Autodesk University 2012

Reverse Engineering with Subassembly Composer for AutoCAD® Civil 3D®

Kati L. Mercier, P.E. – Nathan L. Jacobson & Associates, Inc.

CI3001 Subassembly Composer for AutoCAD Civil 3D 2013 software provides users with an easy-to-use interface for visually creating complex subassemblies without the need for advanced programming knowledge. While you will sometimes find yourself starting with a clean slate, other times you may find yourself needing to modify a Subassembly Composer PKT file created by another user. Intended for existing Subassembly Composer for AutoCAD Civil 3D users, this class explores how to break apart an existing Subassembly Composer flowchart and make it your own. You will see the problems and pitfalls that can make your own flowcharts difficult to understand. You will also learn tips and tricks for making your flowcharts easy to share and be further enhanced by coworkers, colleagues, and the Civil 3D community at large.

Learning Objectives

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

- Deconstruct an existing Subassembly Composer PKT file to identify its functions
- Create new subassemblies to perform custom functions by making modifications to existing PKT files
- Combine components of multiple PKT files together into a composite subassembly using various workflow elements
- Create clear and concise flowcharts that will enhance a your ability to understand and modify your subassembly in the future

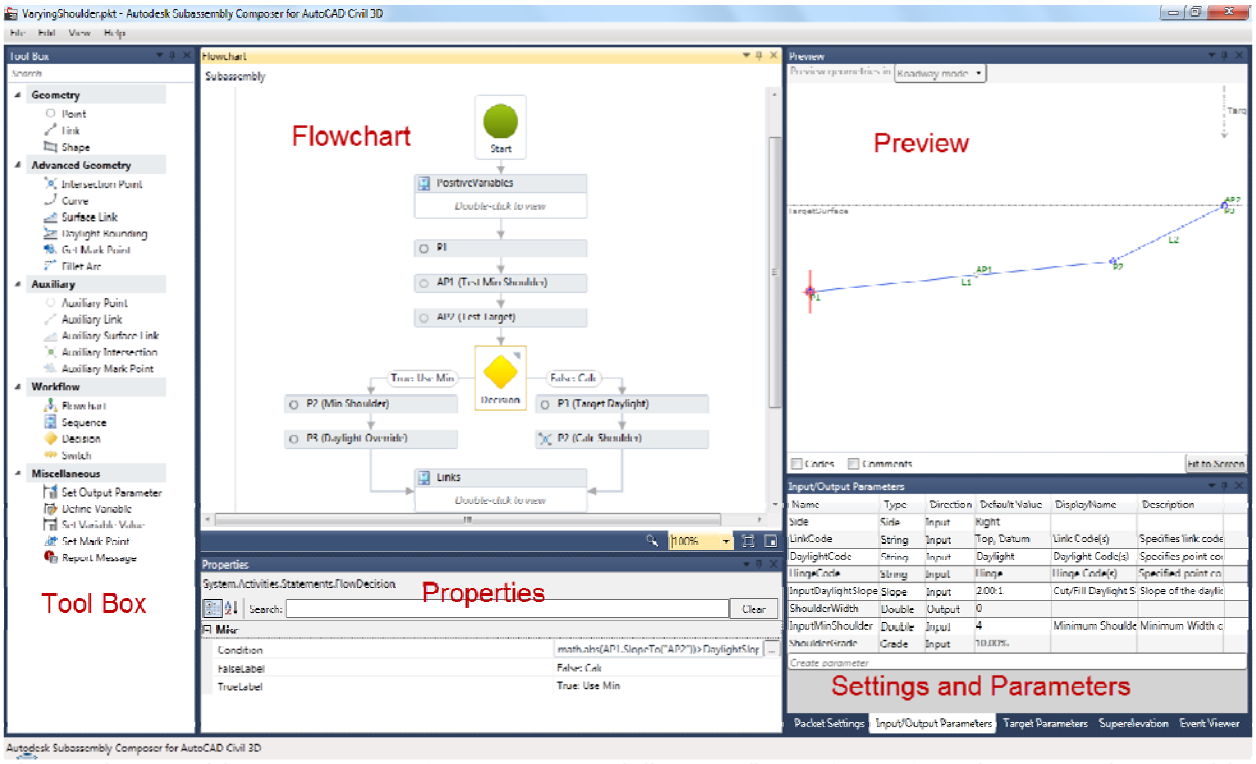
About the Speaker

Kati Mercier is a Professional Engineer licensed in the states of Connecticut and New York. Since 2004 she has served as a project engineer at Nathan L. Jacobson & Associates, Inc., a civil and environmental engineering firm that provides engineering, review, and design consultation to local municipalities and other clients. Most recently, Kati has coauthored Mastering AutoCAD® Civil 3D® 2013. She is an AutoCAD Civil 3D Certified Professional, presented at Autodesk University 2011, and currently operates civil4d.com, a leading site for the discussion of Civil 3D and other civil engineering topics.

katimercier@gmail.com

www.twitter.com/KDinCTPE

Autodesk Subassembly Composer for AutoCAD® Civil 3D®



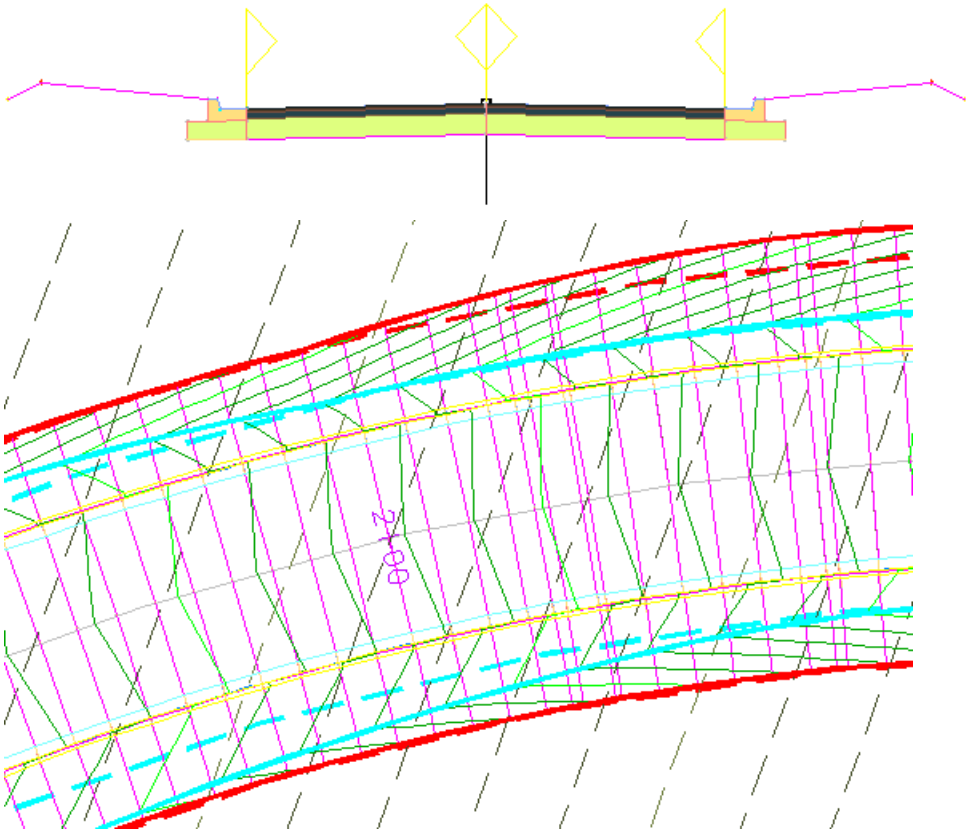
The 'Subassembly Composer® for AutoCAD Civil 3D®' (hereafter referred to as Subassembly Composer) program is used to create subassemblies to supplement the stock subassemblies that come with Civil 3D. This lecture assumes that you have a working knowledge of the program and have created basic subassembly PKT files. All of the examples in this lecture utilize Autodesk Subassembly Composer 2013.

As with many things in life though, there are multiple ways to reach the same end result. Everyone's thought process is unique and therefore even the same subassembly concept will come out looking different when made by a different person. As long as both produce the desired end result, how you get from point A to point Z doesn't really matter. In this lecture we will explore how to create the subassembly you need without necessarily starting from scratch.

Deconstruct an existing Subassembly Composer PKT file to identify its functions

While you will sometimes find yourself starting with a clean slate, other times you may find yourself needing to modify a Subassembly Composer PKT file created by another user, or even 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. Import the subassembly into Civil 3D and try it; examine and manipulate the Settings and Parameters in Subassembly Composer; and dig into the Flowchart.

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 targets, 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 or to turn off optional targets while keeping on required targets. So in order to test these scenarios you will need to use the subassembly in Civil 3D.

Information	
Name	VaryingShoulder - Right
Description	
Show Tooltips	Yes
General	
Data	
ADVANCED	
Parameters	
Side	Right
Link Code(s)	Top, Datum
Daylight Code(s)	Daylight
Hinge Code(s)	Hinge
Cut/Fill Daylight Slope	2.00:1
Minimum Shoulder Width	4.00'
ShoulderGrade	10.00%

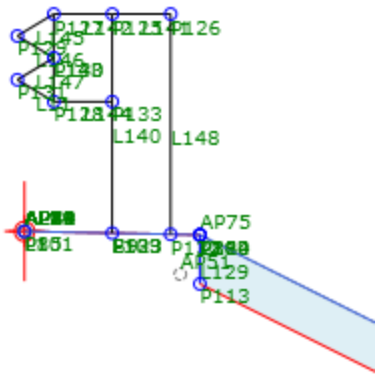
- **What does it ask for as input parameters?**
  - Are the dimensions what you want?
  - Does it ask for slopes where you want slopes and grades where you want grades?
  - Does it give you too many or not enough parameters?
  - Does it allow you to do all the targets you want?

Target	Object Name	Subassembly	Assembly Group
Surfaces	<Click here to set all>		
Target Surface	EG	VaryingShoulder - Left	Group - Left
Target Surface	EG	VaryingShoulder - Right	Group - Right
Width or Offset Targets			
Width Alignment	<None>	LaneSuperelevationAOR - Left	Group - Left
Target Daylight Offset	Polyline- 5	VaryingShoulder - Left	Group - Left
Width Alignment	<None>	LaneSuperelevationAOR - Right	Group - Right
Target Daylight Offset	Polyline- 6	VaryingShoulder - Right	Group - Right
Slope or Elevation Targets			
Outside Elevation Profile	<None>	LaneSuperelevationAOR - Left	Group - Left
Outside Elevation Profile	<None>	LaneSuperelevationAOR - Right	Group - Right

- **What does it give you as output?**
  - Does it handle cuts and fills correctly?
  - Does it have the point and link codes that you want?

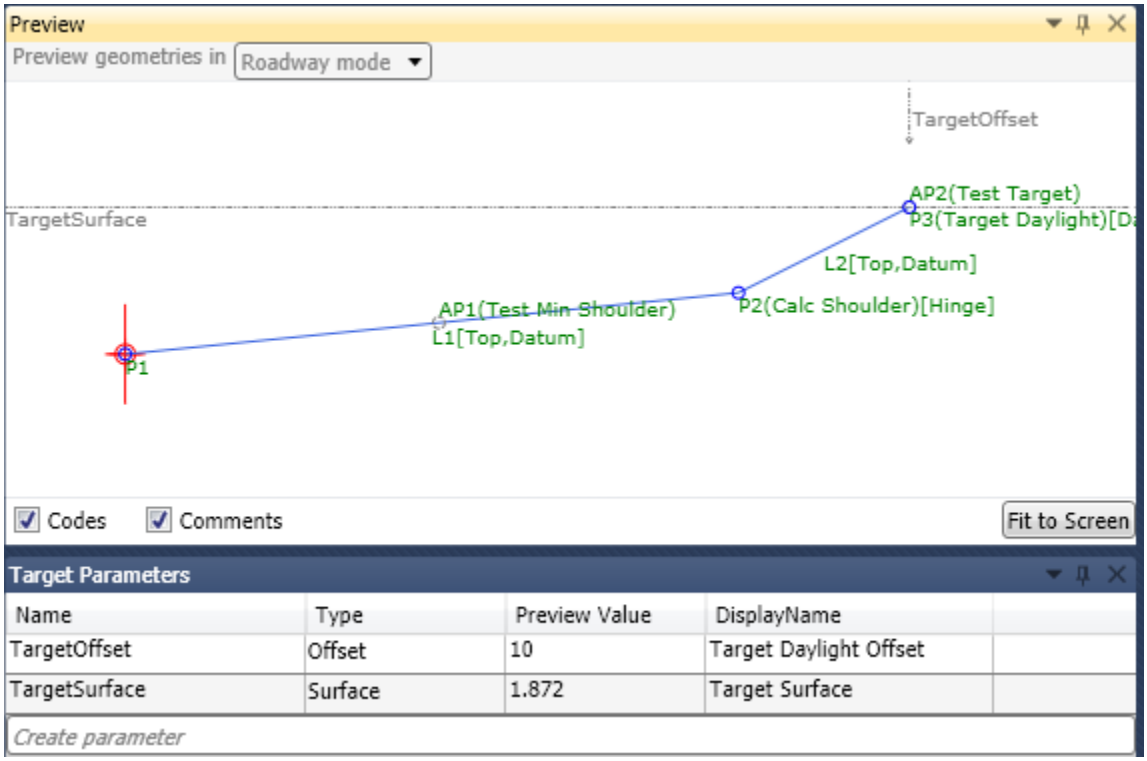
Name	Description	Style	Label Style	Render Ma…	Material Ar…	Pay Item
Link						
<No Codes>		Uncoded	<none>	<none>	<none>	<none>
Top	Top formati…	Top	<none>	<none>	<none>	<none>
Pave	Any finishe…	Pave	<none>	Sitework.Pa…	<none>	<none>
Pave1	First pavem…	Pave1	<none>	<none>	<none>	<none>
Pave2	Second pav…	Pave2	<none>	<none>	<none>	<none>
Base	Link on the …	Base	<none>	<none>	<none>	<none>
SubBase	Links on su…	SubBase	<none>	Sitework.Pl…	<none>	<none>
Datum	Bottom finis…	Datum	<none>	<none>	<none>	<none>
Curb	Link compris…	Curb - Top	<none>	Concrete.C…	<none>	<none>
Point						
<No Codes>		Uncoded	<none>			<none>
Flange	Gutter flan…	Flange	<none>			<none>
Flowline_Gutter	Gutter point	Gutter	<none>			<none>
Top_Curb	Top of curb	Curb	<none>			<none>
Back_Curb	Back of a curb	Curb	<none>			<none>
Daylight	Daylight poi…	Daylight	<none>			<none>
Hinge	Hinge point …	Hinge	<none>			<none>
Shape						

Examine and manipulate the Settings and Parameters



Similar to 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. Try dragging the Target Parameters to different values (or inputting different values numerically for the Input Parameters and/or Target Parameters). Turn on and off the Codes and/or Comments. The comments are shown in parentheses while the codes are shown in brackets. As shown at right, notice that the colors signify common codes with links and shapes; in this example, the blue is “Top”, the red is “Datum” and the black is “Guiderail”.

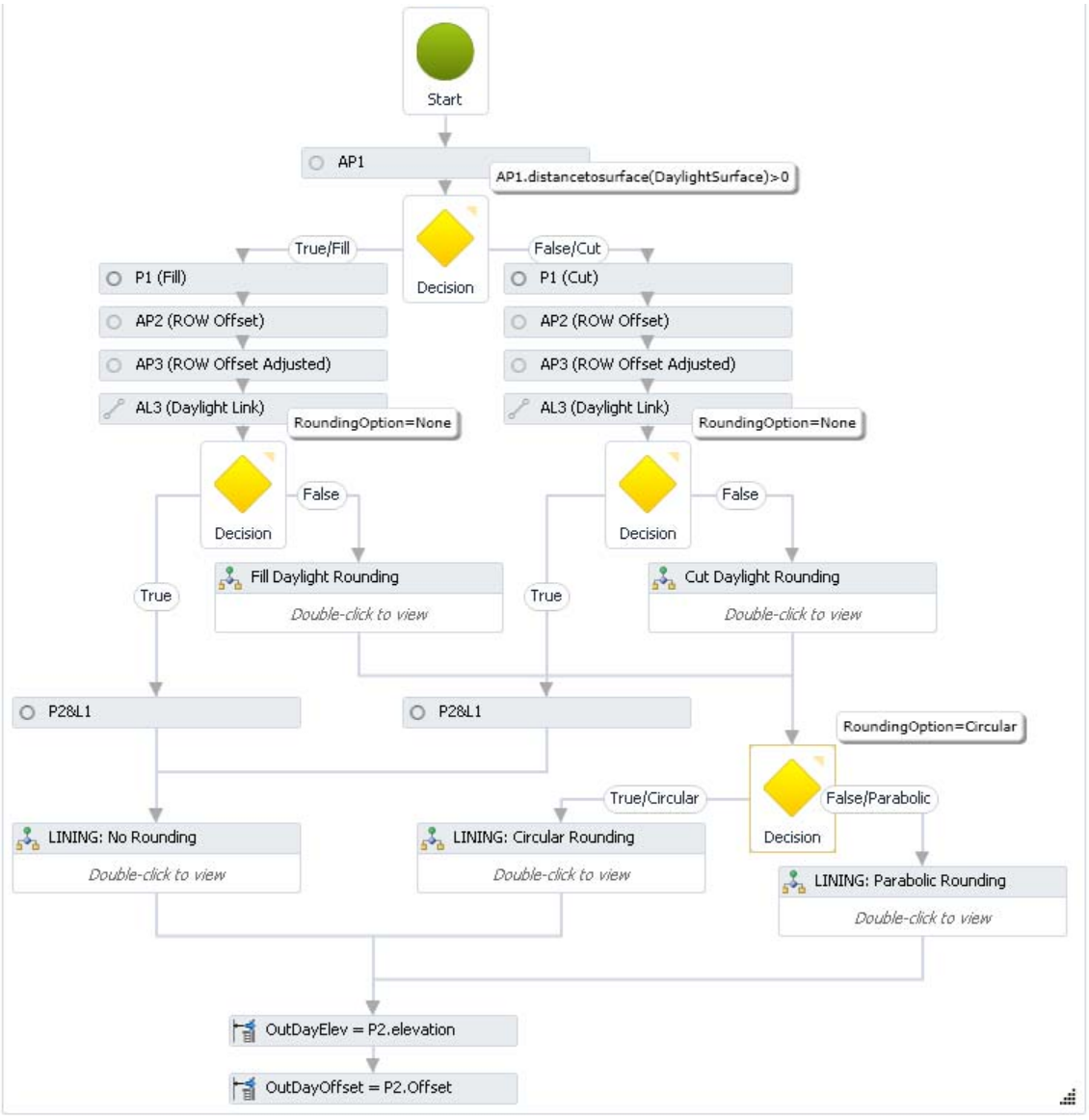


Dig into the Flowchart

Trace flow branches

Take things one branch at a time. Start at the top and work your way down. Remember that you can click the tiny triangle flag at the upper right of a Decision element to display the condition. 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 person creates a subassembly it doesn't magically all appear at once and you can't expect yourself to understand it all at once either.

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 designed did not provide one already.



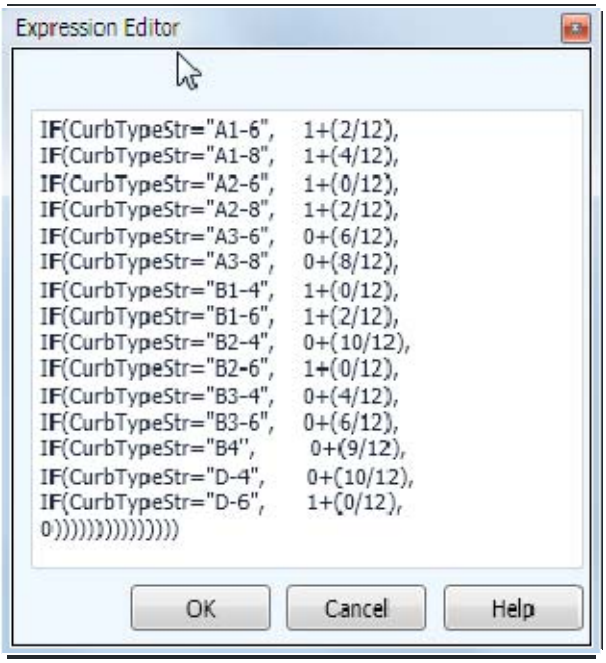
In Sequence elements you can't "break a connection" like you can in a Flowchart element to test only to a certain point, so instead unconnected the original Sequence element, create an empty Sequence element and connect it where the original one was, then cut and paste the elements back into the new 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 may also find it helpful to bypass a section of logic and test a later section by readjusting your connection arrows, just remember that you need to have all the referenced points and links previously defined otherwise you will get errors.

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) and match them up approximately. The unknown calculated value will be displayed in your Target Parameters.
- **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. You may run across a subassembly in which someone has used a function you don't recognize. 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 as shown in the image are right. This example shows how to associate a variable (in this case a curb height) with a table of values based on a string value defining which curb type to use.



- **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.

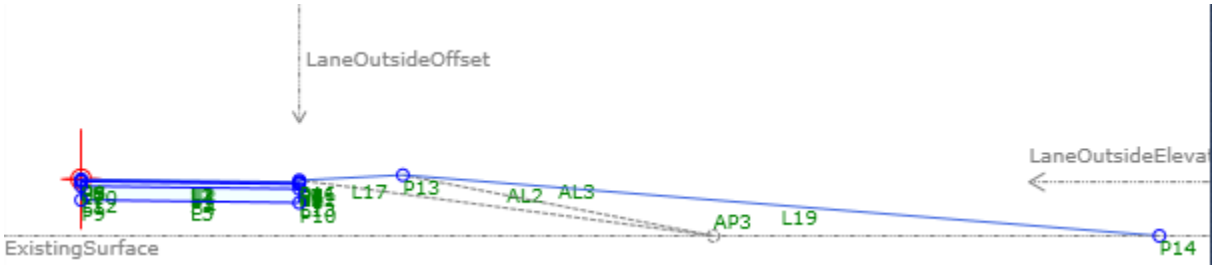
Create new subassemblies to perform custom functions by making modifications to existing PKT files

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 add a shape where there wasn't one previously. Or maybe you want to change the string values for the point or link codes. Maybe the subassembly was originally created to work in feet but now you want to work in inches. All of these changes are doable, but first you must know where to look to make your desired changes.

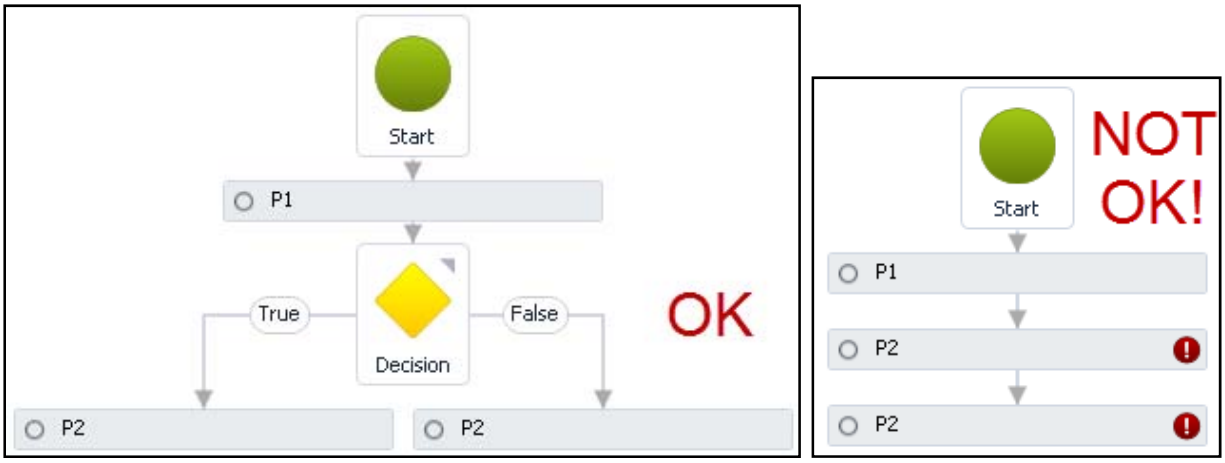
Combine components of multiple PKT files together into a composite subassembly using various workflow elements

The first thing to remember before combining portions of subassemblies into a composite subassembly is to remember that this is a SUBASSEMBLY Composer and not an ASSEMBLY Composer. You will always have the ability to combine your subassemblies together into a complex assembly in Civil 3D.

An instance where you may want to combine a subassembly component 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.



When you combine a portion of one subassembly with another subassembly it is important to remember that Subassembly Composer hates duplicate Names. There can only be one P2 along any given branch of the flowchart; but there can be another P2 on a separate branch, they just cannot coexist as shown in the images below. (Of course this may seem like an over simplified example but it can happen easily once you have lots of points). A good idea is to pick a range of numbers that you know is beyond anything that already exists and change the numbers in the subassembly portion that is being copied to that range of numbers before you copy and paste, this well help avoid some errors.



You may also find it helpful to think ahead of time of elements that you may want to copy to other subassemblies and to compartmentalize them into a Sequence or Flowchart element, this will make copying and pasting the group much easier rather than having to select each element individually each time and having to rearrange them after pasting them into their new location.

Create clear and concise flowcharts that will enhance a your ability to understand and modify your subassembly in the future

By looking at how other people build their logic your subassemblies will improve.

- Use descriptive parameter names and/or parameter display names.
- Add descriptions to Input/Output Parameters.
- Add comments to non-obvious points and links; especially auxiliary ones.
- Break common sections of a subassembly into a Sequence or Flowchart element.
- Define codes to be used throughout the subassembly as a variable so that you can change it in one place and it will change in all the locations instead of having to hunt for all the instances.
- Provide a Help File documenting any complex logic and showing sketches.

For Further Assistance

1. Autodesk® WikiHelp for Autodesk Subassembly Composer: http://wikihelp.autodesk.com/AutoCAD_Civil_3D/enu/2013/Help/Autodesk_Subassembly_Composer

2. Autodesk® Discussion Groups for AutoCAD Civil 3D: http://forums.autodesk.com/t5/AutoCAD-Civil-3D/bd-p/66

3. Twitter hashtag: #SACCivil3D

APPENDIX

VB Expressions: Math

Emphasized values can be changed to reference the applicable value.

Sample VB Expression	Output	Description
math.round(*2.568*,*2*)	*2.57*	Returns a 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 the largest integer that is less than or equal to the specified value (i.e. rounds down)
math.ceiling(*2.568*)	*3*	Returns the 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 the larger of a series of two specified values
math.min(*2.568*,*0.813*)	*0.813*	Returns the smaller of a series of two specified values
math.abs*(-2.568*)	*2.568*	Returns the absolute value
math.pi	*3.14159...*	Returns the value of the constant pi
math.e	*2.71828...*	Returns the value of the constant e
math.sin(*math.pi*)	*0*	Returns the sine of a specified angle measured in radians

Sample VB Expression	Output	Description
math.cos(*math.pi*)	*-1*	Returns the cosine of a specified angle measured in radians
math.tan(*math.pi*)	*0*	Returns the tangent of a specified angle measured in radians
math.asin(*1*)	*1.57079...*	Returns the angle measured in radians whose sine is the specified value
math.acos(*1*)	*0*	Returns the angle measured in radians whose cosine is the specified value
math.atan(*1*)	*0.78539...*	Returns the angle measured in radians whose tangent is the specified value
math.log(*math.e*)	*1*	Returns the natural (base e) logarithm of a specified value
math.log10(*10*)	*1*	Returns the 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 a value raised to the specified power
math.sqrt(*81*)	*9*	Returns the square root of a specified value

VB Expressions: Logic

Emphasized values can be changed to reference the applicable value.

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

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*)
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(*Surface Target*)	Vertical distance from point *P1* to *SurfaceTarget* (*point above = positive, point below = negative*)

Links and Auxiliary Links Class

L1.Slope	Slope of link *L1*
L1.Length	Length of link *L1* (*Always positive*)
L1.Xlength	Horizontal distance between start to end of link *L1* (*Always positive*)
L1.Ylength	Vertical distance between start to 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

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

Elevation Target Class

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

Surface Target Class

SurfaceTarget.IsValid	*SurfaceTarget* is assigned & valid to use (*T/F*)

Superelevation Class

SE.HasLeftLI	Left lane inside superelevation slope is present & valid to use (*T/F*)
SE.HasLeftLO	Left lane outside superelevation slope is present and 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
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.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

EnumerationType.Value	The string value of the current enumeration item

Subassembly Class

SA.IsLayout	Current preview mode is Layout Mode (*T/F*)

Cant Class

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*)