Civil Site Design

# **ScriptX** – A Programmer's Reference Guide

**Cross-Section Scripting Language for Civil Site Design**

## Contents

## Introduction to ScriptX

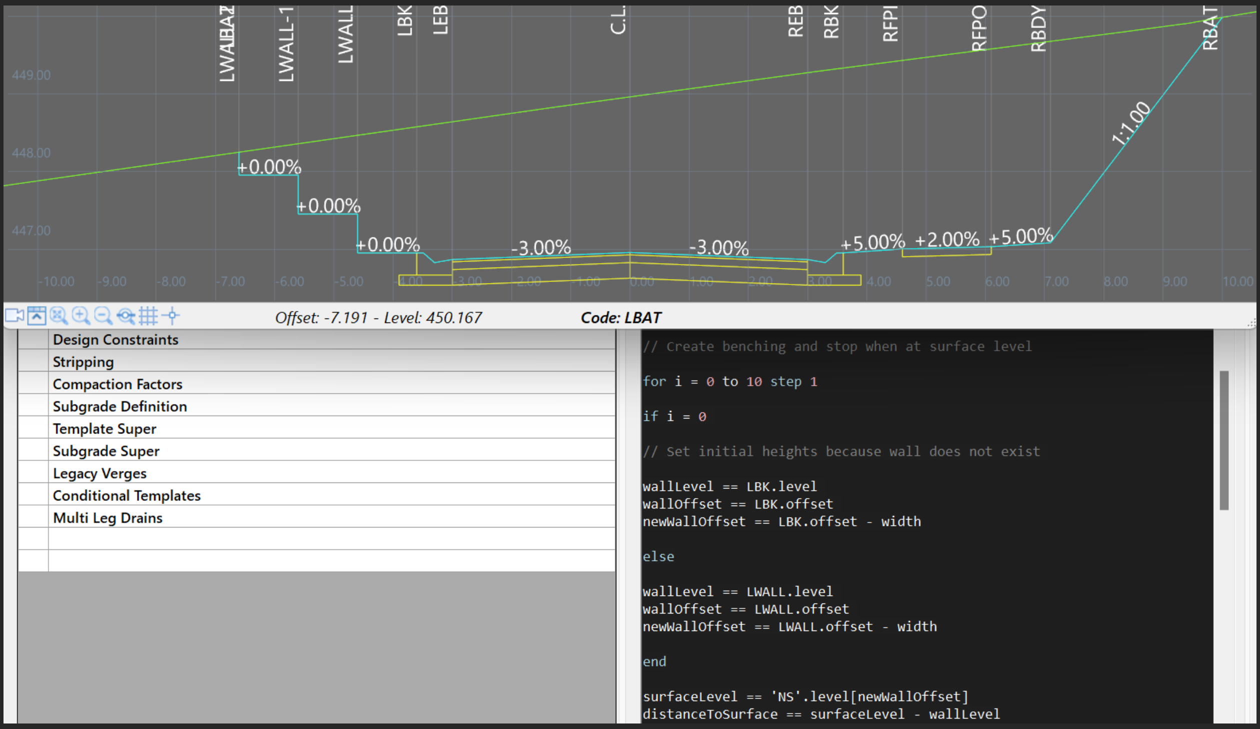
ScriptX is a bespoke scripting language within Civil Site Design, enabling users to craft custom scripts for the modification and manipulation of cross sections.

Through the Variations section in the Design Data Form, users can integrate ScriptX scripts into their workflows. These scripts can work in concert with other standard Variations, although it is crucial to keep in mind the significance of the operation sequence.

What makes ScriptX a useful tool for your toolkit?

ScriptX shines by providing the ability to incorporate logic operations into cross-section editing. Consider a scenario where there's a variation that widens a code to match the alignment. However, what if you need this widening to happen only under specific conditions? Or, what if you want to couple the widening with an increase in the code's offset? This level of nuanced control is exactly what ScriptX empowers you with.

A ScriptX script is incorporated into the Design Data Form by specifying a chainage range. The script is applied to each section within the chainage range individually, processing them one by one.



Offset: -7.191 - Level: 450.167

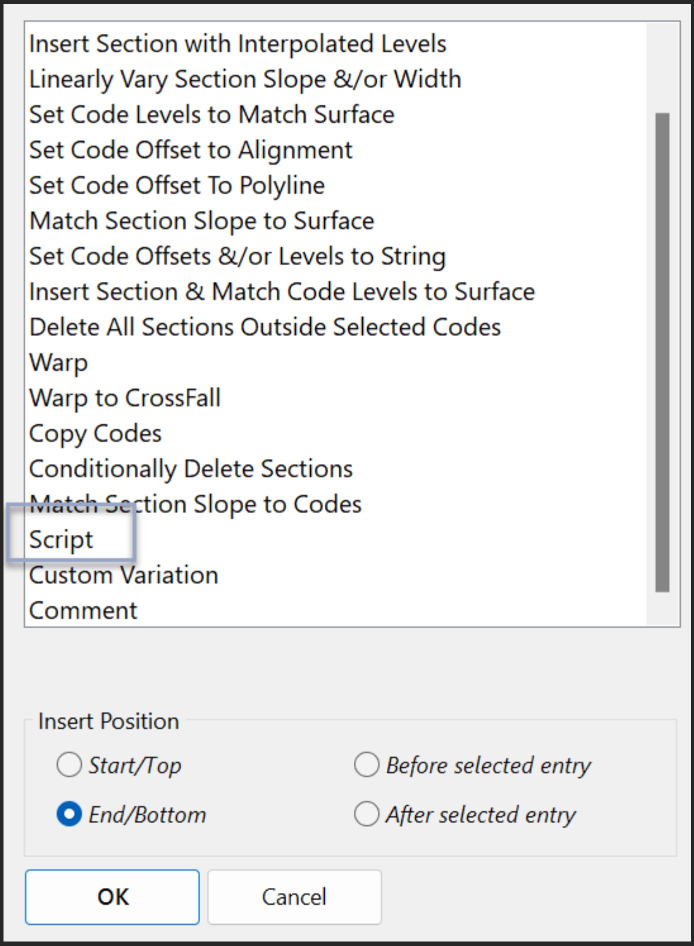
Code: LBAT

Design Constraints
Stripping
Compaction Factors
Subgrade Definition
Template Super
Subgrade Super
Legacy Verges
Conditional Templates
Multi Leg Drains

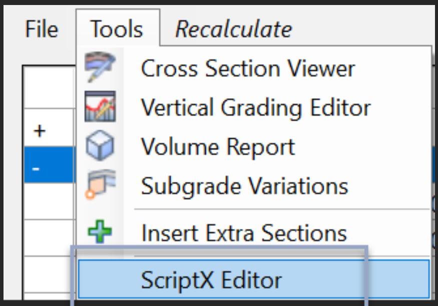
```
// Create benching and stop when at surface level

for i = 0 to 10 step 1

if i = 0

// Set initial heights because wall does not exist

wallLevel == LBK.level
wallOffset == LBK.offset
newWallOffset == LBK.offset - width

else

wallLevel == LWALL.level
wallOffset == LWALL.offset
newWallOffset == LWALL.offset - width

end

surfaceLevel == 'NS'.level[newWallOffset]
distanceToSurface == surfaceLevel - wallLevel
```

## Getting Started

Create new Scripts by including them as a Variation through the Design Data Form.

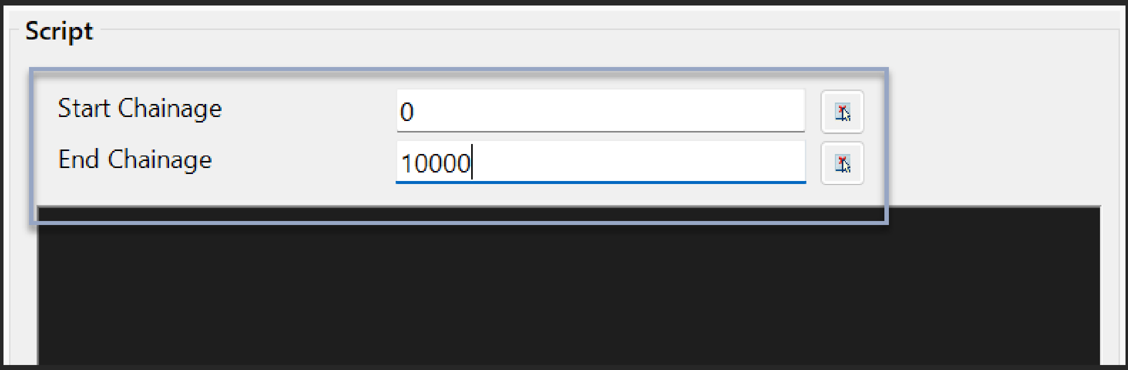

Insert Section with Interpolated Levels
Linearly Vary Section Slope &/or Width
Set Code Levels to Match Surface
Set Code Offset to Alignment
Set Code Offset To Polyline
Match Section Slope to Surface
Set Code Offsets &/or Levels to String
Insert Section & Match Code Levels to Surface
Delete All Sections Outside Selected Codes
Warp
Warp to CrossFall
Copy Codes
Conditionally Delete Sections
Match Section Slope to Codes
Script
Custom Variation
Comment
Insert Position
Start/Top
Before selected entry
End/Bottom
After selected entry
OK
Cancel

Scripts can also be created using the ScriptX Editor, which can be accessed via the Tools drop-down menu in the Design Data Form.



Scripts that have already been created can be run using the **Custom Variation** option in the Variations list. Scripts contained in the ScriptX folder in the Common-10 folder will be listed.

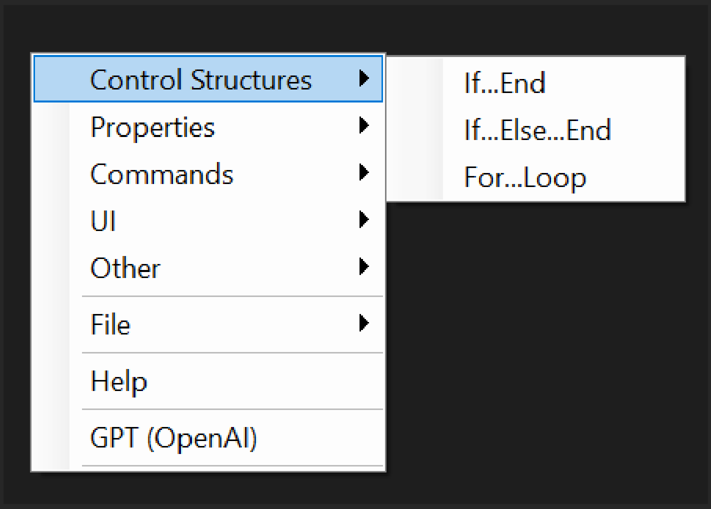
When you are establishing a new Script, you will need to determine and input the start and end chainages, which will define the range over which the Script will be functional. All sections between the chainage range will be processed individually.



| | |
|---|---|
| Start Chainage | 0 |
| End Chainage | 10000 |

Utilise the context menu (accessible by right-clicking) to embed elements like control structures, object properties, and other directives.

Through the File Menu, you have the ability to store and retrieve code snippets from a file.

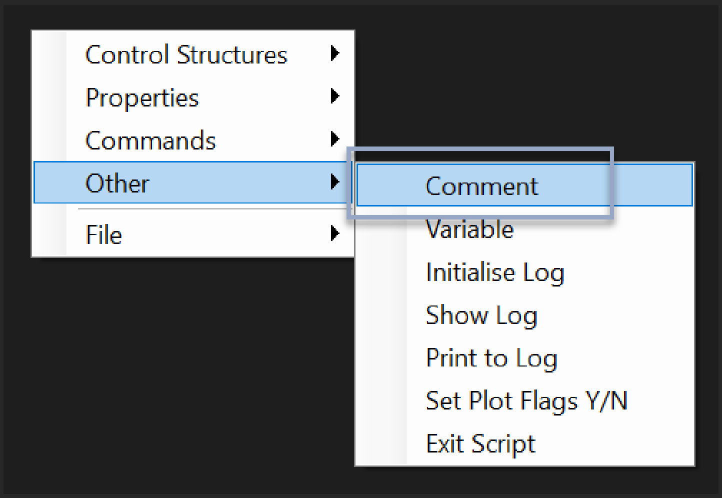


## Language Basics

### Comments

To include a comment within your code, simply prefix the line with "//". It is recommended that you begin your script by explaining what it does.

```
// This script forms benching in cut situations
```

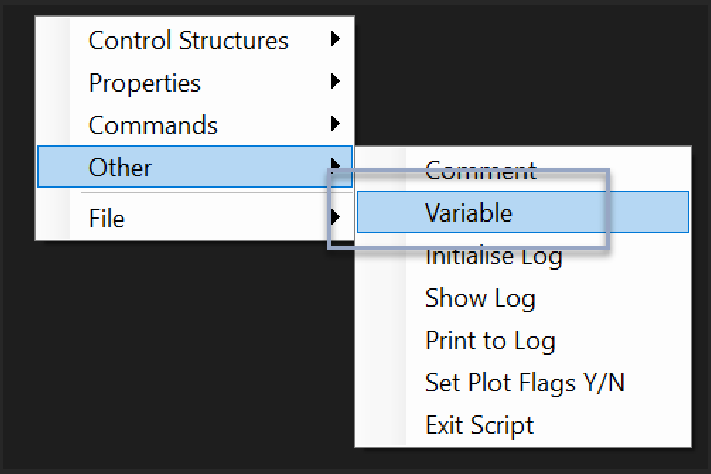


## Variables

In ScriptX, variables are established by starting a line with the variable name, followed by "==" to set its value. Variables can take numeric values or expressions, and they can even reference elements from Civil Site Design objects like cross section codes, alignments, surfaces, and profiles (strings).

For instance, you could write:

```
// Declare and assign variables

x == 10
y == 20

newWidth == LEB.Width + 1
newLevel == LEB.Level + .5
```



In ScriptX, you can assign objects such as codes, alignments, surfaces, or profiles to variables. This functionality is made intuitive by simply enclosing the object's name within single quotation marks. For example, you could write myCode = 'LEB' or myAlignment = 'Widen' to assign the corresponding object to a variable. ScriptX intelligently recognises the assignment of an object due to the use of single quotation marks.

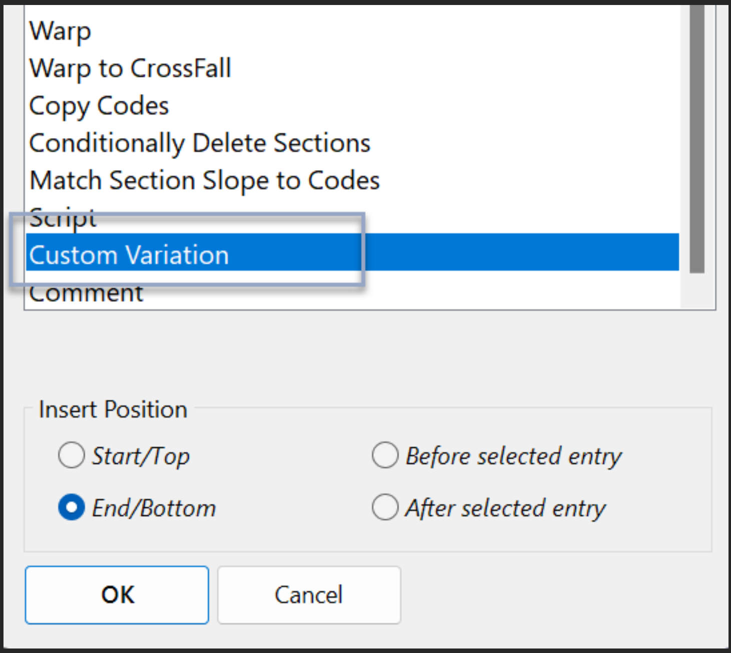
This feature adds a layer of adaptability to your scripts. By assigning objects to variables, you can swiftly adapt your scripts to different designs. All it takes is changing the assigned object once at the top of your script, and ScriptX ensures that the new assignment is correctly referenced throughout the rest of the script. This saves time and reduces potential errors, making it even more effortless to reuse and adapt your scripts across various design scenarios.

For instance, you could write:

```
surface == 'NS'
code == 'LEB'

height == surface.Level[code.Offset]
```

## UI

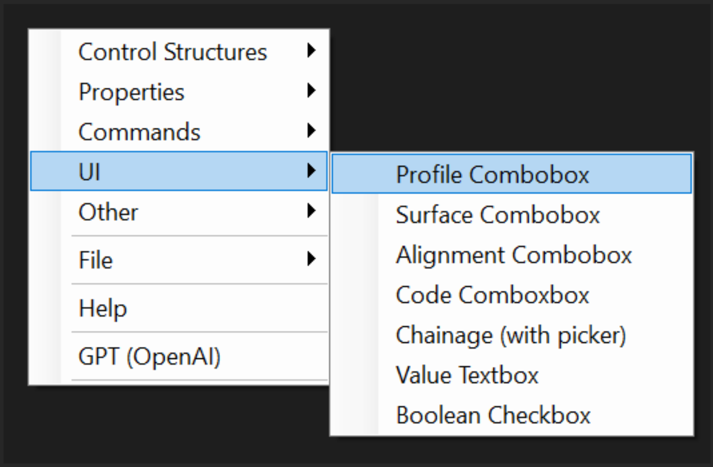
ScriptX, when executed via the "Custom Variation" in the Design Data Form, can incorporate a user interface (UI) defined within the script. This UI presents a panel of controls for user interaction. The UI elements include various types like code selectors, surface and alignment combos, profile selectors, value inputs, boolean toggles, and chainage inputs. Each element is structured with a specific type, a default value, and an accompanying prompt. User selections from these UI elements are captured and stored in variables.


Warp
Warp to CrossFall
Copy Codes
Conditionally Delete Sections
Match Section Slope to Codes
Script
Custom Variation
Comment

Insert Position
( ) *Start/Top*
(●) *End/Bottom*
( ) *Before selected entry*
( ) *After selected entry*

Below are examples of how to integrate UI elements into your script:

```
curCode == <CODECOMBO|LFPI|Code: >
curSurface == <SURFACECOMBO|NATURAL SURFACE|Surface: >
curAlignment == <ALIGNMENTCOMBO|MY ALIGN|Alignment:>
curProfile == <PROFILECOMBO|MY PROFILE|Profile:>
curValue == <VALUE|0|Value: >
curOption == <BOOL|0|Active: >
curChainage == <CHAINAGE|0|Chainage: >
```



## Control Structures

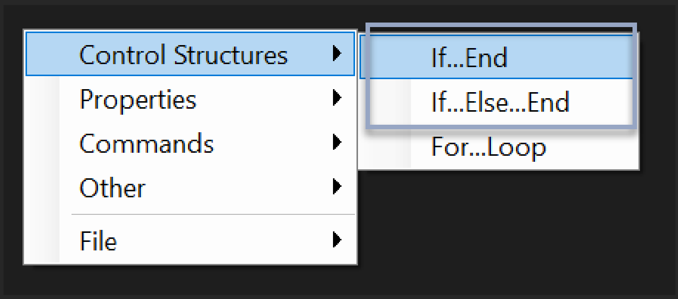
### If statement

The ‘**if**’ statement in ScriptX is a fundamental construct for implementing conditional logic within your scripts. It evaluates a specified condition and, based on the outcome, decides whether to execute a block of code or not. An “**if**” statement begins with the keyword **"if",** followed by a variable and a logical operator, and finally, another variable. The logical operator can be one of the following: "=", "!=", "<", ">", "<=", or ">=". If the condition evaluates to true, the script will execute the block of code following the if statement up to the matching "**end**". If the condition is false, the script skips the code block under the if statement and continues execution from the line following **"end".** This allows for complex and dynamic control flow within your scripts.

For instance, you could write:

```
// Declare and assign variables

LEBWidth == LEB.Width

// Use variables in conditional statements

if LEBWidth > 5

setcodeslope LEB|-4

else

setcodeslope LEB|-5

end
```

Please note that **"if"** statements are not designed to process complex expressions directly. If you need to utilise a complex calculation within an **"if"** statement, you should first perform this calculation and store the result in a variable. Then, you can reference this variable within the **"if"** statement. This approach will ensure proper handling and accuracy of your conditional logic.



## For loop

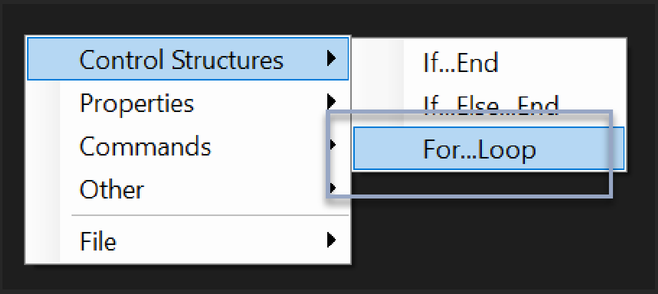
The ‘**for**’ loop in ScriptX provides a way to execute a block of code repetitively over a defined sequence of numbers. It follows the syntax

**for <*variableName*> = <*startValue*> To <*endValue*> Step <*stepValue*>**

Here, <*variableName*> is the name of the variable that will hold the current iteration value. <*startValue*> is the initial value from where the loop starts, and <*endValue*> is the final value where the loop should end. <*stepValue*> is the increment applied in each iteration. Inside the **'for'** loop, the code will execute repeatedly, with the <*variableName*> taking on each value in the range from <*startValue*> to <*endValue*>, increasing by <*stepValue*> each time. The loop ends when the <*variableName*> exceeds the <*endValue*>. Always end the for-loop block with the work ‘**loop**’.

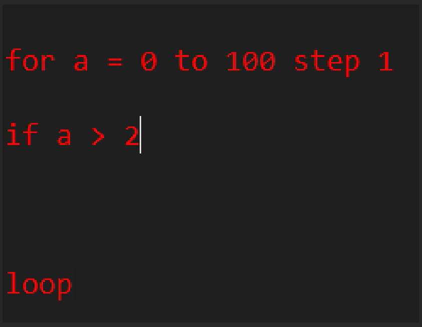
For instance, you could write:

```
// Declare and assign a variable
sum == 0

// Use a for loop to add numbers

for i = 0 to 10 step 1

  sum == sum + i

loop
```



Remember, always end a for-loop with the word **"loop"** and always end an **"if"** statement with the word **"end"**.

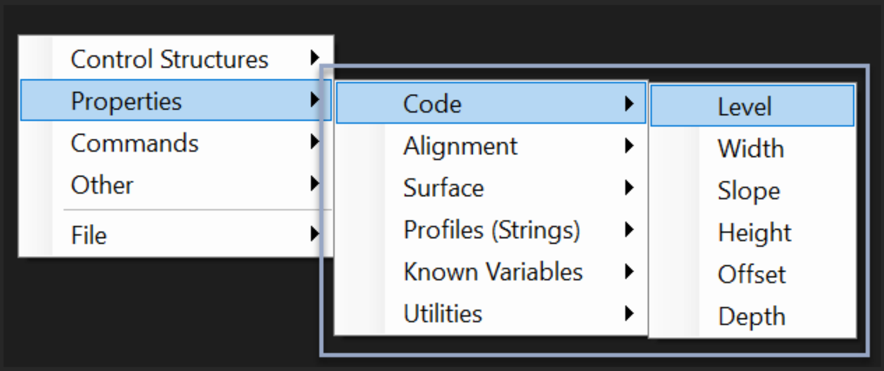
All lines of the script will highlight red if there is a major syntax error.



```
for a = 0 to 100 step 1

if a > 2



loop
```

## Civil Site Design Object Properties

In ScriptX, you can access a wide range of Civil Site Design (CSD) properties within your scripts. These properties provide useful information about the current state of your design and can be utilised to make dynamic adjustments based on existing conditions. For instance, you can retrieve properties related to elevation levels, chainage values, slope information, and more. Leveraging these properties within your scripts enables you to create complex, adaptive scripts that can react to the nuances of your specific design. Keep in mind, these properties are inherently read-only. To modify codes related to your properties, you would use commands or functions within your script.



## Code

To access a property associated with a cross section code, you'll need to use the code's name, followed by a period (".") symbol, and then the specific property name. This notation allows you to directly reference and manipulate properties tied to the particular cross section code.

Example: **LEB**.*Width*

The "Depth" property requires an additional argument. In this case, the argument is the name of the surface. If a property requires multiple arguments, then the ("|") symbol is used to split the arguments.

Example: **LEB**.*Depth[NS]*

| **Property** | **Description** | **Notes** |
|---|---|---|
| Level | The level of the code | |
| Width | The distance of the code from the last code | |
| Slope | The slope of the code from the last code. For example, LEB might have a slope value of -3. | Codes in CSD can be added either with slope or vertical distance. If the code was added using height, then use the height property. |
| Height | The height (vertical distance) of the code from the last code. | Codes in CSD can be added either with slope or vertical distance. If the code was added using slope, then use the slope property. |
| Offset | The offset distance calculated from the main alignment. A negative offset will mean the code is left of the alignment. | |

|  |  |  |
|---|---|---|
| Depth | The depth calculated from a specified surface. | Example: **LEB**.*Depth[NS]* |

**Alignment**

When you're making a reference to an alignment, always enclose the name within single quotation marks (' '), placing these symbols at both the beginning and the end of the name.

Example: **'My Alignment'**.*Offset*

| **Property** | **Description** | **Notes** |
|---|---|---|
| Offset | The offset distance from the current section | |
| Chainage | The "shifted chainage" from the location of the current section | |
| OffsetAtChainage | The offset distance at a defined | |

**Profile (CSD String)**

There are two distinct methods for referencing profile data. One approach is to reference properties directly from the profile itself. Alternatively, you can reference properties from a specific code that resides on the profile.

**Direct from a Profile**

Example: **'My Profile'**.*Grade*

| **Property** | **Description** | **Notes** |
|---|---|---|
| Offset | The offset distance from the main alignment at the current section. | |
| Chainage | The "shifted chainage" from the location of the current section | |

|  |  |  |
|---|---|---|
| Level | The level of the profile calculated at the shifted chainage |  |
| Grade | The vertical grade of the profile calculated at the shifted chainage |  |
| Depth | The level of the profile calculated at the shifted chainage minus the surface level of the specified surface | **‘My Profile’.***Depth[NS]* |

__**Referencing a code from a Profile**__

Example: **‘My Profile’|LEB**.*Level*

| **Property** | **Description** | **Notes** |
|---|---|---|
| Level | The level of the code calculated at the shifted chainage |  |
| Offset | The offset distance of the code from the profile alignment |  |
| CLOffset | The offset distance from the main alignment at the current section to the profile alignment |  |
| Chainage | The “shifted chainage” from the location of the current section |  |
| Depth | The level of the code calculated at the shifted chainage minus the surface level of the specified surface | **‘My Profile’|LEB.***Depth[NS]* |

__**Surface**__

Example: **‘NS’**.*Level[-3.5]*

| **Property** | **Description** | **Notes** |
|---|---|---|
| Level | The level of the surface along the section at the offset specified. | **‘NS’**.*Level[-3.5]*<br><br>The value -3.5 represents the offset. A negative offset means left of the alignment. |

| | | |
|---|---|---|
| OffsetFromProjection | This feature enables users to project from a specified offset and level at a given slope to determine the intersection with the surface. If an intersection is detected with the surface, the corresponding offset value is returned as the result. | Argument 1: Start Offset<br>Argument 2: Start Level<br>Argument 3: Slope<br><br>'**NS'.** *OffsetFromProjection[-4\|450\|50*<br><br>This example will find a surface intersect from offset -4 and elevation 450 at a slope of 50%. The offset will be returned. |
| LevelFromProjection | This feature enables users to project from a specified offset and level at a given slope to determine the intersection with the surface. If an intersection is detected with the surface, the corresponding level value is returned as the result. | Argument 1: Start Offset<br>Argument 2: Start Level<br>Argument 3: Slope<br><br>**'NS'.** *LevelFromProjection[-4\|450\|50*<br><br>This example will find a surface intersect from offset -4 and elevation 450 at a slope of 50%. The level will be returned. |

**Utilities**

Example: **Utils**.*SlopeBetweenPoints[-3.5|450|-2|425*

| **Property** | **Description** | **Notes** |
|---|---|---|
| SlopeBetweenPoints | Calculate the slope between two points. X = Offset and Y = Chainage. The slope is returned as a percentage. | **Utils**.*SlopeBetweenPoints[-3.5\|450\|-2\|425*<br><br>Argument 1: Offset 1<br>Argument 2: Level 1<br>Argument 3: Offset 2<br>Argument 4: Level 2 |
| OffsetFromProjection | This functionality allows users to identify the point of intersection between two line segments. Each segment is characterised by its offset, level, and slope. | **Utils.** *OffsetFromProjection[-4\|450\|50\|-2\|440\|50*<br><br>Argument 1: Start Offset 1<br>Argument 2: Start Level 1<br>Argument 3: Slope 1 |

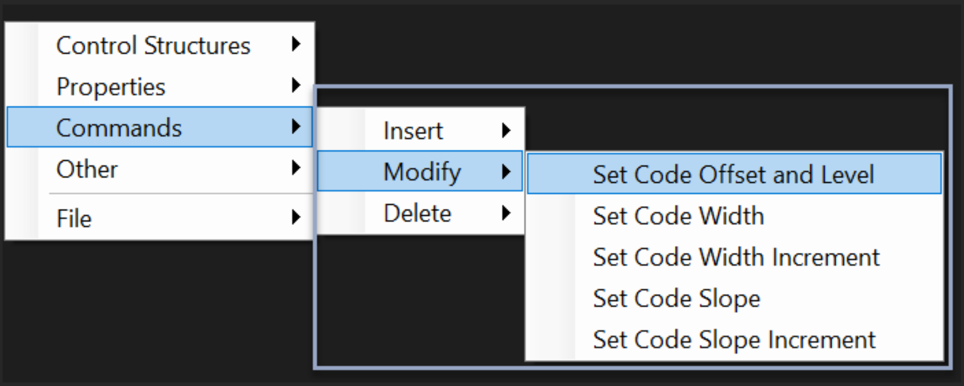
| | | |
|---|---|---|
| | The resulting intersection point is provided in the form of an offset. | Argument 4: Start Offset 2<br>Argument 5: Start Level 2<br>Argument 6: Slope 2 |
| LevelFromProjection | This functionality allows users to identify the point of intersection between two line segments. Each segment is characterised by its offset, level, and slope. The resulting intersection point is provided in the form of a level. | **Utils.** Level*FromProjection[-4\|450\|50\|-2\|440\|50*<br><br>Argument 1: Start Offset 1<br>Argument 2: Start Level 1<br>Argument 3: Slope 1<br><br>Argument 4: Start Offset 2<br>Argument 5: Start Level 2<br>Argument 6: Slope 2 |

## Commands

ScriptX commands, or functions, serve as the building blocks to shape and modify cross-sectional characteristics within the system. These commands provide you with a powerful toolkit to perform a wide array of actions, ranging from inserting new codes, altering existing ones, to deleting unneeded codes from the cross section.

Aside from these manipulation functions, ScriptX also includes other essential commands to aid in script management and flow control. For instance, the **"exit"** command offers a clean, immediate termination of a script, allowing for greater control over the execution process. By leveraging these commands effectively, users can create intricate scripts that can accurately cater to diverse cross-sectional design needs.

The easiest way to add commands is by using the right-click menu.



Example: **InsertAfter** *LWALL|LFPI|.001|2*

**Insert**

| **Command** | **Description** | **Arguments** |
|---|---|---|
| InsertAfter | Inserts a new code after the specified code. | Argument 1: New code<br>Argument 2: After Code (to position the new code)<br>Argument 3: New width<br>Argument 4: New slope |
| InsertCode | Inserts a new code before the specified code. | Argument 1: New code<br>Argument 2: Before Code (to position the new code)<br>Argument 3: New width<br>Argument 4: New slope |
| InsertCodeWithVertical | Inserts a new code before the specified code. The code is inserted with a vertical distance (height value) as opposed to a slope value. | Argument 1: New code<br>Argument 2: Before Code (to position the new code)<br>Argument 3: New width<br>Argument 4: New vertical distance (height) |
| InsertCodeEnd | Inserts a new code at the end of the template (before the batter). | Argument 1: New code<br>Argument 2: New width<br>Argument 3: New slope |
| InsertCodeEndWithVertical | Inserts a new code at the end of the template. The code is inserted with a vertical distance (height value) as opposed to a slope value. | Argument 1: New code<br>Argument 2: New width<br>Argument 3: New vertical distance (height) |

## Delete

| Command | Description | Arguments |
|---|---|---|
| DeleteCode | Deletes the specified code from the cross section | Argument 1: Code to delete |
| DeleteOutside | Deletes all codes outside (after) the specified code from the cross section | Argument 1: Reference code |

## Modify

| Command | Description | Arguments |
|---|---|---|
| SetCodeOffsetLevel | Sets a code's offset and level to specified values | Argument 1: Code to modify<br>Argument 2: New offset<br>Argument 3: New level |
| SetCodeWidth | Sets a code's width | Argument 1: Code to modify<br>Argument 2: New width |
| SetCodeWidthIncrement | Increments the width of the specified code | Argument 1: Code to modify<br>Argument 2: Width increment |
| SetCodeSlope | Sets a code's slope | Argument 1: Code to modify<br>Argument 2: New slope |
| SetCodeSlopeIncrement | Increments the slope of the specified code | Argument 1: Code to modify<br>Argument 2: Slope increment |

**Other**

| **Command** | **Description** | **Arguments** |
|---|---|---|
| SetPlotFlags | Used when inserting new codes. Sets whether the plot flag of new codes is set to 'Y' or 'N' | Example:<br><br>SetPlotFlags Y |
| SetLog | Initialises a log file to be created. This is useful when working on creation of script. Comment out this line when the script is used in production to avoid performance issues | Example:<br><br>SetLog D:\Logger.log |
| Print | Prints a message to the log | Example:<br><br>Print Hello |
| ShowLog | Opens the log file in your default notepad | |
| Exit | Force exit of the current script | |
| SetABS | Sets a variable's value to be the absolute value | Example:<br><br>X == -3<br><br>SetABS X<br><br>(X will be '3') |

## Known Variables

The software retains a set of predefined variables, allowing them to be conveniently referenced within the script. Users don't have to undertake the task of initialising these variables themselves, as the system automatically takes care of this aspect.

| Variable | Description |
|---|---|
| StartChainage | The start chainage of the script |
| EndChainage | The end chainage of the script |
| Chainage | The current chainage of the section being processed |
| vcLevel | The VC level of the current string being processed at the current section |
| vcGrade | The VC grade of the current string being processed at the current section |

## Examples

### Override Code Slope

```
// Override slope behind kerb
newSlope == 7
SetCodeSlope LFPI|newSlope
```

### Widen Code to Alignment with Shift

```
// Widen LEB code to alignment and shift offset
newOffset == 'Widen'.Offset
// Set shifted offset value
shiftedOffset == 1
// Calculate new offset
newOffset == newOffset + shiftedOffset
// Calculate new level
newLevel == vcLevel - (newOffset * (LEB.Slope / 100))
// Apply new offset and level to LEB
SetCodeOffsetLevel LEB|newOffset|newLevel
```

## Set Footpath to Surface with Slope

```
// Set footpath to match surface at slope from LBK

slope == 50

newOffset == 'NS'.OffsetFromProjection[LBK.Offset|LBK.Level|slope
newLevel == 'NS'.LevelFromProjection[LBK.Offset|LBK.Level|slope

SetCodeOffsetLevel LFPI|newOffset|newLevel
```

## Benching

```
// Add some benching behind the kerb

// Set dimensions for benching

width == 1
height == .5

// Turn off plot flags

setplotflag N

// Delete all codes after LBK

deleteoutside LBK

// Create benching and stop when at surface level

for i = 0 to 10 step 1

if i = 0

// Set initial heights because wall does not exist

wallLevel == LBK.level
wallOffset == LBK.offset
newWallOffset == LBK.offset - width

else

wallLevel == LWALL.level
wallOffset == LWALL.offset
newWallOffset == LWALL.offset - width

end

surfaceLevel == 'NS'.level[newWallOffset]
distanceToSurface == surfaceLevel - wallLevel

// Create wall if in cut

if wallLevel < surfaceLevel

if distanceToSurface < height

newHeight == distanceToSurface

insertcodeendwithvertical LBASE|width|0
```

```
setplotflag Y
insertcodeendwithvertical LWALL|.001|newHeight
setplotflag N

exit

else

insertcodeendwithvertical LBASE|width|0

setplotflag Y
insertcodeendwithvertical LWALL|.001|height
setplotflag N

end

else

exit

end

loop
```

## Information From the Developer

- ScriptX enables the computation of expressions while working with variables. However, please note that expression evaluation is not supported within **"If"** statements or "**For"** loops.
- While ScriptX supports nested **"If"** statements, it's advised to refrain from using overly complicated nested structures to ensure readability and smooth execution. It's important to note that any block of code situated between **"If"** statements without an accompanying **"Else"** or **"End"** will not be executed.
- Always check your scripts for errors before running them. Unexpected behaviours or program crashes can occur from syntax mistakes, undeclared variables, mismatched 'If'-'End' statements, or poorly structured loops.
- Utilise comments generously in your scripts. Not only will they help others understand your code better, but they will also serve as reminders for you if you need to revisit the code after some time.
- Be mindful of the efficiency of your scripts. Nested loops and redundant operations can significantly slow down the execution of the scripts.
- Should you create a script that you believe could be beneficial to other Civil Site Design users, we strongly encourage you to reach out to us. We are always interested in collaborative efforts and value the sharing of resources within our user community.