Introduction to Generative Components

A first look for non-users
Bio

My career has included the roles of Draughtsman, Visualiser, Trainer, Cad Manager, IT Consultant, beginning over 25 years ago, helping the AEC industry. The last 14 years have been with Bentley supporting users across EMEA in their technology journeys, supporting their projects and business aspirations.
When to Use GenerativeComponents

• When standard CAD tools are not delivering capabilities necessary to produce the desired results and the design and creation of design tools is needed.
• When changes are anticipated within a predictable range that can be captured in a parametric model.
• When exploration of variants is required to find a design solution.
• When simulation of motion, kinetic, or dynamic behaviour is part of the design exploration with a parametric system being able to capture the range of motion.
• When resolution of geometry can be described and processed in an algorithmic fashion.
• When automating repetitive or tedious workflows.

• + 1 When any combination of the above conditions occurs.
Agenda

• The interface
• User Generated Node Types
• A basic introduction to scripting inside GenerativeComponents
The Interface
The Interface
The Interface

The Coordinate System is a Point, has Planes, has Directions
The Interface

A Point has ~40 different Placement Options

X/Y/Z Values free

X Value fixed. X-Handle disabled

No Handles. Point is constraint by „Distance along Curve“
The Interface

Nodes have Inputs (blue optional) and Outputs. Outputs can control Inputs from other Nodes.
The Interface

Single Value

List of Values

Value via Function
Series(start,end,increment)
The Interface

Unwind / Rewind
Supress / Unsupress
Split / Consolidate
Rebuild
Cut / Delete / Copy
Insert Script

Edit Entire transaction list
Introduction to GenerativeComponents
User Generated Node Type
Generated Node Types (GNT)

A major aspect of GenerativeComponents is the ability to expand the set of existing nodes with user-defined nodes.

- GNT’s should be modular and re-usable
- GNT’s can easily be shared between different files and between different systems supporting collaboration.
- The inputs define how a new node fits into a defined "context" (of existing nodes) and the outputs define the context that this node provides for other "successor" nodes.
Exercise

Create a New File: GC_Crossbar.dgn

Add a baseCS

Set the Active Plane to baseCS.XZPlane

Select the Point icon

a. Place 4 points in the Isometric view to form a rectangle. Exact placement is not important, but they should roughly represent the corners of one curtainwall panel. (3000mm x 4000 mm) The order of the points will matter, so place the points in a clockwise order, as if you were drawing the rectangle, starting at the lower left corner.
Exercise

Select the Polygon node from the Node Types tab.
Change the Technique to ByVertices.
Link the first input, VertexPoints, to the point1, point2, point3 and point4.
Select the 4 point nodes, right-click and toggle on Is Construction, right-click again and toggle off Constructions Visible.
All remaining geometry must be created from the polygon1 node and not the baseCS or point1, point2, point3 or point4.
Select the polygon01 node, right-click and toggle on Is Construction.
Exercise

Select the point node from the Geometry Node Types and drag and drop it onto the Graph view.

Rename the node pointCenter.

Set the Technique to CentroidOfSet.

SetToFindCentroidFrom: polygon1.Vertices

Select the line node from the Geometry Node Types and drag and drop it onto the Graph view.

Set the Technique to ByPoints.

StartPoint: polygon1.Vertices

EndPoint: pointCenter
Exercise

Place a **Solid node** on the **graph**.

**Technique:** Cone

Change the name to **Crossbar**.

**Line:** line1

**StartRadius:** 200 mm

**EndRadius:** 100 mm

**Part:** {'CurtainWall','matl_CurtainWall_Frame'}
**Exercise**

Place a Slider node on the graph.
Change the name to **StartDiameter**.
Value: 200 mm  
Minimum: 0  
Maximum: 500 mm  
Resolution: 50 mm  
Snap To Ticks: True  
Tick Interval: 50 mm  
Ticks Visible: True  

Right-click on the **StartDiameter** node and select **Copy**. Right-click on the Graph and select **Paste**.  
Click in the name field on the node and change the name to **EndDiameter**  
Value: 300 mm  

Link the **StartDiameter** to the **StartRadius** input on the **Crossbar** node and add /2.  
Link the **EndDiameter** to the **EndRadius** input on the **Crossbar** node and add /2.
Exercise

Place a bsplineSurface node on the graph.

Technique: Ruled.

Change the name to surfaceGlass.

StartCurve: line1

EndCurve: line1.Rotate(1)

Part: {'CurtainWall','matl_CurtainWall_Glazing'}
Exercise

In the Generate Node Type dialog set the Type name to Crossbar.
Under the Input Properties select Select from GC Model
Select the polygon1 node from the graph.
Exercise

Green = Outputs
Blue = Inputs
Exercise

The dialog will be populated with this node as the **Input Properties** and all other nodes downstream of the **polygon01** node as **Output Properties**.

Under **Input Properties** the **Property Name** is how the input will display on the node.

**Change the Property Name** for **polygon01** to **Polygon**.

**Turning on Replication** for the input indicates that when the node is placed it will accept multiple polygons.

**Turn On Replicatable** for **polygon1**.

**Turn On Optional** for **StartDiameter** and **EndDiameter** and set the **Initial Value** for these inputs.

**Turn On Construction** for **pointCenter** and **line1**. **Turning on Construction** on the **Output Properties** indicates that those outputs will not be visible when the node is placed, the only geometry actually created will be the crossbars and the glass panels.

**Turn On** the setting to **Lock-in each constituent node’s current color, level and symbology**.

**Turn On** the option to **Store the resultant GNT in your GC environment**.
Exercise
Exercise
Exercise

Create a New File: GC_Wall.dgn
Add a baseCS
Set the Active Plane to baseCS.XYPlane
Select the Point icon
Place 4 points in the Isometric view to form a rough line, they do not need to be in perfect alignment but they should cover a distance greater than 20 meters.
Exercise

Add a BSplineCurve using the Technique ByPoles

Connect in order point1, point2, point3 and point4. to the poles input on the new BSplineCurve
Exercise

Add a `CoordinateSystem` using the Technique `ByCartesianCoordinates`.
Change the `ZTranslation` to a value of 30000mm

Add a `BSplineCurve` using the Technique `CopyTransformGeometricContents`
`NodeToCopy`: `bsplineCurve1`
`CopyFrom`: `baseCS`
`CopyTo`: `coordinateSystem1`
Exercise

Add a `bsplineSurface` using the Technique `LoftCurves`.
Set `Curves` input to `bsplineCurve1, bsplineCurve2`,
Set `ON Is Construction`  
Set `OFF Visible`

Add a point using the Technique `ByUVParametersOnSurface`  
Surface: `bsplineSurface1` 
U: 10  
V: 10  
Set `ON Is Construction`  
Set `OFF Visible`
Exercise

Add a polygon using the Technique points.
Set Points input to point5
Set ON Is Construction
Set OFF Visible
Exercise

Add a **crossbar node** using the Technique **polygon**.
Set **Points** input to **point5**
Set **ON** Is Construction
Set **OFF** Visible
Exercise

It is possible to expose the Start and End diameter values and adjust their values.
Introduction to GenerativeComponents

GC Scripting
GCScript

• **GCScript** is a C-style programming language (like C, C++, Java, and C#) that’s optimized to work with GenerativeComponents.

• **GCScript** is a modern, robust programming language providing type safely, conditional statements, repeating statements, block statements, functions and sub-functions, objects and methods, arguments passed by value or by reference, a rich set of expression operators, and many more aspects.

• Furthermore, **GCScript** provides its own unique aspects, including automatic replication – the ability to use a list, or a nested set of lists, wherever a single value is expected.

• **GCScript** connects all of the features in a GenerativeComponents model. **GCScript** appears at several different levels:
GCScript

- **Node Properties** - Whenever you fill in the value of a property, that value is a **GCScript** expression, which can be as simple or as complex as you like.
- **FunctionCall nodes and 'ByFunction' techniques** - All of the functions you create are written entirely in **GCScript**.
- **Script Transactions** - In a transaction file, a script transaction lets you enter any number of arbitrary **GCScript** statements, which are executed in sequence when that transaction is performed.
- **Script Files** - A script file is a text file designed to contain a complete **GCScript** program. You can run the program, manually, whenever you wish, or you may run it automatically within any other **GCScript** code.

In this part, we will learn about the important fundamental concepts of **GCScript**. After covering the basics, we will create required geometry leveraging **GCScript**.
GCScript

Script Console and Variables

- **Variable Name**: int
- **Variable Type**: A
- **Operator**: =
- **Variable Value**: 3
- **End of Line**: ;
## GCScript: Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>This type can hold a floating-point (real) number value. This is generally used for holding numbers with a decimal point</td>
</tr>
<tr>
<td>number</td>
<td>This type can hold values of both int and double</td>
</tr>
<tr>
<td>string</td>
<td>This type can hold a sequence of zero or more text characters. A string must always come inside ‘ ’ or “ “. Either is correct</td>
</tr>
<tr>
<td>bool</td>
<td>This type can hold a bool value of true or false</td>
</tr>
<tr>
<td>var</td>
<td>This type allows us to hold any type of values</td>
</tr>
</tbody>
</table>
GCScript: Variables

Variables must be declared by Type before they are used.

```java
int A = 3;
int B = 2;

int C = A + B  // Correct
Print(C)     // Correct

// Incorrect
int A = 3;
int B = 2;
int C = A + B;
Print(C)     // Correct
```

**Examples:**

- Correct: `Print(C)`
- Incorrect: `Print(D)`
GC Script: Naming Variables

- The names of variables and other symbolic values follow the standard convention of most programming languages: A letter or an underscore character, followed by any number of letters, digits, and/or underscore characters, but not spaces.
- Names are case-sensitive; the names `lineLength` and `LineLength` are considered different names.
- The name cannot start with a number.
- Special characters (@!#$%^&*) cannot be used. However, ‘_’ is allowed.
- Words that have a special meaning within the GCScript language cannot be used.
GCScript: Numeric Operators

• The ‘+’ is a type of numeric operator. Let’s look at other commonly used numeric operators
• $A - 3$; ‘-‘ subtraction
• $A * B$; ‘*’ multiplication
• $A / B$; ‘/’ division
• $A \backslash B$; ‘\’ floored division (rounded to nearest whole number in the direction of negative infinity)
• $A \% B$; ‘%’ remainder when $A$ is divided $B$
GCScript: P.O.D.M.A.S

```c
int A = 4, B = 10, C, D = 3;
C = A * B + 9 / D - 1;
Print(C);
```

Each operator has a precedence level just like in mathematics we have the famous acronym **PODMAS** (*Parenthesis, Operations, Division, Multiplication, Addition, Subtraction*). Operators ‘∗’, ‘/’, ‘\’’, ‘%’ have the same precedence level, but they have higher precedence than ‘+’ and ‘-‘.
GCScript: Relational Operators

A==B; Test if the value of A is equal to the value of B.
A!=B; Test if the value of A is not equal to the value of B.
A>B; Test if the value of A is greater than the value of B.
A<B; Test if the value of A is less than the value of B.
A>=B; Test if the value of A is greater than or equal to the value of B.
A<=B; Test if

This type of operator compares two variables to understand their relationship, for instance, if one variable is greater than the other or equal to the other, etc. The operations always return a Boolean value (true or false) based on the test performed.
**GCScript: Lists**

```plaintext
int A = {1, 3, 5, 6};
```

<table>
<thead>
<tr>
<th>DATA</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>

A.First(), A.Last(), A.Count

*GSScript* works with lists of values as easily as single values. A *List* is a collection of values where each value can be managed individually. To specify a list we have to put the series of values separated by commas within **curly braces**.
GCScript: Functions

Function (argument1, argument2)
{
    statement A
    statement B
    return output;
}
GCScript: Functions

functionCall1

Create/edit this node's custom script function

function(int number1, int number2) //Arguments
{
    int sum = number1+number2;
    return sum;
}
A control statement is a statement that determines the flow of execution based on logical condition. In other words, such statements allow us to specify whether a statement is to be skipped or repeated based on the conditions sets. Some common types are if, if-else, for, and while.
A control statement is a statement that determines the flow of execution based on logical condition. In other words, such statements allow us to specify whether a statement is to be skipped or repeated based on the conditions sets. Some common types are *if*, *if-else*, *for*, and *while*.

```plaintext
function (int Number) //arguments – input
{
    if (Number%6==0) //test
    {
        return Number+1; //return - output
    }
}
```
GCScript: Exercise

Steps
1. Create a baseCS
2. Place a Point Node (ByCartesianCoordinates)
3. Attach GeometricSequence to Point Node (Xtranslation) input
4. Place a Line Node (ByChaining)
GCScript: Exercise

Create a `functionCall` node and rename to `GeometricSequence`.
Open the `Script Editor` and input the following:

```csharp
function (int CommonRatio, int XValue, int Count) //arguments - inputs
{
    int GS = { }; //declare the variables, GS is an empty list
    GS.Add(XValue); //statement – add the value of XValue to the list
    for (int i = 0; i < Count; ++i) //for expression - initialize the loop at 0, increase by 1 until i is no longer less than Count
    {
        XValue = XValue * CommonRatio; //statement – multiply the current XValue by the CommonRatio
        GS.Add(XValue); //statement – add the value of XValue to the list
    }
    return GS; //return - output
}
```
GCScript: Exercise

**CommonRatio:** 2

**XValue:** 1000

**Count:** 10.
GCScript: Exercise

You will generate a list of points where the distance between each point and the next point is twice the distance from the previous point.
GC Resources

Bentley User forum:
GenerativeComponents Forum - GenerativeComponents - Bentley Communities

Youtube Channels:
5D Design – YouTube
GenerativeComponents – YouTube
Bentley OpenBuildings – YouTube

Bentley Learn*:
GC Learning Path(bentley.com)

* May require a commercial subscription or if a student or academic staff, free registration with the Bentley Education Program: Bentley Education - Unlimited access to Bentley engineering applications
Thank you

Brenden Roche

designcomputation.org/2021