AUTODESK. UNIVERSITY

CES322839

Custom Civil 3D Subassemblies – Why Would I Need That??

Matthew Dalton
Digital Engineering Lead
WSP

Learning Objectives

- Understand why Subassembly Composer for Civil3D is such a powerful tool for their business.
- Learn why a custom subassembly pack should be something critical to their company standards kit
- Learn how subassemblies can be used to drive more consistent outputs to drawings, models and asset information.

Description

Infrastructure BIM development is racing ahead, and every year new software versions and releases come out at break-neck pace to make sure that we all deliver the best possible output to our clients in highways design. This class will show why custom subassemblies instead of the standard content are a must-have for any business or project where design, drawing and model consistency are important. We will cover how to control and maintain model consistency, to reduce user-error, to drive drawing outputs and even how you can use subassemblies to automate asset data extraction.

The class will overview Subassembly Composer creation, some tips and tricks that have been learned on the UK's current largest infrastructure design project (HS2) and will cover how subassemblies can be used to drive, standardize and reduce error. We will demonstrate why businesses need to think seriously about subassemblies and make them a key part of their internal standards and practices

Speaker(s)

Matthew Dalton is currently a Digital Engineering Lead in WSP UK in the Transport and Infrastructure business. He has worked in infrastructure design consultancy for over twenty years with experience in environmental, water, mining and highways sectors. Matt has been a speaker at Autodesk Civil3D and Infrastructure user group meetings in the UK, as well as presenting to Welsh Government on use of Augmented and Virtual reality use on highways BIM projects. Matthews current role within WSP is to continuously drive, author and support improvements in Digital design across all of our Autodesk suite including Civil3D, Infraworks,

Navisworks and other software. This includes content creation, management and workflows, evaluation of new tools and developments, mentoring staff, promoting best practice and driving design automation as we move into the future of BIM for infrastructure.

AUTODESK. UNIVERSITY

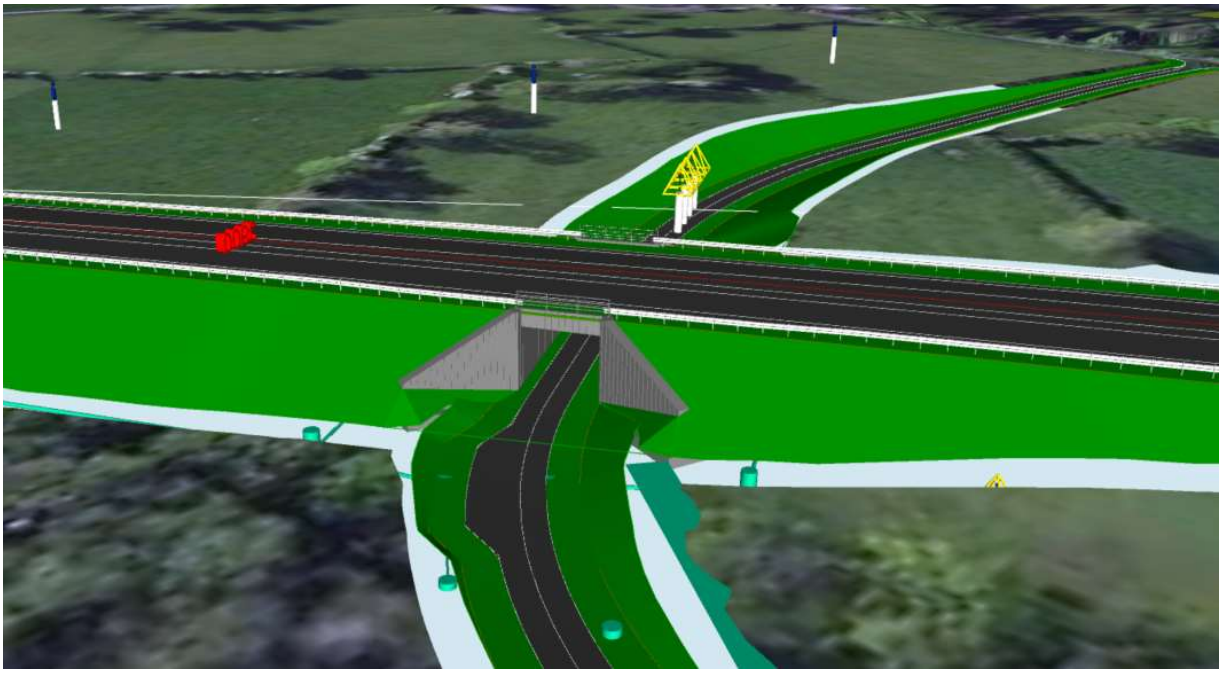
CONTENTS

Common projects, Common Deliverables, Common Engineering

So, we've all worked on a range of projects, right? Some are smaller, local projects with simply junction layouts and others can be much larger, 'major' highway schemes that could have grade-separated junctions, major intersections, roundabouts, and often with complex drainage or other requirements. The common thing to almost all major civil projects in 2019 and looking beyond, is now BIM has moved from a concept clients don't understand to one where they not only expect it, but their requirements and demands are surging forward at breakneck pace. All of which is being accompanied by a need to reduce costs, demonstrate ever-higher efficiency savings and provide end clients with more and more data-rich models. Furthermore, clients are becoming far more tech-savvy and also looking for GIS content as part of handover packages, often for them to use in their own Asset Information Model systems, or what we were up to a year ago referring to as the 'digital twin'.

Within this process, most of us will be familiar with the 'common deliverables' for most all major infrastructure projects, namely:

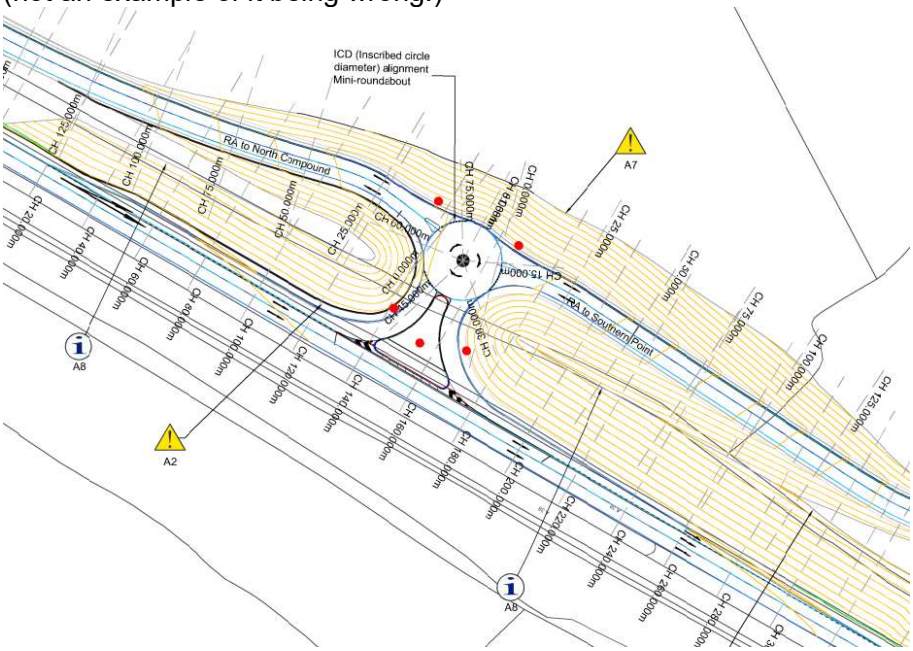
- Traditional CAD drawings
- Plan view 'string' output files (corridor feature lines, grading feature lines, alignments, drainage networks etc.)
- 3D Solids for use in clash detection and in a federated Project Information Model (PIM)
- Setting out information (feature lines, points)
- GIS outputs (to link to client Asset Information Models – Digital Twins)
- Asset Data



Civil 3D ‘out of the box’ and Common project issues

Moving from these standard and familiar deliverables then, it’s in these areas that we see some of the most common ‘mistakes’ within a working project. Especially on those where you have large numbers of modelling staff who often will be working on multiple projects, occasionally with different client standards. Some of the most common errors we see within the Civil 3D environment for example include feature lines going onto the wrong layer or line type; 3D solids requiring post-processing from the initial creation to swap them to different layers; link codes for hatch areas not behaving correctly and also on occasion, the corridor model itself not behaving quite as we would expect where a different subassembly has been used. All of these common errors are easily made, familiar and entirely human mistakes often caused by something as simple as a typo error when adding point code behavior or where a client asks for something more detailed in the solid model. However, when we at WSP started to look really into these small errors, we found there was a significant impact on our projects and a significant impact on staff at several levels. Checking, re-works, reviewing and post-processing all add up to a fairly significant cost, even for those of us with rigorous CAD standards and verification processes. As we all know, “to err is human”!

(not an example of it being wrong!)



Custom Subassemblies – So what can they do that's beyond conventional?

So, let's get into the real meat of things, why you're all here and why I've been fortunate enough to convince my bosses to let me come all the way here to the US to this amazing venue to share with you some of what we have learned over the last two years. I'd like to take a moment to acknowledge a few people who have helped massively with this project. First is Autodesk consulting who were able to step in and help with delivery of the initial set of subassemblies when we had some project time restraints and who's support was much appreciated. Secondly would be Ian Philpott who was hoping to be giving this talk with me but was unable to attend, his knowledge and experience of all things Civil3D is almost unrivalled in the UK. Last but not least, would be anyone and everyone who contributes to the Autodesk user forums, a great source of knowledge for anyone starting out in subassembly composer!

The basics – Point, link and shape codes

As I'm sure most of you are aware, Civil3D used subassemblies to primarily drive three key 'codes' in order to produce drawing and corridor models and can be varied in their behaviors using 'Code Set Styles' within a Civil 3D '.dwg' file.

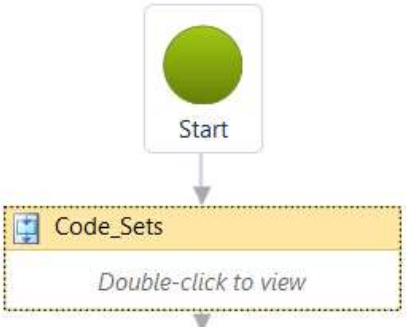
Point codes – Which drive point and line outputs such as corridor feature lines
Link Codes – Which are used to create surfaces and can also be used to create hatch patterns for given areas within a code set style
Shape Codes – Which Civil 3D uses to drive the 3D solid elements of a corridor model and can also be utilized when outputting a corridor to 3D solids.

In a traditional 'out of the box' subassembly such as those included with a country kit, these are all controlled by manual entry of the information by the engineer or designer and their functions are reasonably limited.

Next Step – Keeping it simple!

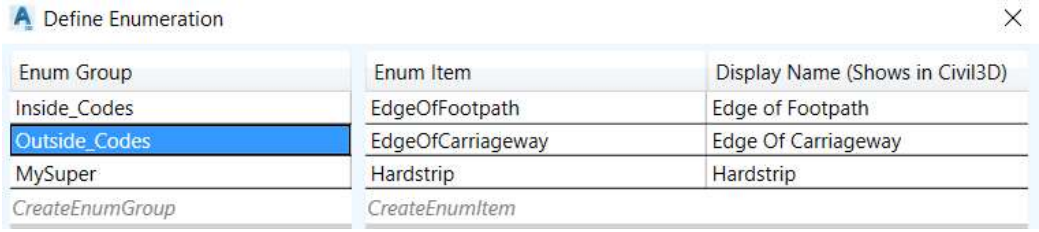
So, one of the first things we considered, was how to simplify subassemblies in a way that means we could reduce the likelihood of error with the most simple part. The point code. One of the things we decided early on was that we wanted to try and reduce the need for modellers on major projects to be editing and changing code set styles as we took on and started using reference templates. So, point codes needed to have three functions built-in to them, firstly we wanted to automatically include the 'side' to the point code so that we would be able to differentiate between carriageway feature lines when driving GIS outputs. Secondly, we decided to give the modeller the choice over which codes to use as part of the subassembly/assembly creation and finally, to give the modellers a choice over which code was applicable while taking out the need for manual entry and therefore effectively hard-coding in the point codes eliminating typo errors. The aim, to create an approach common to all of our subassemblies which was simple, easy to teach, easy to use and gives a reliable and consistent output.

The layout:



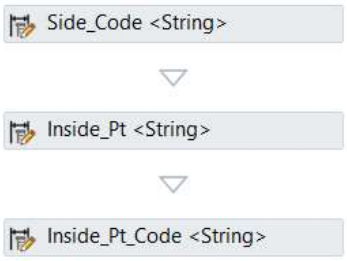
I like my subassemblies neat and tidy, so I use sequences as a good way to ‘group’ objects and variables that will form part of a set within a subassembly. It’s not essential, but I find it a better way of setting things out that will be easier for others to pick up and for me to edit as well if I haven’t looked at a particular subassembly for a while.

Enumeration lists:



Enumeration lists are simply brilliant, and you can use them for a whole range of things from potential output values for asset data to hard-coded point code options. In this example from our footpath, we set three values each of which were most common for users to choose from. Then, how do we set these to be selectable for the point code?

Create a parameter for your enumeration list. Then create a string variable and set that to read the parameter name and use the '.value' function. We can then use a second variable to link the side and name together which is our code variable.



And the code we use?

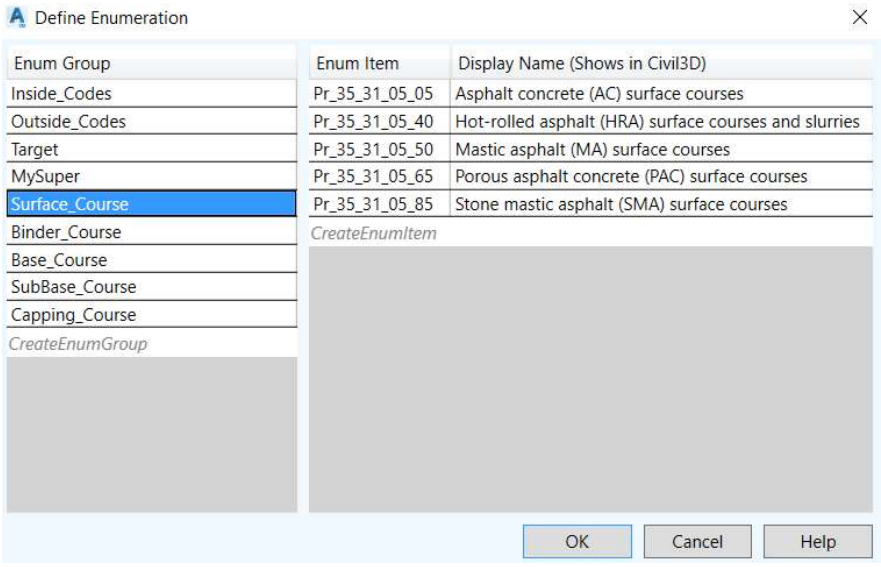
This allows us via a yes/no parameter to choose if the point code is to be used or not, adds the side code automatically to the output and sets the point code to whatever value we choose from our enumeration list. Simple, easy to use and locks the outputs to a value already recognised in the template so you're not going to miss layers when you run your GIS outputs, you're going to be able to spot any sections of the corridor that aren't on the correct layer and you will be able to guarantee quality on your drawing production!

Getting clever – Linking it all up! Complex link codes, shape codes and how that affects your 3D solid modelling

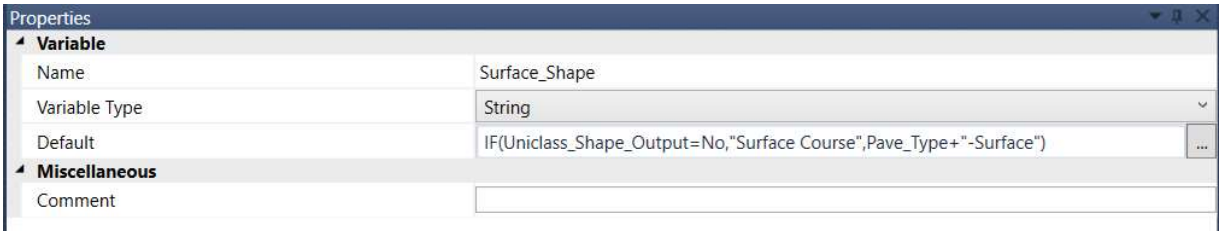
So, we've demonstrated how you can use point codes to do something really basic but that gives you a big effect on your outputs and reduces your error and re-work costs, but taking that similar logic and applying it to link codes and even to shape codes what else could we do? Well, firstly and as a standard feature we can simply 'lock' our link codes to a simple default value by just creating the link code as a string variable.

Alternately, you could set an enumeration list in exactly the same way as we did with the point code if you wanted to have a selection of potential codes to choose from. All of this you can apply to link codes and even to shape codes if you prefer.

However, when we think beyond the standard methods most people use link and shape codes for which is to create surfaces or 3D solids and start thinking about drawing production, then you realise you can use these things a little more intelligently again. Namely, you can link your shape codes to a layer convention via an enumeration list and also tie that list to link codes to create specifics you can use to drive hatch patterns, for example on a carriageway subassembly which we use to drive our paving type drawings! To do this, we created an enumeration list, with the variable in our case set to match the UK standard layering convention, uniclass…



Map that to a string variable just as we have before using a variable with the '.value' function and then we can tie that up to our link and shape codes, by creating another hard-coded variable, for example called 'Pave_Type' and then mapping that to our shape object as shown here using a fixed variable for shape code.



As you can see, we can use the 'IF' function to tell the shape code to work in both standard form or layer form if required. We effectively tell the shape code that if the 'uniclass shape output' parameter is set to 'yes' then it should use the value from our enumeration list, in this case with a suffix to act as the descriptor on the shape code.

Now if you're sat reading this and scratching your head, why would you want to drive layer information to a shape code? Then the answer lies in Civil 3D and how it creates 3D solids. You can choose as an option in Civil 3D (and set the variable to match in your templates) to output to solid using the shape code as the target layer. For anyone who's worked on a BIM project where you have to output solids regularly and then spend time post-processing these to a layer convention, if you're not doing this already then, I guarantee your modellers will be pulling your arms off for it!!

The future – Leveraging your subassemblies and your corridor models to have dynamic 3D solid models with automatic asset data extraction and model federation.. Yes, really, no more property dataset manual entry or clever script writing!!

Ok so that's a really long section title, probably too long but I can't help it because this is the bit that I think is really exciting for subassemblies. To this point really all we have done is used some of the functions in subassembly composer to help standardise outputs, simplify use around point and link codes in order to try and make the design process more efficient. There is, however, something else you can do with subassemblies using the .dwg API that we are using at WSP which I think you will love (assuming you've not already found it!)…

Firstly, when we look at a dwg file in which we have been working on a corridor model the API reveals that corridor models have a set data tree…

```
- Corridor: TestCorridor - (1) (2319600249968)

  - Solid3d:  (2319600253072)
    *A1:
      CorridorName: TestCorridor - (1)
      CorridorDesc:
      BaselineName: BL - M-Mainline Dual Carriageway-1 - (2)
      HorizontalBaseline: M-Mainline Dual Carriageway-1
      VerticalBaseline: M-Mainline Dual Carriageway-1 - Training_Topo - SP 2
      RegionName: RG - TestAssembly - (1) - (2)
    *A3:
      CodeName: Topsoil_Shape, Grass
      Side: Left
      RateItem:
      ClassificationCode:
      AssemblyName: TestAssembly - (1)
      AssemblyStartStation: 0.000m
      AssemblyEndStation: 6711.992m
```

If we further expand this tree, you can also find that the assembly is displayed, as is the subassembly and interestingly so are all of the parameters that are set in that subassembly…

```
  - Assembly: TestAssembly - (1) (2319600249824)

    - Subassembly: Subassembly3 (2319600249952)
      Subassembly Properties:
        Verge Grade: -0.025
        Sub-Surface Depth: 0.3
        Topsoil Depth: 0.15
        Verge Width: 2.5
        Formation Link Code: Formation
        Inside Point Code: Verge_In
        Outside Point Code: Verge_Out
        Subsoil Shape Code: Subsoil_Shape
        Subsoil Top Link Code: Subsoil_Top_Link
        Topsoil Base Link Codes: Topsoil_Base_Code
        Top Link Codes: Top, Verge
        Topsoil Shape Code: Topsoil_Shape
        Verge Sub Material: Sub-Soil
        Side: Left
        Verge Top Material: Grass
```

Now, when we look at the corridor solids, we see the same hierarchy displayed only now the subassembly information isn't present as the solid doesn't carry this through. However, since the combination of the data tree is constant and unique for every single solid in a model file or across multiple model files (assuming you have a good naming convention for your corridors, assemblies and subassemblies) we can use this tree to guarantee that there is no repetition in a

model as you may get if you want to use the GUID when looking at automated asset data solutions.

CorridorName -> CorridorDesc -> BaselineName -> HorizontalBaseline -> VerticalBaseline -> RegionName -> AssemblyName -> CodeName (Shape code)

You can use this information to drive model outputs direct to an external database, giving you clear and visible information on assets based on subassembly parameters and you can also use this same ‘extraction’ path, should you wish for a ‘push’ to the 3D solid.

The result, a ‘single source of truth’ in your design model which is consistent and dynamically linkable to a database for asset data fed by subassembly parameters. From which, you can dynamically feed data onto your 3D solid models as property datasets either in their entirety or as a hyperlink to your database depending on the client requirements for your project.

You can’t expect me to show you everything though!!

Good luck and thanks for reading!