CES322249

Computational Design for Civil Engineers

Paolo Emilio Serra, Implementation Consultant
Autodesk

Safi Hage, Designated Support Specialist
Autodesk

Learning Objectives

- Learn how to create automated workflows for infrastructures with Dynamo
- Learn how to apply computational design to civil workflows
- Learn how to write Python script to interact with AutoCAD and Civil 3D .NET API
- Learn about generative design for infrastructures

Description

This class demonstrates how to use automation for transportation, land development, and pipe networks workflows in Civil 3D software through Dynamo and Python. The class introduces Dynamo for Civil 3D capabilities, focusing on the implementation of computational design for infrastructures. We will cover how to expand the possible workflows through Python scripting to use the AutoCAD and Civil 3D APIs.
This class is for Civil engineers with a working experience of Civil 3D interested in automating workflows to do more, better with less effort to maintain data across software platforms.

Speakers

Paolo Emilio Serra

Is a construction engineer by trade, he worked as BIM Manager in an architectural firm for 5 years in Milan Italy. Since 2014 he is works as an Implementation Consultant BIM for Autodesk. With Autodesk he has been delivering Customer Success Services to engineering Companies, supporting BIM workflows and Digital Transformation in their business processes. Paolo’s main focuses are on automation, generative design, integration between AEC and ENI industries. Paolo is an Architecture enthusiast and a Revit user since 2006. He started to discover the possibilities of automation and customization with the Autodesk product APIs and Dynamo. He developed the CivilConnection Dynamo package that creates dynamic relationships between Civil 3D and Revit for Linear Structures BIM workflows. He also provided support to the product team in Autodesk to introduce Dynamo for Civil 3D. He owns the blog Punto Revit.

Safi Hage

Is a structural engineer by trade. Since 2014 he is a Designated Support Specialist at Autodesk working with enterprise priority customers worldwide focused on the AEC industry. Before joining Premium Support Organization, he was an Autodesk Technical Consultant BIM for 6 years.

Dynamo and Computational Design

Dynamo is a platform enabling designers to explore visual programming, solve problems and create custom tools.

Visual programming is the process of establishing rules that can describe the relationships among the parts of a design, the rules and the relationships are formalized into an algorithm.

An algorithm is a set of instructions to follow to perform a task. The instructions can be textual in plain English, Italian or French, etc. or can be expressed in a graphical way, for example using languages such as UML or BPMN.

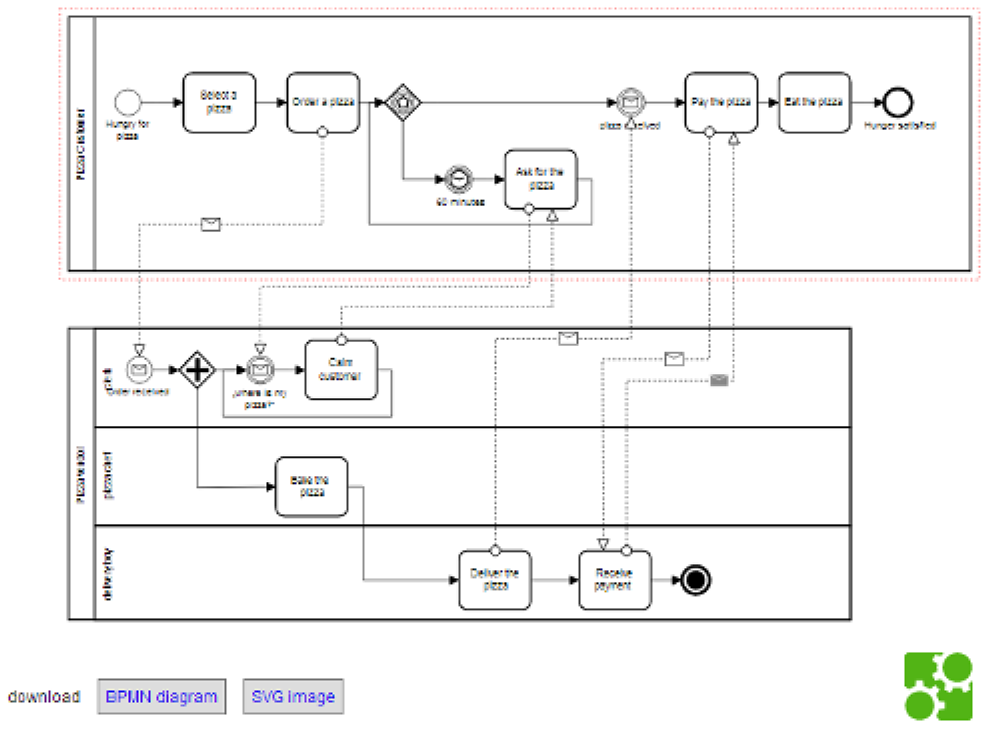
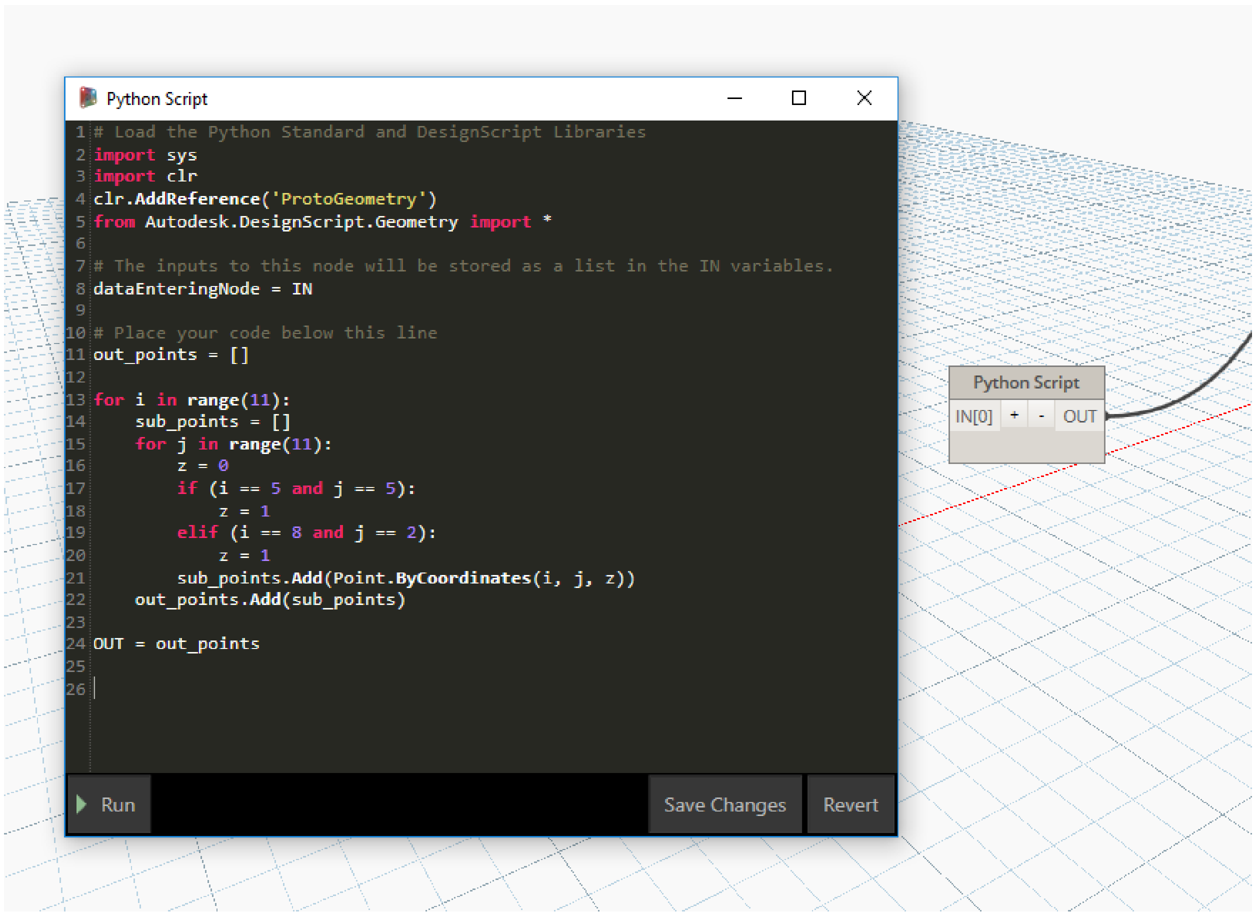


FIGURE 1: AN EXAMPLE OF VISUAL PROGRAMMING IN BPMN

For a computer to be able to understand these instructions there is need to adopt an intermediate language that is human readable but that a computer can also understand.
There are several possible traditional scripting languages such as C#, Python, JavaScript.



```python
# Load the Python Standard and DesignScript Libraries
import sys
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
dataEnteringNode = IN

# Place your code below this line
out_points = []

for i in range(11):
    sub_points = []
    for j in range(11):
        z = 0
        if (i == 5 and j == 5):
            z = 1
        elif (i == 8 and j == 2):
            z = 1
        sub_points.Add(Point.ByCoordinates(i, j, z))
    out_points.Add(sub_points)

OUT = out_points

```

FIGURE 2: AN EXAMPLE OF TRADITIONAL SCRIPTING IN PYTHON

Dynamo is a platform that can define algorithms visually and the sequence of instructions are generated using block called Nodes that perform predefined tasks, joined together in sequence using Connectors. A Dynamo algorithm is called Graph to emphasize the visual programming approach which is one of the main differences between Dynamo and any other scripting language.

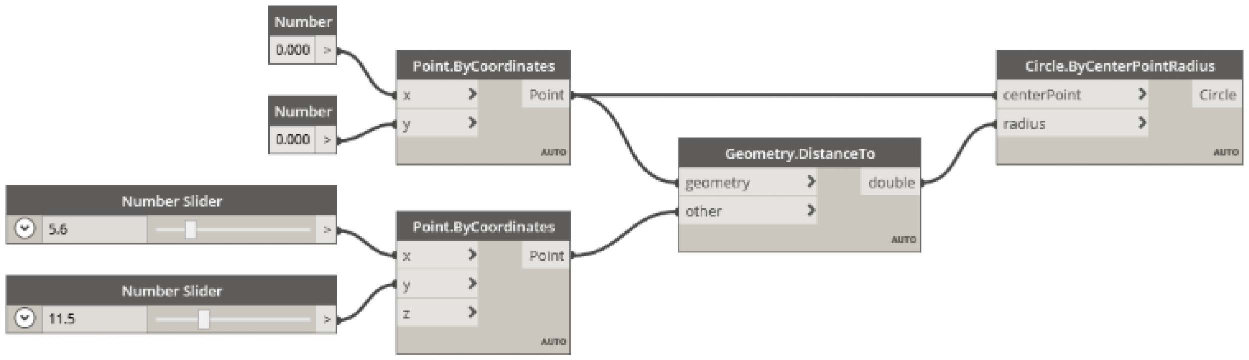


FIGURE 3: VISUAL PROGRAMMING IN DYNAMO TO CREATE A CIRCLE USING NODES AND CONNECTORS

Dynamo is some sort of a digital Swiss-knife that can contain just right blade for the job and even if it is very intuitive in the way it can be used, it is also very effective. Another important characteristic of Dynamo is that, for whatever reason, a blade should be missing it supports several ways to either add one from a public repository of external packages or create more sophisticated functionalities using traditional scripting languages such as Python or C#.

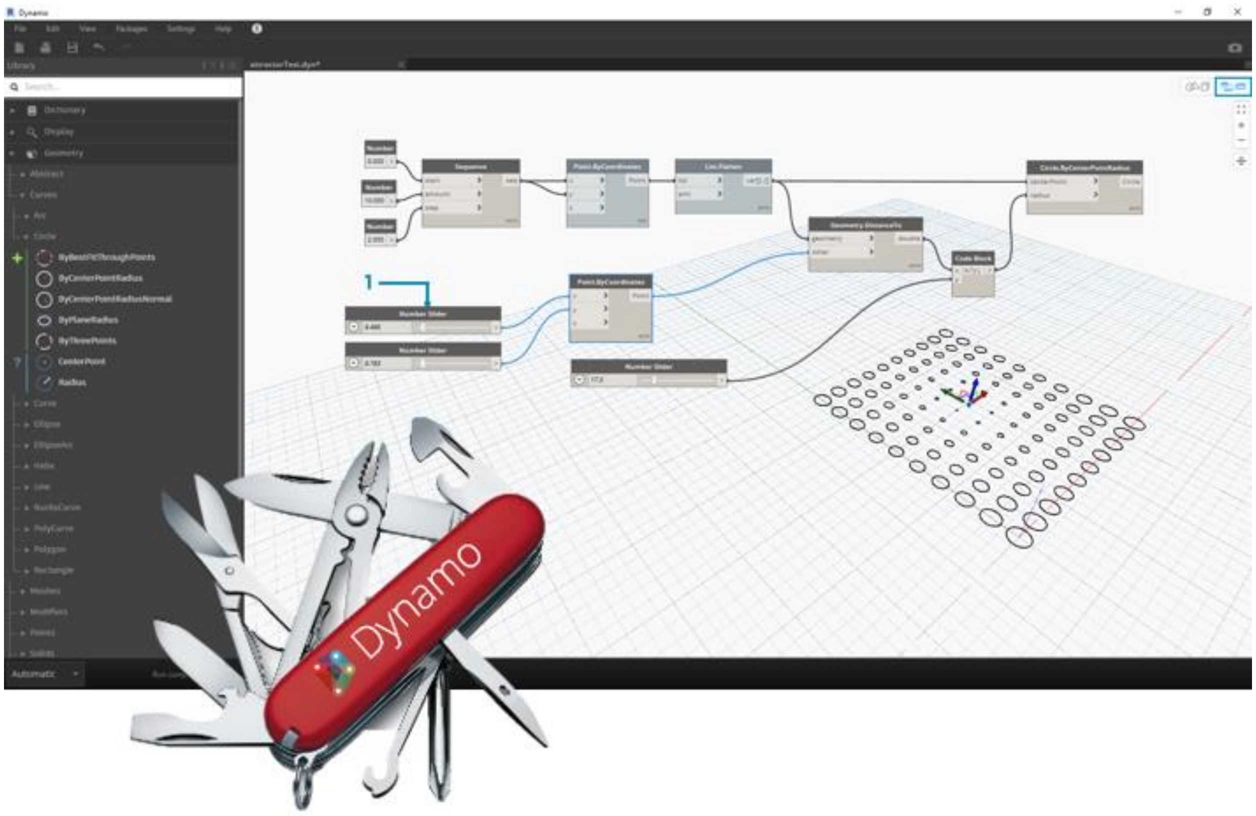


FIGURE 4:DYNAMO IS A PLATFORM FOR AUTOMATION THAT SUPPORTS CUSTOMIZATION

Dynamo commoditizes programming and brings it closer to the designers, taking care of the sophisticated operations such as handling databases and transactions, serializing changes and updating the models rather than performing a “fire-and-forget” automation. Dynamo is in the sweet spot in between interactive tools such as Civil 3D or Revit and more traditional scripting languages to leverage product APIs.

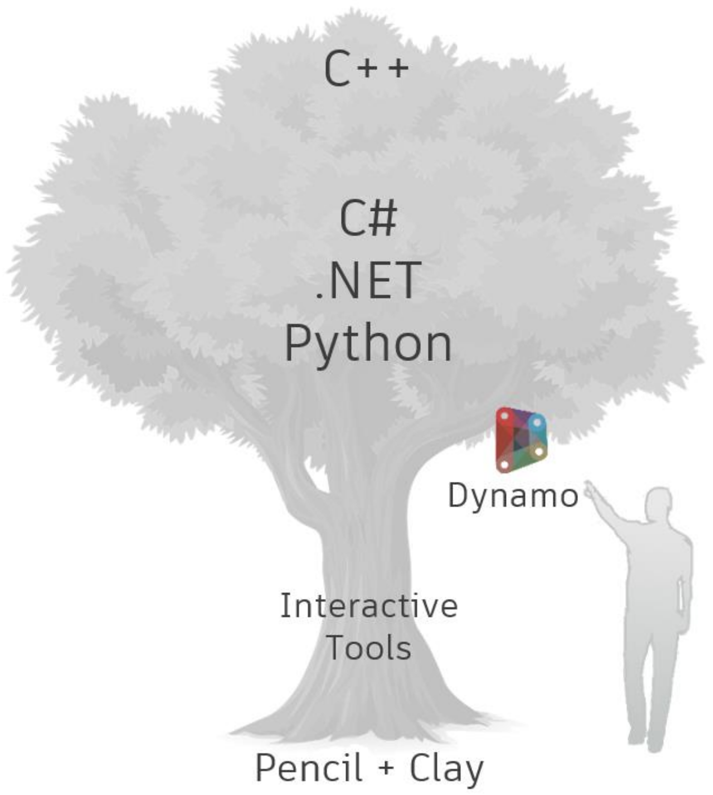


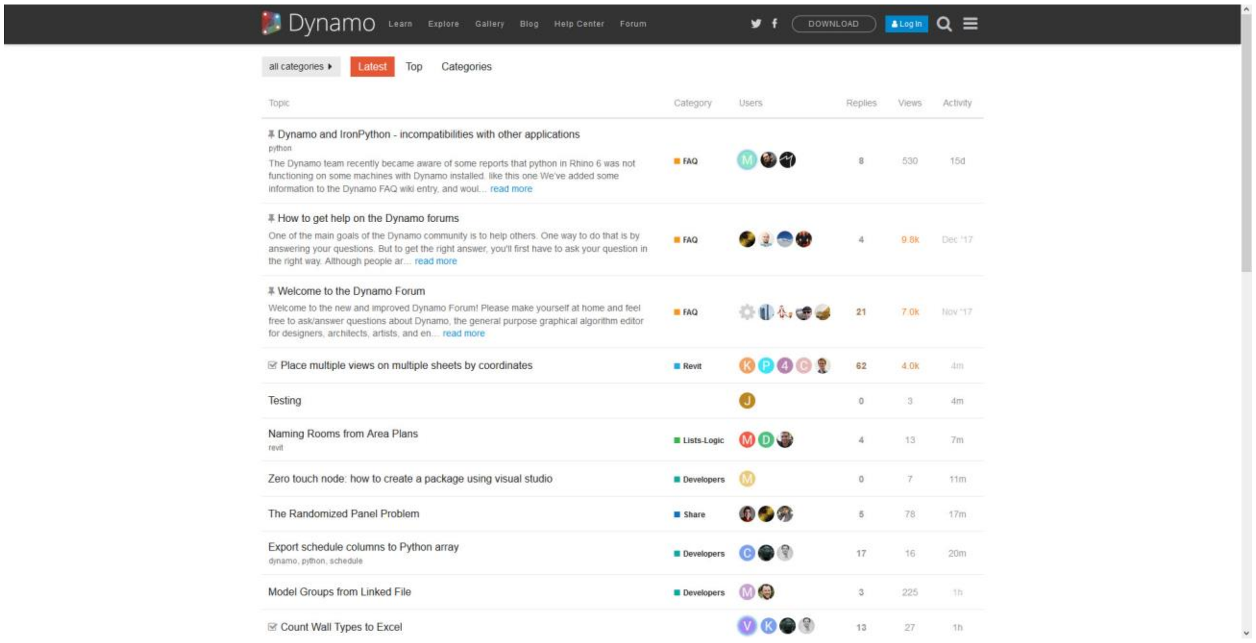
FIGURE 5: DYNAMO AS LOW-HANGING FRUIT OF THE TREE OF AUTOMATION TOOLS

Dynamo can be found as a standalone application called Sandbox but also integrated with a host application such as Revit or Civil 3D. For more information about Dynamo integrations and where to get it visit the link https://dynamobim.org/download/

FIGURE 6:DYNAMO IS INTEGRATED WITH A NUMBER OF HOST APPLICATIONS

The community of Dynamo users has grown with a very fast and steady pace, mostly because of the ease of use of visual programming and because the community started to collect and share useful sets of nodes for free. These are bundled into packages to extend Dynamo capabilities and simplify the creation of more sophisticated workflows and boost adoption and productivity.

The community is also proactively engaging in knowledge sharing on the Dynamo forums: a must for anyone and everyone who wants to connect, learn, find creative solutions to problems and contribute back.


Dynamo Learn Explore Gallery Blog Help Center Forum DOWNLOAD Log In

all categories Latest Top Categories

Topic	Category	Users	Replies	Views	Activity
Dynamo and IronPython - incompatibilities with other applications python The Dynamo team recently became aware of some reports that python in Rhino 6 was not functioning on some machines with Dynamo installed. like this one We've added some information to the Dynamo FAQ wiki entry, and woul... read more	FAQ	M	8	530	15d
How to get help on the Dynamo forums One of the main goals of the Dynamo community is to help others. One way to do that is by answering your questions. But to get the right answer, you'll first have to ask your question in the right way. Although people ar... read more	FAQ		4	9.8k	Dec '17
Welcome to the Dynamo Forum Welcome to the new and improved Dynamo Forum! Please make yourself at home and feel free to ask/answer questions about Dynamo, the general purpose graphical algorithm editor for designers, architects, artists, and en... read more	FAQ		21	7.0k	Nov '17
Place multiple views on multiple sheets by coordinates	Revit	K P 4 C	62	4.0k	4m
Testing		J	0	3	4m
Naming Rooms from Area Plans revit	Lists-Logic	M D	4	13	7m
Zero touch node: how to create a package using visual studio	Developers	M	0	7	11m
The Randomized Panel Problem	Share		5	78	17m
Export schedule columns to Python array dynamo, python, schedule	Developers	C	17	16	20m
Model Groups from Linked File	Developers	M	3	225	1h
Count Wall Types to Excel		V K	13	27	1h

FIGURE 7: THE DYNAMO FORUM ON THE OFFICIAL WEBSITE DYNAMOBIM.ORG

For more mature Dynamo integrations with host applications such as Revit, Dynamo comes with an optional simplified interface called Dynamo Player that allows to select a graph from a list, provide the necessary user inputs via a dynamic interface and run the execution of the graph. Dynamo Player can turn the Dynamo graphs into something that resembles in many ways' custom add-ons for the host application with the advantage that they can be developed without requiring a deep understanding of the host application API.

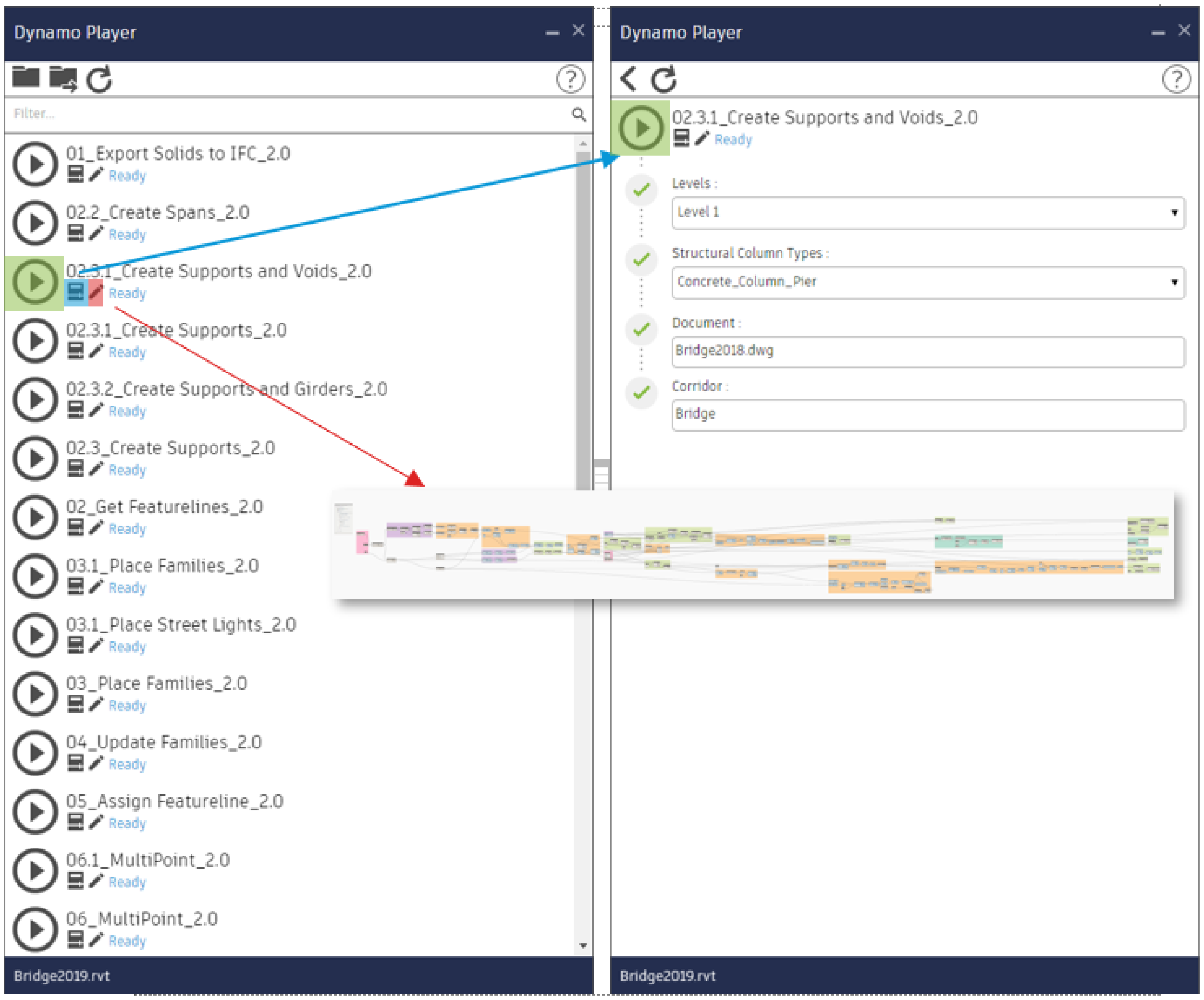


FIGURE 8: DYNAMO PLAYER FURTHER SIMPLIFIES THE ADOPTION OF AUTOMATION IN A HOST APPLICATION

Why Use Dynamo

Dynamo allows the designers to do **more** and control complex modeling that would be otherwise impossible or extremely difficult or time consuming to do manually with the mere out of the box functionalities of the host application.

Dynamo can provide a way to achieve a higher level of model data consistency to ensure **better** quality across multiple projects and teams without relying on the skills of the individuals, capturing best practices and scaling them throughout the business with automation.

Dynamo allows to automate repetitive tasks, **reducing the effort** required to achieve the same outcome and allowing designers to focus of the part of their jobs that require critical thinking.

Dynamo is particularly versatile and allows designers with no coding background to prototype solutions to solve immediate problems and express their needs in a language that in easy to comprehend by a professional developer.
A developer has the possibility to focus on ample development and implementations without risking of becoming a bottleneck for other departments. If necessary, the experienced developer can extend the nodes in Dynamo and even overwrite or add new behaviors to the application to suit Company needs and project requirements.

As a plus, Dynamo provides out of the box features such as reading and writing Excel spreadsheets or implement a Create-Read-Update-Delete paradigm.
Dynamo also reduces the need for updating the code when a new API is released as the visual programming approach is almost immune to deprecation cycles.

A Dynamo file is structured as a JSON file, an organized collection of data that is used by the Dynamo application to implement a logic interpreting on the fly what is in this text.
This implies that a Dynamo file cannot be “complied” like a plugin written in C# or locked down for preventing someone from fiddling with the logic of the nodes, at the end of the day it is just a text file that anyone can open with a text editor.

CivilConnection and CivilPython

CivilConnection is the name of the Dynamo custom package for Dynamo for Revit that enables the Civil 3D - Dynamo - Revit technology stack for Autodesk Consulting engagements that have been known as "Linear Structures". It enables the usage of a linear reference system defined in an open Civil 3D document (e.g. a corridor, alignment, feature line, etc.) to create and/or update elements in a Revit document. It also enables to import and update the elements from a Revit document into a Civil 3D document.

FIGURE 9: CIVILCONNECTION & CIVILPYTHON DYNAMICALLY CONNECT CIVIL 3D AND REVIT VIA DYNAMO

This experience allowed to collect since 2016 several potential use cases that helped in shaping the development of Dynamo for Civil 3D.

Automation is not something new in the infrastructure space. There have been several attempts in the past, recurring to plug-ins based on a constrained form to execute very specific tasks. Although they do not require the users to understand the automation steps, these plug-ins could not be flexible enough to accommodate all the project requirements and needs. On top of that some plug-ins were retired and left a technology gap to be addressed.
After considering all these factors, a custom tool developed directly by the designers for the designers is still the best solution.

CivilConnection eases the definition of prototypes with a toolkit philosophy: it contains a range of functionalities generic enough to cover most use cases that can be combined to solve very specific project challenges. And all this with a visual programming approach rather than recurring to traditional scripting languages.

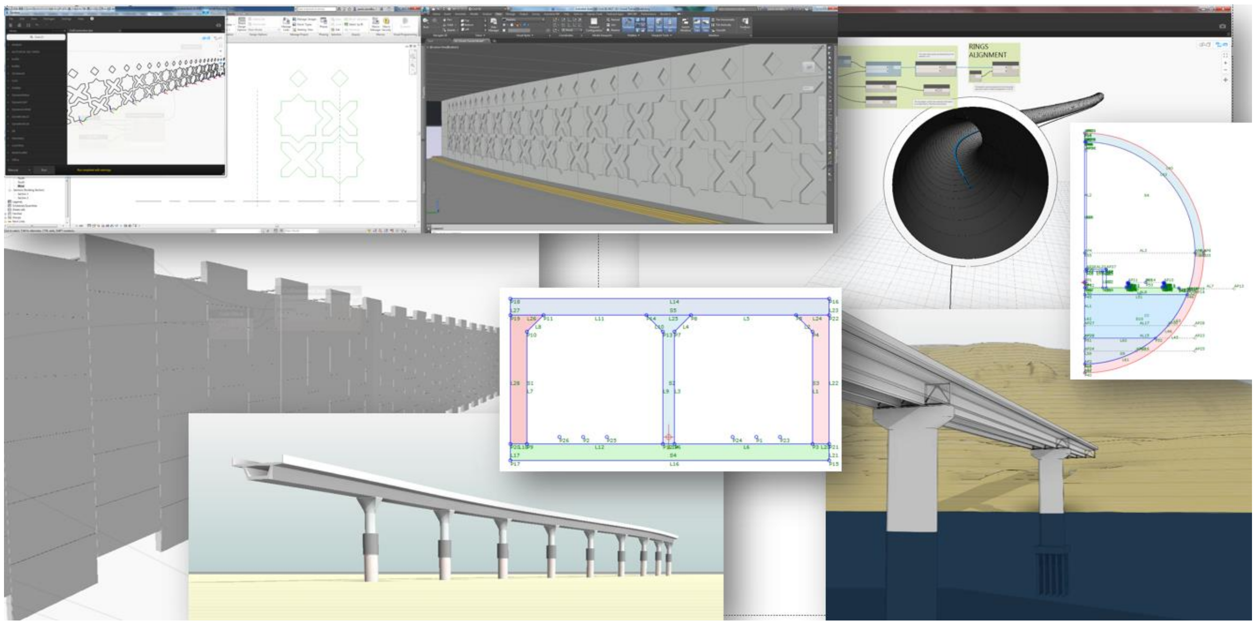


FIGURE 10: LINEAR STRUCTURES USE CASES ENABLED VIA CIVILCONNECTION AND CIVILPYTHON

CivilConnection is mostly based on a technology called Component Object Model or COM; over the year's Civil 3D and in general most Autodesk products introduced a more efficient and performant API based on .NET.

The intention behind CivilPython was to enable the same kind of prototyping approach that was available for Dynamo users in Revit, using the Python syntax to leverage the .NET Revit API. CivilPython is a custom command that enables to select and execute Python scripts in AutoCAD and Civil 3D. It also has a command line version of the command that can be launched from CivilConnection through COM. That was it, now CivilPython also contains some commands that are necessary for CivilConnection to access data using .NET rather than COM.

FIGURE 11: CIVILPYTHON FURTHER EXTENDS THE CAPABILITIES OF CIVILCONNECTION

CivilConnection and CivilPython are open source and can be found at this link along with examples and documentation https://github.com/Autodesk/civilconnection

Design Automation

Traditionally designers apply their intuition to define a design of a building or a bridge. The design intent is captured in detailed drawings and the relationships of the parts of the design, their qualities and quantities are explicated using dimensions, text notes, etc.
We can call this kind of process "static design".

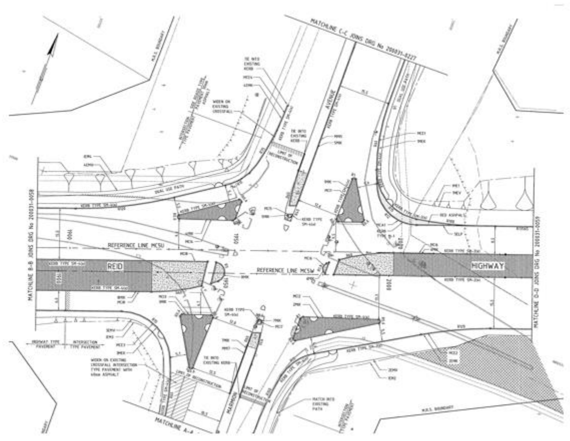
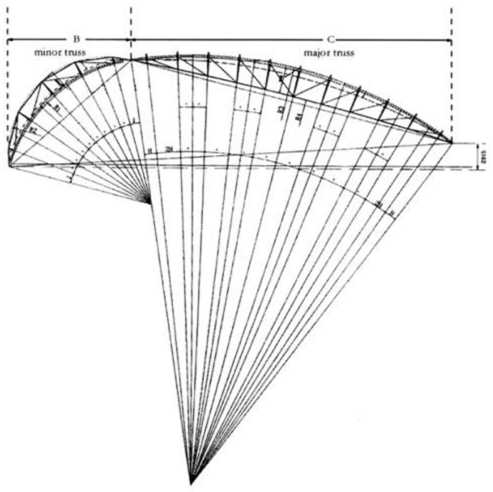



FIGURE 12: EXAMPLES OF STATIC DESIGN FOR A ROAD INTERSECTION OR A TRUSS

With dedicated modelling software and intelligent parametric objects, the design process changed radically. Designers can now explore more options more quickly; the software takes care of updating a complex system of relationships between parts in three dimensions thanks to the parameters expressed either analytically (e.g. changing a "thickness" in a subassembly) or graphically (e.g. overriding the "width" or "elevation" parameters in a subassembly using targets). For example, in Civil 3D the designer defines the following inputs: the alignment and vertical profile, the baseline regions and assigns an assembly, specifies the sampling frequency and specifies the targets. Then the software takes care of generating all the corridor sub-elements (e.g. feature lines, surfaces, solids) and updating them dynamically if any of the user inputs changes.

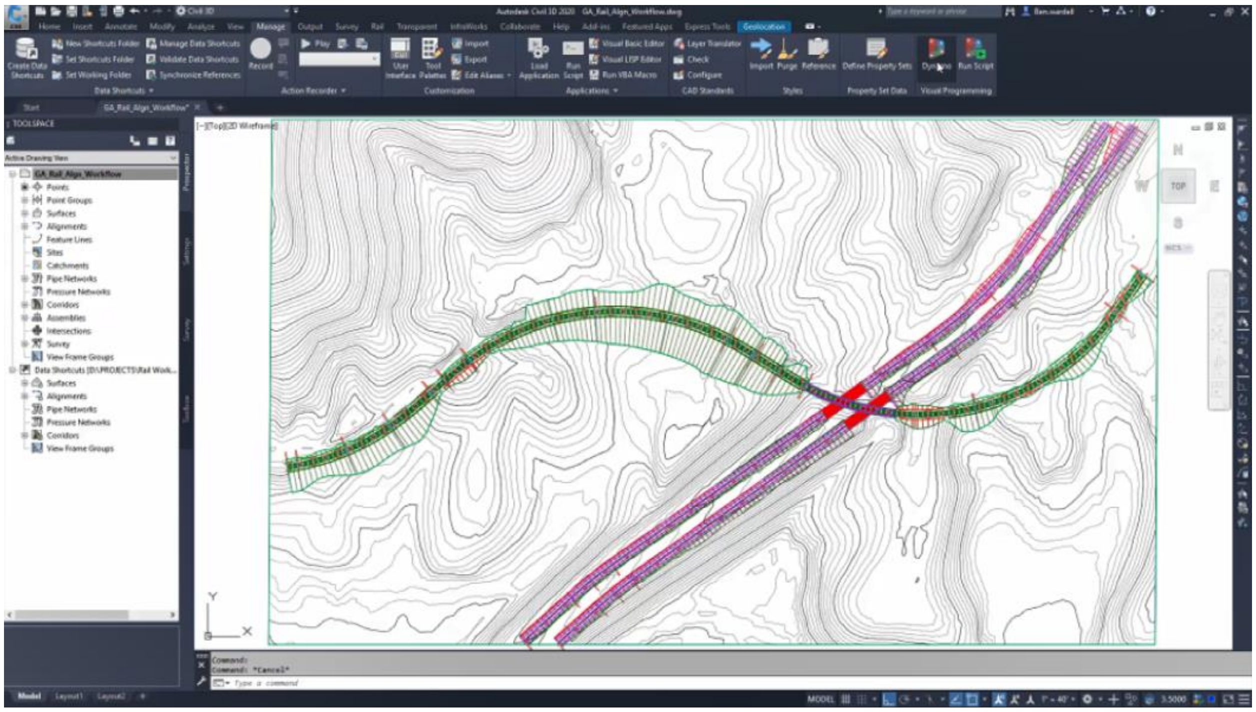


FIGURE 13: CIVIL 3D OFFERS PARAMETRIC TOOLS FOR CIVIL WORKFLOWS

The next step in the evolution of design is the possibility to introduce algorithms, a set of instructions, a system of rules that allows to generate the objects and that can produce predictable results. This approach enhances the ability of the designers to introduce new dynamic relationships in the software, relationships that the software was not originally designed for.
A very simple application in Civil 3D is the creation of discrete elements along a corridor (e.g. concrete ties along a rail track), so that the location and orientation of the discrete objects is connected to the feature lines in the corridor.

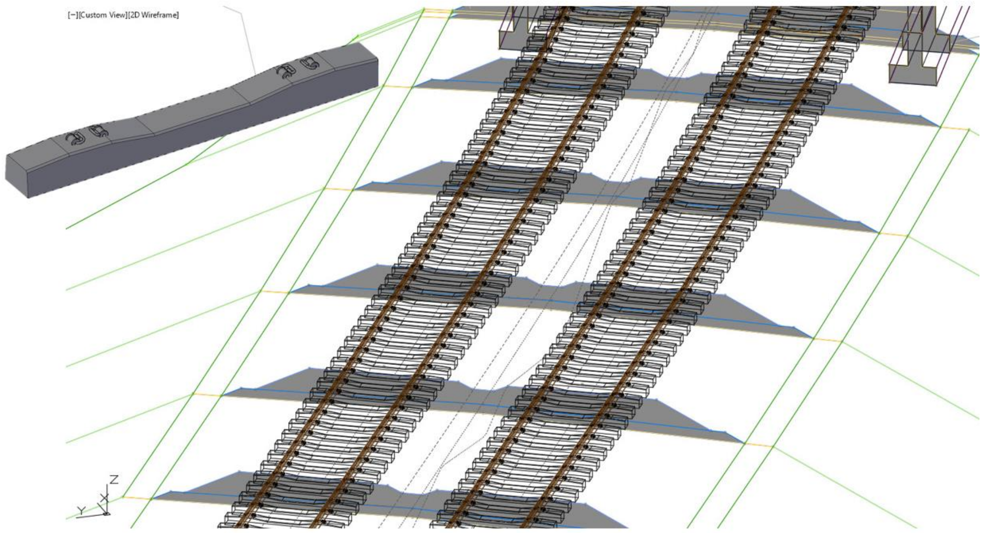
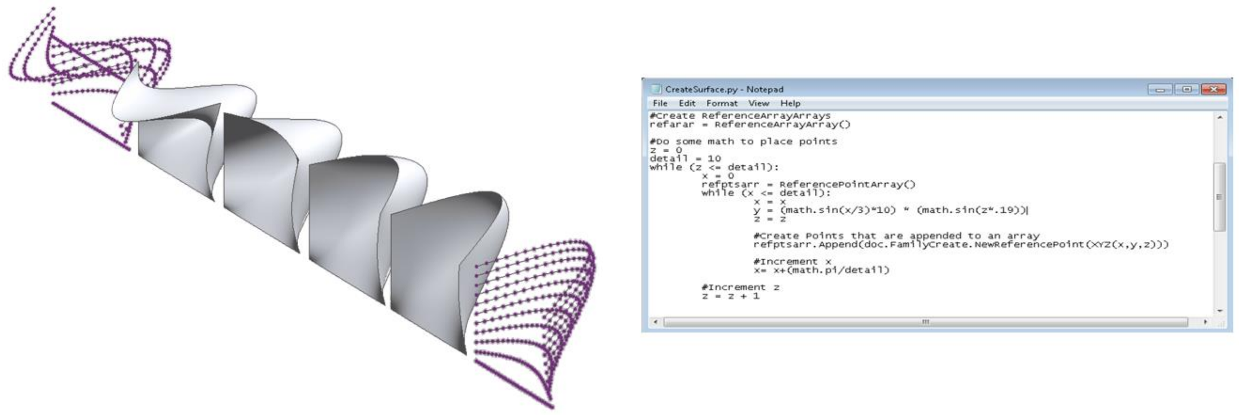


FIGURE 14: CONCRETE TIES DYNAMICALLY CONNECTED TO A CIVIL 3D RAIL CORRIDOR VIA DYNAMO

Computational design needs an intermediate environment between the designer and the software to prototype and assemble these new relationships. It also needs an intermediate language between the design and the software to translate the rules of the design intent into instructions for the software.

The intermediate environment could be a VBA macro editor, a Visual Lisp editor, MS Visual Studio, etc. to leverage some sort of traditional scripting to define the sequence of instructions.



```
#Create ReferenceArrayArrays
refarar = ReferenceArrayArray()

#Do some math to place points
z = 0
detail = 10
while (z <= detail):
        x = 0
        refptsarr = ReferencePointArray()
        while (x <= detail):
                x = x
                y = (math.sin(x/3)*10) * (math.sin(z*.19))
                z = z

                #Create Points that are appended to an array
                refptsarr.Append(doc.FamilyCreate.NewReferencePoint(XYZ(x,y,z)))

                #Increment x
                x= x+(math.pi/detail)

        #Increment z
        z = z + 1
```

FIGURE 15: DEFINE A SEQUENCE OF INSTRUCTIONS VIA TRADITIONAL SCRIPTING

Dynamo is an intermediate environment that allows the designers to leverage both visual programming and traditional scripting via Python and explore a computational design approach in their daily jobs.

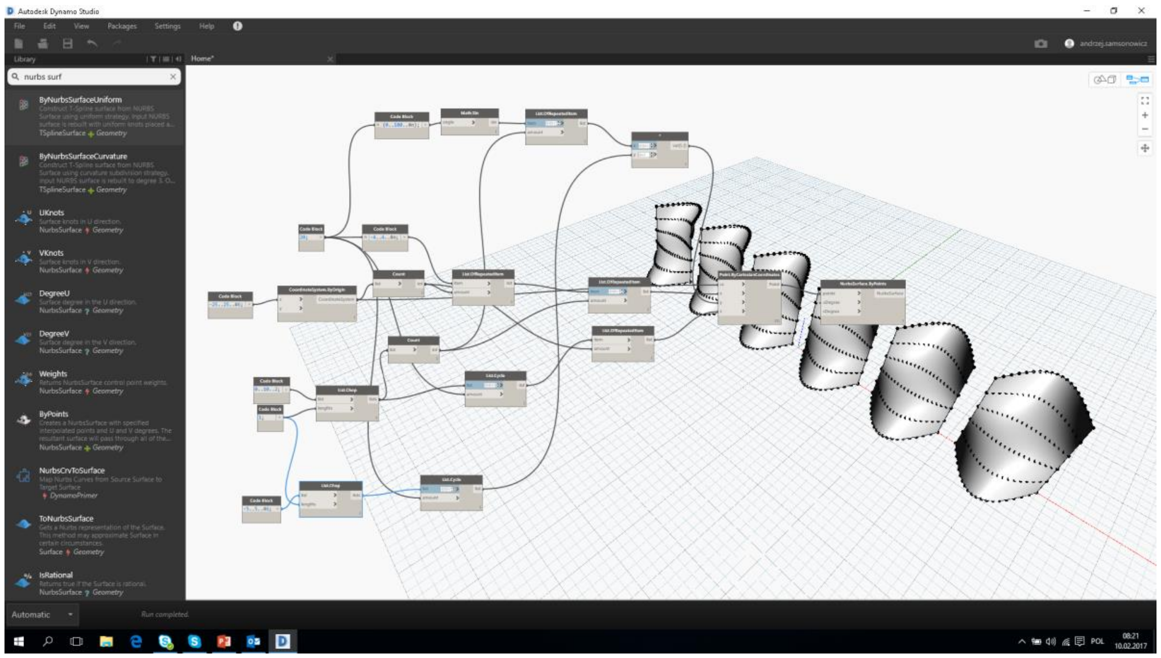


FIGURE 16:DYNAMO AS AN ENVIRONMENT FOR COMPUTATIONAL DESIGN

The best advice I can give to someone that is starting to approach computational design, regardless the level of expertise in coding, is to plan the automation. Defining all the steps in the algorithm, identifying the inputs and their suppliers, defining a process to consume the inputs and create suitable output that can be consumed later for other BIM uses.

A very intuitive tool that allows to draft any algorithm in a visual way is called Business Process Modeling Notation, or BPMN. This approach allows to abstract and analyze the problem at hand and subdivided it into smaller problems that can be solved at a later stage, but without losing focus on the overall objective.
This allows to identify opportunities for the automation to improve efficiency with less effort.
For examples, a process can be greatly simplified (e.g. reducing number of steps, reducing number of decisions to handle, etc.) if the inputs are provided in a slightly different way from the usual, something that the input supplier can do with little effort (e.g. adopting a naming convention for Point Codes in subassemblies) but that can bring great benefits when it comes to acquire, interpret and process the data.
BPMN can also be used to “debug” existing processes or algorithms (regardless if they were developed using traditional scripting or visual programming).

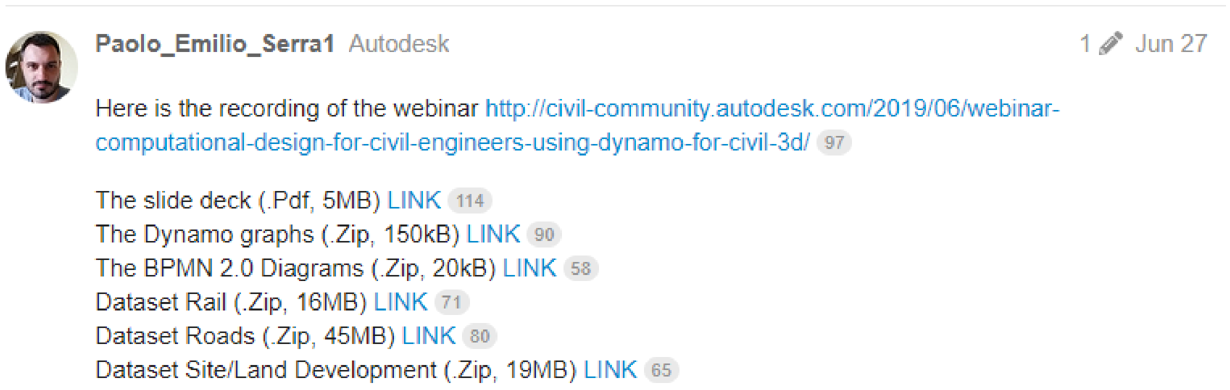
There is a free web based tool called BPMN.IO that was used to create the diagrams in this document, it is highly recommendable to introduce something similar in any design process to capture ideas, plan the development and frame a problem to formulate a very precise question when asking for help.

On June 26th, 2019 aired a live webinar on Design Automation that was presenting Dynamo for Civil 3D for the first time after the Civil 3D 2020 global launch in April of the same year.

In the webinar were presented examples of how to use automation for civil workflows rail, road and site development. At this LINK you can find the recording of the webinar.

FIGURE 17: LIVE WEBINAR ON DESIGN AUTOMATION FOR CIVIL WORKFLOWS

The presentation and the dataset used in the examples presented in the webinar can be found in this post (LINK) on the Civil 3D section of the Dynamo Forum.


Paolo_Emilio_Serra1 Autodesk 1 Jun 27

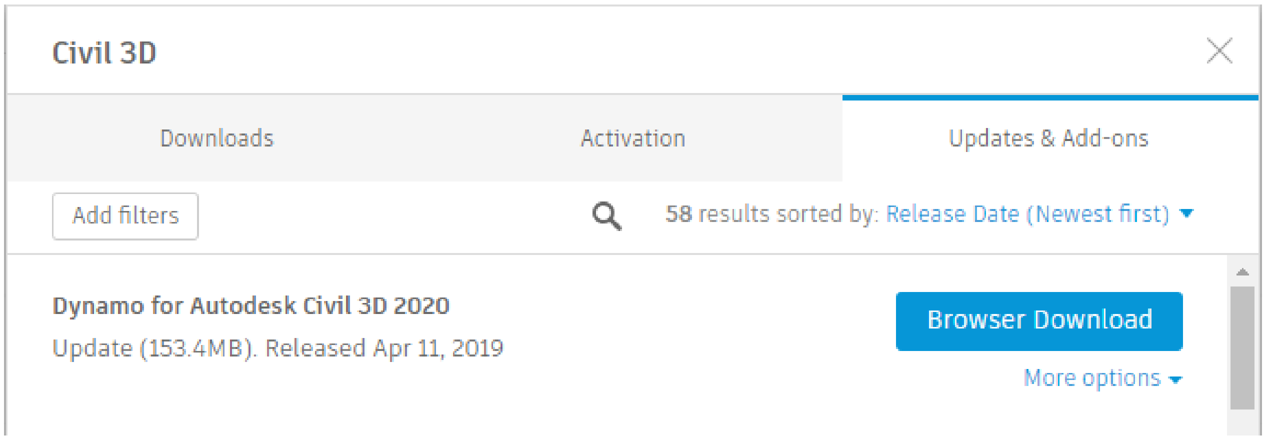
Here is the recording of the webinar http://civil-community.autodesk.com/2019/06/webinar-computational-design-for-civil-engineers-using-dynamo-for-civil-3d/ 97

The slide deck (.Pdf, 5MB) LINK 114
The Dynamo graphs (.Zip, 150kB) LINK 90
The BPMN 2.0 Diagrams (.Zip, 20kB) LINK 58
Dataset Rail (.Zip, 16MB) LINK 71
Dataset Roads (.Zip, 45MB) LINK 80
Dataset Site/Land Development (.Zip, 19MB) LINK 65

FIGURE 18: LINK TO THE DATASET USED IN THE WEBINAR

Dynamo for Civil 3D

In Civil 3D 2020 it is possible to find and integration with Dynamo. Initially Dynamo for Civil 3D was available as a separate install accessible from the manage.autodesk.com under the addons for Civil 3D 2020. With Civil 3D 2020.1, Dynamo comes directly with the product.


Civil 3D

Downloads | Activation | Updates & Add-ons

Add filters

58 results sorted by: Release Date (Newest first)

Dynamo for Autodesk Civil 3D 2020
Update (153.4MB). Released Apr 11, 2019

Browser Download
More options

FIGURE 19: DYNAMO IN CIVIL 3D 2020 AFTER GLOBAL LAUNCH

It is possible to access to Dynamo for Civil 3D when the workspace is set to Civil 3D, from the Mange tab under the Visual Programming panel on the ribbon. There are currently two icons, one that launches the Dynamo application interface, the other that launches a headless Dynamo session that runs a script.

Alternatively, Dynamo can be launched at the command line typing the *AECCLAUNCHDYNAMO* command.

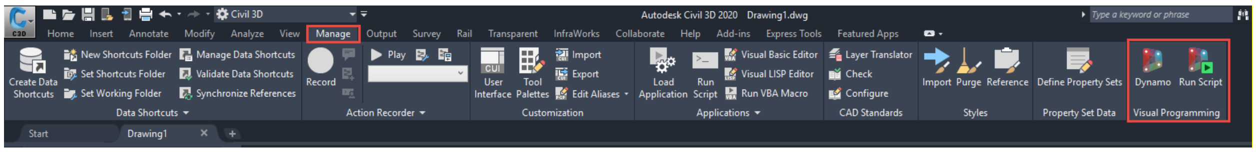


FIGURE 20: DYNAMO FOR CIVIL 3D IS ACCESSIBLE FROM THE MANAGE TAB IN THE RIBBON

Dynamo for Civil 3D comes with a set of examples use cases to show how to apply computational design to enhance transportation workflows with automation.
There are two components to consider: the AutoCAD objects and the Civil 3D objects.
In these first releases of Dynamo for Civil 3D, the nodes available are mainly focus around enabling transportation model authoring workflows dealing with Alignments and Corridors, in the future we can expect to find more objects accessible via Dynamo and to implement functionalities that can enable other civil disciplines and BIM uses (e.g. drawing production, visualization).

Dynamo can read the data in a DWG file and for objects it can return proxies, or “wrapper” that represents a node in the Dynamo library.
Dynamo can also write back to the DWG a given set of objects (e.g. Line, Polyline, Circle, Text, Block Reference, etc.) and in doing so it keeps track of which objects have been created through Dynamo.

AutoCAD and Civil 3D Nodes

The aim of this section of the document is not to list the nodes currently present in Dynamo for Civil 3D, that would be rather pointless as they can change quite a lot in between versions. It is more important understanding how the nodes are organized in the Dynamo library, what are the relationships between object types and what are the current limitations.

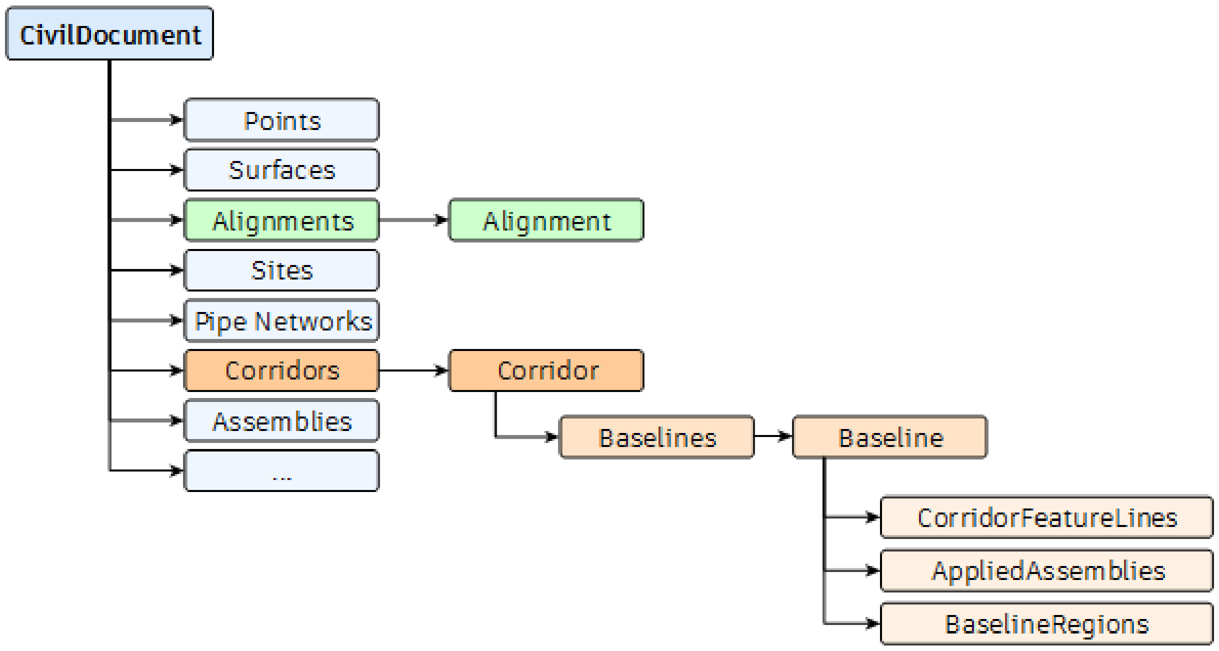


FIGURE 21: NODES STRUCTURE FOR DYNAMO FOR CIVIL 3D

Dynamo Bindings

When the Dynamo application is visible and the graph creates objects in the DWG, the trace of the Dynamo owned objects can be serialized in the Dynamo file itself. This mechanism is called “Binding”: for each node that creates objects from Dynamo into the DWG, Dynamo stores their “fingerprints” in the Bindings section of the Dynamo JSON structure, so that if the inputs change, Dynamo can confidently update only the objects that are owned by the graph accordingly. The user needs to save the file after the execution for Dynamo to be able to store the trace data into the Bindings section.

If there are no valid Bindings in the Dynamo graph (e.g. the Bindings were removed from the JSON file, or someone manually deleted the objects that were generated by Dynamo, or the Dynamo graph was previously used on a different DWG file), after the execution new objects are created rather than update existing ones, and the bindings relationships in the JSON file are overridden as soon as the user saves the Dynamo file.

By design, Dynamo for Civil 3D connects with one DWG document at the time and interacts only with that document for the entire session. This is a limitation that mimics the same implementation that we find in Dynamo for Revit. This is in part due to how Bindings in Dynamo work: you can have a dynamic behavior only if you make sure you have a 1:1 relationship between a Civil 3D model and a Dynamo file.

When using the Run Script option, the Dynamo interface is not visible which means that after the execution, the user cannot save the graph. This implies that the Bindings section of the JSON file is wiped out after the execution and the Dynamo graph is more of a “fire-and-forget” rather than dynamically updating existing objects. We find a similar behavior (no bindings) if objects are created through a Python Script rather than using Dynamo nodes.
Because of this, it is recommended to define a naming convention for your Dynamo graphs that shows the relationship between the graph and a Civil 3D model.

For example:

[<DWG name>_][<Sequence>_]<Category>_<Task>_<Version>.dyn

Dynamo Standards

As soon as Dynamo allowed the creation of node groups and colors in 2015 (0.8.1) I've developed a standard to unify the look and feel of Dynamo graphs (LINK).
The standard is based on a very simple fact: every automation on a model, regardless the language used, is a sequence of functional tasks. I've associated a color in the palette in Dynamo, from left to right like reading a book, for these sequences of logical functions.

TABLE 1: AUTODESK CONSULTING DYNAMO STANDARD COLOR CODING

#	Function	Color	
1	Gather data from the model or external sources	🟪	Purple
2	Gather user inputs	🟥	Pink
3	Process the inputs	🟧	Orange
4	Generate some key outputs	🟩	Green
5	Set property values of existing objects in the model	🟩	Blue Green
6	Display intermediate results or debug the automation logic	🟦	Blue
7	Provide comments and documentation for logic	⬛	Dark Grey
8	Identify portions of the logic under development	⬜	Light Grey

Adopting a standard improves readability and supports adoption of the Dynamo graphs within a company. Also leaving exhaustive commentary around the logic is an added value and best practice that should always be applied in any development.
For node groups though it is better to be very concise and assign a short sentence to express the task, something like Verb + Noun, something that forces the developer to convey a high-level idea of what the collection of nodes in the group is supposed to do. That is why it is recommendable to plan the development using, for example, a BPMN diagram to use as a draft for renaming the node group in Dynamo.

Automated Workflows in Dynamo

At a very high level, a Dynamo graph can enable some form of the following workflows.

Extract data from the model

The automation reads the model objects data programmatically and then sends it to another application for consumption.
To enable this workflow the automation should provide functionalities to:

- Select objects, either directly or based on a property or classifier (e.g. select all the lines in a model)
- Get properties, every objects in a host application is more than just an abstract piece of geometry and contains information such as handles/ids, type, and other kind of metadata associated (e.g. the length of a line, the color, the name of the layer etc.), the data needs to be extracted and interpreted (e.g. it is a piece of text, a number or another object, etc.)
- Write the values to an external storage, this can take any form and requires managing the creation of data streams and serialization towards different formats (e.g. text, CSV, XLSX, XML, JSON, image formats, etc.)

An example of this workflow could be extracting the elevation of feature lines at given station intervals and write the resulting information to a custom report in Excel.

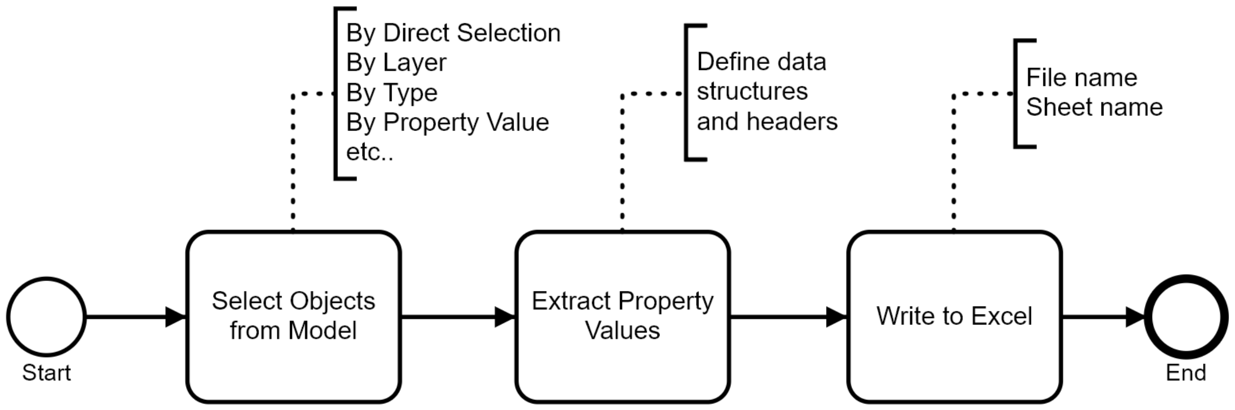


FIGURE 22: VISUAL REPRESENTATION OF THE AUTOMATION TO EXTRACT DATA TO EXCEL

Modify existing objects with external data

The automation reads the external data, accesses the model objects properties, updates the values and saves the changes to the database (e.g. the location of Cogo Points, the starting station of an alignment, etc.).
To enable this workflow the automation should provide functionalities to:

- Select objects, either directly or based on a property or classifier (e.g. select all the lines in a model)
- Read an external source or input, for example a numeric value specified by the user or the location of the external source (e.g. server, cloud-based repository, etc.); the automation should be able to interpret the data contained in the source/input and return the values in the automation environment for further processing.

- Modify objects, the automation should be able to open a transaction with the host application, find the properties that needs to be modified as the user intended (metadata and non) assign the new value and commit the changes to the model database.

An example of this workflow could be reading the property set values from an Excel spreadsheet and update the values on the correspondent objects in the model.

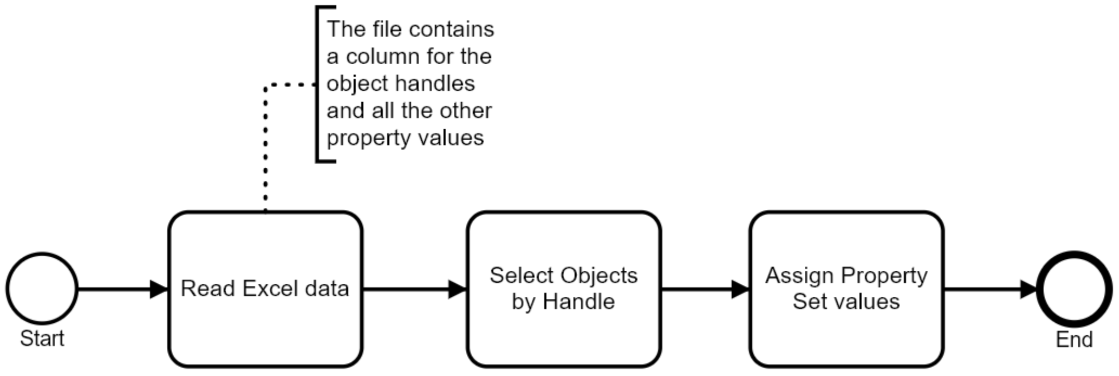

The file contains a column for the object handles and all the other property values
Start
Read Excel data
Select Objects by Handle
Assign Property Set values
End

FIGURE 23: LOGIC FOR EDITING PROPERTY SETS VALUES FROM EXCEL

Create or update objects

The automation receives some inputs, processes the data following the computational rules and creates objects in the host application model. If the inputs change, executing the automation one more time updates the definition of the elements previously created they are new or update their definition if there are any existing objects connected to a previous execution of the automation.
To enable this workflow the automation should provide functionalities to:

- Acquire user inputs, either directly or from an external source
- Process data define the logical steps to transform the inputs into abstract Dynamo geometry items that can be used to drive the creation of model objects.
- Create/Update model objects, a set of functionalities that can transform a Dynamo abstract object into a concrete model object for the host application.

An example of this workflow could be creating bollards along the side of a road.

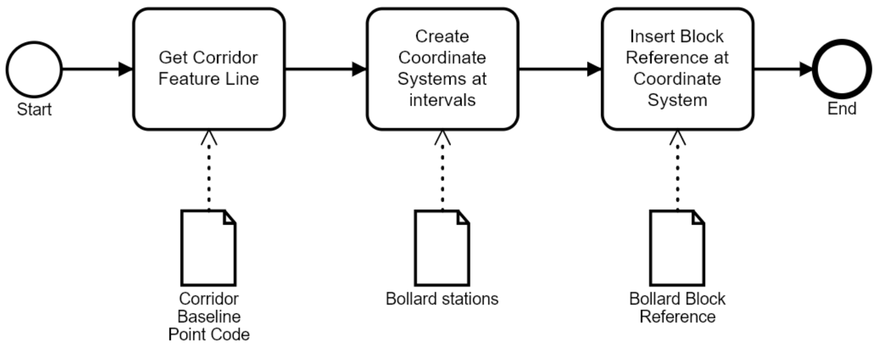

Start
Get Corridor Feature Line
Create Coordinate Systems at intervals
Insert Block Reference at Coordinate System
End
Corridor Baseline Point Code
Bollard stations
Bollard Block Reference

FIGURE 24: LOGIC TO CREATE BOLLARDS ALONG THE SIDE OF A ROAD

Computational Design for Civil Workflows

In the webinar were briefly discussed some use cases to demonstrate design automation applications of Dynamo for Civil 3D in three different areas: Rail, Roads and Site Development.
The scope in this document is to provide some context and highlight the logic behind the graphs that were used during the webinar and point out the Dynamo for Civil 3D functionalities used to obtain the results. The approach used in this document explains the goal and the logic used to reach it, the object types involved and how to find the relationships that keep the logic together using Dynamo nodes.

Rail

With Civil 3D 2020 there are many improvements to support rail workflows, a complete list of these improvements can be found on the Civil 3D online help under the Release Notes. When it comes to automation it is always useful to identify the components of the tasks in relation to their outcomes such as Do More, Do it Better or Do it with Less Effort.

Civil 3D already provides a set of tools to deal with the modeling and management of the linear elements such as the actual rails, so what can an automation do to improve the modelling workflow? The use cases presented below cover different aspects, from detailed modelling to usage of external data source through visual programming, taking advantage of Dynamo for Civil 3D functionalities.

Concrete Ties

The key learnings for this example are:

- How to get a Corridor Feature Line from the Document
- How to get a Coordinate System on a Corridor Feature Line
- How to create a range of stations
- How to create Block References

A concrete tie or sleeper is a typical example of discrete element that participates to the detailed modeling of a rail track that is hard to position and maintain in Civil 3D. In AutoCAD terms, a concrete tie is a Block Reference, something that could come for Revit or Inventor, then exported to DWG. It is important to know the local origin and insertion point for Block Reference and its name before using it in the automation.

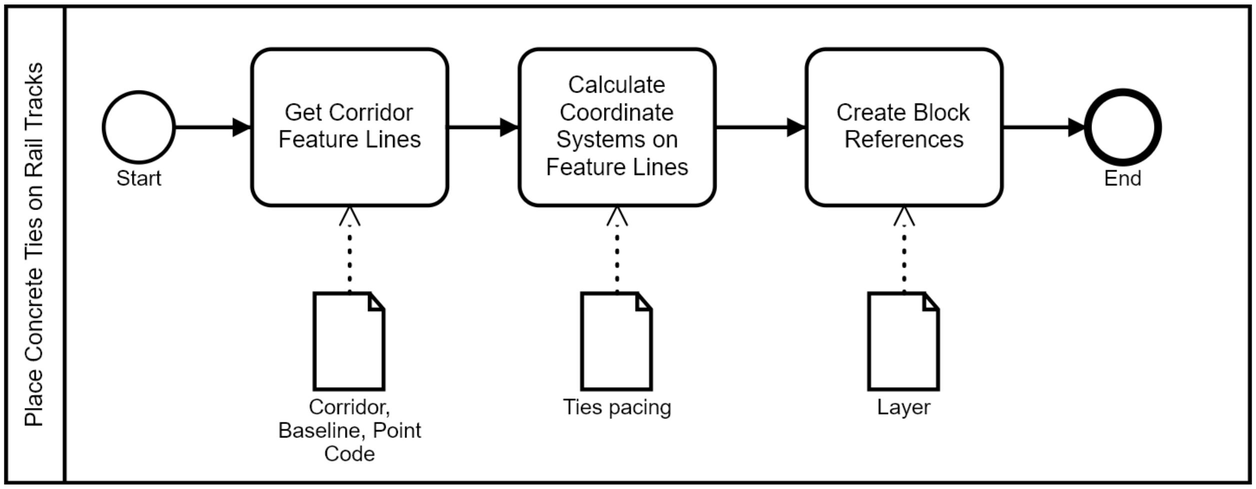

Place Concrete Ties on Rail Tracks
Start
Get Corridor Feature Lines
Calculate Coordinate Systems on Feature Lines
Create Block References
End
Corridor, Baseline, Point Code
Ties pacing
Layer

FIGURE 25: LOGIC FOR THE CONCRETE TIES

The concrete ties follow the Center Line of the tracks of the rail corridor, the Block References are placed at regular intervals of 2' (ca 0.6m), orthogonally to the Alignment. At a high level it is relatively easy to think about what is needed to place a Block Reference: the name, the layer, the location in X, Y and Z (where Z is the elevation of a linear reference such as the Baseline or a rail Feature line) and the rotation around the vertical axis. To place a Block Reference there is need for some more data, but it is safe to assume that these are default values for now (e.g. the scaling factors, and of course, the target block where we need the Block Reference to be created, in other words, the Model Space of the document).

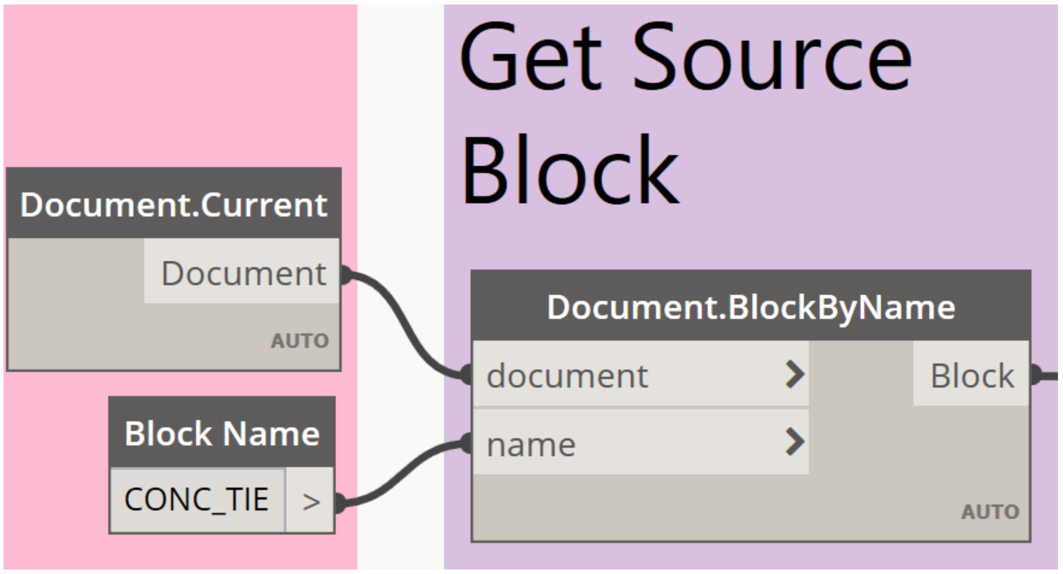


FIGURE 26: SELECTING THE DEFINITION OF A BLOCK REFERENCE IN THE DOCUMENT BY NAME

This is an effective technique when it comes to automation: starting with the end in mind and building the automation going "upstream" so to speak.

The next step then is to understand what the inputs are to provide to the function that creates the Block Reference and so on.

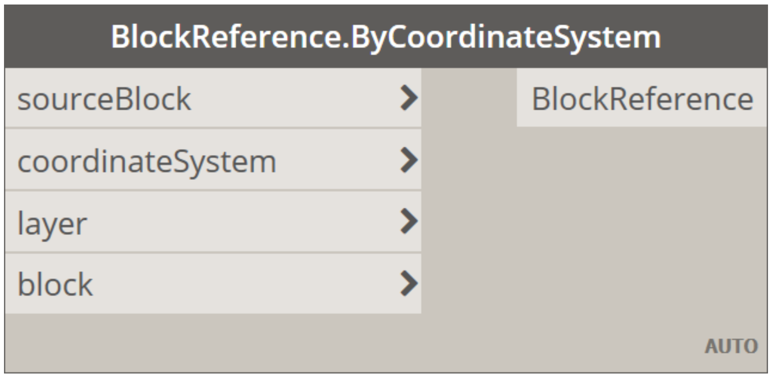


FIGURE 27: CREATE A BLOCK REFERENCE WITH A COORDINATE SYSTEM

Let's focus on the location: the request is to tessellate the linear reference (e.g. the Baseline) at regular intervals. The intervals may or may not coincide with the sampling intervals defined in the Corridor parameters. This is already powerful: it is possible to consider a Corridor and its Baselines and query some stations that may not be in the definition of the Corridor on the fly, without the need to rebuild the Corridor!

Now it's time to look at the rotation of the concrete ties: the ask is to make these elements orthogonal to the Alignment. The default orientation of the Block Reference is known (e.g. parallel to the global X axis), so the ask is to calculate the rotation necessary to have the Block Reference orthogonal to the Alignment at a given station.
In Dynamo, to calculate angles there is need to operate with Vectors. On DynamoPrimer.com there are very useful notions of trigonometry that every Dynamo user should be familiar with and the assumption here is that the reader understands how this is done.
In the 2020.1 release of Dynamo, the Block Reference creation has been simplified greatly with the introduction of a node that takes as an input a Coordinate System (link).
This object conveys at once the information of the location and orientation and saves us a lot of trouble trying to calculate the rotation for each concrete tie.

In geometrical terms, the ask is to calculate a Point that is sitting on the Baseline or a Feature Line curve at a given station (the station is measure along the Alignment used to create the corridor) and the direction of the curve in that point. There is a Dynamo node that does exactly that.

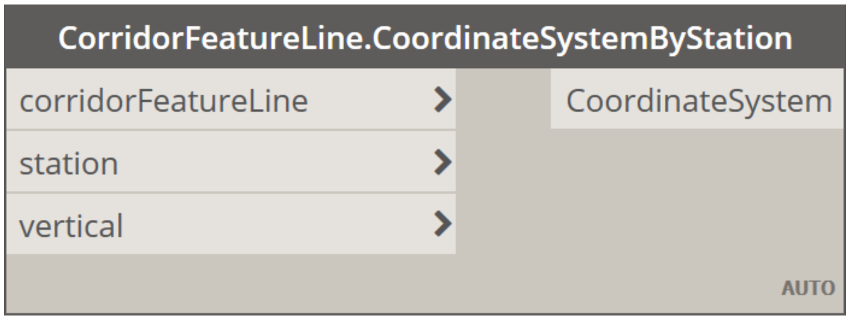


FIGURE 28: GETTING A COORDINATE SYSTEM ON A FEATURE LINE BY STATION

Traversing the data in the Document it is possible to recall the Corridor and the Baseline by their names and after select the Corridor Feature Line by the Point Code assigned in the subassembly definition.

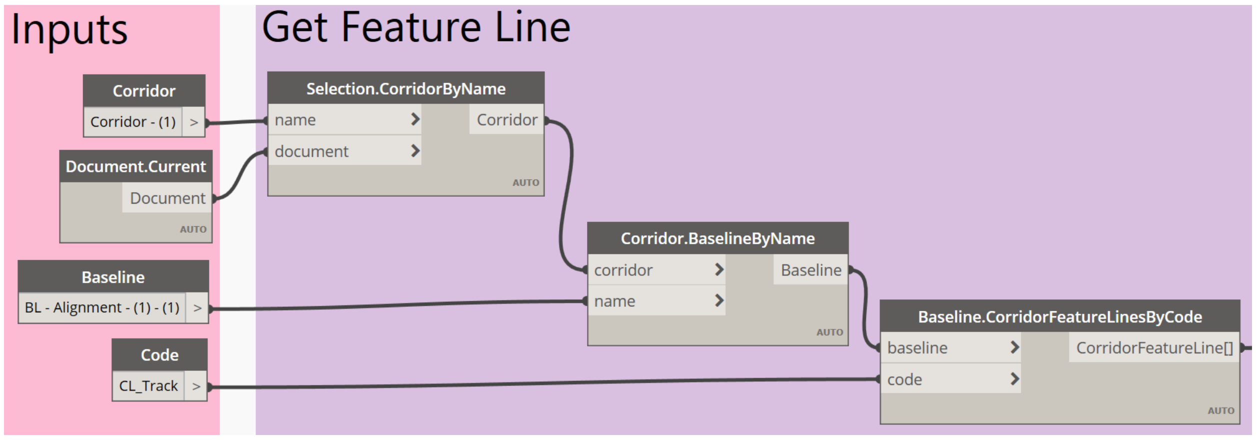


FIGURE 29: GET A FEATURE LINE FROM A CORRIDOR VIA ITS POINT CODE

This is the logic used to place a single Block Reference, what about the others though? This is done via a “loop”, something that tells Dynamo how many times to repeat a set of instructions providing the correct inputs at the beginning of each repetition.

Loops in Dynamo are handled through a couple of paradigms called **lacing strategy** and **replication guides**. For a detailed explanation of how this works refer to the Wiki pages on the Dynamo GitHub, starting from this link.

In a nut shell: each input port on a node has a default “rank” of input (a single input, a list of inputs, a list of lists of inputs, etc.). If the input provided has a rank higher than the default, this will trigger a loop and the function defined by the node will be repeated several times. At each repetition the function will traverse the list of inputs. To determine exactly how this is done, for how many times and how the output is structured the user can specify the lacing strategy on the node via right click and/or using the chevrons next to the input ports of the node.

So now the question is how to specify a list of stations to calculate multiple Coordinate Systems on a Feature Line. To do that there is need for a range, Dynamo provides a node and several Code Block syntaxes to generate range of number, in this case a range of stations on a Feature Line (note: these are not the stations used to sample the Feature Line in the Corridor!).

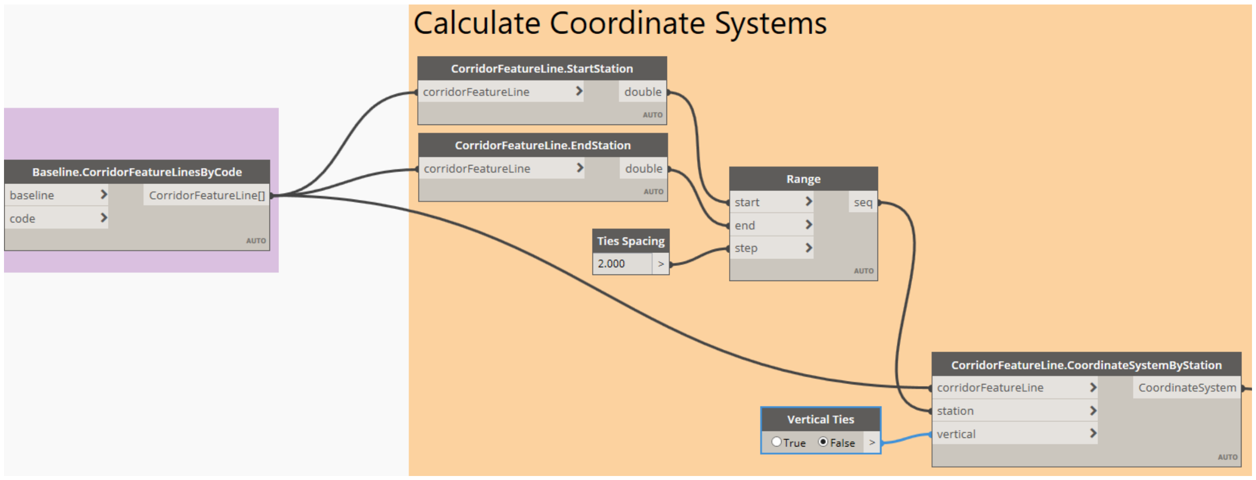


FIGURE 30: CALCULATE A RANGE OF STATIONS TO RETRIEVE COORDINATE SYSTEMS ON THE FEATURE LINE

Putting it all together returns an automation that in very few nodes can place discrete Block References along a Corridor and dynamically update the model to react to design changes (e.g. the alignment, the vertical profile, the block definition, the spacing of the block references, etc.). This allows to do more with Civil 3D, create better models and update the models if something changes with less effort.

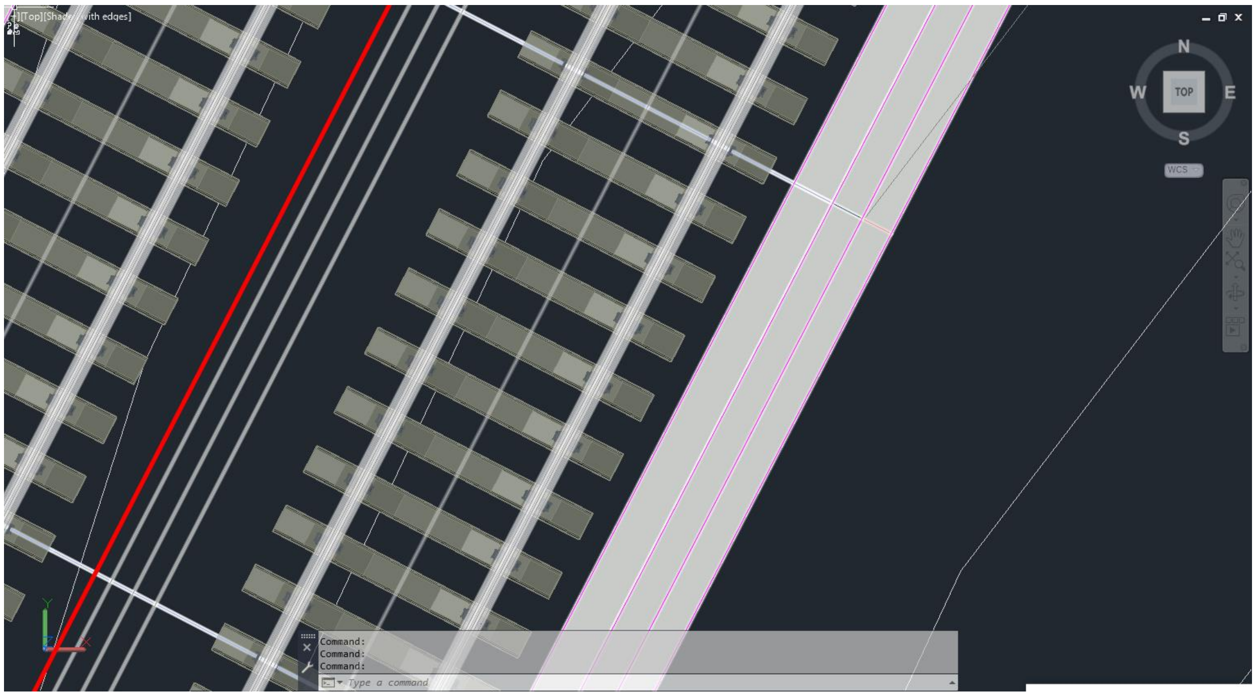


FIGURE 31: FINAL RESULT OF THE AUTOMATION FOR THE CONCRETE TIES UNDER THE RAIL TRACKS

Overhead Catenary Systems

The key learnings for this example are:

- How to group and filter objects by a property value
- How to retrieve Station, Offset, Elevation and relative rotation of an object
- How to write data to Excel

Placing Block References presents many similarities with the previous example: the OCS masts are indeed Block References following a linear reference but they are also some key differences: the Block References are not all of the same type (e.g. centered, on the left and on the right side of the tracks), their location is not defined at regular intervals but they are specified by the designer. It should not be difficult for the reader derive the necessary logic for the placement that considers different Block sources and stations, nevertheless it is interesting leverage an out of the box functionality in Dynamo that allows to read and write Excel spreadsheets natively. This opens the possibilities for the user when it comes to import data in and extract data from Civil 3D without recurring to external plugins.

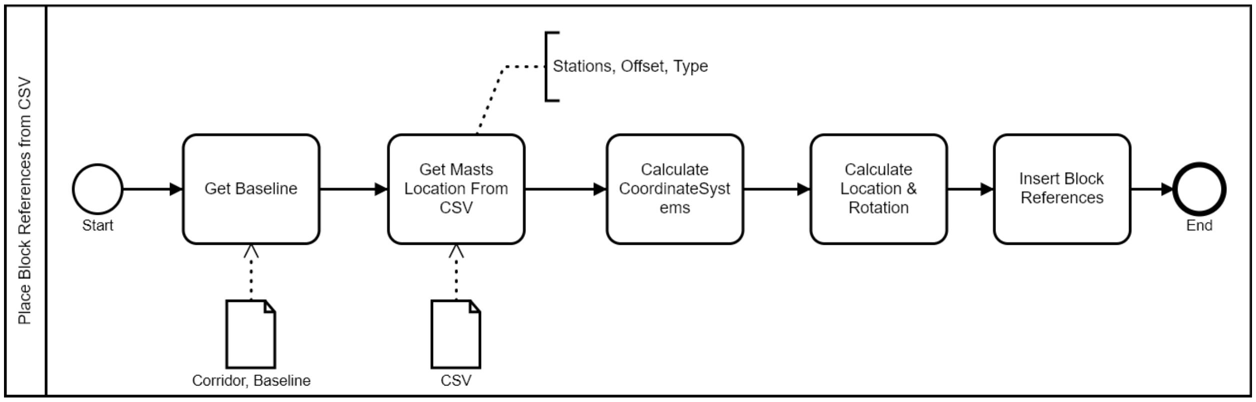

Place Block References from CSV
Stations, Offset, Type
Start
Get Baseline
Get Masts Location From CSV
Calculate CoordinateSystems
Calculate Location & Rotation
Insert Block References
End
Corridor, Baseline
CSV

Figure 32: Logic for the OCS Masts

In this example it could be of interest to understand how to create and maintain an external Excel file (or CSV) that lists the names of the masts type, their station, offset and elevation from the corridor baseline and other properties as needed.

Dynamo offers the capabilities to read and write Excel files natively, process the data following a custom logic and maintain the connection between the external data and the Civil 3D model so that if the inputs change, Dynamo can update the model and the Excel file accordingly.

The diagram below shows the logic to follow to **extract** the data from the model, in this case all the Block References and then filter those that are representing Overhead Catenary System (OCS) masts. With the Corridor and Baseline names it is possible to select the Baseline to retrieve the relative coordinate of the Block Reference insertion points. This function uses the Alignment underlying the Baseline to calculate Station and Offset and the Elevation is simply the Z coordinate of the insertion Point. Lastly it is necessary to calculate also the relative rotation of the Block References in relation to the tangent to the Baseline in the same Station.

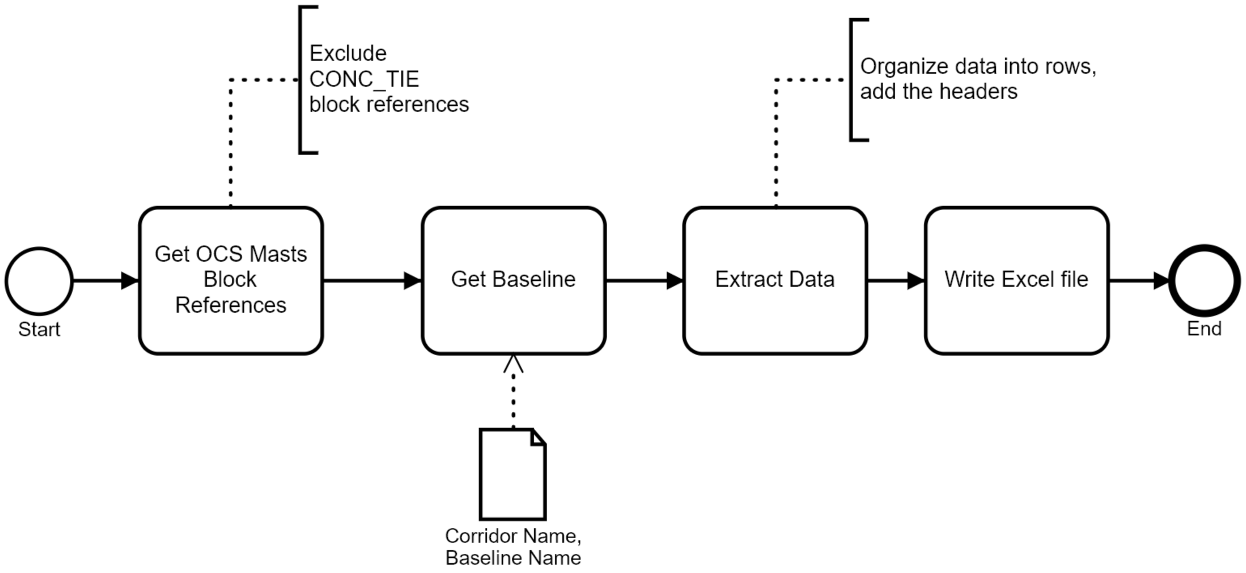


FIGURE 33: LOGIC TO EXTRACT BLOCK REFERENCE DATA TO EXCEL

Using the Select by Object Type it is possible to extract all the Block References in the Document. To do this there is need to specify the Block in which the Block References are created, this is going to be the Document Model Space (1).

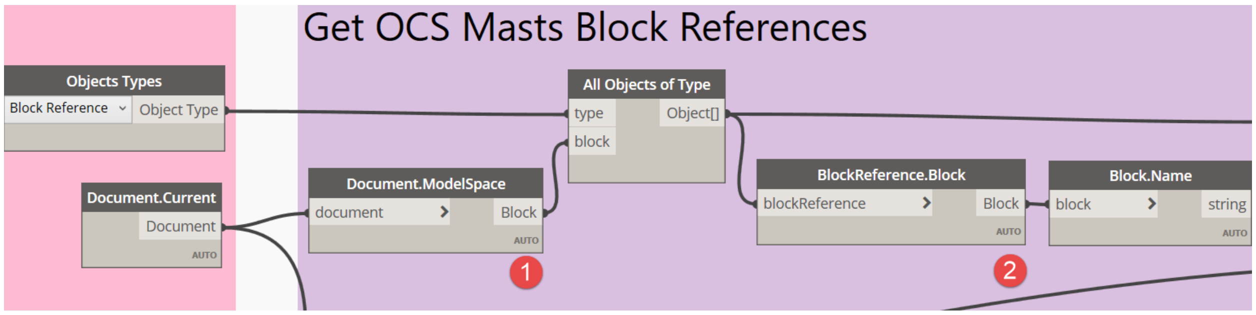


FIGURE 34: SELECT ALL BLOCK REFERENCES IN THE DOCUMENT AND ACCESS THEIR DEFINITION

The Block References don't have a Name property directly accessible, to differentiate them it is necessary to access to the Blocks used to define the Block References (2). This can be rather confusing, but the Model Space and the definition of a Block Reference are the same kind of object.
In AutoCAD, for example, it is possible to insert an external Document drawing as a Block Reference and, by default, it is the content in the Model Space of the external Document that is going to be used as a definition for the Block Reference. The nodes in Dynamo are reflecting the same relationship.
It is possible to use the Name of the Blocks as a "Key" to first group (3) and then filter (4) the collection of Block References. The Block References with the same name are

grouped together and the goal is to select only groups that satisfy a given criteria. This can be done with a list of Boolean values (true and false) that are used to separate an input list into two categories: those that are passing the filter and those that are not. The criteria used in the example is to discard the Block References called "*CONC_TIE*" and keep all the others (in this case these are the OCS Masts we are after).

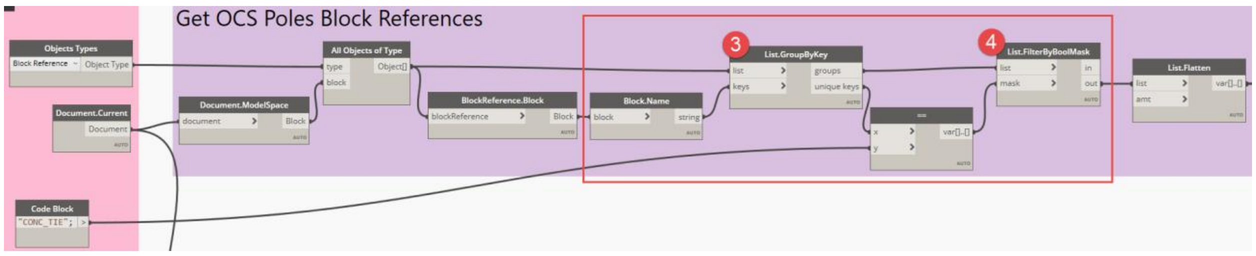


FIGURE 35: GROUPING AND FILTERING DATA IN DYNAMO

The data can be organized into rows where each row contains the Block name, the Station, the Offset, the Elevation and the relative Rotation. The rotation can be obtained measuring the angle between two vectors, for example the Y axis of the Coordinate System respectively of the Block Reference and the Baseline at the same station.

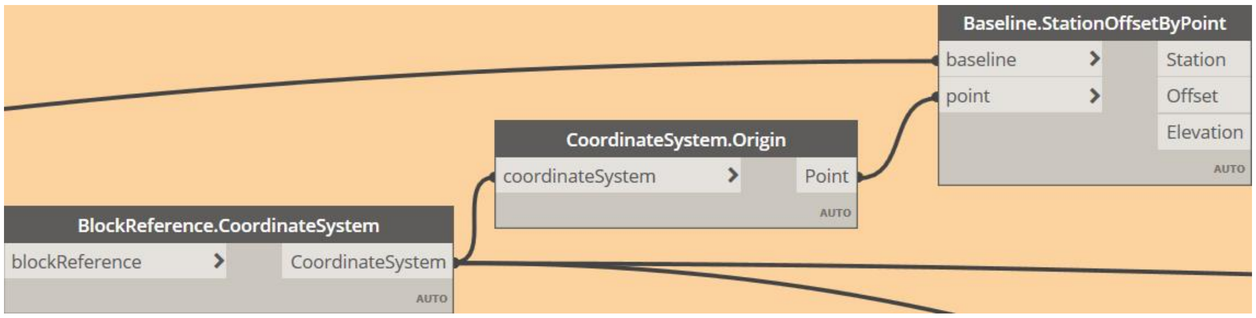


FIGURE 36: LOGIC TO EXTRACT STATION, OFFSET AND ELEVATION FROM A POINT AGAINST A BASELINE

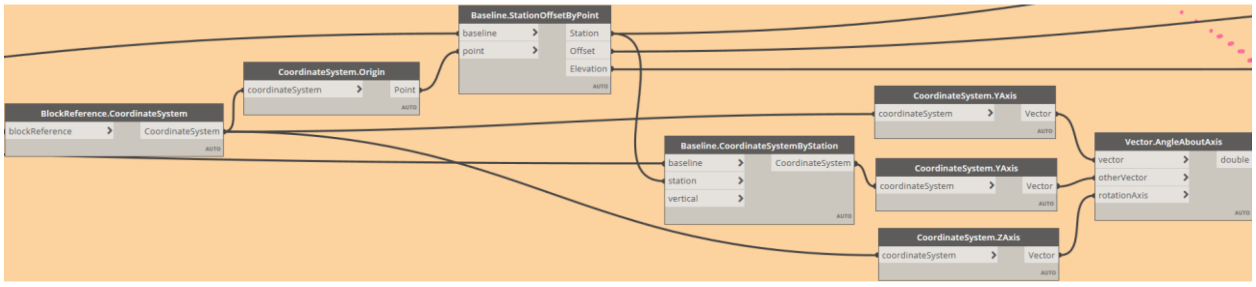


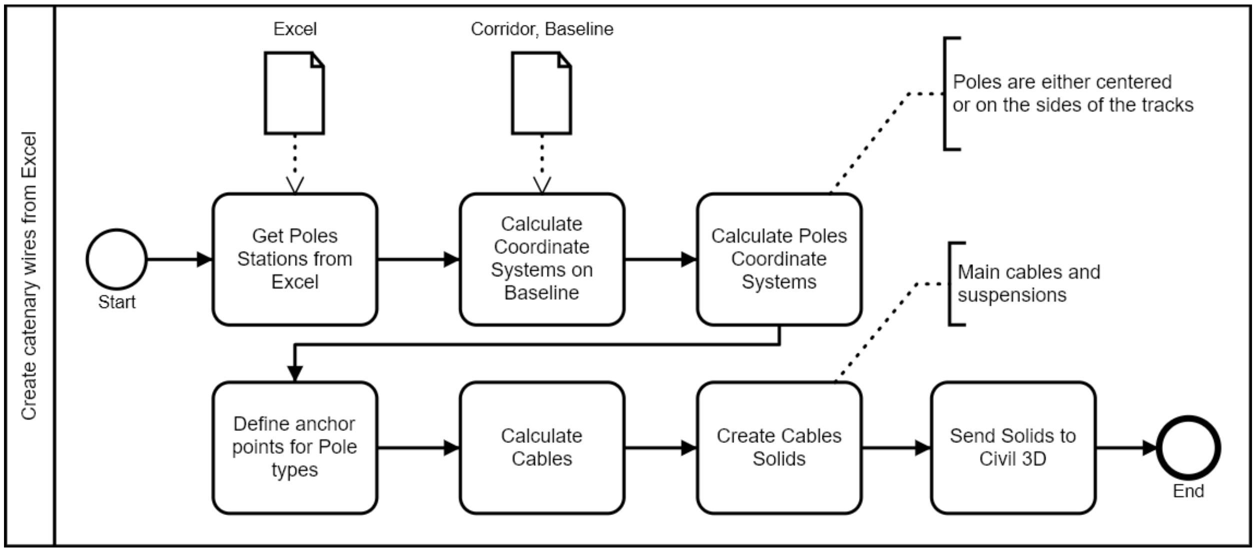
FIGURE 37: LOGIC TO MEASURE THE RELATIVE ROTATION

The next examples are going to only show an overview of how to apply design automation and visual programming for other components.

Overhead Catenary Systems Cables

The key learnings for this example are:

- How to read data from Excel
- How to extract objects from a Block Reference
- How to create solids from Dynamo to Civil 3D



Create catenary wires from Excel
Excel
Corridor, Baseline
Poles are either centered or on the sides of the tracks
Start
Get Poles Stations from Excel
Calculate Coordinate Systems on Baseline
Calculate Poles Coordinate Systems
Main cables and suspensions
Define anchor points for Pole types
Calculate Cables
Create Cables Solids
Send Solids to Civil 3D
End

FIGURE 38: LOGIC FOR THE OCS CABLES

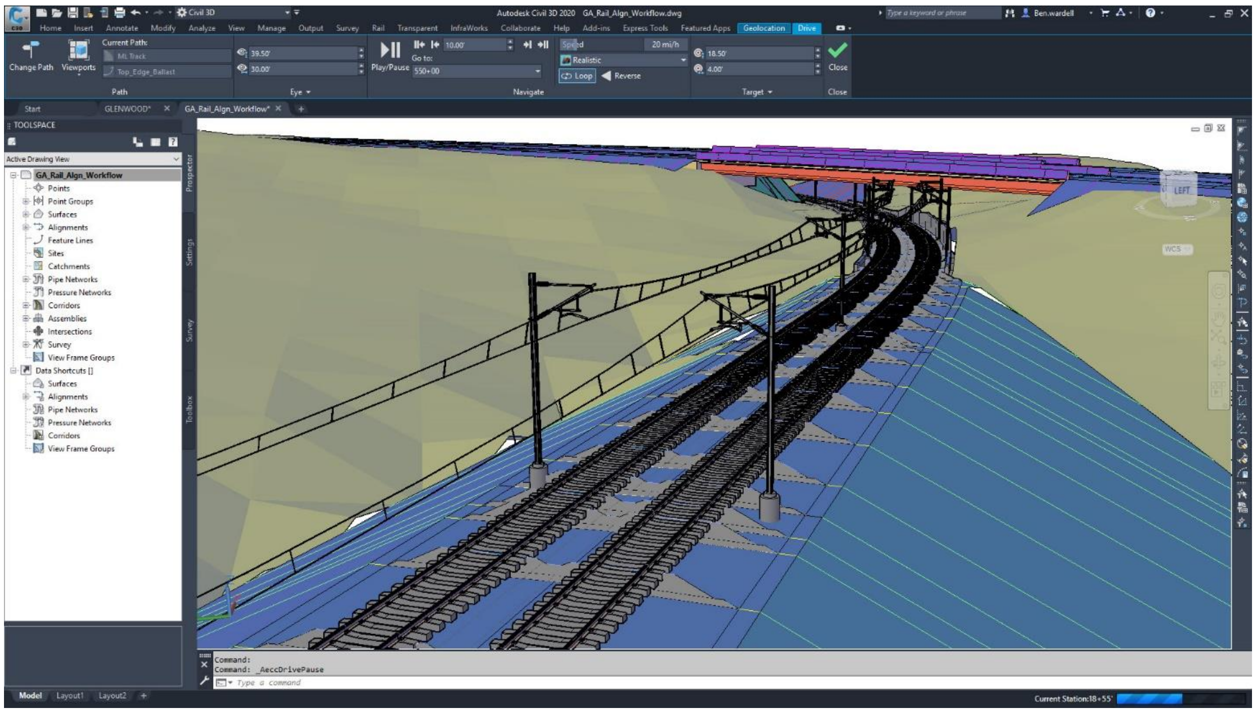


FIGURE 39: FINAL STATE OF THE RAIL MODEL

Roads

A few examples on how to apply design automation and Dynamo for Civil 3D for road design.

Retaining Walls

The key learnings for this example are:

- How to use Coordinate Systems to transform geometry objects
- How to create dynamic relationships to create solids

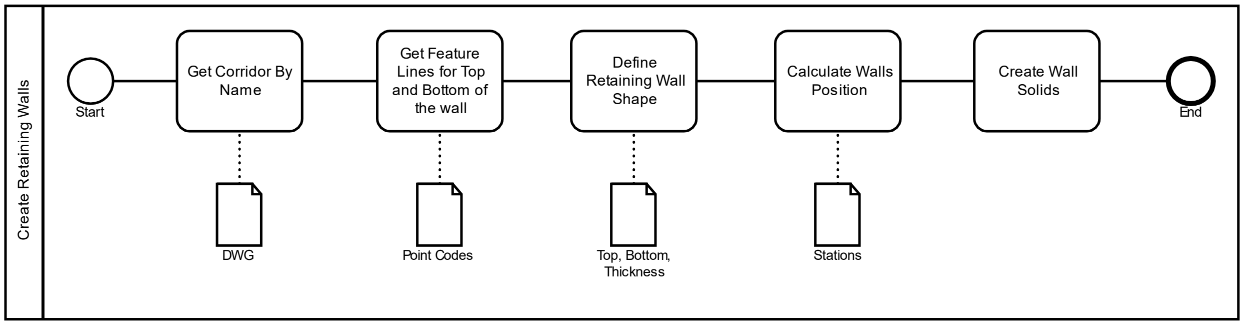
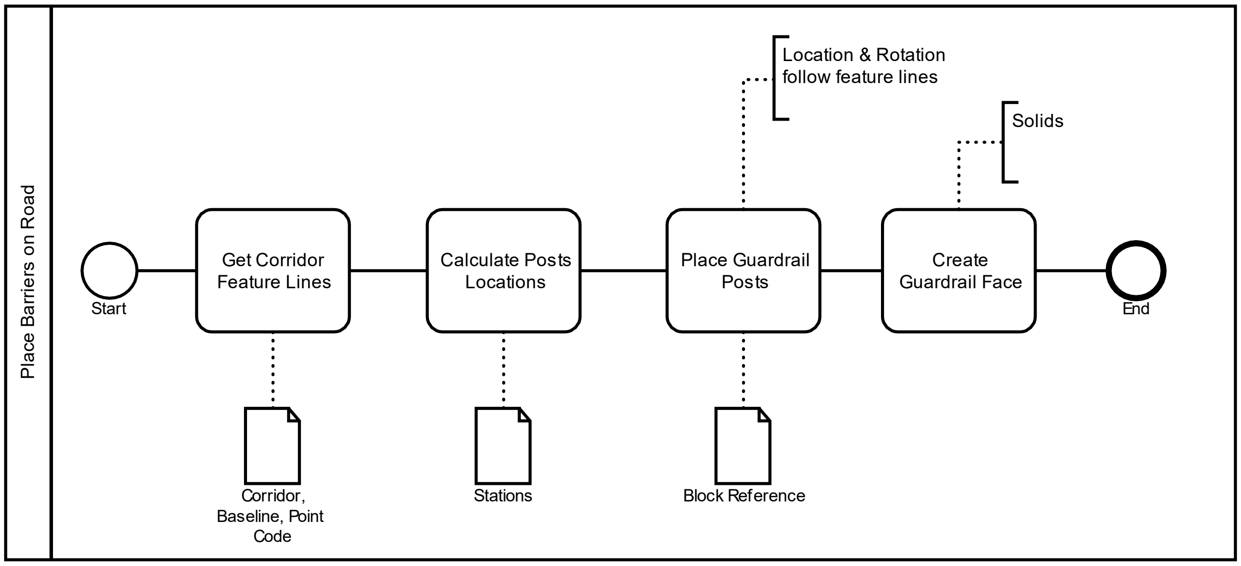


FIGURE 40: LOGIC FOR RETAINING WALLS

Barriers

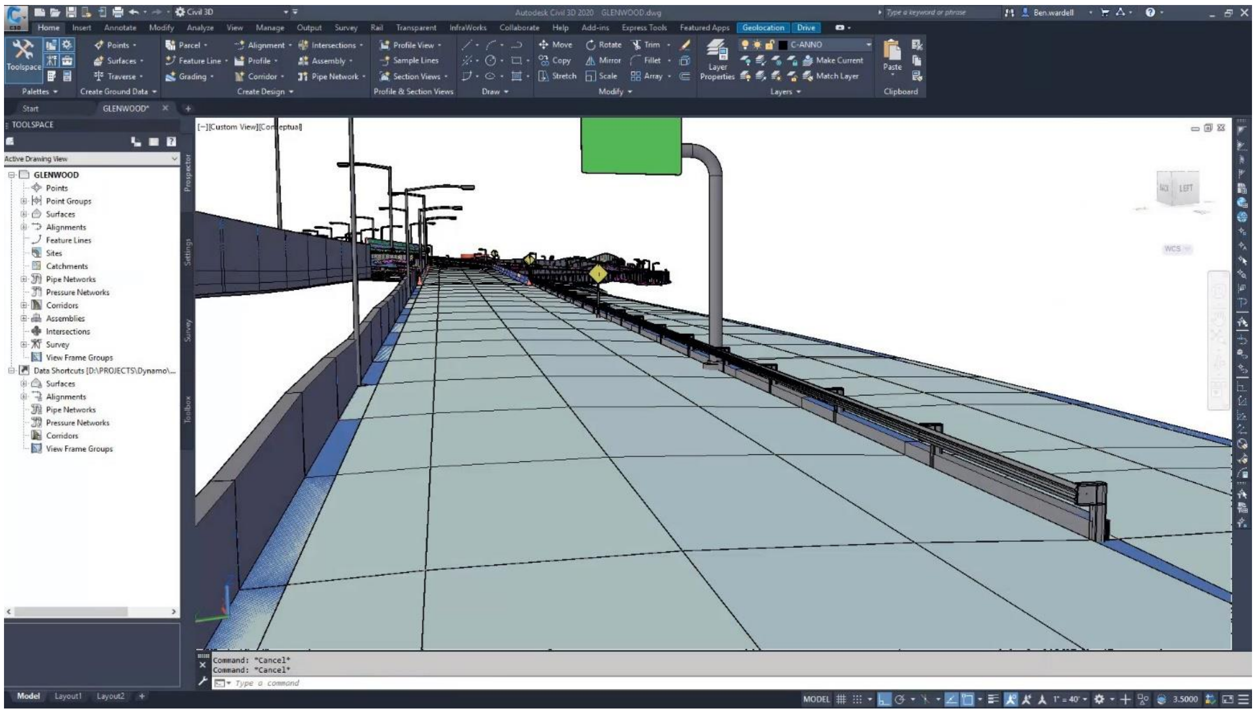
The key learnings for this example are:

- How to combine Block References and Solids



Place Barriers on Road
Location & Rotation follow feature lines
Solids
Start
Get Corridor Feature Lines
Calculate Posts Locations
Place Guardrail Posts
Create Guardrail Face
End
Corridor, Baseline, Point Code
Stations
Block Reference

FIGURE 41: LOGIC FOR BARRIERS



```
Command: *Cancel*
Command: *Cancel*
```

FIGURE 42: FINAL STATE OF THE ROAD MODEL

Site Development

A few examples on how to apply design automation and Dynamo for Civil 3D site development.

Ramps

The key learnings for this example are:

- How to ask the users for inputs (e.g. a Code Block or via a spreadsheet)
- How to create dynamic relationships to create linework

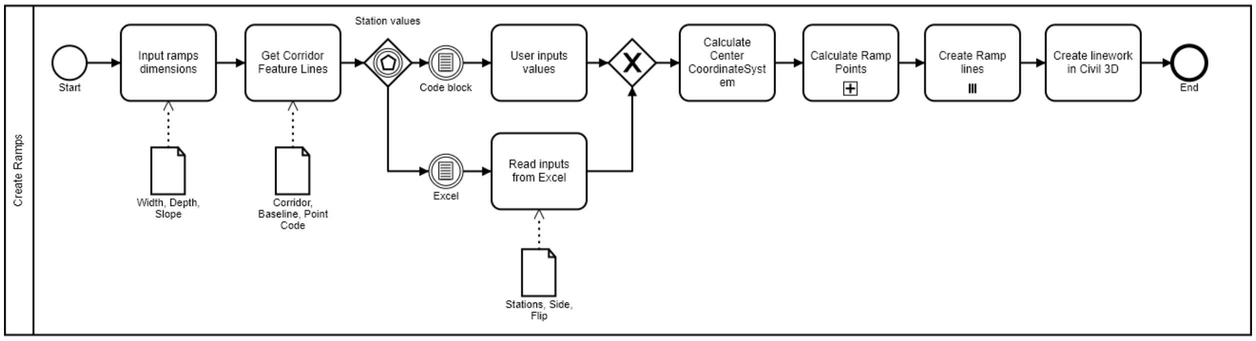


FIGURE 43: LOGIC FOR RAMPS

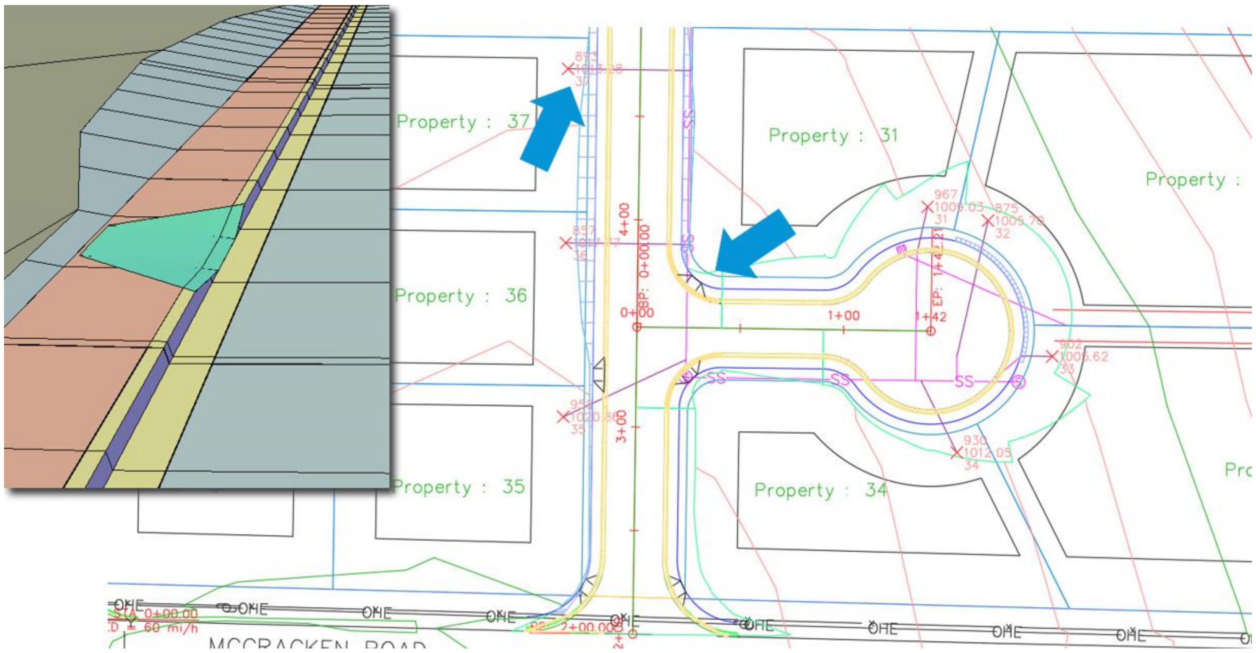


FIGURE 44: RAMPS AND COGO POINTS IN CIVIL 3D CREATED VIA DYNAMO

Point Labels & Report

The key learnings for this example are:

- How to create dynamic relationships to create points, circles and texts
- How to write data to a spreadsheet

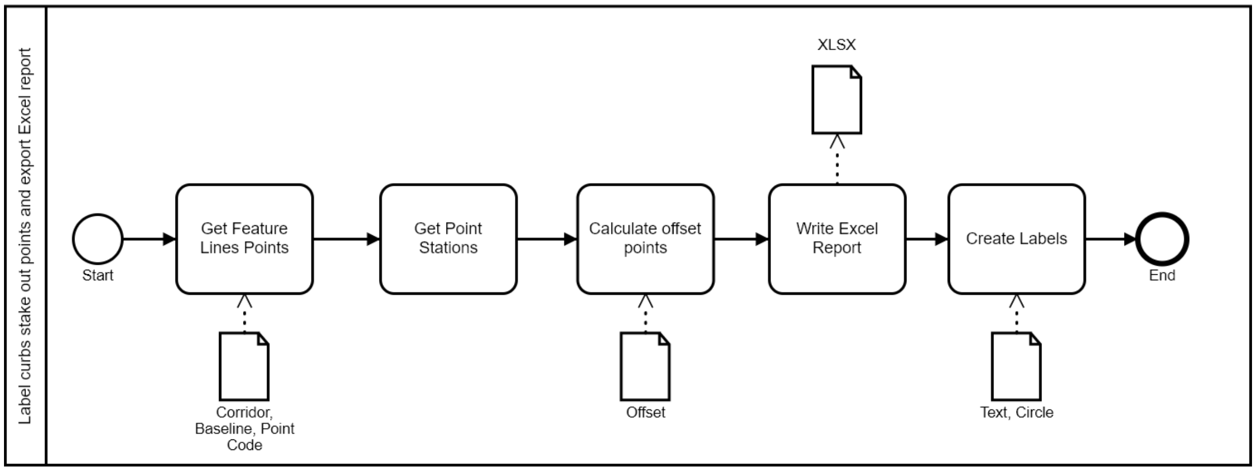
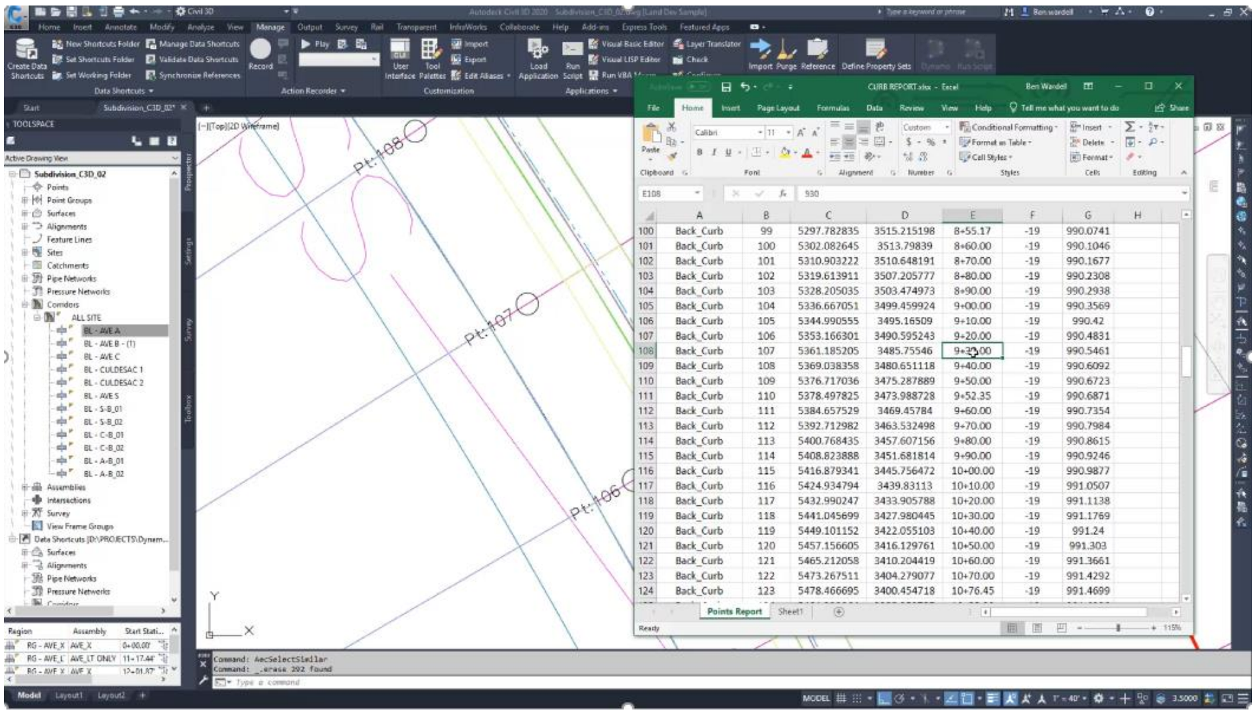


FIGURE 45: LOGIC FOR POINT LABELS AND REPORT



	A	B	C	D	E	F	G	H
100	Back_Curb	99	5297.782835	3515.215198	8+55.17	-19	990.0741	
101	Back_Curb	100	5302.082645	3513.79839	8+60.00	-19	990.1046	
102	Back_Curb	101	5310.903222	3510.648191	8+70.00	-19	990.1677	
103	Back_Curb	102	5319.613911	3507.205777	8+80.00	-19	990.2308	
104	Back_Curb	103	5328.205035	3503.474973	8+90.00	-19	990.2938	
105	Back_Curb	104	5336.667051	3499.459924	9+00.00	-19	990.3569	
106	Back_Curb	105	5344.990555	3495.16509	9+10.00	-19	990.42	
107	Back_Curb	106	5353.166301	3490.595243	9+20.00	-19	990.4831	
108	Back_Curb	107	5361.185205	3485.75546	9+30.00	-19	990.5461	
109	Back_Curb	108	5369.038358	3480.651118	9+40.00	-19	990.6092	
110	Back_Curb	109	5376.717036	3475.287889	9+50.00	-19	990.6723	
111	Back_Curb	110	5378.497825	3473.988728	9+52.35	-19	990.6871	
112	Back_Curb	111	5384.657529	3469.45784	9+60.00	-19	990.7354	
113	Back_Curb	112	5392.712982	3463.532498	9+70.00	-19	990.7984	
114	Back_Curb	113	5400.768435	3457.607156	9+80.00	-19	990.8615	
115	Back_Curb	114	5408.823888	3451.681814	9+90.00	-19	990.9246	
116	Back_Curb	115	5416.879341	3445.756472	10+00.00	-19	990.9877	
117	Back_Curb	116	5424.934794	3439.83113	10+10.00	-19	991.0507	
118	Back_Curb	117	5432.990247	3433.905788	10+20.00	-19	991.1138	
119	Back_Curb	118	5441.045699	3427.980445	10+30.00	-19	991.1769	
120	Back_Curb	119	5449.101152	3422.055103	10+40.00	-19	991.24	
121	Back_Curb	120	5457.156605	3416.129761	10+50.00	-19	991.303	
122	Back_Curb	121	5465.212058	3410.204419	10+60.00	-19	991.3661	
123	Back_Curb	122	5473.267511	3404.279077	10+70.00	-19	991.4292	
124	Back_Curb	123	5478.466695	3400.454718	10+76.45	-19	991.4699	

```
Command: AecSelectSimilar
Command: _.erase 392 found
```

FIGURE 46: LABELS AND REPORT IN EXCEL GENERATED VIA DYNAMO

Stakeout points / Parcels

The key learnings for this example are:

- How to use Python scripting to access the AutoCAD Document
- How to filter the objects by layer or type/name
- How to extract the parcels segments coordinates
- How to use Dynamo polygons containment test
- How to project points on a TIN Surface
- How to create COGO Points

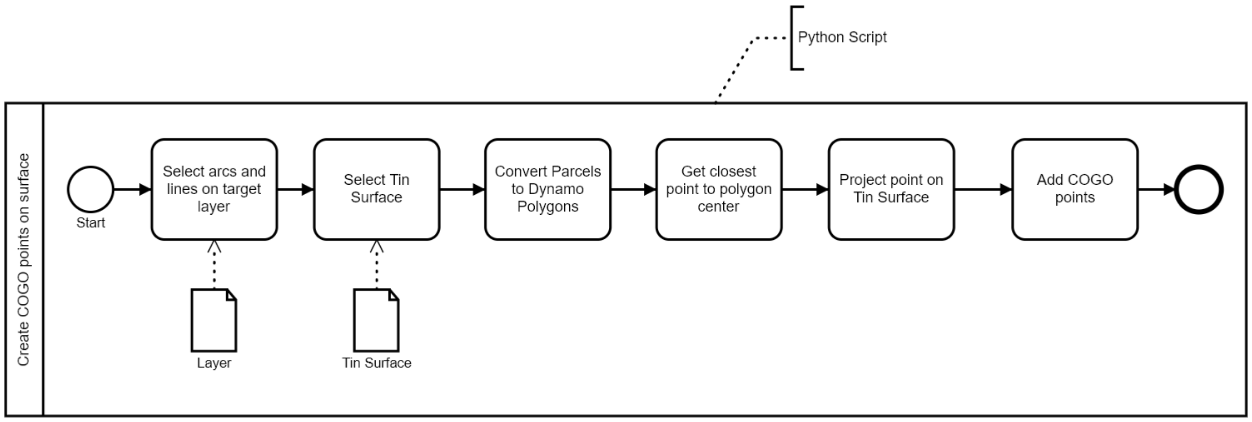

Python Script
Create COGO points on surface
Start
Select arcs and lines on target layer
Select Tin Surface
Convert Parcels to Dynamo Polygons
Get closest point to polygon center
Project point on Tin Surface
Add COGO points
Layer
Tin Surface

FIGURE 47: LOGIC FOR THE COGO POINTS ON TIN SURFACE

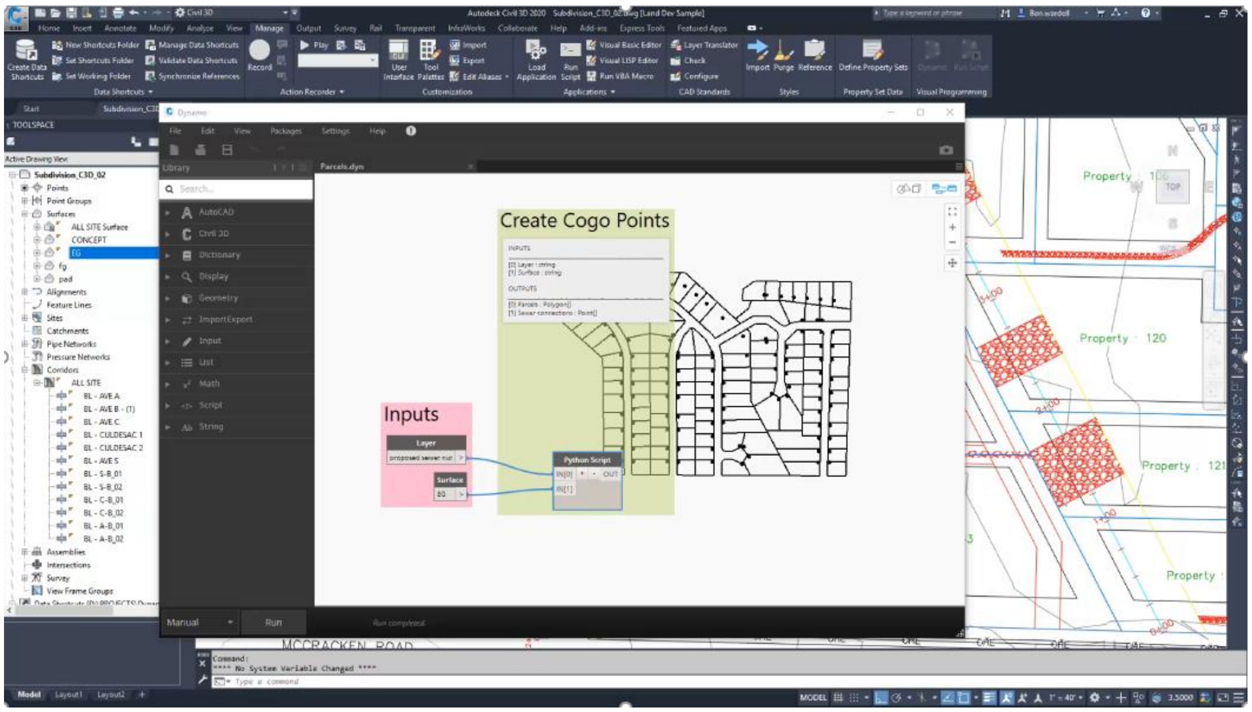


FIGURE 48: PARCELS AND COGO POINTS IN DYNAMO VIA PYTHON SCRIPT

Civil 3D Toolkit

One great way to expand Dynamo capabilities is to create custom packages. A Dynamo package is a collection of custom nodes that can be embed frequently used node structures (a custom node created via User Interface) or brand-new functionalities that are defined via C# in Visual Studio (the so called Zero Touch nodes).
These packages can be found on the Dynamo Package Manager but since Dynamo for Civil 3D has been introduced only in 2019, there are not so many packages dedicated to Civil workflows.

Another fact to consider is that the Civil 3D team that is responsible for the integration of Dynamo has a focus around transportation and specifically rail. That is part of the reason why the initial implementation of Dynamo for Civil 3D is so transportation oriented.

Civil 3D supports more than just transportation workflows and that is why we took the decision to create a Civil 3D Toolkit to enable more disciplines and workflows for Civil 3D users.

In this first release there is a collection of nodes that are extending existing objects or introducing new objects to support new civil workflows.

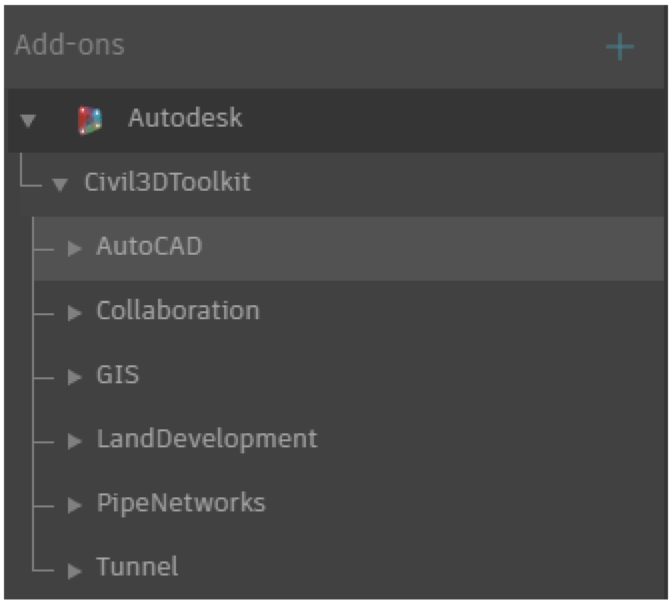


FIGURE 49: CIVIL 3D TOOLKIT SHELVES IN THE DYNAMO LIBRARY

The Civil 3D Toolkit will be available on the Dynamo Package Manager after Autodesk University Las Vegas 2019. For more information and used cases covered refer to the presentation that accompanies this handout.

Python Script for AutoCAD and Civil 3D

Python is a traditional scripting language that is available in Dynamo via a Python Script node. This represent another way to expand what is currently possible through Dynamo.

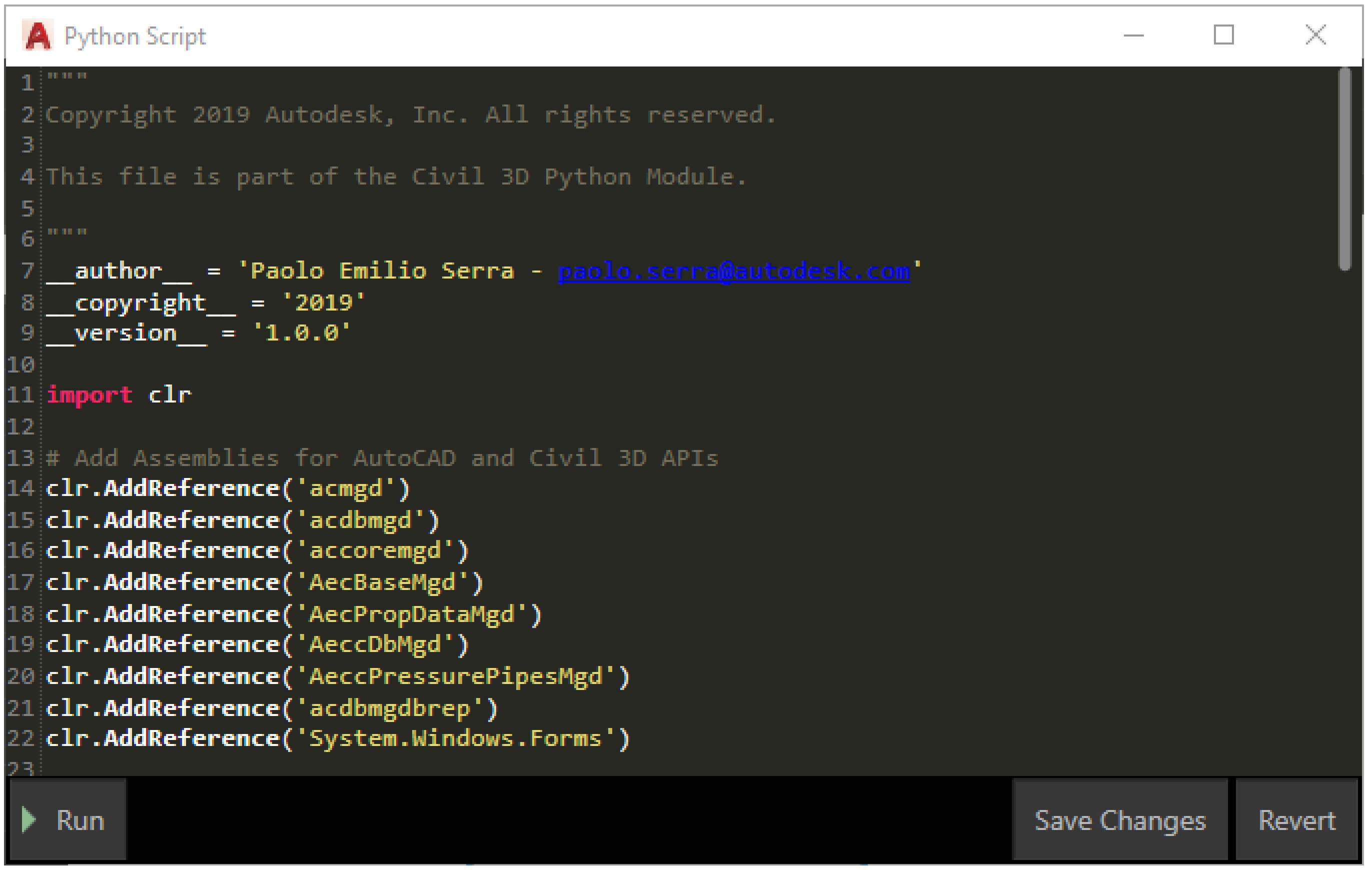
All started with the Autodesk Consulting engagements in which CivilPython was used, that created a lot of use cases around automation for example for the entire life cycle of the Property Sets.

As a matter of fact, that is actually Iron Python, a Python distribution that is compatible with the .NET framework. This framework is very popular, and it can be found as a mean to interact with many software with a public Application Program Interface (API). Autodesk products such as AutoCAD, Civil 3D or Revit have .NET APIs.

Let's clear something right away: Python is currently available with two main language versions 2.7 and 3.7, they are very similar at first sight but not quite compatible.
The Iron Python version available in the Dynamo Core is based on Python 2.7.9, this means that the Python syntax and libraries to be used are those compatible with 2.7.

For more information about the Iron Python syntax you can visit this link to the documentation that covers the .Net integration: https://ironpython.net/documentation/dotnet/dotnet.html

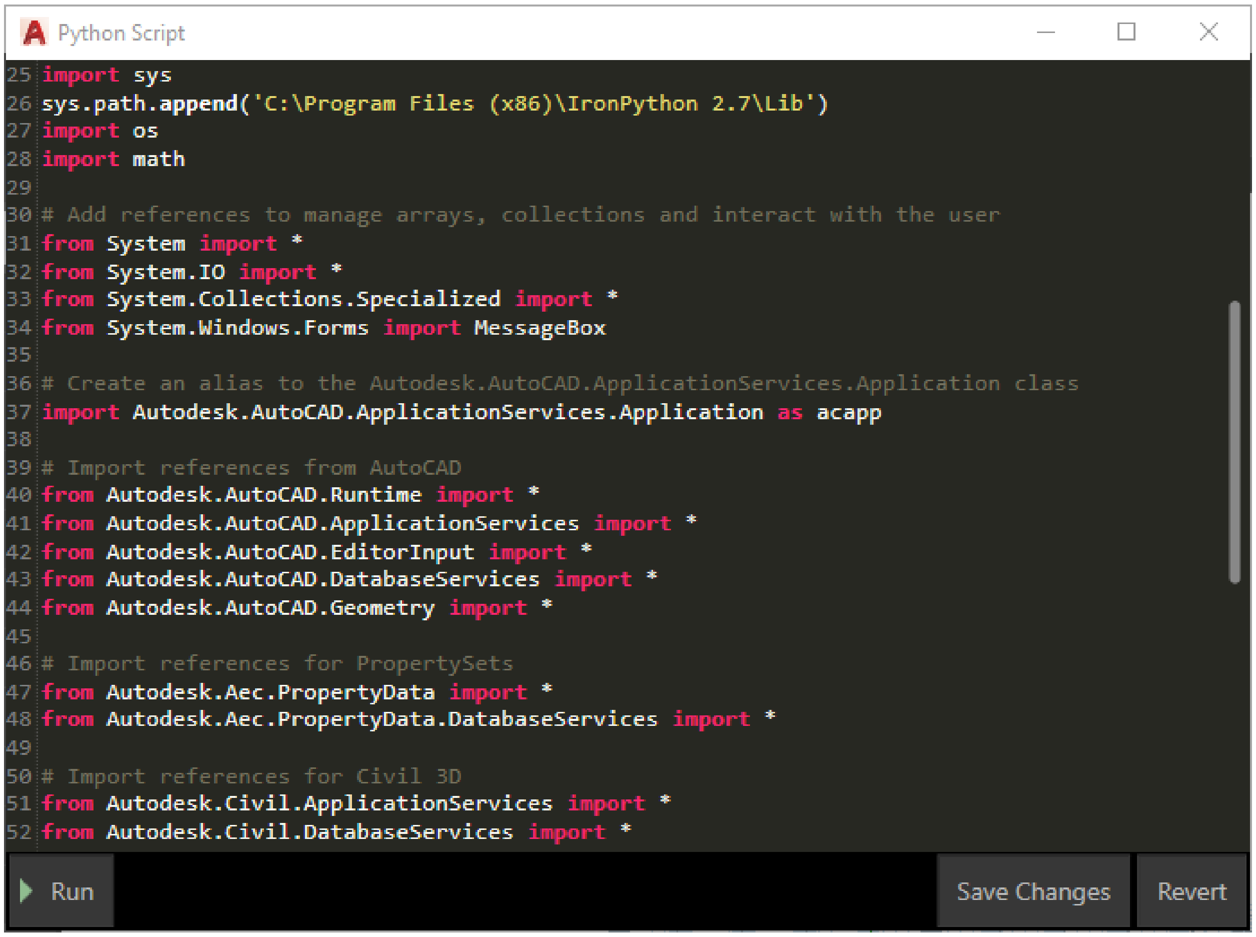
To access a Python Script node in the graph, just double click on it or right click and select Edit.
The Python Script editor appears showing the default template that contains the references to libraries used to interact with AutoCAD and Civil 3D APIs.



```
"""
Copyright 2019 Autodesk, Inc. All rights reserved.

This file is part of the Civil 3D Python Module.

"""
__author__ = 'Paolo Emilio Serra - paolo.serra@autodesk.com'
__copyright__ = '2019'
__version__ = '1.0.0'

import clr

# Add Assemblies for AutoCAD and Civil 3D APIs
clr.AddReference('acmgd')
clr.AddReference('acdbmgd')
clr.AddReference('accoremgd')
clr.AddReference('AecBaseMgd')
clr.AddReference('AecPropDataMgd')
clr.AddReference('AeccDbMgd')
clr.AddReference('AeccPressurePipesMgd')
clr.AddReference('acdbmgdbrep')
clr.AddReference('System.Windows.Forms')
```

FIGURE 50: PYTHON SCRIPT TEMPLATE

After the references have been made available to Python it is necessary to specify what namespaces to recall in the code via the **import** directives. It is also possible to recall specific objects via the **from** directive and even provide aliases with **as** directive. Using the * symbol instead imports all the objects and functions defined in a namespace.

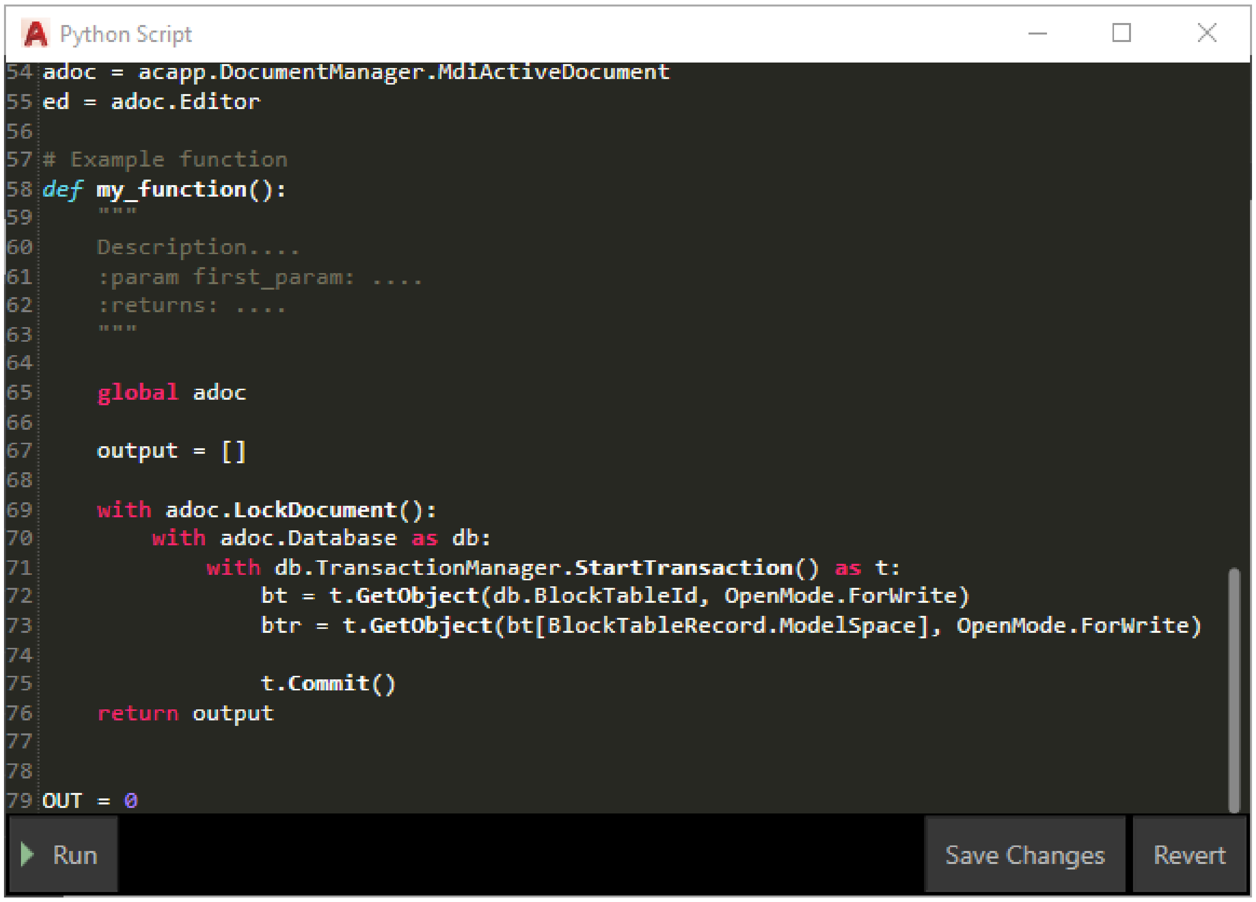


```python
import sys
sys.path.append('C:\Program Files (x86)\IronPython 2.7\Lib')
import os
import math

# Add references to manage arrays, collections and interact with the user
from System import *
from System.IO import *
from System.Collections.Specialized import *
from System.Windows.Forms import MessageBox

# Create an alias to the Autodesk.AutoCAD.ApplicationServices.Application class
import Autodesk.AutoCAD.ApplicationServices.Application as acapp

# Import references from AutoCAD
from Autodesk.AutoCAD.Runtime import *
from Autodesk.AutoCAD.ApplicationServices import *
from Autodesk.AutoCAD.EditorInput import *
from Autodesk.AutoCAD.DatabaseServices import *
from Autodesk.AutoCAD.Geometry import *

# Import references for PropertySets
from Autodesk.Aec.PropertyData import *
from Autodesk.Aec.PropertyData.DatabaseServices import *

# Import references for Civil 3D
from Autodesk.Civil.ApplicationServices import *
from Autodesk.Civil.DatabaseServices import *
```

FIGURE 51: IMPORTING NAMESPACES IN THE PYTHON MAIN SCRIPT

In the template there are a couple of global variables defined to get a reference to the AutoCAD active document and its editor to write messages to the command line.

There is a sample function whose purpose is to demonstrate how to use the Python constructs to interact safely with the AutoCAD document.
The use of the **with** directive, or context manager, that allows to safely interact with and dispose the connection after of the document, its database and the transaction object.
There are three nested levels to safely access to the BlockTable and the BlockTableRecord.
With the BlockTableRecord it is possible to loop through all the ObjectIds in the document.



```
54 adoc = acapp.DocumentManager.MdiActiveDocument
55 ed = adoc.Editor
56
57 # Example function
58 def my_function():
59     """
60     Description....
61     :param first_param: ....
62     :returns: ....
63     """
64
65     global adoc
66
67     output = []
68
69     with adoc.LockDocument():
70         with adoc.Database as db:
71             with db.TransactionManager.StartTransaction() as t:
72                 bt = t.GetObject(db.BlockTableId, OpenMode.ForWrite)
73                 btr = t.GetObject(bt[BlockTableRecord.ModelSpace], OpenMode.ForWrite)
74
75                 t.Commit()
76     return output
77
78
79 OUT = 0
```

FIGURE 52: THE PYTHON SCRIPT TEMPLATE

That is where the automation really begins, in the image above, at line 74. At this point the reader should really get familiar with the AutoCAD and Civil 3D APIs as it is beyond the scope of this document. Nevertheless, the Civil 3D section of the Dynamo forum is a very good place to start for asking questions and finding answers.

In terms of learning resources there is good news and bad news. The good news is that there are A LOT of resources about AutoCAD API examples and a good number of Civil 3D API examples too. The bad news is that very few examples are written in Python as it is not officially supported. This is a perfect opportunity for the community to develop and share some knowledge!

Just a few references for learning about AutoCAD and Civil 3D APIs:

- The *ADN Dev Blog for AutoCAD* (LINK), has great contributions from a lot of Autodesk experts around AutoCAD API
- The *ADN Dev Blog for Infrastructures* (LINK), has a lot of examples for Civil 3D and Map 3D, very useful readings
- Kean Walmsley's blog *Through the Interface* (LINK) has very good examples of how to access the AutoCAD API with .NET, only downside is that they are in C#
- Isaac Rodriguez's blog *Civilized Development* (LINK) has a good collection of examples for the Civil 3D API but it has not been active in a while.

The template can be customized, and the users are encouraged to do so under:
%update%\ Autodesk\C3D 2020\Dynamo\2.4

Creating your custom *PythonTemplate.py* file and updating the *DynamoSettings.xml* at the tag:
<PythonTemplateFilePath>*YOUR TEMPLATE PATH GOES HERE*</PythonTemplateFilePath>

It is possible to run the Python code via the Run button in the bottom left corner of the editor or form the Dynamo user interface.

The next time you want to execute a Python Script node you need to make sure that at least one input that feeds the Python script has changed, otherwise it will not take part to the execution by design. What I normally do is to click on the "+" and "-" buttons to add or remove extra inputs on the Python Script to mark the node as "dirty" so that it will be recalculated at the next execution. If there are a lot of Python Scripts, it is probably easier to close the Dynamo graph and reopen it or change an input at the very beginning.

Best Practices

As a recommendation, develop your Python modules and libraries in an external editor of your choice, I recommend PyCharm Community Edition (LINK).

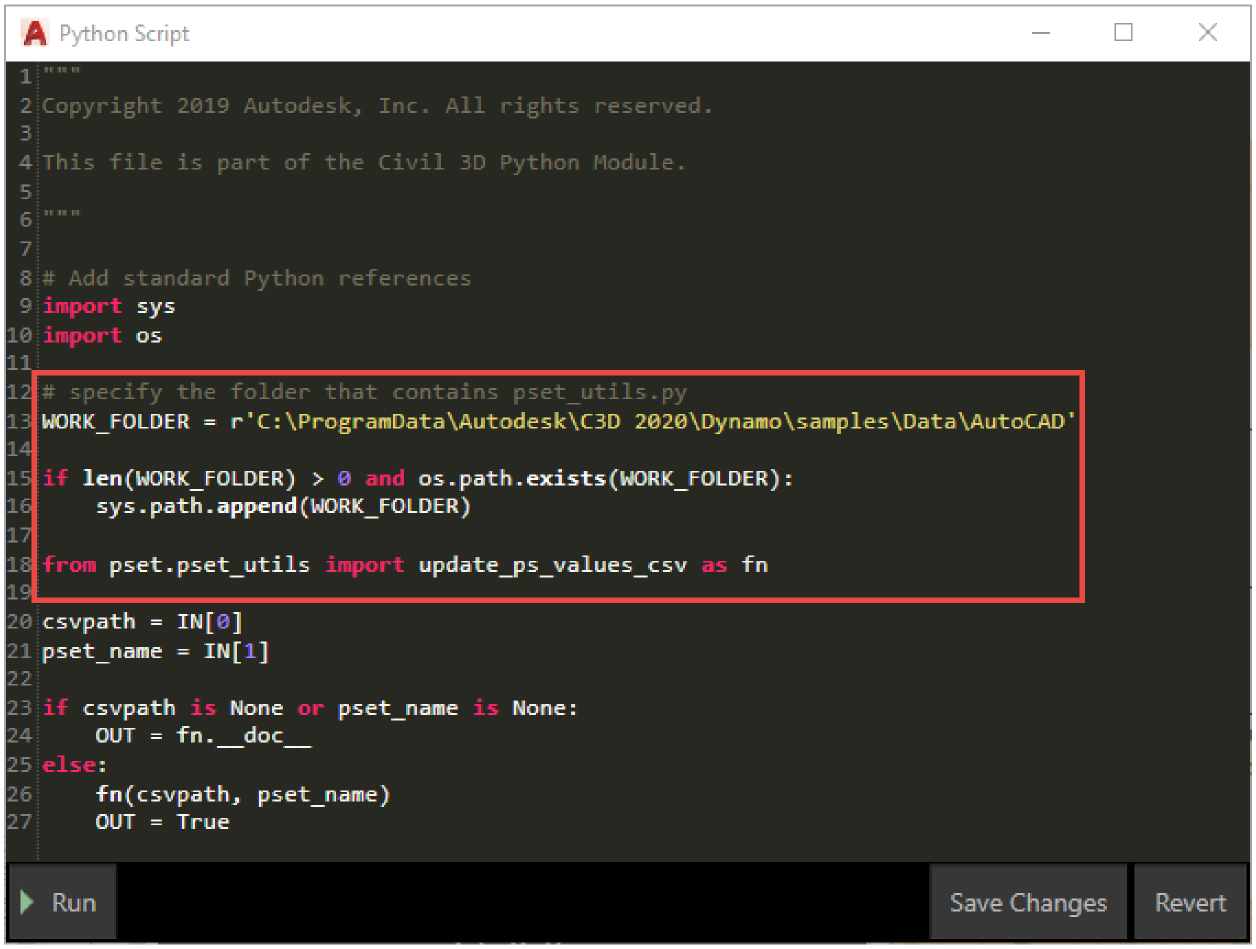
Afterwards it is possible to recall a custom Python code into a Python Script node in Dynamo for Civil 3D. A good example that shows how this is done is the Example 9 shipped with Dynamo around Property Sets.

The whole logic is developed in a couple of Python files under the **pset** folder, they are added to the **sys.path** that contains the list of search paths for modules. In the Python Script code, the objects can then be added with an **import** directive.

There are multiple advantages in following this approach:

- The external Python development environment is more versatile to write and maintain code because it is specifically designed for that and has a lot of features to catch typos, replace names etc.
- This allows to decouple the Python definitions from the Dynamo graph, which means that they can be updated separately without the end user noticing the differences.
- With a centralized Python library, it is easier to propagate changes or add new functionalities available to all Dynamo graphs and to all users.
- It can happen, especially when learning about the API, something goes wrong, and Civil 3D encounters a fatal error and it is forced to close; having the Python code stored outside the Dynamo graph prevents the loss of data in the code.
- It is possible to leverage the __doc__ document string on function and objects to pass instructions and useful information to the end user (see following image at line 24).
- It is possible to take advantage of the **logging** module to create detail reports of the operations that are happening during the processing of the Python script.

In general it is a good practice trying to follow the style guide for Python (the so called PEP8 and PEP257) available under the Python official website.


Python Script

```
"""
Copyright 2019 Autodesk, Inc. All rights reserved.

This file is part of the Civil 3D Python Module.

"""

# Add standard Python references
import sys
import os

# specify the folder that contains pset_utils.py
WORK_FOLDER = r'C:\ProgramData\Autodesk\C3D 2020\Dynamo\samples\Data\AutoCAD'

if len(WORK_FOLDER) > 0 and os.path.exists(WORK_FOLDER):
    sys.path.append(WORK_FOLDER)

from pset.pset_utils import update_ps_values_csv as fn

csvpath = IN[0]
pset_name = IN[1]

if csvpath is None or pset_name is None:
    OUT = fn.__doc__
else:
    fn(csvpath, pset_name)
    OUT = True
```

FIGURE 53: AN EXAMPLE OF HOW TO LOAD AN EXTERNAL MODULE IN A PYTHON SCRIPT NODE

Generative Design for Infrastructures

Dynamo is the gateway to enter a new exciting future of possibilities enabled by a collaborative design process between humans and machines to find optimal solutions to design challenges.
The designer defines the parameters and goals for the design and provides the feedback to the process. The computer generates design alternatives, analyzing their performance to rank the options by their fitness and evolves the design over time based on the previous iterations and the feedback received.

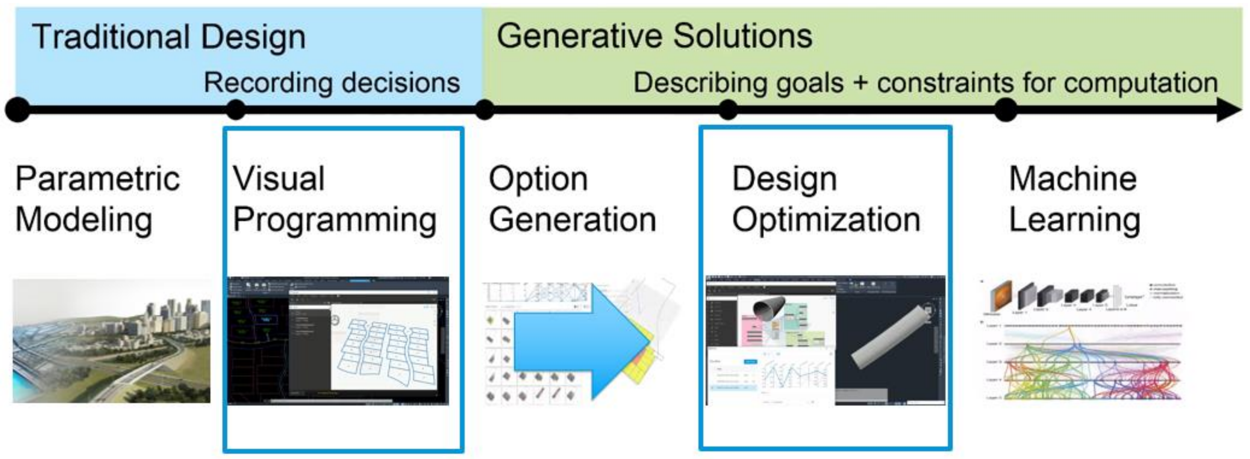


FIGURE 54: THE JOURNEY OF THE EVOLUTION OF DESIGN

Currently, Autodesk is developing a tool called **Project Refinery** that can be found in the Autodesk Feedback Community (LINK).
Project Refinery is leveraging computational design strategies and visual programming through Dynamo to investigate the design space and return solutions that can ben then selected by the designer and curated inside a parametric modeling environment.

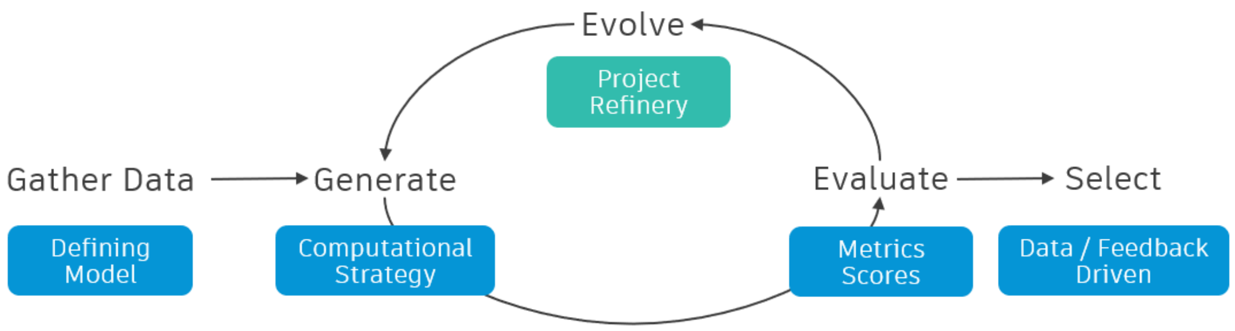


FIGURE 55: GENERATIVE DESIGN APPROACH

The first use cases that come to mind in the Infrastructures spaces are related to the definition of transportation Alignments with criterion such as defining the shortest path, avoiding obstacles, respecting constraints related to the speed and so on.
The system must be able to generate an Alignment object and to allow the define a scoring system (e.g. the criterion listed above are all measurable, which makes things very easy to implement in

Dynamo). These scoring system collects Objectives for a Generative Design study that can be Maximized or Minimized.

Introducing the vertical Profile as another object to generate calls for additional Objectives such as maximizing the line of sight or minimizing the amount of dirt movement or the cost for retaining walls or maximizing the comfort and safety for a driver, etc.

Another example could be related to Pipe Networks in an urban context with potable, reclaimed water and sewage. In countries like Sweden this is not a trivial task as there are constraints at the top (the potable water must be below the freezing line) and at the bottom (the bedrock is relatively high, and designers try to avoid the use of dynamite).

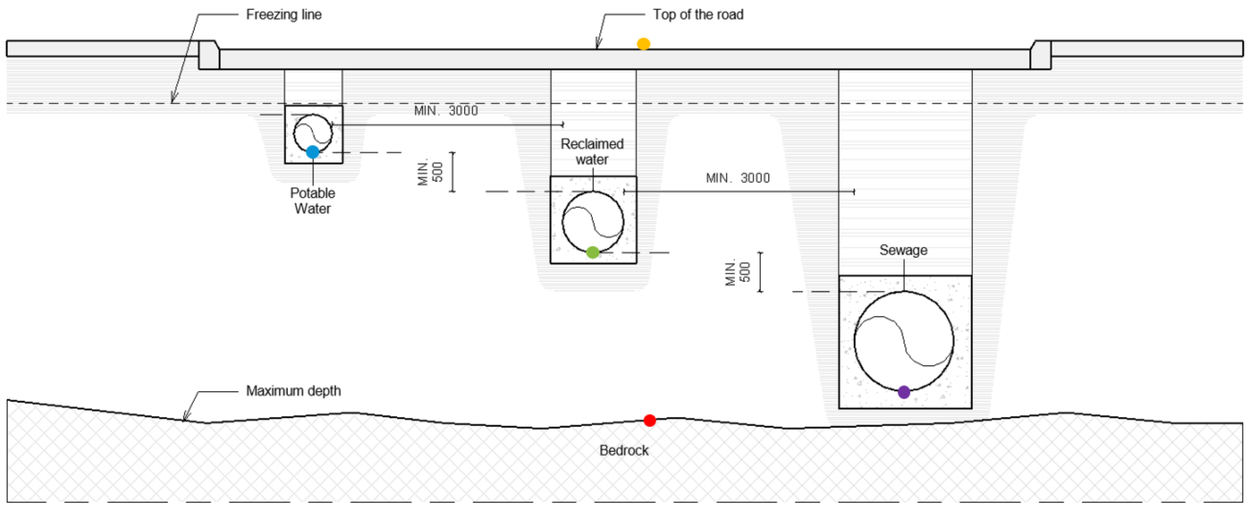


FIGURE 56: DIAGRAM REPRESENTING THE CONSTRAINTS FOR PIPE NETWORKS IN AN URBAN CONTEXT

Generative design can help explore a lot of design alternatives to see if a solution is possible, minimizing the depth, the crossings of different networks, the overall costs, maximize the accessibility, etc.

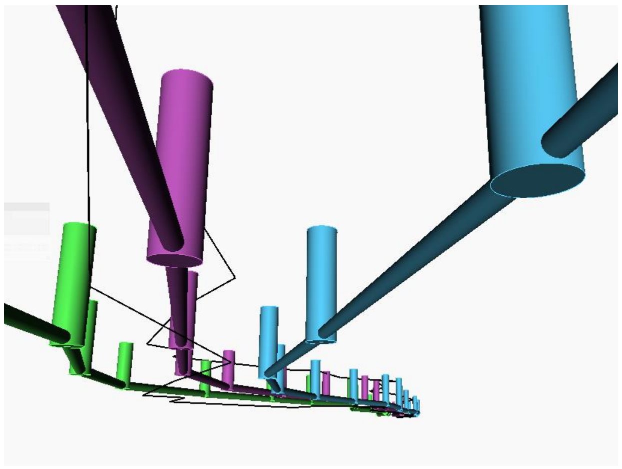


FIGURE 57: A VIEW OF A PIPE NETWORK GENERATED IN DYNAMO

Currently the Project Refinery Team has been working on integrations with mature hosts for Dynamo such as Revit, but it is also available for the standalone version of Dynamo called Sandbox.
Refinery is a View Extension for Dynamo and as such it comes as an external package; this can be copied from let's say the Core folder for Sandbox into the packages for Dynamo for Civil 3D.

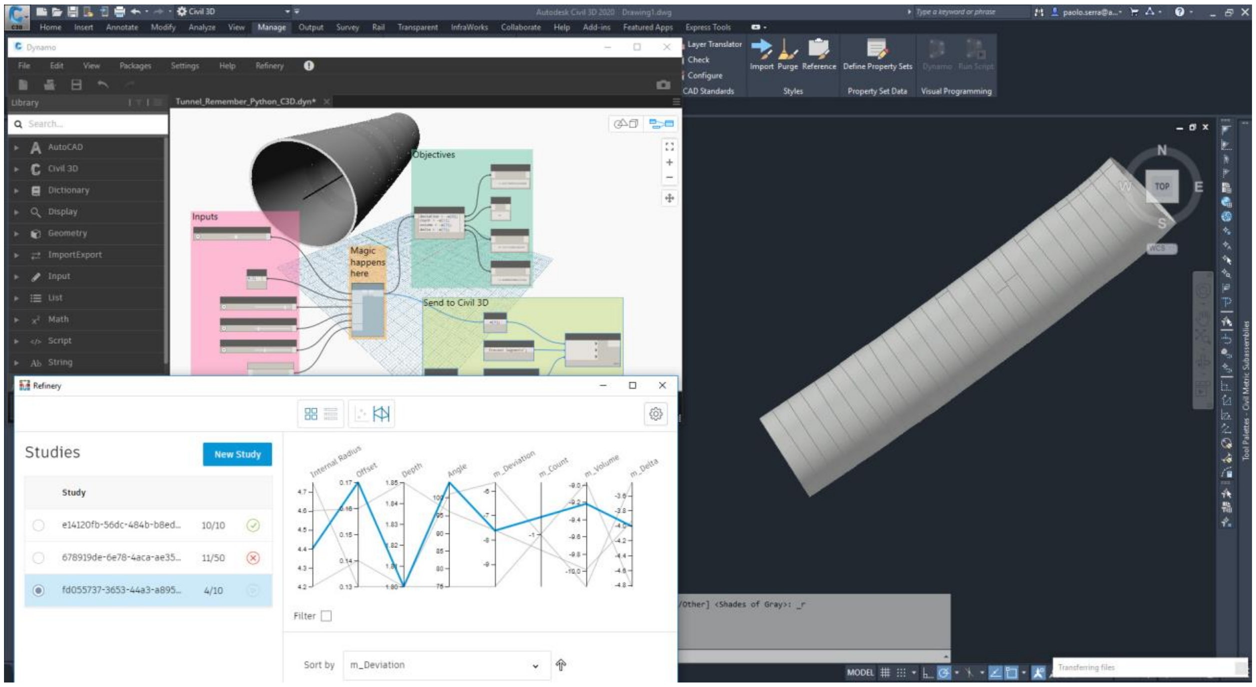


FIGURE 58: AN EXAMPLE OF PROJECT REFINERY IN DYNAMO FOR CIVIL 3D