
- Weakest Building Element
- High Fragment Hazard

Materials

- Annealed
- Heat Strengthened
- Thermally Tempered Glass
- Polycarbonate

Mitigation Techniques

New Construction

- Laminated Glass

Existing Conditions

- Blast Window Film
- Blast Curtains
- Energy Absorbing Materials
- Muntin Window System

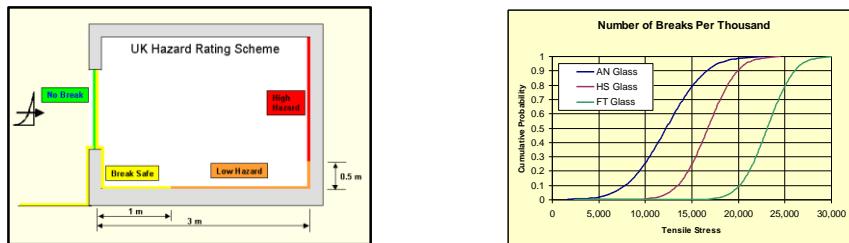
Balanced Window Design

- Glass Fail First
- Match Capacity of Glass to Mullions and Anchorages
- Ensure Post Damage Glazing Attachment

Choice of Glass

Materials and Thickness Selected Using SDOF Hazard Response Software

- WINGARD, HAZL and WINLAC
- Hazard Response Level Specified for the Project



Window Mullion Design

Designed for glazing edge reactions

Mullion & its connections cannot be the “Weak Link”

Glazing attached to frames & mullions with a mechanical bite and/or silicone bead

Performance measured in terms of ductility and rotation

Curtain Walls

The Challenge

- Reduce Architectural Constraints
- Increase the Blast Worthiness for Curtain Walls
- Reduce Load Transfer to Supporting Structural Elements

Capitalize on Flexible Window Systems

Better Understanding Gained Through Computational Simulation and Experimentation

Blast Loads

Blast wave expands spherically from charge

- Strength of the blast weakens with distance

Peak pressure and impulse

- Peak pressure considers intensity
- Impulse considers duration

Intensity of Explosive Threat

- Function of standoff (Z) and charge weight (W)

Blast Load Methodologies

Empirical – 0th Order

- Response Tables

- ConWep
- 3D Blast

Ray Tracing

- BlastX

Finite Element Method – nth Order

- ANSYS
- LS-Dyna
- ABACUS
- Maze
- WALAIR

Blast Loads

Side-on or incident pressure

- Pressure on surface over which the shock wave passes unimpeded (ground surface, flat roof, side wall)

Reflected pressure

- Pressure on surfaces obstructing the shock wave (wall facing explosion) may be significantly greater than incident pressures

Urban Blast Tool (UBT)

- Fast running tool developed for NYC Financial District and Mid Manhattan
- Geared toward the design community and first responders
- Provides guidance on Air blast loads based on CFD analysis
- Addresses column damage and potential for progressive collapse
- Displays glass debris hazards
- Use for evaluating emergency evacuation rescue and recovery (EERR) systems after an event
- Based on extensive R&D on Blast and its effects

Understand Extraction of Geometric Data Using the Revit API and Proper API Coding Practices

API Object Wrapping

The benefits of wrapping the Revit API is increased readability and maintenance of the code base. It is common in the AEC coding community to write code in a linear, scripting style which creates commands containing many hundreds of lines of code. By wrapping the API and separating the code into objects, we can create a more readable and maintainable code base. Below is the primary code in the command. We have clearly delineated where we collect our Revit elements, where the primary analysis is computed, and where the analysis results are visualized.

Code Block 1 - Primary Executed Code from Main Command

```
//Initialize element collectors
RevitCurtainPanelCollector panelColl = new RevitCurtainPanelCollector(doc);
RevitColumnCollector colColl = new RevitColumnCollector(doc);
RevitThreatCollector threatColl = new RevitThreatCollector(doc);
//Collect elements
List<RevitCurtainPanel> panels = panelColl.GetAllPanels();
List<RevitColumn> interiorCols = colColl.GetSelectedColumns();
RevitThreat threat = threatColl.GetSelectedThreats()[0];
//Compute Blast Loads
BlastAnalyser.ComputeAnalysis(panels, threat, interiorCols);
//Render Analysis
RevitView3D view = new RevitView3D(doc.UIDocument.ActiveView as View3D);
view.LoadPressureValues(panels, 1, "Test", 0);
view.CreateAndSetAnalysisDisplayStyle();
```

Extracting Geometric Data from Panels for Analysis

A key line in Code Block 1 is the `BlastAnalyser.ComputeAnalysis(...)`. It is in this black box the magic happens. See Code Block 2 for the method.

Code Block 2 - Compute Analysis

```
public static void ComputeAnalysis(List<RevitCurtainPanel> panels, RevitThreat threat,
List<RevitColumn> interiorCols) {
    foreach (RevitCurtainPanel panel in panels) {
        XYZ interiorPt = panel.GetClosestColumnPointToPanelVolumeCenter(interiorCols);
        panel.InitializeBlastAnalysisValues(interiorPt);
        BlastAnalysisValues interiorValues = panel.InteriorBlastValues;
        BoundingBoxUV bb = interiorValues.AnalysisFace.GetBoundingBox();
        UV midParam = (bb.Max + bb.Min) / 2.0;
        XYZ facePt = interiorValues.AnalysisFace.Evaluate(midParam);
        double pressure = ComputePressure(threat, facePt);
        double impulse = ComputeImpulse(threat, facePt);
        interiorValues.Add(midParam, new List<double>() { pressure, impulse });
    }
}
```

For each panel we need to reliably and consistently store the location at which our analysis values apply and the analysis values themselves. To do this, I use a Dictionary which stores the UV parameter as the key and a list of doubles as the value which is shown in Code Block 3. I maintain two dictionaries for each panel, one for the interior face and one for the exterior.

Code Block 3 – Analysis Value Dictionary Container

```

namespace RevitAnalysisSample.Analysis {

    public class BlastAnalysisValues : IDictionary<UV, List<double>> {

        Face face;
        Dictionary<UV, List<double>> paramValues;

        public BlastAnalysisValues(Face face) {
            this.face = face;
            this.paramValues = new Dictionary<UV, List<double>>();
        }

        public Face AnalysisFace {
            get { return this.face; }
        }

        public ICollection<UV> Keys {
            get { return this.paramValues.Keys; }
        }

        public ICollection<List<double>> Values {
            get { return this.paramValues.Values; }
        }

        .

        .

        public void Add(UV key, List<double> value) {
            this.paramValues.Add(key, value);
        }

        public bool ContainsKey(UV key) {
            return this.paramValues.ContainsKey(key);
        }

        .

        .

    }
}

```

To determine which face is internal or external, we need an interior point. The parsing of the interior columns to find the closest interior column point is a task taken care of by the panel object. Once the interior point is identified, we parse through each face of the panel and identify the interior and exterior faces as illustrated in Code Block 4.

Code Block 4 – Get Panel Faces

```

public Face PanelFace(XYZ interiorPt, PanelSide side) {
    List<Face> panelFaces = RevitElementPrimaryGeometryFaces();
    Face f1, f2;
    GeometryUtil.GetTwoFacesWithLargestArea(panelFaces, out f1, out f2);
    BoundingBoxUV f1BoundBox = f1.GetBoundingBox();
    BoundingBoxUV f2BoundBox = f2.GetBoundingBox();
    XYZ f1Cent = f1.Evaluate((f1BoundBox.Max + f1BoundBox.Min) / 2.0);
    XYZ f2Cent = f2.Evaluate((f2BoundBox.Max + f2BoundBox.Min) / 2.0);
    if (PanelSide.Exterior == side) {
        return (GeometryUtil.DistanceXY(f1Cent, interiorPt) >
            GeometryUtil.DistanceXY(f2Cent, interiorPt)) ?
            f2 : f1;
    } else {
        return (GeometryUtil.DistanceXY(f1Cent, interiorPt) >
            GeometryUtil.DistanceXY(f2Cent, interiorPt)) ?
            f1 : f2;
    }
}
public List<Face> RevitElementPrimaryGeometryFaces() {
    List<Face> faces = new List<Face>();
    foreach (Solid sol in RevitElementPrimaryGeometrySolids()) {
        faces.AddRange(FacesOfSolid(sol));
    }
    return faces;
}
public List<Solid> RevitElementPrimaryGeometrySolids() {
    List<Solid> solids = new List<Solid>();
    foreach (GeometryObject go in this.revitelement.get_Geometry(new Options())) {
        Solid solid = go as Solid;
        GeometryInstance geomInst = go as GeometryInstance;
        if (null != solid && 0 < solid.Faces.Size) {
            solids.Add(solid);
        } else if (null != geomInst) {
            List<Solid> geomInstSolid = RevitInstanceGeometrySolids(geomInst);
            if (0 < geomInstSolid.Count) {
                solids.AddRange(geomInstSolid);
            }
        }
    }
    return solids;
}
List<Solid> RevitInstanceGeometrySolids(GeometryInstance geomInst) {
    List<Solid> solids = new List<Solid>();
    foreach (GeometryObject symGo in geomInst.GetInstanceGeometry()) {
        Solid instSolid = symGo as Solid;
        if (null != instSolid && 0 < instSolid.Faces.Size) {
            solids.Add(instSolid);
        }
    }
    return solids;
}

```

Notice that after we get the faces of the panel form, we still need to determine which of the six faces of the panel solid could be interior or exterior. We simply use the two faces with the largest area cover. This method is not fool proof. Clearly, we can have panels where the faces with largest areas are the edge faces but these are isolated cases and can be planned for. Now that we have identified the faces of the panel, we compute the pressures and impulses and store the values along with the face midpoint the UV parameter. We will use these values for the visualization

Understand the Revit API Analysis Visualization Framework

Analysis Visualization Basics

The basic process to visualize any set of data onto a group of faces is as follows:

1. Get Spatial Field Manager of view
2. Register new Analysis Result Schema
3. Add each face as a unique Spatial Field Primitive to the schema
4. Set view Analysis Display Style

What is important to notice is that the faces are individually added to the same schema. Each schema represents an analysis result set. In our result set, we have the panels, their specified faces and the values associated with the midpoints of those faces.

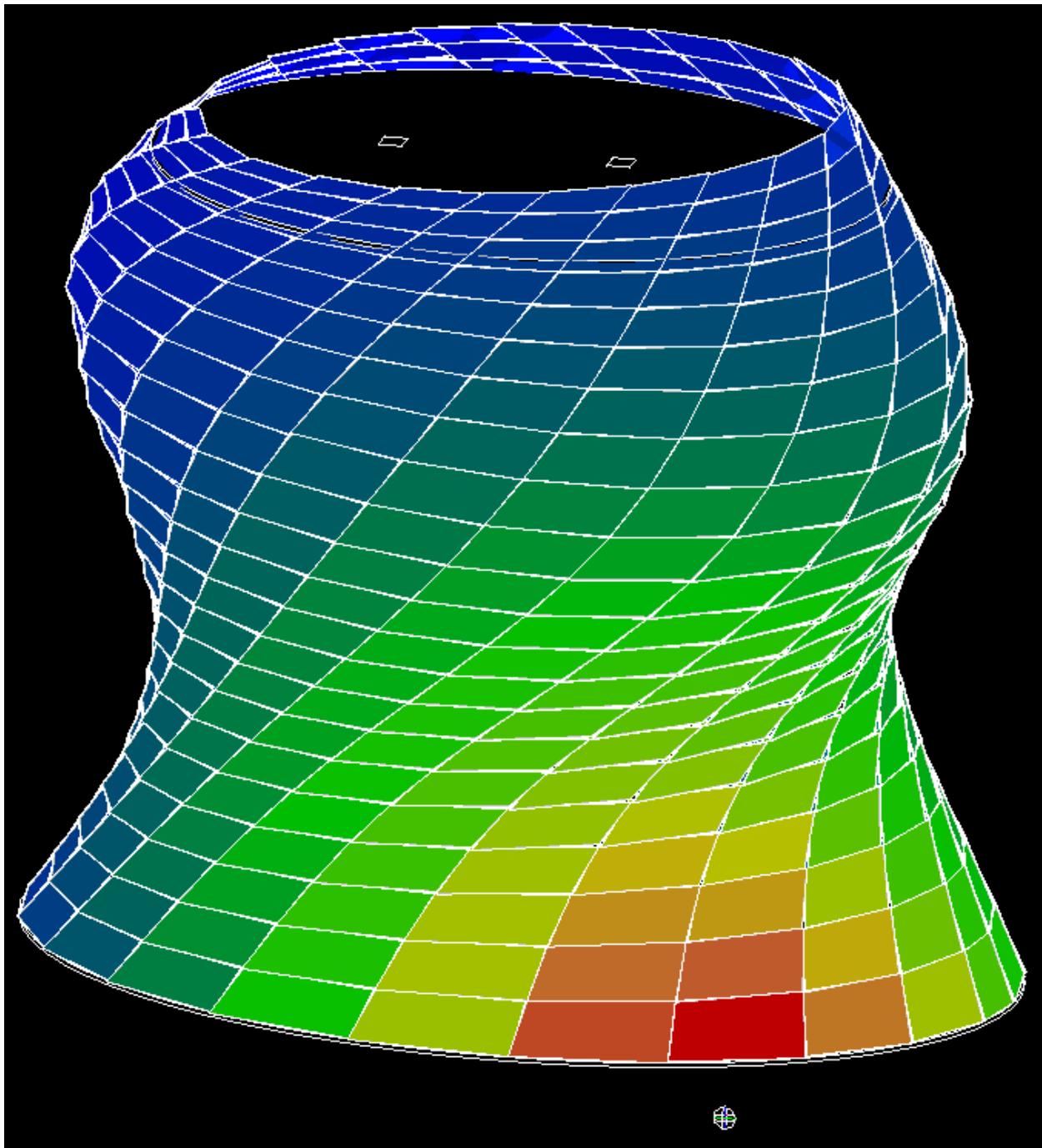
Visualizing Our Results

To visualize the results we just calculated, we will follow the steps prescribe above. We will first get a handle on the active view with our Revit View object. It is within this object we load the result data and then set the Analysis Display Style. Code Block 5 demonstrates how we use our dictionary structure for each panel to populate the result schema.

Code Block 5 – Create the Analysis Result Schema

```
public void LoadPressureValues(List<RevitCurtainPanel> panels, int threatCt, string optionName,
int valueIdx) {
    SpatialFieldManager sfm = GetSpatialFieldManger(1);
    string schemaName = String.Format("{0} - Threat {1} Schema", optionName, threatCt);
    AnalysisResultSchema resultSchema = new AnalysisResultSchema(schemaName, "Description");
    int schemaIdx = sfm.RegisterResult(resultSchema);
    foreach (RevitCurtainPanel panel in panels) {
        BlastAnalysisValues analysisVal = panel.InteriorBlastValues;
        if (null != analysisVal) {
            int primIdx = sfm.AddSpatialFieldPrimitive(analysisVal.AnalysisFace,
                Transform.Identity);
            List<UV> paramUVs = new List<UV>();
            List<ValueAtPoint> paramUVVals = new List<ValueAtPoint>();
            foreach (KeyValuePair<UV, List<double>> val in analysisVal) {
                paramUVs.Add(val.Key);
                paramUVVals.Add(
                    new ValueAtPoint(
                        new List<double>(){ val.Value[valueIdx] }));
            }
            FieldDomainPointsByUV pnts = new FieldDomainPointsByUV(paramUVs);
            FieldValues vals = new FieldValues(paramUVVals);
            sfm.UpdateSpatialFieldPrimitive(primIdx, pnts, vals, schemaIdx);
        }
    }
}
```

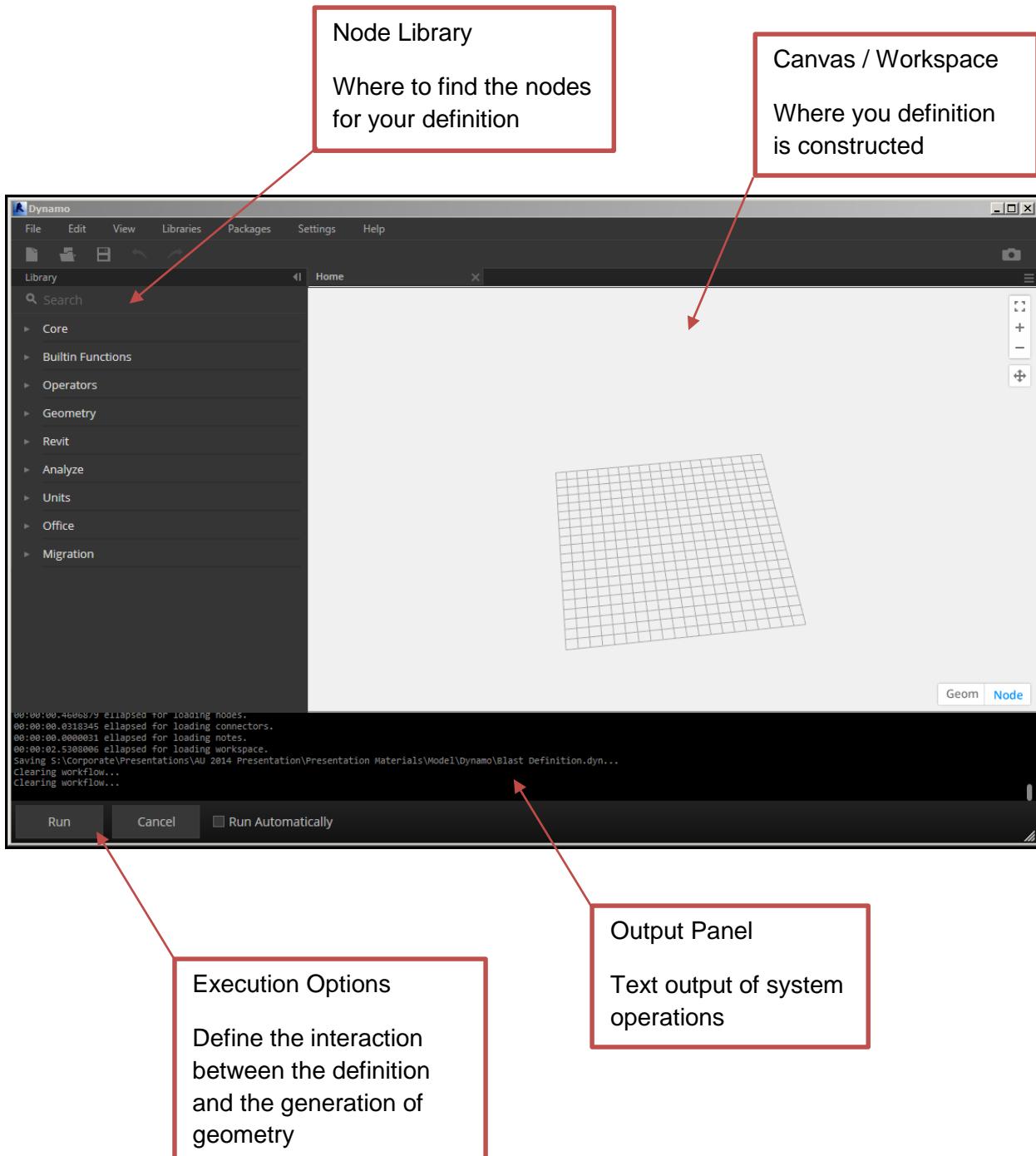
Sample Output

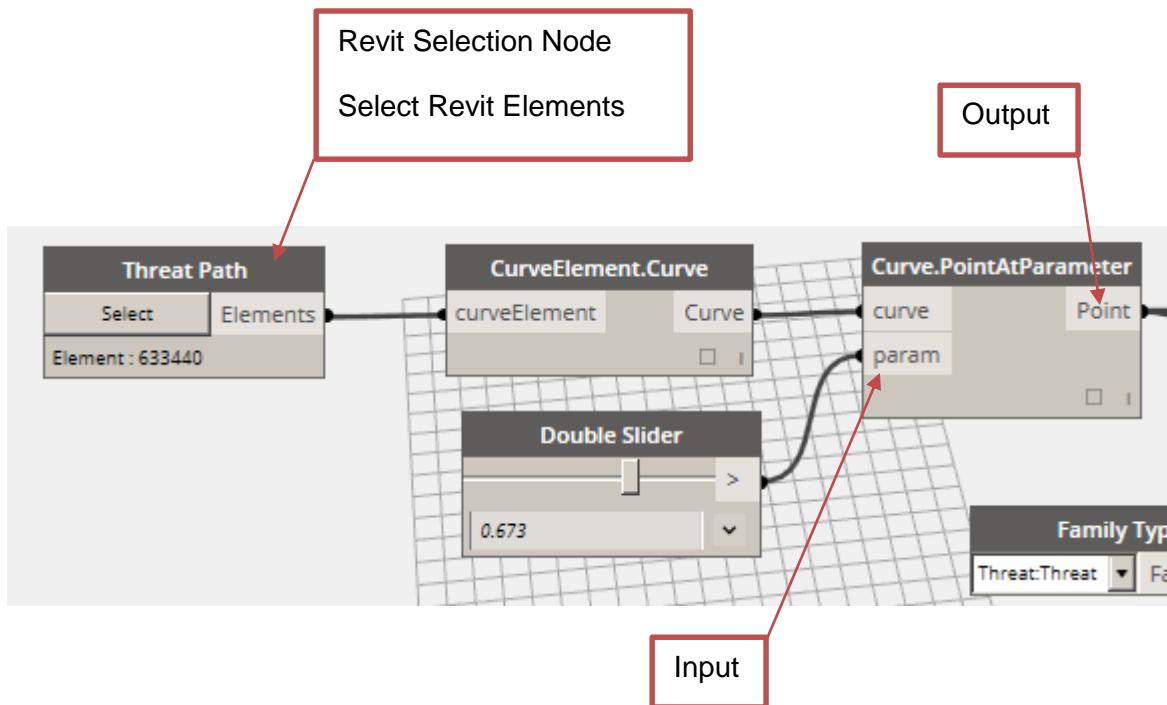


Understand the Dynamo Environment

The Interface Anatomy

The Whole Interface



The Definition

Understand Zero Touch Development to Extend Dynamo Functionality

To extend the functionality of Dynamo is an easy and intuitive process. If we want to work with the elements in a more finely grained specification, we will use the Zero Touch Plugin Development (ZTD) workflow. With this model of extension we can easily port any library of functions into our Dynamo environment

References

There is no primary dll with which you will interface with dynamo, ZTD will accept any .NET dll and make the objects contained available for use to the user. However if we want to work with certain objects defined in Dynamo, we will need to reference the ProtoInterface.dll and the appropriate dll for the objects we need. All needed dlls can be found in C:\Program Files\Autodesk\Dynamo 0.7 or one of its child folders. If we want to work with Dynamo Geometry, we will need to reference ProtoGeometry.dll; to work with Revit Nodes, we will need to reference RevitNodes.dll from the Revit_2015 directory. If we are working with Revit Nodes and want to work with them using the Revit API, we will need to import that dll as well which can be found in C:\Program Files\Autodesk\Revit 2015.

The Code

In order to extend Dynamo with ZTD, we need code. For our Blast example, we need to break down the problem into Node-sized chunks. We look to how we have done this in our previous solution above. We first isolate the faces which we want to analyze. We then calculate the loads. Finally, we render the analysis onto the isolated faces. Code Block 6 highlights the primary process to get the interior and exterior faces of the panels and the other code is available in the additional materials.

Code Block 6 – Isolating Faces

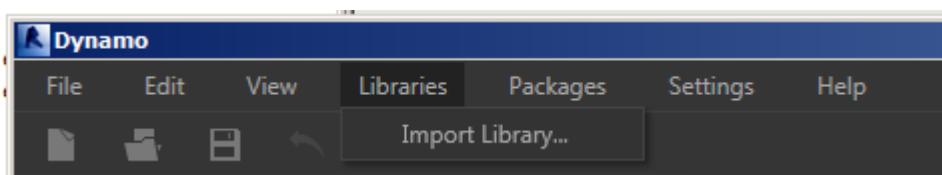
```
[MultiReturn(new[] { "Exterior Faces", "Interior Faces" })]
public static Dictionary<string, object> GetRevitFacesOfPanel(List<CurtainPanel> panels,
    DyElement interiorCol) {
    List<RFace> interiorRevitFaces = new List<RFace>();
    List<RFace> exteriorRevitFaces = new List<RFace>();
    XYZ interiorRevPt = GetRevitPointFromColumn(interiorCol);
    GetRevitFaces(panels, interiorRevPt, ref interiorRevitFaces, ref exteriorRevitFaces);
    return new Dictionary<string, object> {
        { "Exterior Faces", exteriorRevitFaces },
        { "Interior Faces", interiorRevitFaces }
    };
}

static void GetRevitFaces(List<CurtainPanel> panels, XYZ interiorPt, ref List<RFace>
    interiorRevitFaces, ref List<RFace> exteriorRevitFaces) {
    foreach (CurtainPanel el in panels) {
        RElement e = el.InternalElement;
        if (e is Panel) {
            List<RFace> panelFaces = RevitElementPrimaryGeometryFaces(e);
            RFace f1, f2;
            GetTwoFacesWithLargestArea(panelFaces, out f1, out f2);
            BoundingBoxUV f1BoundBox = f1.GetBoundingBox();
            BoundingBoxUV f2BoundBox = f2.GetBoundingBox();
            XYZ f1Cent = f1.Evaluate((f1BoundBox.Max + f1BoundBox.Min) / 2.0);
            XYZ f2Cent = f2.Evaluate((f2BoundBox.Max + f2BoundBox.Min) / 2.0);
            if (f1Cent.DistanceTo(interiorPt) > f2Cent.DistanceTo(interiorPt)) {
                exteriorRevitFaces.Add(f1);
                interiorRevitFaces.Add(f2);
            } else {
                exteriorRevitFaces.Add(f2);
                interiorRevitFaces.Add(f1);
            }
        }
    }
}
```

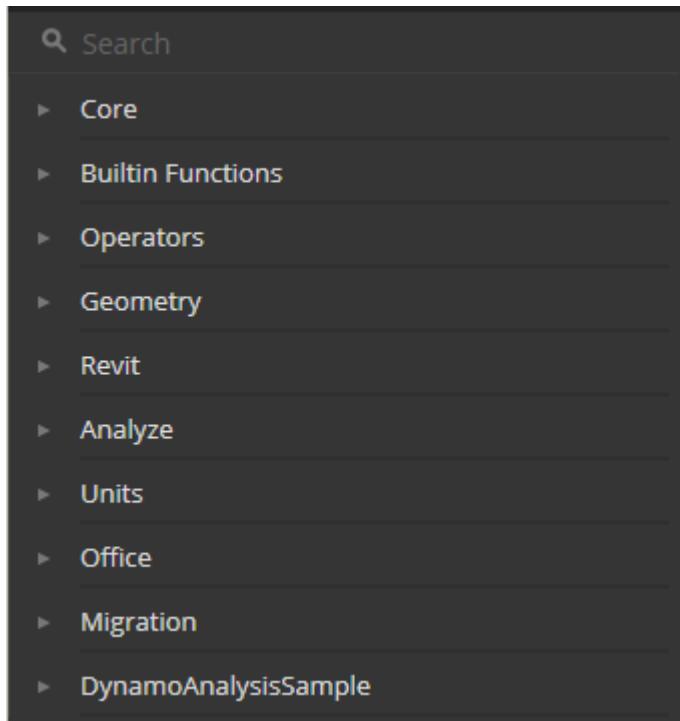
There are a few features about the above code which I would like to highlight. First is the addition of multiple outputs of a node. This feature is implemented with a Dictionary with takes associates an object with a string. The string is the name of the output node and the object is the returned value. The next feature is that we are using Revit API operations. What enables us to do this and basically just reuse our code from before is the property InternalElement which all Dynamo Revit Elements have. This property returns the Autodesk.Revit.DB object and, since we have the Revit API referenced, we can work with the elements

Load the Code

To load the code, simple find “Import Library” in the dynamo interface and then locate the compiled dll with your code.

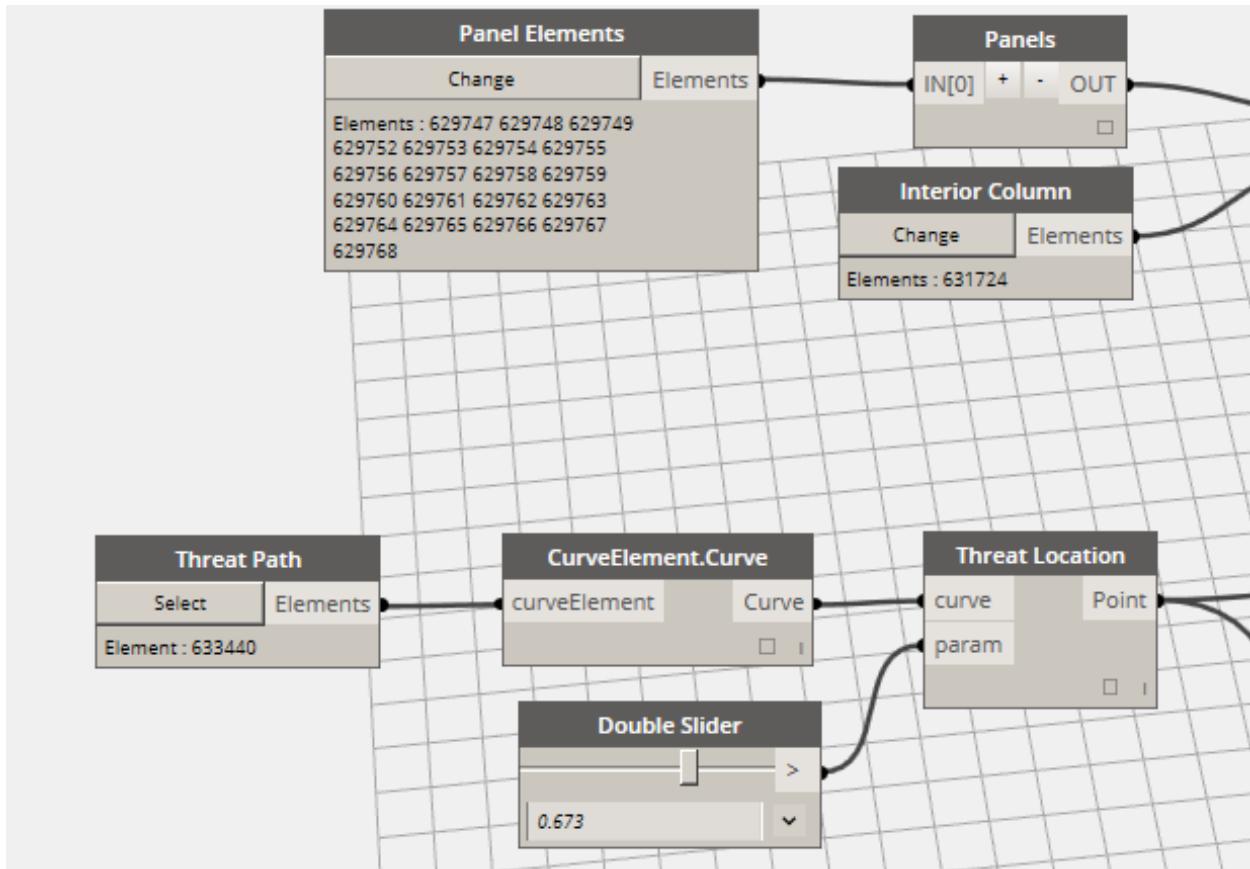


You will see that in the Node library a new category has been added with your nodes available to you. This

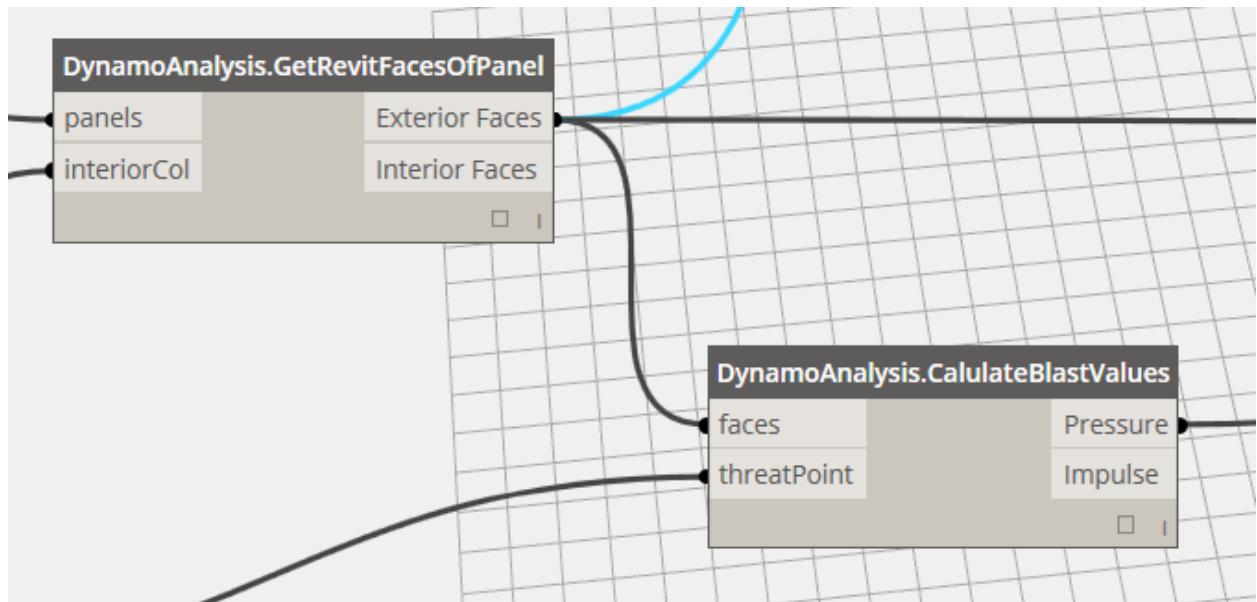


Add the Nodes

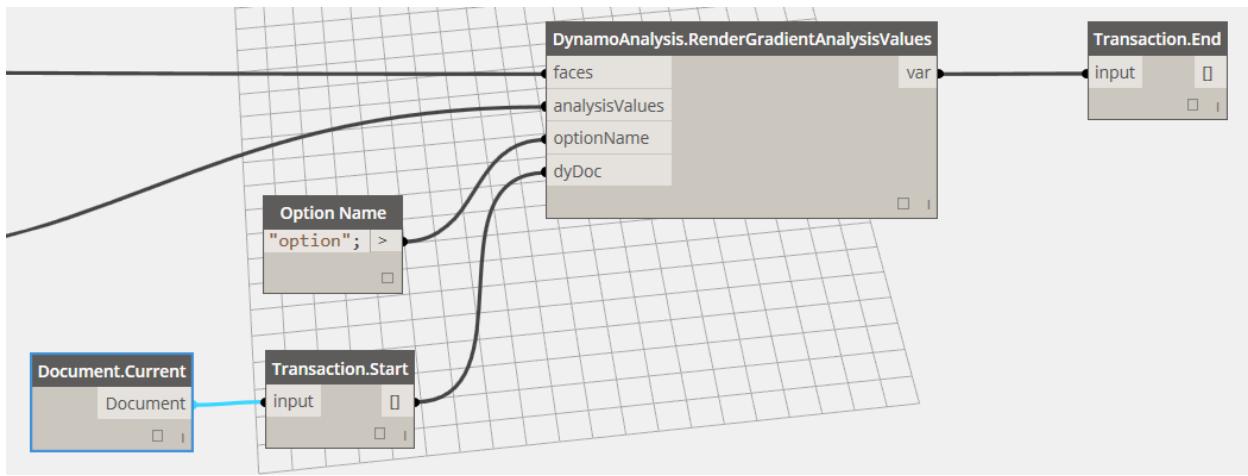
Now that we have our code loaded, we will build our definition. We need to select our panels, and interior column and a path which we will evaluate to get our threat point.



Next we need to add our custom nodes to build out the blast analysis.



We finally place our Analysis Visualization Node.



What is important to note in our analysis visualization is how transactions are handled. In a typical Revit API command, we handle our Transactions within the code. This is not the case for a ZTD custom node. Each time the definition is run, it will spin up a transaction for us, we just need to be sure to end it.

The Results

