CES321590

Analyze and Revise Existing Subassembly Composer PKT Files for AutoCAD Civil 3D

Kati Mercier, P.E.
Martinez Couch & Associates, LLC

Learning Objectives

- Deconstruct the logic in an existing Subassembly Composer PKT file to identify its functions
- Learn how to modify and improve existing PKT files to maximize Autodesk Civil 3D workflows specific to your needs
- Learn how to troubleshoot your PKT files to provide a clean and complete final product
- Learn how to use your PKT files to build assemblies that will make your corridors better than ever before

Description

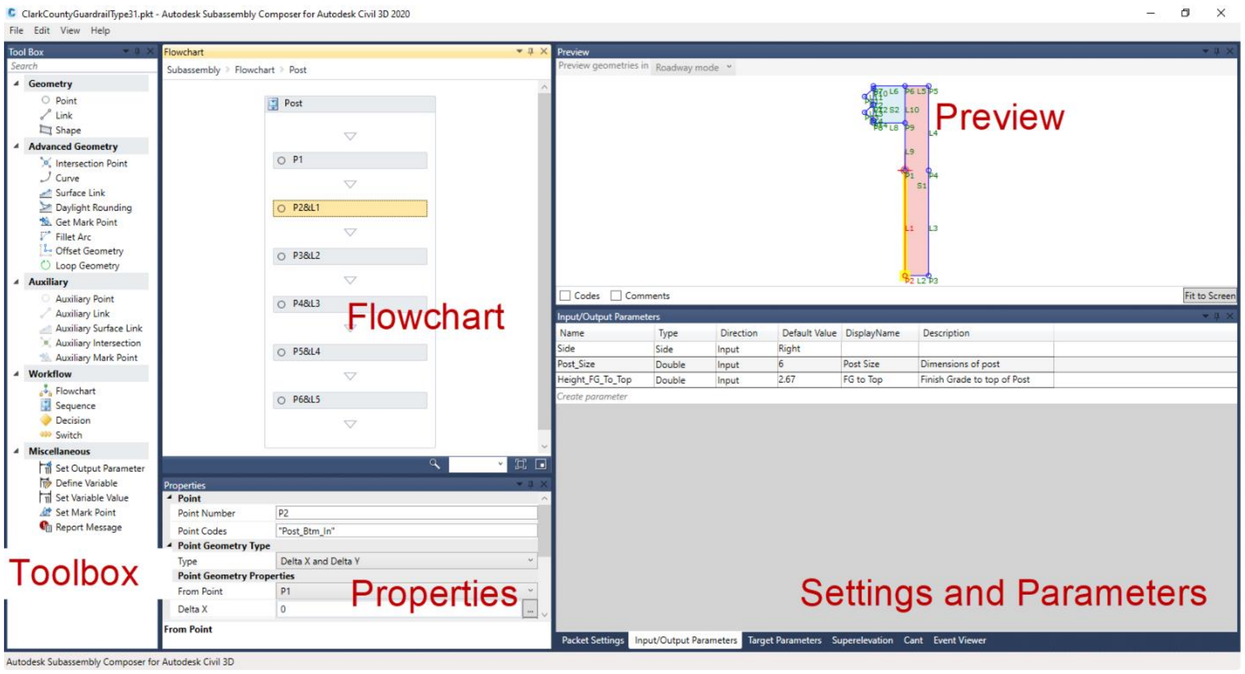
Autodesk's Subassembly Composer for Autodesk Civil 3D software provides users with a flowchart interface for building complex subassemblies for use in Autodesk Civil 3D assemblies and corridors without the need for advanced programming knowledge. There are times when starting with a clean slate makes the most sense; but with more and more Subassembly Composer PKT files available to download with Autodesk Civil 3D, it is becoming increasingly possible to modify an existing subassembly PKT file to fit your needs. Knowing how to navigate and troubleshoot these existing files, while modifying them to fit your particular needs, will enhance the power of your corridors and provide you with a personalized solution. This class is for users who have basic knowledge of Subassembly Composer and want to improve their ability to understand PKT files that were built using someone else's logic, and to make modifications to the input parameters, output parameters, target parameters, enumeration groups, and flowchart elements.

Speaker

Kati Mercier is a Professional Engineer licensed in the states of Connecticut and New York. Kati coauthored Mastering AutoCAD® Civil 3D® 2013. She is an Autodesk® Expert Elite, AutoCAD Civil 3D Certified Professional, presented at Autodesk University 2011, 2012, and 2017, and has a special place in her heart for Autodesk® Subassembly Composer.

katimercier@gmail.com

Autodesk Subassembly Composer for Autodesk Civil 3D 2020



Properties

Point	
Point Number	P2
Point Codes	"Post_Btm_In"
Point Geometry Type	
Type	Delta X and Delta Y
Point Geometry Properties	
From Point	P1
Delta X	0

From Point

Input/Output Parameters

Name	Type	Direction	Default Value	DisplayName	Description
Side	Side	Input	Right		
Post_Size	Double	Input	6	Post Size	Dimensions of post
Height_FG_To_Top	Double	Input	2.67	FG to Top	Finish Grade to top of Post
Create parameter					

SUBASSEMBLY COMPOSER 2020: CLARKCOUNTYGUARDRAILTYPE31.PKT

The ‘Subassembly Composer® for Autodesk Civil 3D®’ (hereafter referred to as Subassembly Composer) program is used to create subassemblies to supplement the stock subassemblies that come with Autodesk Civil 3D and also to modify the subassembly PKT files that also now come with Civil 3D. This handout assumes that you have a working knowledge of the program and have created basic subassembly PKT files. All the examples in this handout utilize Autodesk Subassembly Composer 2020.

Subassembly Composer can be a powerful tool and help provide the customization that many companies need that the stock subassemblies are unable to fully provide. That said, the stock subassemblies are also powerful tools and should not be overlooked. Subassembly Composer should not be used to recreate the wheel; if there is already a stock subassembly that does what you need for a part of your assembly use it! Remember, this is a **SUB**assembly Composer and is meant to give you the building blocks needed to build a full assembly and therefore you do not necessarily need to build the full assembly in Subassembly Composer.

PKT Files Downloaded With Civil 3D 2020

Autodesk Civil 3D 2020 (also Civil 3D 2019, the Civil 3D 2018.2 update, and Civil 3D 2017 v1 Enhancements) comes with a variety of subassemblies that were created in Subassembly Composer that are available for users to modify to fit their needs. These subassemblies are different than the stock subassemblies that come in the default Tool Palette with Civil 3D. The .PKT subassemblies are installed by default to Imperial and Metric folders in the following location: C:\ProgramData\Autodesk\C3D 2020\enu\Subassemblies. Reference files for each of these PKT subassemblies are provided in Autodesk Help.

Reviewing and manipulating these PKT files are a great way to become more familiar with the Subassembly Composer program's capabilities and make the customizations that the stock subassemblies don't have available.

In order to make the best use of the existing Subassembly Composer PKT files this handout will walk the user through four steps:

1. Deconstruct the logic in an existing Subassembly Composer PKT files to identify its functions;
2. Modify and improve existing PKT files to maximize Autodesk Civil 3D workflows specific to your needs;
3. Troubleshoot the PKT files to provide a clean and complete final product; and
4. Use the modified PKT files to build assemblies that will make your corridors better than ever before.

Deconstruct the logic in an existing Subassembly Composer PKT file to identify its functions

While you will sometimes find yourself wanting to start a subassembly from a clean slate, there will be other times that you may find yourself wanting to modify a Subassembly Composer PKT file that comes with Civil 3D, one that was created by another user, or even one created by yourself but so long ago that you don't remember what it does. There are a few procedures you can go through in order to help unravel the mystery to identify what it does and whether it will work for you.

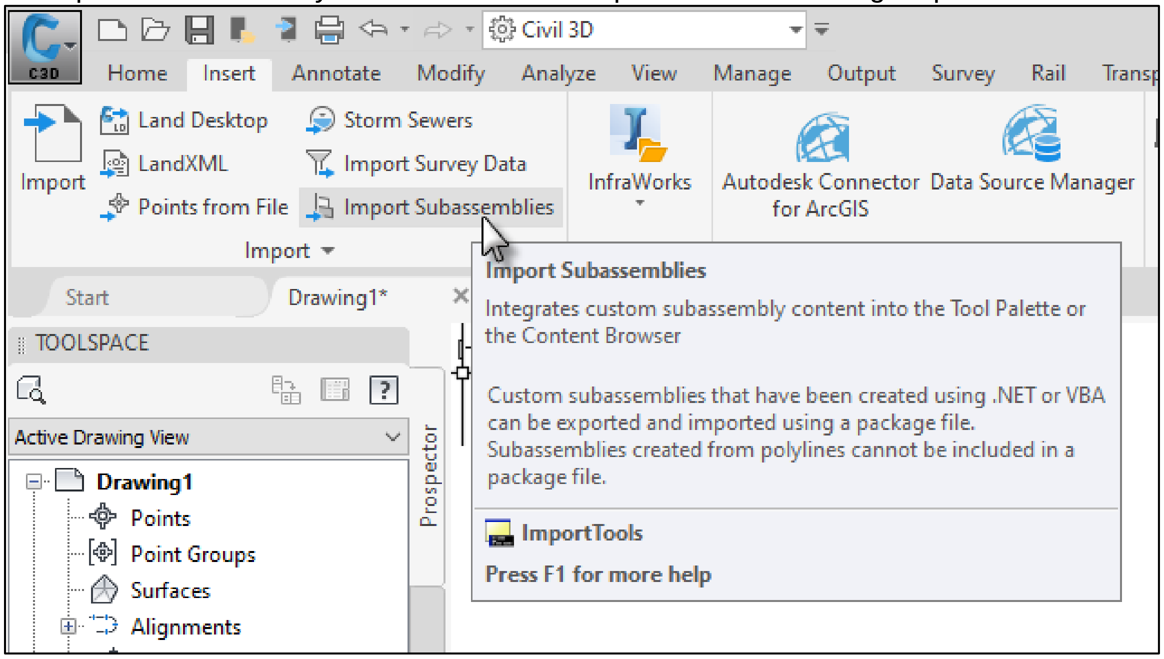
- Import the subassembly into Civil 3D and try it;
- Examine and manipulate the Settings and Parameters in Subassembly Composer; and
- Dig into the Flowchart.

Let's look at each of these procedures…

Import the subassembly into Civil 3D and try it!

One of the best ways to figure out what a subassembly does is to simply try it. By doing this you will realize what does and doesn't work for your needs. One of the benefits to testing it in Civil 3D first, especially for subassemblies that have surface targets, is the ability to manipulate these is more realistic in Civil 3D than in Subassembly Composer. In the Subassembly Composer you do not have the ability to analyze based on a sloped or irregular surface target. So, in order to test these scenarios, you will need to use the subassembly in Civil 3D.

To import a subassembly PKT file into Civil 3D perform the following steps:



Import Subassemblies

Integrates custom subassembly content into the Tool Palette or the Content Browser

Custom subassemblies that have been created using .NET or VBA can be exported and imported using a package file. Subassemblies created from polylines cannot be included in a package file.

ImportTools

Press F1 for more help

1. In Civil 3D, on the Insert ribbon, in the Import panel, click Import Subassemblies.
2. In the Import Subassemblies dialog box, to specify the Source File, click the browse button and navigate to the location of the PKT file you would like to import.
    a. As noted previously, there are a selection of PKT files installed by default to Imperial and Metric folders in the following location: C:\ProgramData\Autodesk\C3D 2020\enu\Subassemblies (or whatever location the PKT file you would like to import is located).
3. Confirm that the Import To: Tool Palette checkbox is checked.
4. Use the drop-down menu below the Tool Palette checkbox to select which Tool Palette you would like to import the subassembly file to.
    a. At the bottom of the drop-down menu is an option to Create New Palette. For ease of use, it may be helpful to create a new palette called "*TESTING PKT Subassemblies*" and another called "*PKT Subassemblies*". The first, being a palette that will contain subassemblies that you know you shouldn't necessarily trust just yet, and the latter being a palette to keep track of your growing library of PKT files utilized on various projects.
5. Click OK.

There may be instances when you have updated a PKT file and would like to re-import it into Civil 3D. There is a further discussion on how to do this later in this handout in the discussion about troubleshooting.

Once you have imported the subassembly PKT file onto your tool palette then you can utilize it like any other subassembly by attaching it to an assembly and utilizing the assembly in a corridor. As you test the PKT subassembly in Civil 3D you can ask yourself a variety of questions that will help you determine what modifications will be most beneficial to make in Subassembly Composer.

What does it ask for as input parameters?

- Does it give you too many or not enough parameters?
- Are the dimensions what you want?
- Does it ask for slopes where you want slopes and grades where you want grades?
- Does it allow you to do all the targets you want?

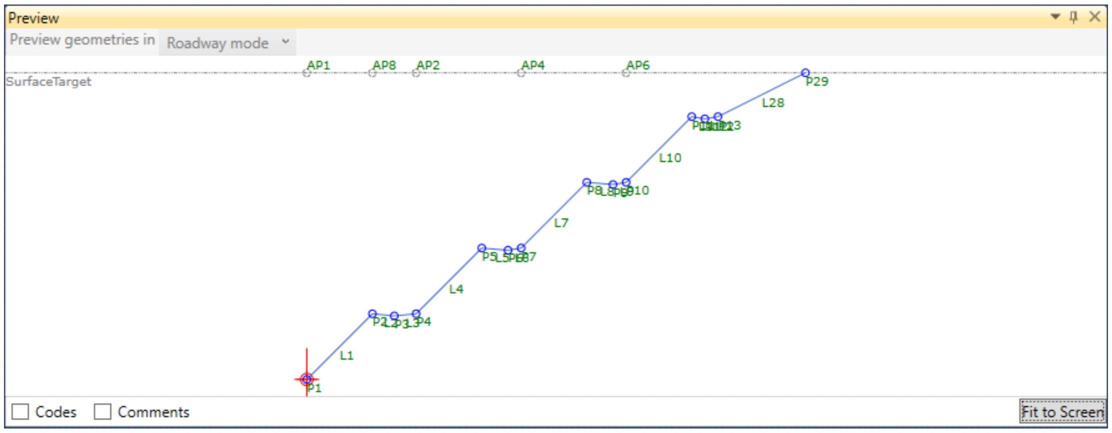
What does it give you as output?

- Does it have the point and link codes that you want?
- Is the attachment point in the correct location?
- Does it handle cuts and fills correctly?

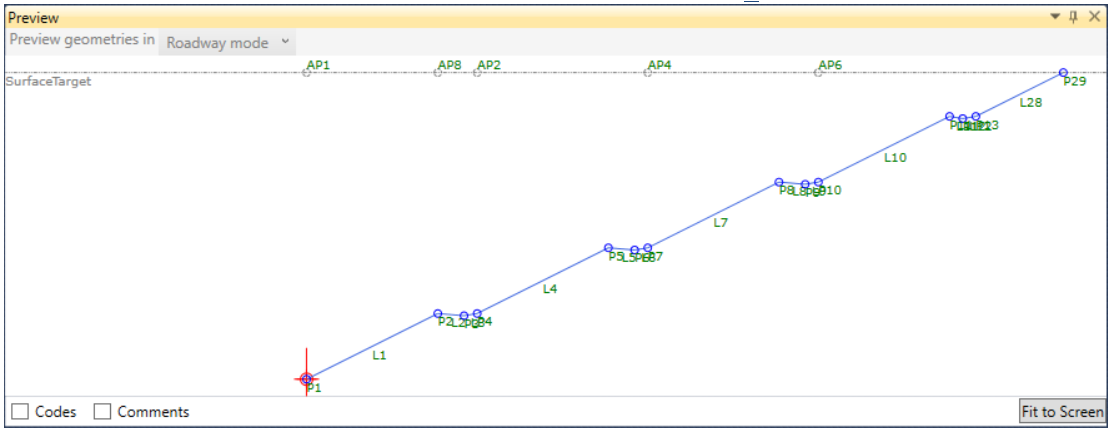
Examine and manipulate the Settings and Parameters in Subassembly Composer

Like viewing it in Civil 3D you will also want to examine and manipulate it in Subassembly Composer and ask yourself the same questions.

- Try inputting different values for the Input Parameters.

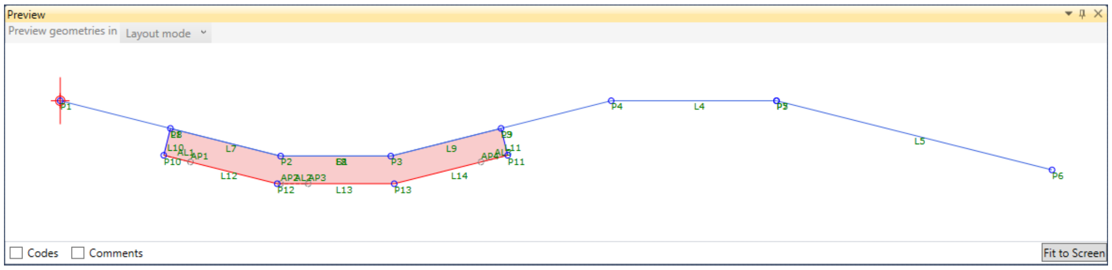


DEFAULT INPUT PARAMETERS: MULTIPLEBENCH_DEPTH.PKT

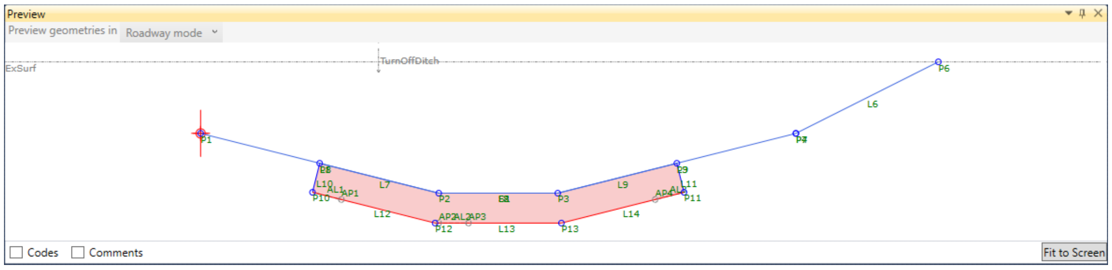


MODIFIED INPUT PARAMETERS: MULTIPLEBENCH_DEPTH.PKT

- Try switching between Layout Mode and Roadway Mode to view the subassembly without and with targets.

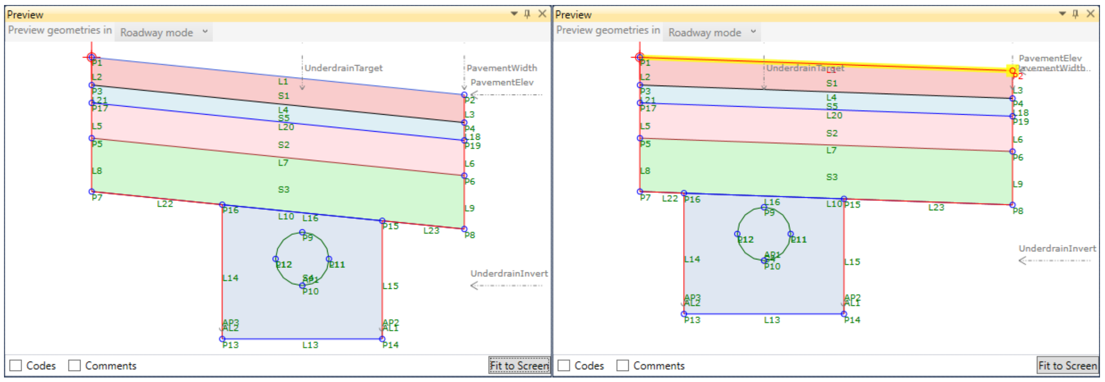


LAYOUT MODE: SIDESLOPECUTDITCH.PKT



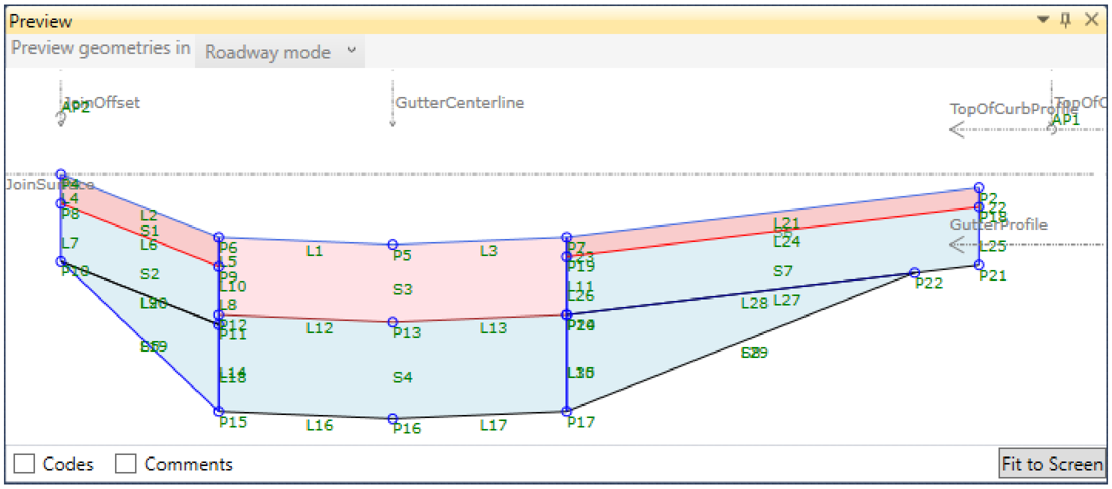
ROADWAY MODE: SIDESLOPECUTDITCH.PKT

- In Roadway Mode, try dragging the Target Parameters to different values (or inputting different values numerically for the Input Parameters and/or Target Parameters).

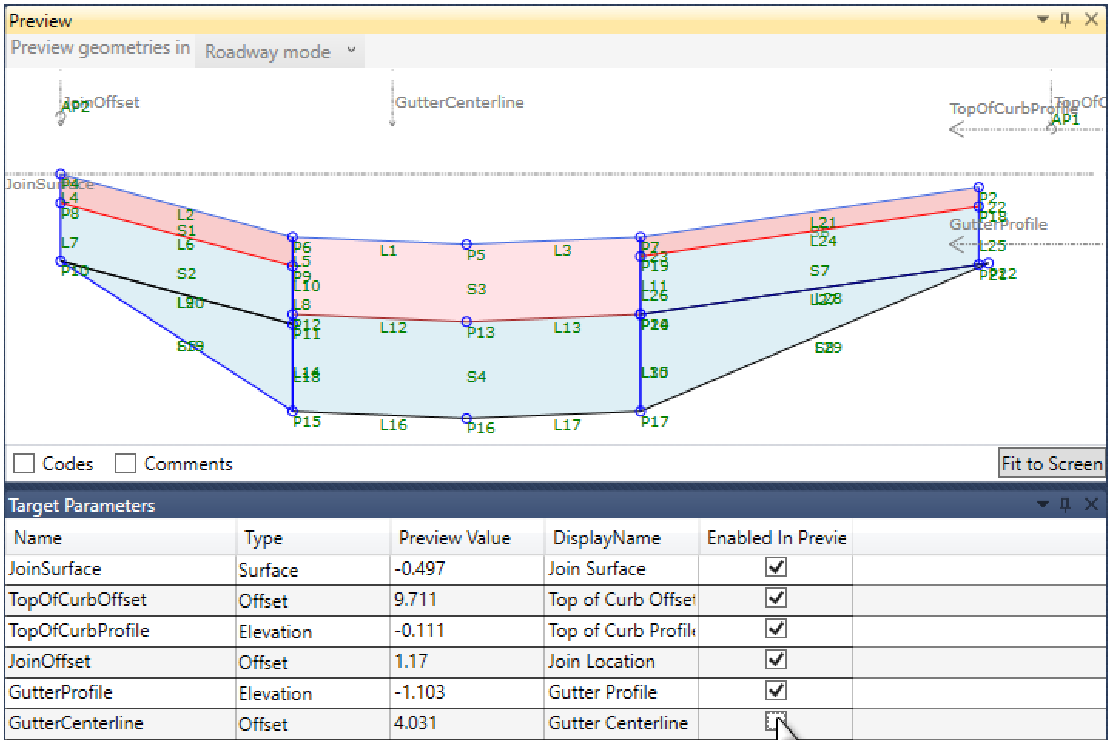


CHANGING TARGET PARAMETERS: PERMEABLEPAVEMENT.PKT

- Try switching Enabled in Preview on/off for a combination of various Target Parameters or Superelevation Parameters, as applicable to your subassembly.



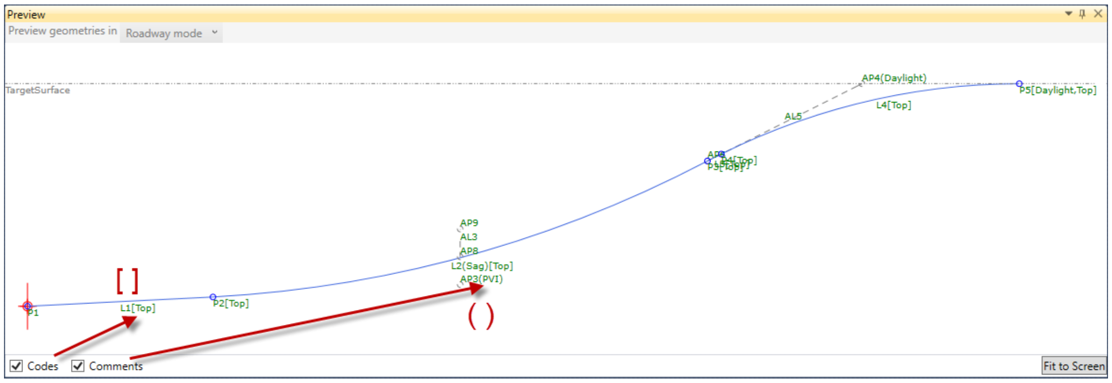
ENABLED GUTTERCENTERLINE TARGET PARAMETER: PARKINGADJACENTTOROAD.PKT



Name	Type	Preview Value	DisplayName	Enabled In Previe
JoinSurface	Surface	-0.497	Join Surface	✓
TopOfCurbOffset	Offset	9.711	Top of Curb Offset	✓
TopOfCurbProfile	Elevation	-0.111	Top of Curb Profil	✓
JoinOffset	Offset	1.17	Join Location	✓
GutterProfile	Elevation	-1.103	Gutter Profile	✓
GutterCenterline	Offset	4.031	Gutter Centerline	☐

UNENABLED GUTTERCENTERLINE TARGET PARAMETER: PARKINGADJACENTTOROAD.PKT

- Turn on and off the Codes and/or Comments. The codes, which will be utilized in Civil 3D, are shown in brackets while the comments, which are just for additional Subassembly Composer user information, are shown in parentheses.



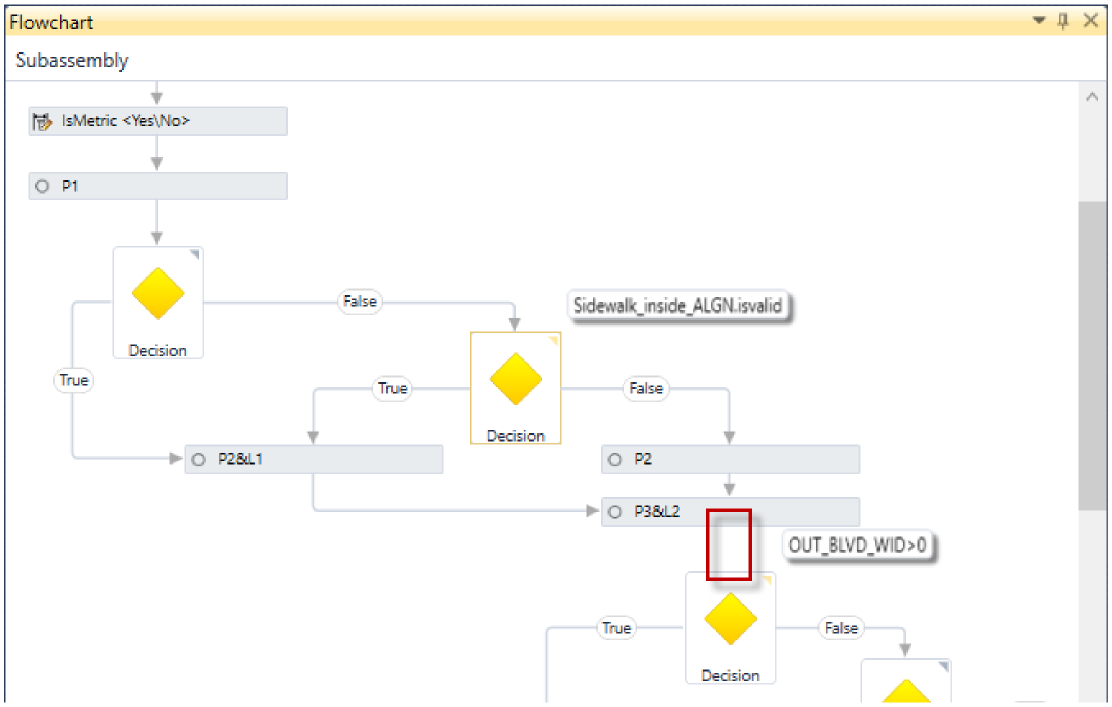
CODES AND COMMENTS: SHOULDERPARABOLICDAYLIGHT.PKT

Dig into the Flowchart

Trace flow branches

Take things one branch at a time. Start at the top and work your way down.

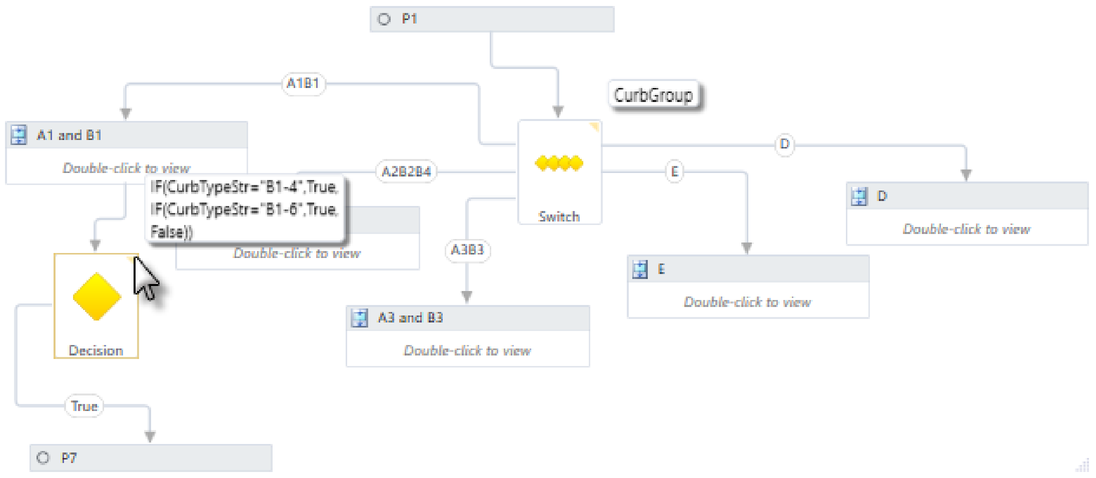
- If needed, delete connection arrows to break the flow in the logic and slowly work your way through figuring out what it is calculating as you go. When a user creates a subassembly, it doesn't magically all appear at once and you can't expect yourself to understand it all at once either.



TEMPORARILY DELETE CONNECTION ARROWS: CLARKCOUNTYSIDEWALK.PKT

- In Sequence elements you can't "break a connection" like you can in a Flowchart element to test only to a certain point. Instead you can either click on each element and it will highlight in the Preview pane. Alternatively, if you only want to see a portion of the sequence instead un-connect the original Sequence element, create an empty Sequence element and connect it where the original one was, then cut and paste the elements from the unconnected Sequence back into the new Sequence element. You can select multiple elements by holding down the Ctrl key while clicking with your mouse to select. You can then use CTRL+X to cut and CTRL+V to paste. When you paste them, any variables may be suffixed with _copy. You will want to delete the suffix so that the references to these variables do not remain broken.

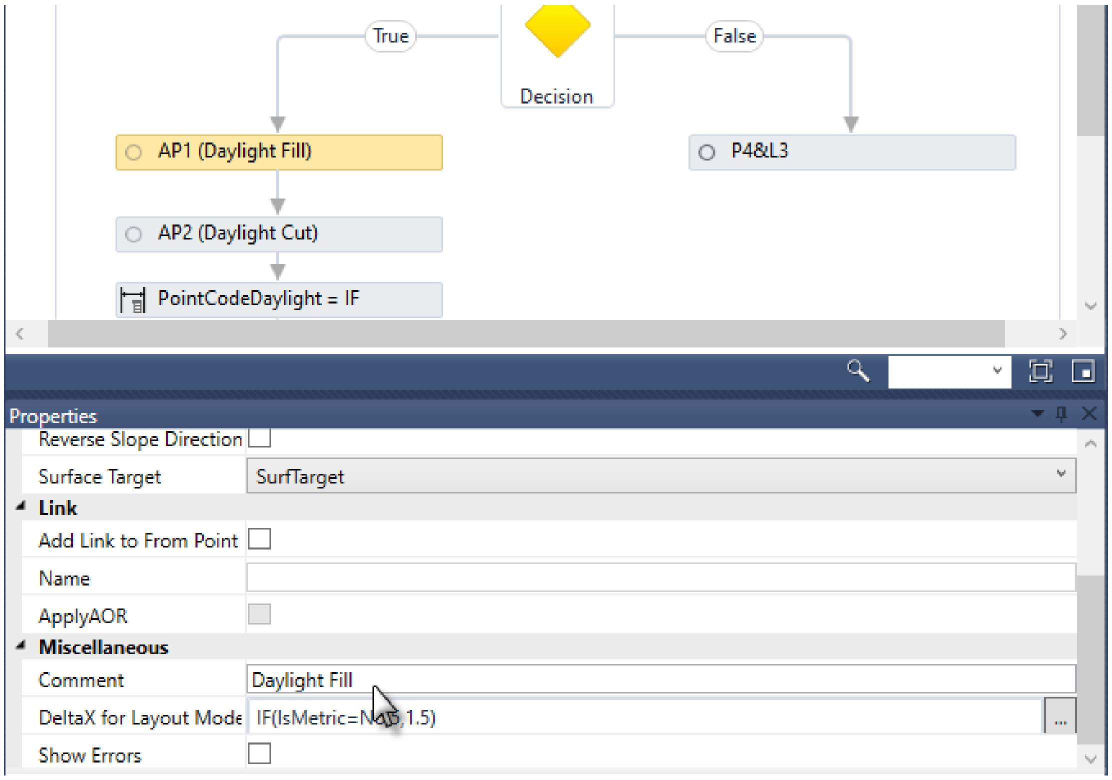
- You can click the tiny triangle flag at the upper right of a Switch or Decision element to display the condition without looking at the Properties each time.



IF(CurbTypeStr="B1-4",True,
IF(CurbTypeStr="B1-6",True,
False))

FLAG INFO FOR DECISIONS AND SWITCHES: CALTRANSCURBS.PKT

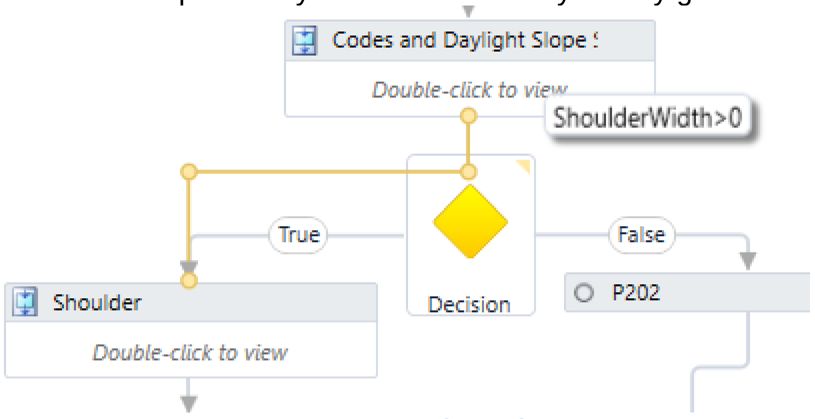
- As you figure things out add comments along the way to any elements which need a little extra explaining. You may also find it helpful to add description to Input Parameters as you discover what they are calculating if the original user did not provide one already.



Reverse Slope Direction	☐
Surface Target	SurfTarget
Link	
Add Link to From Point	☐
Name	
ApplyAOR	☐
Miscellaneous	
Comment	Daylight Fill
DeltaX for Layout Mode	IF(IsMetric=N05,1.5)
Show Errors	☐

ADD COMMENTS: BERM.PKT

- You may also find it helpful to bypass a section of logic and test a later section by readjusting your connection arrows and connecting them directly to other elements, especially at decisions, just remember that you need to have all the referenced points and links previously defined otherwise you may get errors.



BYPASS A DECISION: CLARKCOUNTYBASICDAYLIGHT.PKT

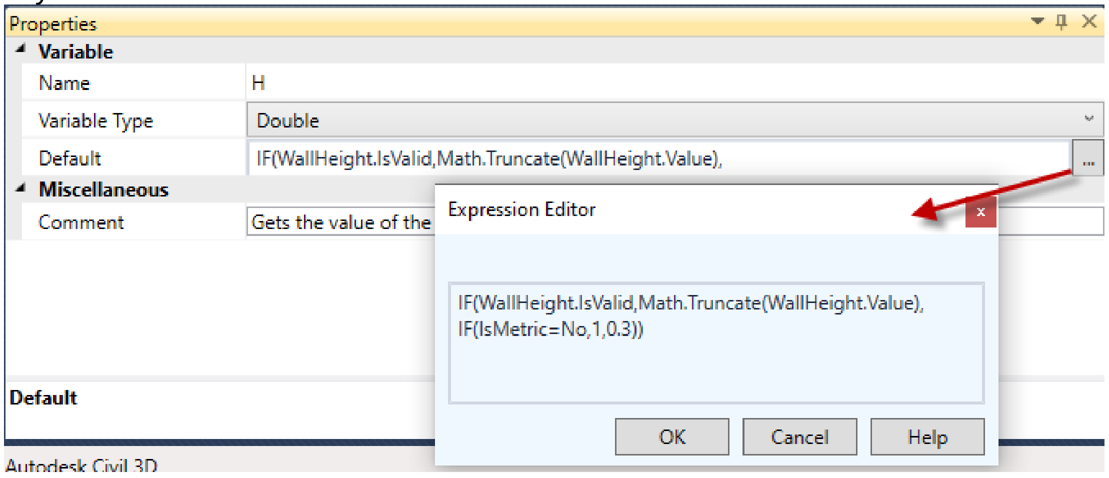
Decode equations and conditions

Sometimes it is hard to tell what value an equation is calculating or a condition is outputting. If you run into this problem, there are a few different tricks you can use to help figure out what you are working with:

- **Don't know what value is being calculated?**: Create a temporary Point element with the equation as your delta X or delta Y value. Then create a temporary target parameter (offset or elevation) that you can use like a temporary dimension line to query an approximate length or depth. The unknown value will be displayed in your Target Parameters. Don't forget to delete the temporary target parameter before importing to Civil 3D.

- **Don't recognize a function?**: In the Appendix of this document you will find a listing of all of the API functions which are available for use in equations and conditions. The portion of the function before the period will give you a hint at what class of functions you will want to look in (for example, P1.elevation belongs to the point class, whereas baseline.elevation belongs to the baseline class).

- **Super long equation?**: Remember that you can always click the ellipses button next to the equation to open up a separate window which you can expand to see the full equation. You can also separate the equation on multiple lines by using the Enter key.



Name	H
Variable Type	Double
Default	IF(WallHeight.IsValid,Math.Truncate(WallHeight.Value),
Comment	Gets the value of the

Default

```
IF(WallHeight.IsValid,Math.Truncate(WallHeight.Value),
IF(IsMetric=No,1,0.3))
```

EXPRESSION EDITOR: CALTRANSB3RETAININGWALL.PKT

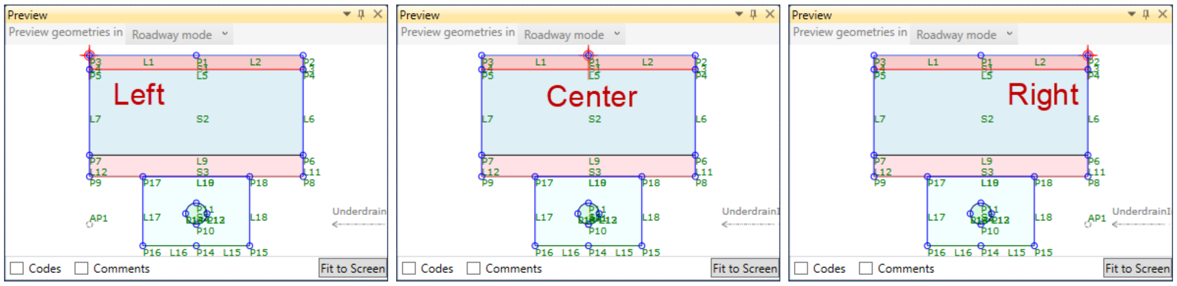
- **Don't know what conditional is being used?**: Put the conditional in a temporary Decision element, place a temporary Point element that is at a known location (for example, 5 units above the origin) on either the true or false side. Now change the parameters and see when the point appears or disappears. (for example, you may have the condition AP1.distancetosurface(DaylightSurface)>0 and will learn that the true side means that the point is in a fill condition and the false side means that the point is in a cut condition.

Modify and improve existing PKT files to maximize Autodesk Civil 3D workflows specific to your needs

Once you have figured out how an existing PKT file works, then you will be able to modify it. Modifications may involve either simplifications to the subassembly or adding more complex features. Maybe you want to modify the insertion point. Or you want to modify input parameters. Or you want to modify the string values for the point or link codes. Or you want to change dimensions. Or you want to add a shape where there wasn't one previously. Or you want the Civil 3D user to be able to modify the point or link codes as input parameters. Or you want both the cut and fill linework to display in Layout Mode. Or you want to set an input parameter to taper from one value to another through a region. The following are several short examples that modify the PKT subassemblies that come with Civil 3D. These similar modifications could be made to other subassemblies as well.

Modify the insertion point and add an Enumeration for use with an Input Parameter type

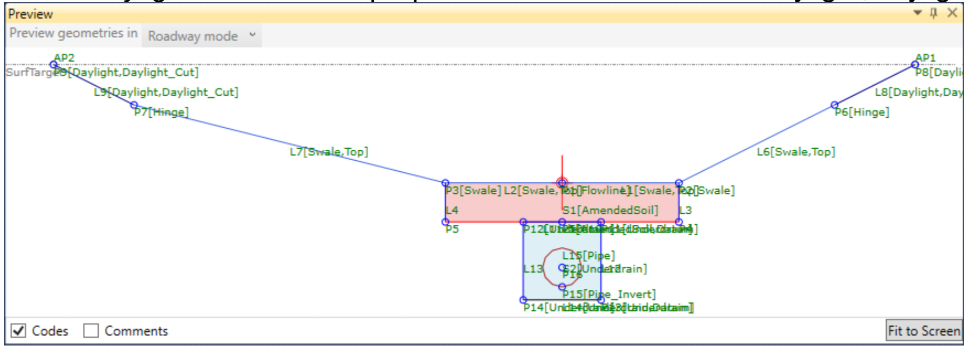
1. Open BioRetention.pkt and save as BioRetention_v1.pkt.
    a. The BioRetention.pkt file defaults to the insertion point being located at the center of the top link. This example will modify the insertion point by allowing the Civil 3D user to select whether to attach at Left, Right, or Center.
2. From the View menu select Define Enumeration.
    a. Enumeration Groups are used to provide a different Type for Input and Output Parameters that provides the Civil 3D user a list to select from.
3. In the Define Enumeration window, click CreateEnumGroup.
4. Change the name of the Enum Group from Enum0 to AttachPt.
5. Click CreateEnumItem three times.
6. Change the three Enum Items from Item0 to Left, Item1 to Right, and Item2 to Center.
7. Click OK to close the Define Enumeration window.
8. In the Input/Output Parameters window, click Create parameter.
9. For the new parameter (Parameter11) created, set the Name to AttachmentPoint, Type to AttachPt, and Default Value to Center.
10. In the Flowchart window, select point P1.
11. In the P1 properties, change the Delta X formula to:
    if(AttachmentPoint=Left, Biowidth/2,if(AttachmentPoint=Right,-Biowidth/2,0))
12. Try changing the Default Value for the AttachmentPoint input parameter to left, right, or center to see how the new formula adjusts the location of the origin.
    a. The origin is the red cross symbol and is where the subassembly will attach in Civil 3D.



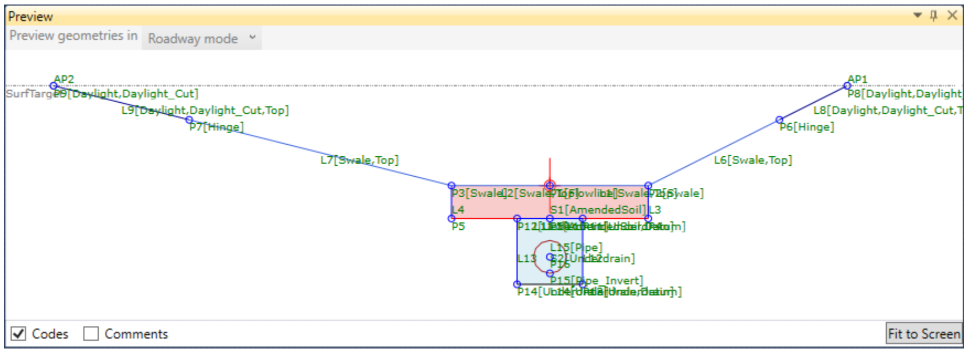
MODIFIED INSERTION POINT: BIORETENTION_V1.PKT

Modify input parameters and modify link codes

1. Open VegetatedSwale.pkt and save as VegetatedSwale_v1.pkt.
    a. The VegetatedSwale.pkt file defaults to utilizing the DaylightSlope for both the left and the right sides. This example will modify the subassembly to reference DaylightSlopeLeft and DaylightSlopeRight.
2. In the Input/Output Parameters window, change DaylightSlope to DaylightSlopeLeft and change the DisplayName and Description to also reference Daylight Slope Left.
3. In the Input/Output Parameters window, click Create parameter.
4. For the new parameter (Parameter13) created, set the Name to DaylightSlopeRight, Type to Slope, Default Value to 2.00:1, and DisplayName to Daylight Slope Right.
5. In the Flowchart window, double click to enter the Daylights Flowchart element.
6. Select the auxiliary point AP1 and note that it highlights in the Preview window as the auxiliary point that daylights on the right side.
7. In the AP1 properties, change the Slope to DaylightSlopeRight.
8. Try changing the Default Value for DaylightSlopeLeft and DaylightSlopeRight to see how the new formula modifies the left and right daylight links independently of one another.
9. In the Preview window, click the checkbox for Codes. Notice that the subassembly currently has Top link codes for the main portion of the vegetated swale and side slopes but not for the daylight links.
10. In the Flowchart window, in the Subassembly>Flowchart>Daylights text, click the word Flowchart to step back up to the Flowchart level of the flowchart.
11. In the Flowchart window, double click to enter the Codes Sequence element.
12. In the Codes Sequence, select the DaylightCutLinkCode Define Variable element.
13. In the DaylightCutLinkCode properties, set the Default to “Daylight,Daylight_Cut,Top”.



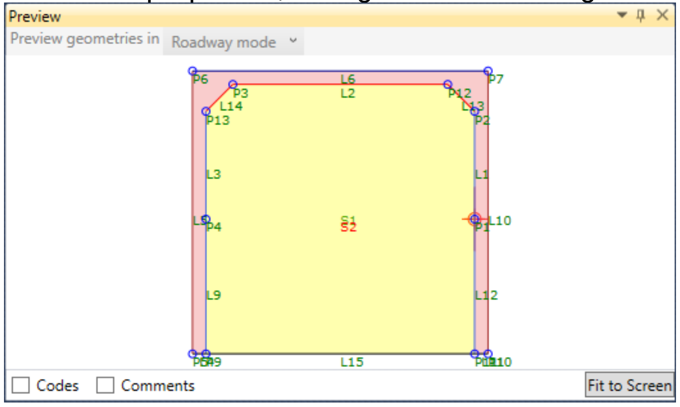
ORIGINAL PKT FILE: VEGETATEDSWALE.PKT



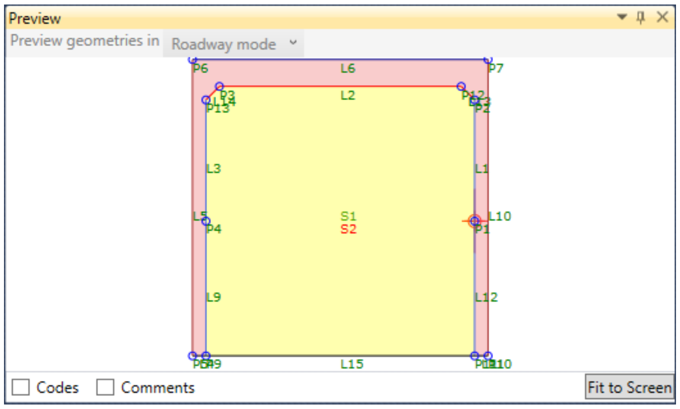
DAYLIGHTSLOPELEFT AND DAYLIGHTSLOPERIGHT AND TOP LINK: VEGETATEDSWALE_V1.PKT

Change dimensions

1. Open SimpleTunnelOpenBottom.pkt and save as SimpleTunnelOpenBottom_v1.pkt.
    a. The SimpleTunnelOpenBottom.pkt file defaults to utilizing an Input Parameter for WallThickness that is also used for the ceiling thickness, and defaults to a 1' haunch height and width. This example will add an Input Parameters for CeilingThickness, HaunchHeight, and HaunchWidth.
2. In the Input/Output Parameters window, click Create parameter three times.
3. For the first new parameter (Parameter13) created, change the Name to CeilingThickness, Type to Double, Default Value to 1, and DisplayName to Ceiling Thickness.
4. For the next new parameter (Parameter14) created, change the Name to HaunchWidth, Type to Double, Default Value to 0.5, and DisplayName to Haunch Width.
5. For the next new parameter (Parameter15) created, change the Name to HaunchHeight, Type to Double, Default Value to 0.5, and DisplayName to Haunch Height.
6. For the WallThickness Input Parameter delete the description.
7. In the P2 properties, change DeltaY to Height-HaunchHeight.
8. In the P12 properties, change DeltaX to -HaunchWidth and DeltaY to HaunchHeight.
9. In the P3 properties, change DeltaX to -Width+2*HaunchWidth.
10. In the P13 properties, change DeltaX to -HaunchWidth and DeltaY to -HaunchHeight.
11. In the P4 properties, change DeltaY to -Height+HaunchHeight.
12. In the P6 properties, change DeltaY to FoundationDepth+Height+CeilingThickness.
13. In the P10 properties, change DeltaY to -Height-FoundationDepth-CeilingThickness.



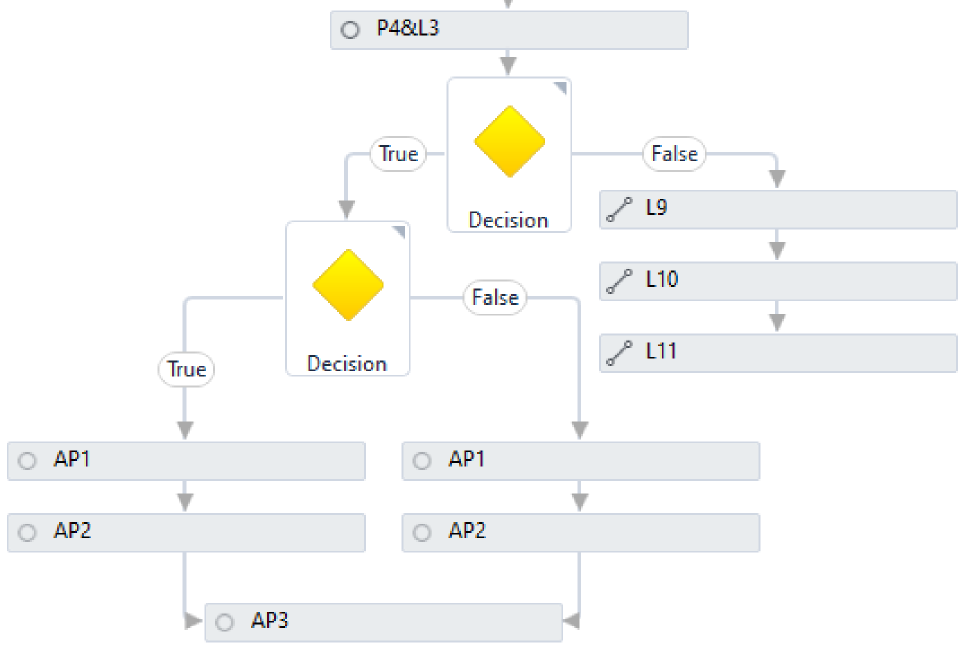
ORIGINAL PKT FILE: SIMPLETUNNELOPENBOTTOM.PKT



CEILINGTHICKNESS, HAUNCHHEIGHT, HAUNCHWIDTH: SIMPLETUNNELOPENBOTTOM_V1.PKT

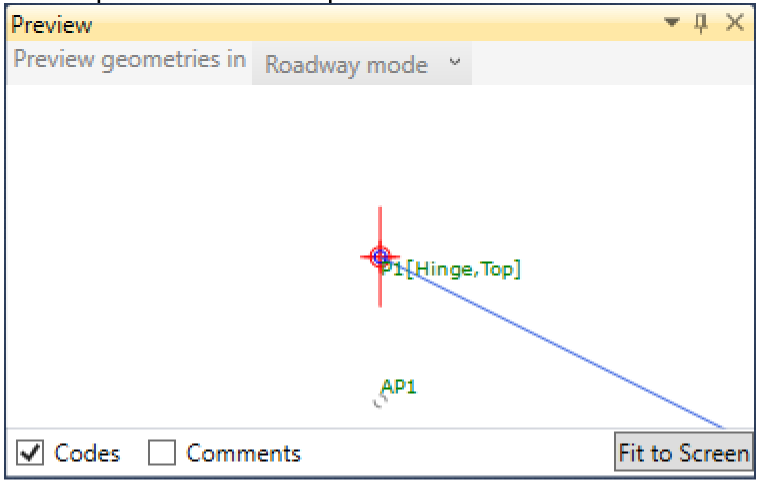
Add a shape and its associated points and links

1. Open EvenSlopeDitch.pkt and save as EvenSlopeDitch_v1.pkt.
    a. The EvenSlopeDitch.pkt file does not include any depth or shape code. This example will modify the subassembly to include a lined material with a depth measured either vertically or perpendicular to the slope.
2. From the View menu select Define Enumeration.
3. In the Define Enumeration window, click CreateEnumGroup.
4. Change the name of the Enum Group from Enum0 to MeasureDirection.
5. Click CreateEnumItem two times.
6. Change the two Enum Items from Item0 to Vertical and Item1 to PerpToSlope.
7. In the Input/Output Parameters window, click Create parameter three times.
8. For the first new parameter (Parameter5) created, change the Name to LinedMaterialDepth, Type to Double, Default Value to 0.25.
9. For the next new parameter (Parameter6) created, change the Name to DepthMeasureDirection, Type to String, Default Value to Top,Datum,Bench.
10. For the next new parameter (Parameter7) created, change the Name to LinedMaterialCode, Type to String, Default Value to Topsoil.
11. In the P1 properties, delete the Point Codes.
12. In the P2&L1 properties, delete the "Datum" from the Link Codes.
13. In the P3&L2 properties, delete the "Datum" from the Link Codes.
14. In the P4&L3 properties, delete the "Datum" from the Link Codes.
15. In the Flowchart, below P4&L3 add the following elements:

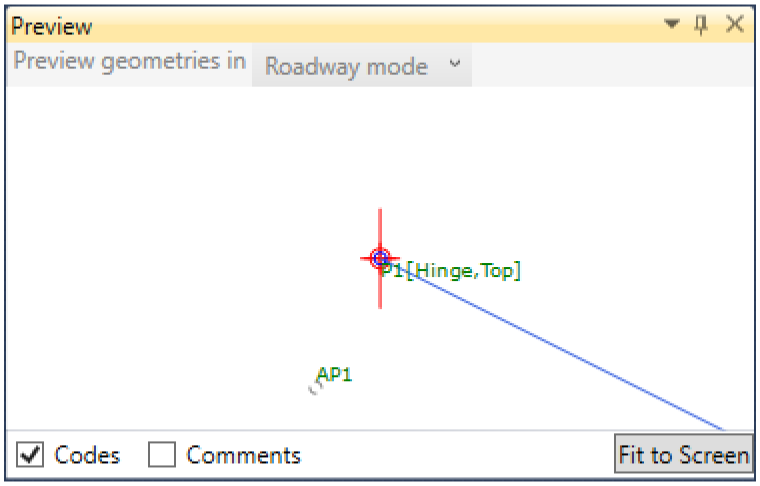


16. In the first Decision, set the Condition to LinedMaterialDepth>0.
17. In the second Decision, set the Condition to DepthMeasureDirection=Vertical.
18. On the True side of the second decision, select AP1 and in the properties, set Type to DeltaX and DeltaY, From Point to P1, DeltaX to 0, and DeltaY to -LinedMaterialDepth.
19. On the True side of the second decision, select AP1 and in the properties, set Type to DeltaX and DeltaY, From Point to P4, DeltaX to 0, and DeltaY to -LinedMaterialDepth.

20. On the False side of the second decision, select AP1 and in the properties, set Type to Angle and Distance, From Point to P1, Reference Point to P2, Angle to 90, and Distance to -LinedMaterialDepth.
    a. This auxiliary point will not default to AP1, you will need to manually change the Point Number to duplicate the auxiliary point on the true side. Duplicating the point number allows for elements later in the flowchart to reference either version of AP1 from the true or false sides in the same way.
21. On the False side of the second decision, select AP2 and in the properties, set Type to Angle and Distance, From Point to P4, Reference Point to P3, Angle to 90, and Distance to LinedMaterialDepth.
    a. Like above, you will need to manually change the Point Number to AP2.
22. In AP3 properties, set Type to DeltaX and DeltaY, From Point to P3, DeltaX to 0, and DeltaY to -LinedMaterialDepth.
23. With Input Parameter DepthMeasureDirection set to Vertical, AP1 should look like this:

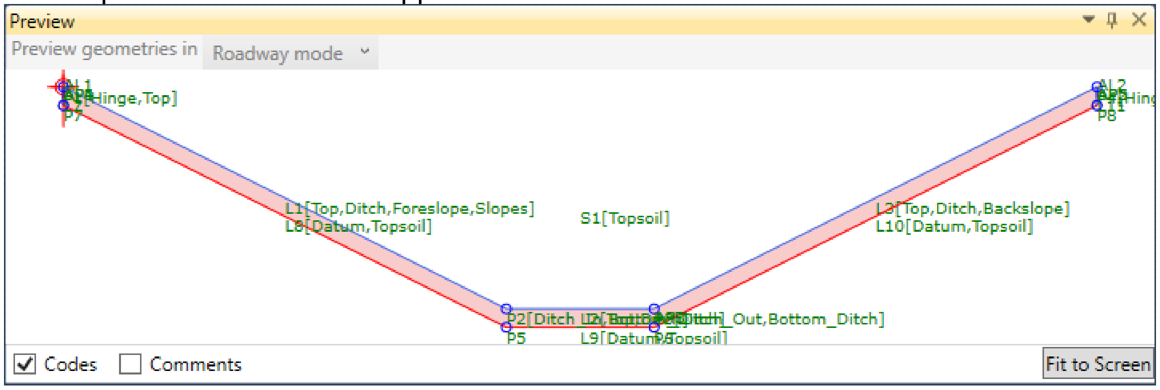


24. With Input Parameter DepthMeasureDirection set to PerpToSlope, AP1 should look like this:



25. Below AP3, add an Intersection Point, in the P5 properties, set Type to Intersection:TwoPointsSlope, Point1 to AP1, Slope1 to L1.slope, Point2 to AP3, Slope2 to L2.slope, and check the box for Extend Slope 2.

26. Below P5, add an Intersection Point, in the P6 properties, set Type to Intersection:TwoPointsSlope, Point1 to AP2, Slope1 to -L3.slope, Point2 to AP3, Slope2 to L2.slope, and check the box for Reverse Slope 1.
27. Below P6, add an Auxiliary Point, in the AP4&AL1 properties, set Type to DeltaX and DeltaY, From Point to P1, DeltaX to 0, and DeltaY to -LinedMaterialDepth.
28. Below AP4&AL1, add an Intersection Point, in the P7 properties, set Type to Intersection:LinkPointSlope, Link to AL1, Point to AP1, check the Extend Link checkbox, and Slope to L1.slope.
29. Below P7, add an Auxiliary Point, in the AP5&AL2 properties, set Type to DeltaX and DeltaY, From Point to P4, DeltaX to 0, and DeltaY to -LinedMaterialDepth.
30. Below AP5&AL2, add an Intersection Point, in the P8 properties, set Type to Intersection:LinkPointSlope, Link to AL2, Point to AP2, check the Extend Link checkbox, Slope to L3.slope, and check the Extend Slope checkbox.
31. Below P8, add a Link, in the L7 properties, set Start Point to P1 and End Point to P7.
32. Below L7, add a Link, in the L8 properties, set Start Point to P7 and End Point to P5, and set the Link Codes to “Datum”,LinedMaterialCode.
    a. For Codes, the quotes referencing a string and the non-quotes reference a variable, in this case the LinedMaterialCode input parameter.
33. Below L8, add a Link, in the L9 properties, set Start Point to P5 and End Point to P6, and set the Link Codes to “Datum”,LinedMaterialCode.
34. Below L9, add a Link, in the L10 properties, set Start Point to P6 and End Point to P8, and set the Link Codes to “Datum”,LinedMaterialCode.
35. Below L10, add a Link, in the L11 properties, set Start Point to P4 and End Point to P8.
36. Below L11, add a Shape, in the S1 properties, set the Shape Codes to LinedMaterialCode and click the green Select Shape in Preview button and click inside the shape area to add all the applicable links.

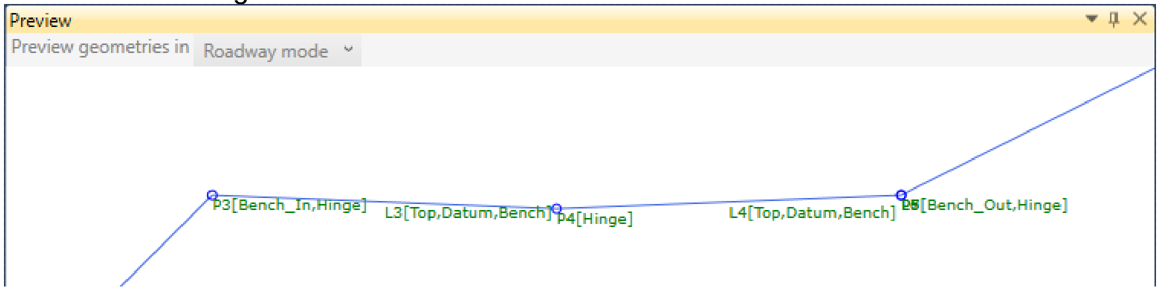


LINED MATERIAL SHAPE: EVENSLOPEDITCH_V1.PKT

Modify the point or link codes to be input parameters

1. Open BasicBench.pkt and save as BasicBench_v1.pkt.
    a. The BasicBench.pkt file includes point codes and link codes manually entered into SAC. This example will add Input Parameters to allow the Civil 3D user to modify the various point codes and link codes.
2. In the Input/Output Parameters window, click Create parameter eight times.
3. For the first new parameter (Parameter10) created, set the Name to BenchSlopeLinkCodes, Type to String, Default Value to Top,Datum.

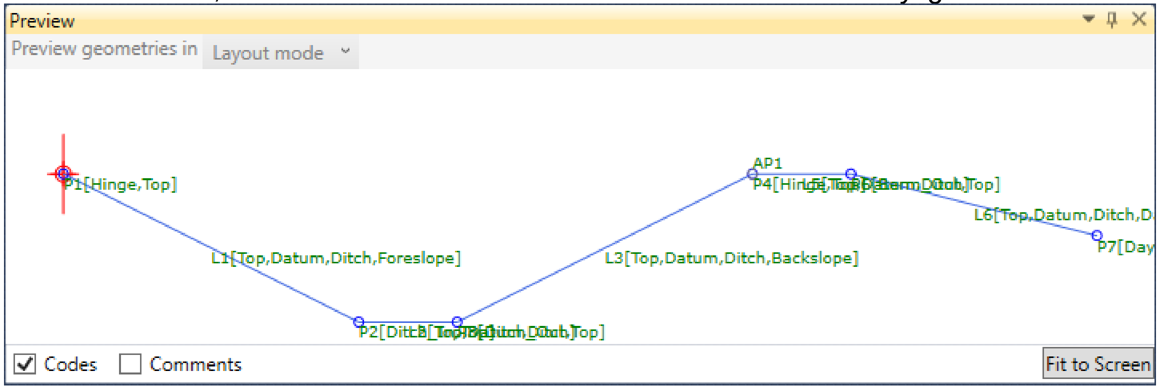
4. For the next new parameter (Parameter11) created, set the Name to BenchInLinkCodes, Type to String, Default Value to Top,Datum,Bench.
5. For the next new parameter (Parameter12) created, set the Name to BenchOutLinkCodes, Type to String, Default Value to Top,Datum,Bench.
6. For the next new parameter (Parameter13) created, set the Name to DaylightLinkCodes, Type to String, Default Value to Top,Datum,Daylight.
7. For the next new parameter (Parameter14) created, set the Name to BenchInPointCodes, Type to String, Default Value to Bench_In,Hinge.
8. For the next new parameter (Parameter15) created, set the Name to BenchHingePointCodes, Type to String, Default Value to Hinge.
9. For the next new parameter (Parameter16) created, set the Name to BenchOutPointCodes, Type to String, Default Value to Bench_Out,Hinge.
10. For the next new parameter (Parameter17) created, set the Name to DaylightPointCodes, Type to String, Default Value to Daylight.
11. In the P1 properties, delete Point Codes.
12. In the P2&L1 properties, change Point Codes to DaylightPointCodes and change Link Codes to DaylightLinkCodes
13. In the P3&L2 properties, change Point Codes to BenchInPointCodes and change Link Codes to SlopeLinkCodes.
14. In the P4&L3 properties, change Point Codes to BenchHingePointCodes and change Link Codes to BenchInLinkCodes.
15. In the P5&L4 properties, change Point Codes to BenchOutPointCodes and change Link Codes to BenchOutLinkCodes.
16. In the P6&L5 properties, change Point Codes to DaylightPointCodes and change Link Codes to DaylightLinkCodes.
17. In the Tool Box, in the Geometry section, drag a Point to next to P5&L4.
    a. Subassembly Composer Loop Geometry will not include the point codes associated with the last point in the loop, to force the point codes for P5 to generate, we must add an empty point and link at the end of the loop geometry.
18. In the P7&L6 properties, set the From Point to P5.
19. In the LO1 properties, add the L6 link to the Links list by clicking Add Link. The P5 point codes will now be generated. If you had added point codes to P7, they would now be the ones that are not generated.



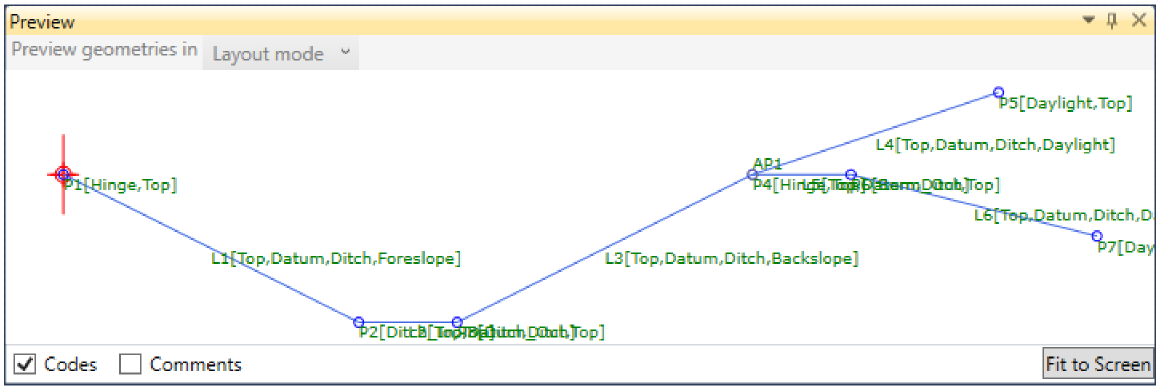
POINT CODE AND LINK CODE INPUT PARAMETERS: BASICBENCH_V1.PKT

Set the Cut and Fill linework to both show in Layout Mode

1. Open SlopeDitch.pkt and save as SlopeDitch_v1.pkt.
    a. The corridor functionality (or Roadway Mode) of the SlopeDitch.pkt will not be modified, this example will modify how the subassembly is displayed in Layout Mode.
2. In the Preview window, change the Preview geometries in dropdown to Layout Mode. Notice that only the Berm linework is displayed.
3. In the Tool Box, in the Workflow section, drag a Decision to below the Berm sequence element in the Flowchart.
4. In the Decision properties, set the Condition to SA.islayout=true.
    a. SA.islayout is an API function that queries whether the subassembly is in Layout Mode.
5. On the true side, connect the True connection arrow to the P5&L4 daylight link.


Preview geometries in Layout mode

LAYOUT MODE SHOWING ORIGINAL PKT: SLOPEDITCH .PKT

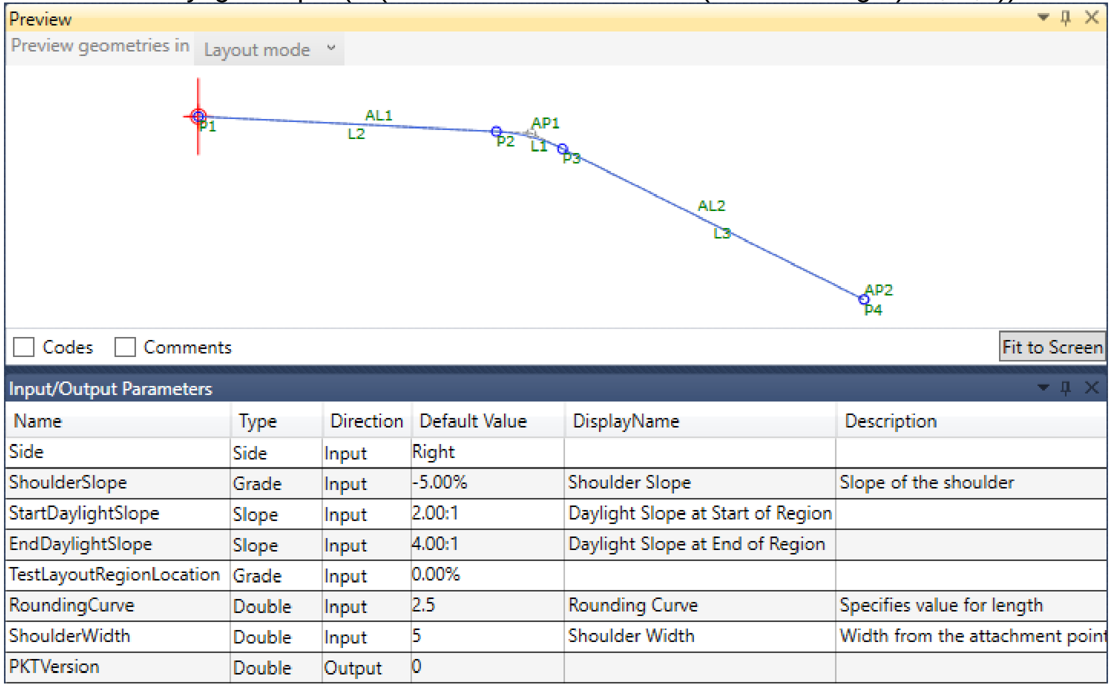

Preview geometries in Layout mode

LAYOUT MODE SHOWING CUT BERM AND DAYLIGHT: SLOPEDITCH_V1.PKT

Set a variable to taper from one value to another value throughout a region

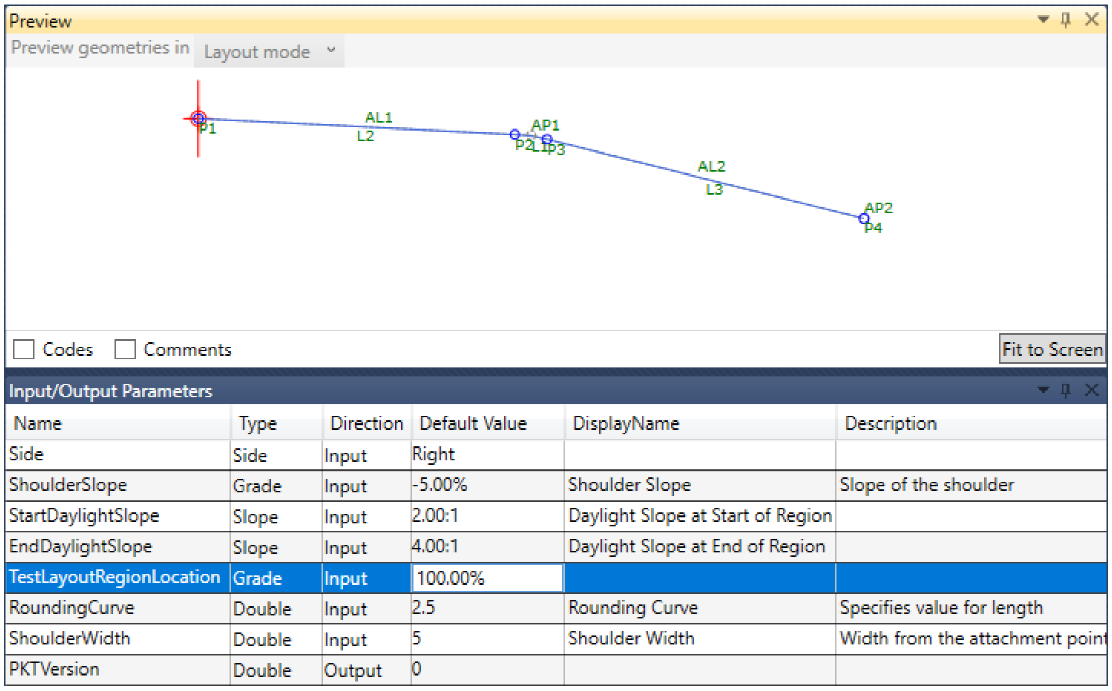
1. Open ShoulderRoundedTarget.pkt and save as ShoulderRoundedTarget_v1.pkt.
    a. The ShoulderRoundedTarget.pkt file defaults to utilizing a constant DaylightSlope throughout the entire region. This example will modify the subassembly to taper the DaylightSlope from the Start to the End of the region that the subassembly is applied.
2. In the Input/Output Parameters window, change DaylightSlope to StartDaylightSlope and change the DisplayName to reference Start Daylight Slope.
3. In the Input/Output Parameters window, click Create parameter two times.

4. For the first new parameter (Parameter6) created, set the Name to EndDaylightSlope, Type to Slope, Default Value to 4.00:1, and DisplayName to End Daylight Slope.
    a. If you did not want to taper the daylight slope through the region, you would set both Values to the same slope.
5. For the next new parameter (Parameter7) created, set the Name to TestLayoutRegionLocation, Type to Grade, Default Value to 0%.
    a. This Input Parameter allows the user to vary the percentage along the region, therefore 0% applies the StartDaylightSlope and 100% applies the EndDaylightSlope with the percentages in between tapering the slopes the applicable percentage for viewing/checking purposes.
6. Define Variable, set Name to CalclulatedDaylightSlope, VariableType to Double, and Default to if(SA.islayout=true,ctype(StartDaylightSlope-(StartDaylightSlope-EndDaylightSlope)*ctype(TestLayoutRegionLocation,double),double),ctype(StartDaylightSlope+(Baseline.Station-Baseline.RegionStart)/(Baseline.RegionEnd-Baseline.RegionStart)*(EndDaylightSlope-StartDaylightSlope),double)).
    a. Baseline API functions cannot calculate in Roadway Mode in SAC, therefore switch to Layout Mode to view the generated subassembly.
7. In AP2&AL1 properties, change the Slope to CalclulatedDaylightSlope*(IF(AP1.DistanceToSurface(SurfaceTarget)>0,-1,1))



Name	Type	Direction	Default Value	DisplayName	Description
Side	Side	Input	Right		
ShoulderSlope	Grade	Input	-5.00%	Shoulder Slope	Slope of the shoulder
StartDaylightSlope	Slope	Input	2.00:1	Daylight Slope at Start of Region	
EndDaylightSlope	Slope	Input	4.00:1	Daylight Slope at End of Region	
TestLayoutRegionLocation	Grade	Input	0.00%		
RoundingCurve	Double	Input	2.5	Rounding Curve	Specifies value for length
ShoulderWidth	Double	Input	5	Shoulder Width	Width from the attachment point
PKTVersion	Double	Output	0		

LAYOUT MODE SHOWING THE START OF REGION SLOPE: SHOULDERROUNDEDTARGET_V1.PKT

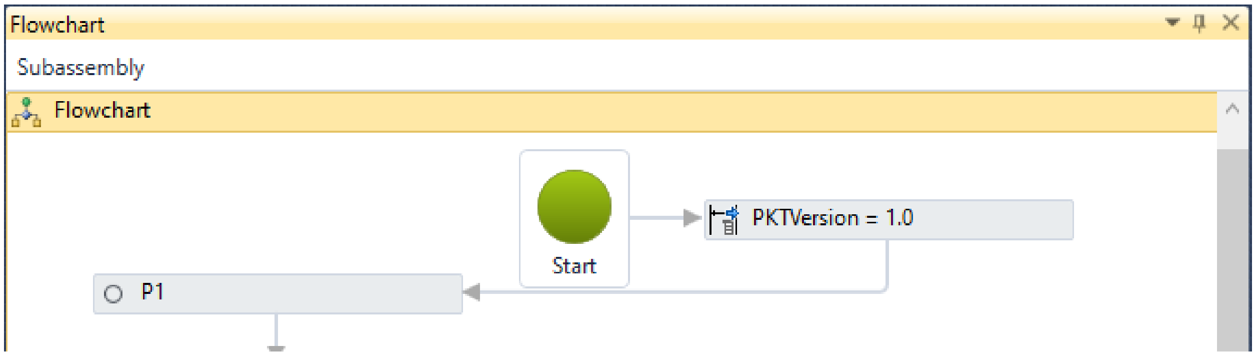


Name	Type	Direction	Default Value	DisplayName	Description
Side	Side	Input	Right		
ShoulderSlope	Grade	Input	-5.00%	Shoulder Slope	Slope of the shoulder
StartDaylightSlope	Slope	Input	2.00:1	Daylight Slope at Start of Region	
EndDaylightSlope	Slope	Input	4.00:1	Daylight Slope at End of Region	
TestLayoutRegionLocation	Grade	Input	100.00%		
RoundingCurve	Double	Input	2.5	Rounding Curve	Specifies value for length
ShoulderWidth	Double	Input	5	Shoulder Width	Width from the attachment point
PKTVersion	Double	Output	0		

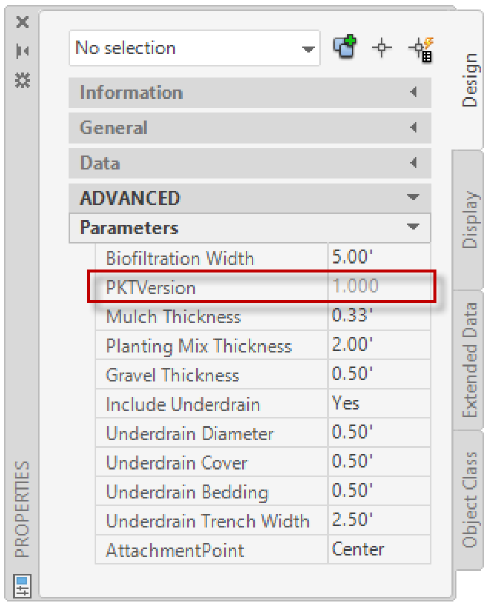
BLAYOUT MODE SHOWING THE END OF REGION SLOPE: SHOULDERROUNDEDTARGET_V1.PKT

Troubleshoot your PKT files to provide a clean and complete final product

A good practice when you start working in a PKT file, whether it is one you are starting or one that you are editing, is to add an Output Parameter named PKTVersion, Type=Double. Once the PKTVersion output parameter is created you can add a Set Output Parameter element from the Toolbox's Miscellaneous section to the Flowchart and connect Start to this element and then connect the rest of the flowchart. Set the PKTVersion initially to 1.0. As you revise and save the PKT subassembly before each import into Civil 3D you can manually iterate this output parameter to 1.1 or 2.0 or whatever value is appropriate. By setting this output parameter in Subassembly Composer, you will be able to see the value in Civil 3D and know which version of the PKT file you are utilizing in your assembly and have confidence in which version of the file you are using as you work through various iterations.



PKTVERSION OUTPUT PARAMETER IN SUBASSEMBLY COMPOSER



Biofiltration Width	5.00'
PKTVersion	1.000
Mulch Thickness	0.33'
Planting Mix Thickness	2.00'
Gravel Thickness	0.50'
Include Underdrain	Yes
Underdrain Diameter	0.50'
Underdrain Cover	0.50'
Underdrain Bedding	0.50'
Underdrain Trench Width	2.50'
AttachmentPoint	Center

PKTVERSION OUTPUT PARAMETER IN CIVIL 3D

Use your PKT files to build assemblies that will make your corridors better than ever before

As noted earlier, it is important to remember before combining portions of subassemblies into a composite subassembly that this is a **SUB**assembly Composer and not an ASSEMBLY Composer. You can always combine your subassemblies together into a complex assembly in Civil 3D without making a complex Flowchart.

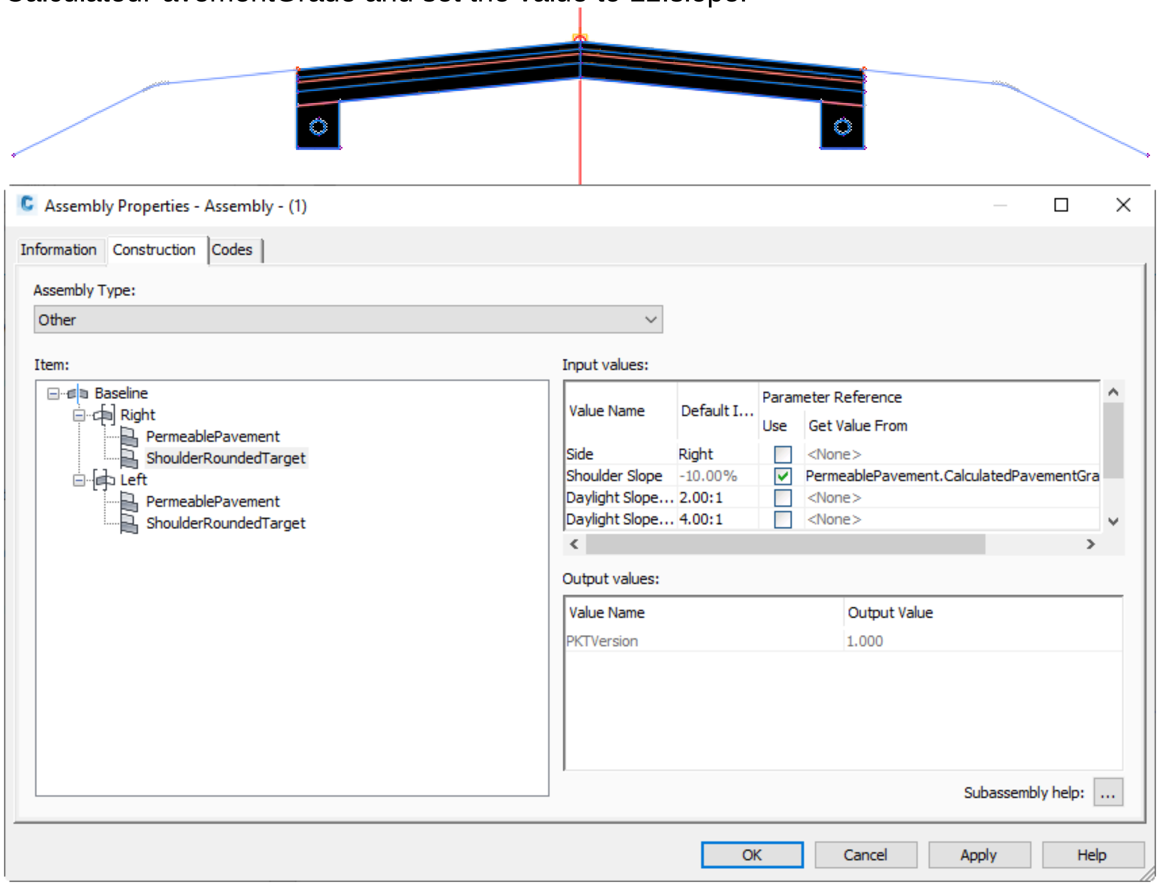
An instance where you may want to combine subassembly components is when you want the additional component in one instance but not in another; for example, you want guide rail in fill but not in cut. An instance where you may not want to combine the subassembly components is your road subassembly with your shoulder/daylight subassembly, because there may come a time when you want to use your shoulder/daylight subassembly with something other than a road, such as a swale.

As a final example, we will add an Output Parameter to one of the subassemblies in order to allow the user to use it in a subsequent subassembly. This is a great way to keep your subassemblies small and focused by handing off information in the form of output parameters.

Add Output Parameters for calculated values

1. Open PermeablePavement.pkt and save as PermeablePavement_v1.pkt.
    a. The PermeablePavement.pkt file allows for the top slope of the pavers to be calculated based on various target parameters. This example will add an output parameter for the top slope that will allow it to be referenced as an Output Parameter for another subassembly used subsequently in the same assembly, such as a shoulder link.
2. In the Input/Output Parameters window, click Create parameter.

3. For the new parameter (Parameter12) created, set the Name to CalculatedPavementGrade, Type to Double, and Direction to Output.
    a. While the Types available for Output Parameters are the same as the Input Parameters, they behave best with the Types: Double, Integer, or String.
4. In the Flowchart window, double click to enter the Pavers Sequence element.
5. In the Tool Box, in the Miscellaneous section, drag a Set Output Parameter to the bottom of the Pavers Sequence below S1.
6. In the Output Parameter Properties, select the Output Parameter as CalculatedPavementGrade and set the value to L1.slope.


Assembly Properties - Assembly - (1)

Information | Construction | Codes

Assembly Type: Other

Item:
- Baseline
  - Right
    - PermeablePavement
    - ShoulderRoundedTarget
  - Left
    - PermeablePavement
    - ShoulderRoundedTarget

Input values:

Value Name	Default I...	Parameter Reference Use	Get Value From
Side	Right	☐	<None>
Shoulder Slope	-10.00%	✓	PermeablePavement.CalculatedPavementGra
Daylight Slope...	2.00:1	☐	<None>
Daylight Slope...	4.00:1	☐	<None>

Output values:

Value Name	Output Value
PKTVersion	1.000

Subassembly help: ...

OK Cancel Apply Help

USE OUTPUT PARAMETER REFERENCE FOR SUBSEQUENT SUBASSEMBLIES: PERMEABLEPAVEMENT_V1.PKT

Endless Possibilities

Subassembly Composer is a versatile tool and with the PKT subassemblies provided by Autodesk you no longer always have to start from a blank Flowchart. By seeing what subassemblies are already out there, determining how they work, figuring out how to modify them to fit your needs, and troubleshooting along the way, before you know it your corridors will have endless possibilities. The best way to learn a program is by using it, so go ahead and try out the existing PKT subassemblies that come with Subassembly Composer for Civil 3D 2020. Before you know it, your mind will be opened to a whole different way of approaching corridors in your design.

For Further Assistance

1. Autodesk® Knowledge Network for Autodesk Subassembly Composer 2020: https://knowledge.autodesk.com/support/civil-3d/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Civil3D-SubassemblyComposer/files/GUID-C569F4E7-D548-410E-B7D6-942A927FFD0B-htm.html
2. Autodesk® Knowledge Network Subassembly Reference for Autodesk Civil 3D PKT Subassemblies: https://knowledge.autodesk.com/support/civil-3d/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Civil3D-Subassembly-Ref/files/GUID-4E4F898A-D521-4CFA-BBCB-FA6A01ABB8AD-htm.html
3. Autodesk® Forums for Civil 3D: http://forums.autodesk.com/t5/AutoCAD-Civil-3D/bd-p/66

Appendix: VB Expressions and API Functions

VB Expressions: Math

Emphasized values can be changed to reference the applicable value.

Math VB Expression	**Output**	**Description**
math.round(*2.568*,*2*)	2.57	Returns value rounded to the nearest specified decimal places (*ex. -2 = hundreds, -1 = tens, 0 = whole number, 1 = tenths, 2 = hundredths, etc.*)
math.floor(*2.568*)	2	Returns largest integer that is less than or equal to the specified value (i.e. rounds down)
math.ceiling(*2.568*)	3	Returns smallest integer that is greater than or equal to the specified value (i.e. rounds up)
math.max(*2.568*,*0.813*)	2.568	Returns larger of two specified values
math.min(*2.568*,*0.813*)	0.813	Returns smaller of two specified values
math.abs(*-2.568*)	2.568	Returns absolute value
math.pi	3.14159...	Returns value of the constant pi
math.sin(*math.pi*)	0	Returns sine of a specified angle measured in radians
math.cos(*math.pi*)	-1	Returns cosine of a specified angle measured in radians
math.tan(*math.pi*)	0	Returns tangent of a specified angle measured in radians
math.asin(*1*)	1.57079...	Returns angle measured in radians whose sine is the specified value
math.acos(*1*)	0	Returns angle measured in radians whose cosine is the specified value
math.atan(*1*)	0.78539...	Returns angle measured in radians whose tangent is the specified value
math.log(*math.e*)	1	Returns natural (base e) logarithm of a specified value
math.log10(*10*)	1	Returns base 10 logarithm of a specified value
math.exp(*1*)	2.71828...	Returns e raised to the specified power
math.pow(*2*,*3*)	8	Returns value raised to the specified power
math.sqrt(*81*)	9	Returns square root of a specified value
math.ieeeremainder(*7,2*)	1	Returns remainder of the first value divided by the second value
math.sign(*-2.1*)	-1	Returns an integer (-1 or +1) indicating the sign of the number

VB Expressions: Casting

Emphasized values can be changed to reference the applicable value.

Ctype VB Expression	Output	Description
CType(*10%,double*)	0.10	Converts first value into the specified variable type (integer, double, string)

VB Expressions: Logic

Emphasized values can be changed to reference the applicable value.

Logic VB Expression	Description
IF(P1.Y>P2.Y,2,3)	Used in a VB Expression, returns a value depending on whether the condition (*P1.Y>P2.Y*) is true (*value of 2*) or false (*value of 3*)
P1.Y>P2.Y	Returns true if *P1.Y* is greater than *P2.Y*
P1.Y>=P2.Y	Returns true if *P1.Y* is greater than or equal to *P2.Y*
P1.Y<P2.Y	Returns true if *P1.Y* is less than *P2.Y*
P1.Y<=P2.Y	Returns true if *P1.Y* is less than or equal to *P2.Y*
P1.Y=P2.Y	Returns true if *P1.Y* is equal to *P2.Y*
P1.Y<>P2.Y	Returns true if *P1.Y* is not equal to *P2.Y*
(P1.Y>P2.Y)AND(P2.X>P3.X)	Returns true if both the condition (*P1.Y>P2.Y*) AND the condition (*P2.x>P3.X*) are true
(P1.Y>P2.Y)OR(P2.X>P3.X)	Returns true as long as either the condition (*P1.Y>P2.Y*) OR the condition (*P2.x>P3.X*) is true
(P1.Y>P2.Y)XOR(P2.X>P3.X)	Returns true if only one of the two conditions (*P1.Y>P2.Y*), (*P2.x>P3.X*) is true (if both are true or both are false, then false is returned)

VB Expressions: Subassembly Composer Application Programming Interface (API) Functions

Emphasized values can be changed to reference the applicable element.

Points and Auxiliary Points Class

Point API Function	Description
P1.X	Horizontal distance from point *P1* to Origin
P1.Y	Vertical distance from point *P1* to Origin
P1.Offset	Horizontal distance from point *P1* to assembly baseline
P1.Elevation	Elevation of point *P1* relative to 0
P1.DistanceTo("*P2*")	Distance from point *P1* to point *P2* (*Always positive*)

Point API Function	Description
P1.SlopeTo("*P2*")	Slope from point *P1* to point *P2* (Upward = positive, Downward = Negative)
P1.IsValid	Point *P1* assigned & valid to use (*T/F*)
P1.DistanceToSurface(*SurfaceTarget*)	Vertical distance from point *P1* to *SurfaceTarget* (point above = positive, point below = negative)

Links and Auxiliary Links Class

Link API Function	Description
L1.Slope	Slope of link *L1*
L1.Length	Length of link *L1* (*Always positive*)
L1.Xlength	Horizontal distance between start and end of link *L1* (*Always positive*)
L1.Ylength	Vertical distance between start and end of link *L1* (*Always positive*)
L1.StartPoint	A point located at the start of link *L1* (Can be used in API Functions for P1 Class)
L1.EndPoint	A point located at the end of link *L1* (Can be used in API Functions for P1 Class)
L1.MaxY	Maximum Y elevation from a link's points
L1.MinY	Get the minimum Y elevation from a link's points
L1.MaxInterceptY(*slope*)	Apply the highest intercept of a given link's points to the start of another link
L1.MinInterceptY(*slope*)	Apply the lowest intercept of a given link's points to the start of another link
L1.LinearRegressionSlope	Slope calculated as a linear regression on the points in a link to find the best fit slope between all of them
L1.LinearRegressionInterceptY	The Y value of the linear regression link
L1.IsValid	Link *L1* is assigned & valid to use (*T/F*)
L1.HasIntersection("*L2*") *L1*.HasIntersection("*L2*", *true*, *true*)	*L1* and *L2* have an intersection, second input is a Boolean defining whether to extend *L1* with default of false, third input is a Boolean defining whether to extend *L2* with default of false (*T/F*)

Offset Target Class

Offset API Function	**Description**
OffsetTarget.IsValid	*OffsetTarget* is assigned & valid to use (*T/F*)
OffsetTarget.Offset	Horizontal distance from *OffsetTarget* to assembly baseline

Elevation Target Class

Elevation API Function	**Description**
ElevationTarget.IsValid	*ElevationTarget* is assigned & valid to use (*T/F*)
ElevationTarget.Elevation	Vertical distance from *ElevationTarget* to assembly baseline

Surface Target Class

Offset API Function	**Description**
SurfaceTarget.IsValid	*SurfaceTarget* is assigned & valid to use (*T/F*)

Superelevation Class

Superelevation API Function	**Description**
SE.HasLeftLI	Left lane inside superelevation slope is present & valid to use (*T/F*)
SE.HasLeftLO	Left lane outside superelevation slope is present & valid to use (*True/False*)
SE.HasLeftSI	Left shoulder inside superelevation slope is present & valid to use (*T/F*)
SE.HasLeftSO	Left shoulder outside superelevation slope is present & valid to use (*T/F*)
SE.HasRightLI	Right lane inside superelevation slope is present & valid to use (*T/F*)
SE.HasRightLO	Right lane outside superelevation slope is present & valid to use (*T/F*)
SE.HasRightSI	Right shoulder inside superelevation slope is present & valid to use (*T/F*)
SE.HasRightSO	Right shoulder outside superelevation slope is present & valid to use (*T/F*)
SE.LeftLI	Left lane inside superelevation slope
SE.LeftLO	Left lane outside superelevation slope
SE.LeftSI	Left shoulder inside superelevation slope
SE.LeftSO	Left shoulder outside superelevation slope
SE.RightLI	Right lane inside superelevation slope

Superelevation API Function	Description
SE.RightLO	Right lane outside superelevation slope
SE.RightSI	Right shoulder inside superelevation slope
SE.RightSO	Right shoulder outside superelevation slope

Baseline Class (*Note assembly baseline may or may not be the subassembly origin)

Baseline API Function	Description
Baseline.Station	Station on assembly baseline
Baseline.Elevation	Elevation on assembly baseline
Baseline.RegionStart	Station at the start of the current corridor region
Baseline.RegionEnd	Station at the end of the current corridor region
Baseline.Grade	Grade of assembly baseline
Baseline.TurnDirection	Turn direction of assembly baseline (*Left = -1, Non-curve = 0, Right = 1*)

EnumerationType Class

Enumeration API Function	Description
EnumerationType.Value	The string value of the current enumeration item

Subassembly Class

Subassembly API Function	Description
SA.IsLayout	Current preview mode is Layout Mode (*T/F*)

Cant Class

Cant API Function	Description
Cant.PivotType	Pivot method assigned to the current curve: Low Side Rail (left rail) = -1 Center Baseline = 0 High Side Rail (right rail) = 1
Cant.LeftRailDeltaElevation	Differential elevation for the left rail
Cant. RightRailDeltaElevation	Differential elevation for the right rail
Cant.TrackWidth	Track Width assigned to the alignment
Cant.IsDefined	Cant has been calculated on the alignment (*T/F*)