Cscript

C : Compression, Computer, Base 3 (a,b,c)

Cscript is a constructed script, in short it is a writing system for alphabets..

Cscript is based on Base 3 (aka. trinary/ternary). Is is designed with the following goals

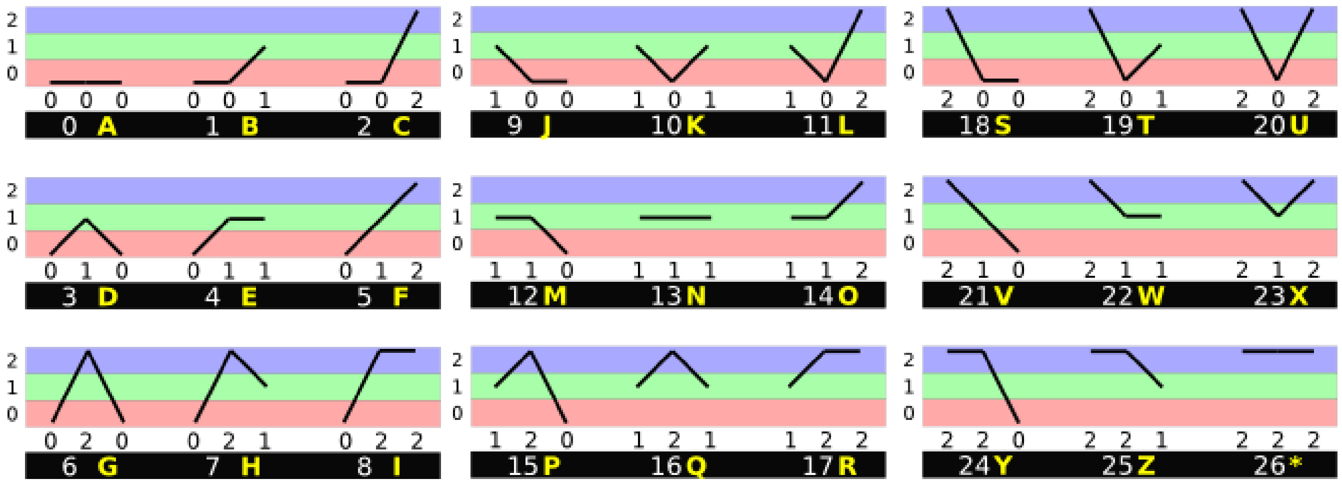
1. Easy to read/write programmatically as well as by hand
2. Ability to spatially compress information

There several “levels” to Cscript, they are...

1. **The Alphabet**
2. **Vertical Compression**
3. **Vertical Stacking**
4. **4 Point Compression**
5. **5 Point Compression**
6. **Gap Compression**
7. **Vertical Sequencing**
8. **Line Segmentation**
9. **Custom Shortcut Glyphs**

*compression refers to the ability to conserve pen-strokes/pen-lifts or horizontal spacing

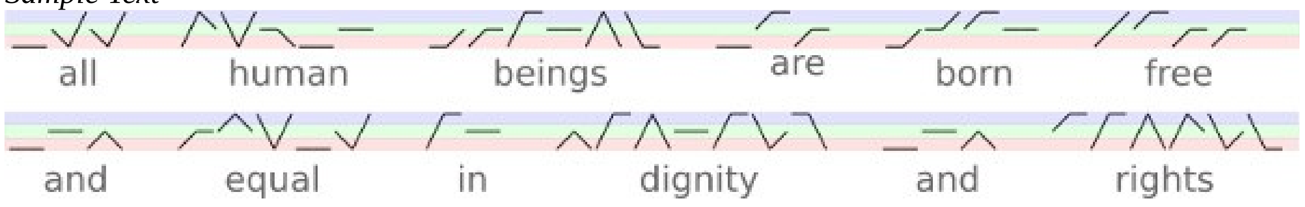
Level 1 - The Alphabet



The writing line is divided into Bottom / Middle / Top, each section represents a ternary value.

Always read **LEFT → RIGHT**

Sample Text

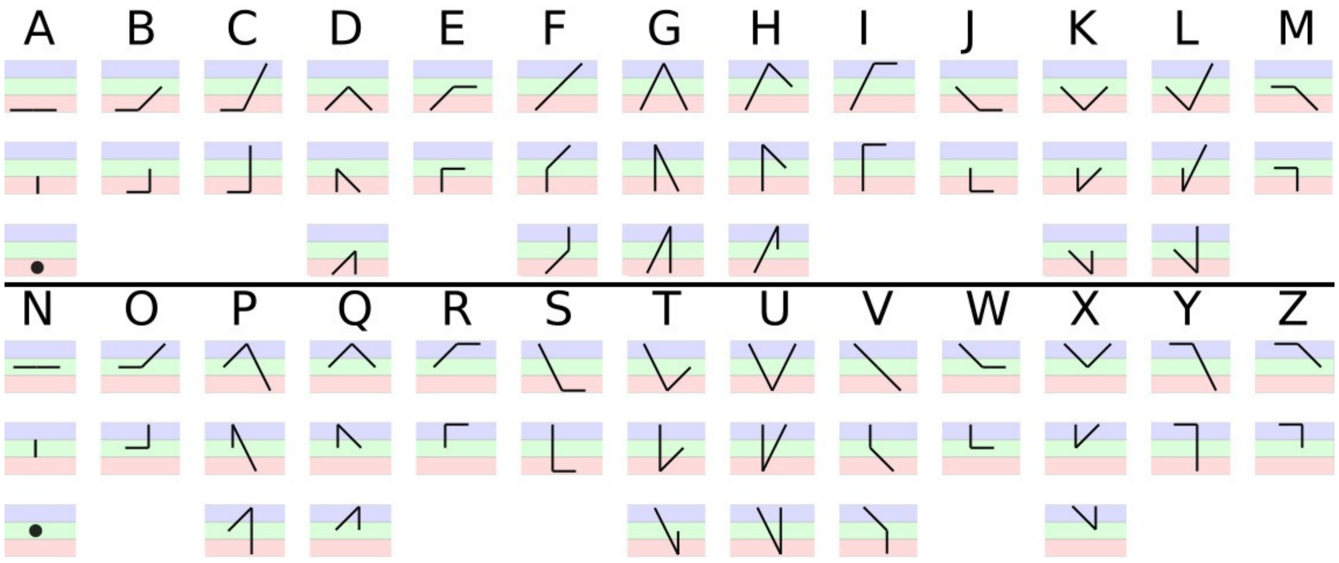


***Level 2** – Vertical Compression*

Here we save horizontal space by allowing allowing vertical lines

This works on the line connecting 2 values (half of a letter) if...

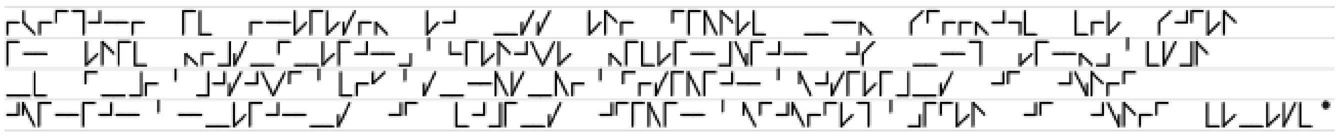
1. The values are not the same (0-1 and 0-2 can be vertical, 0-0 cannot)
2. It is not immediately after a vertical (2 verticals looks the same as 1 vertical)



The A and N do not have any corners, this makes their middle value "invisible". For this reason the A and N are treated as single lines without corner, and also allowed to also exists as dots.

In the following example we do not use the alternate forms of A or N.

The comma (,) and the period (.) use the extra character space (top section) and we use the same alternate forms that we use for the A and N. A vertical dash at the top means comma and a dot at the top means period

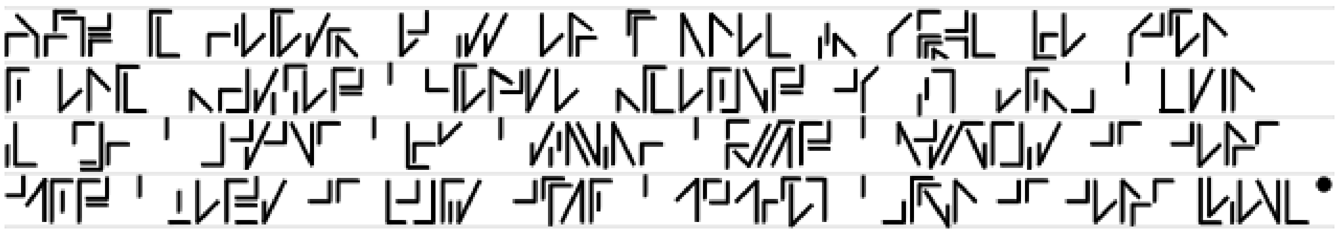


Text reads :

Everyone is entitled to all the rights and freedoms set forth in this Declaration, without distinction of any kind, such as race, colour, sex, language, religion, political or other opinion, national or social origin, property, birth or other status. (There's a typo, bet ya can't find it in 5 minutes :P)

Level 3 *– Vertical Stacking*

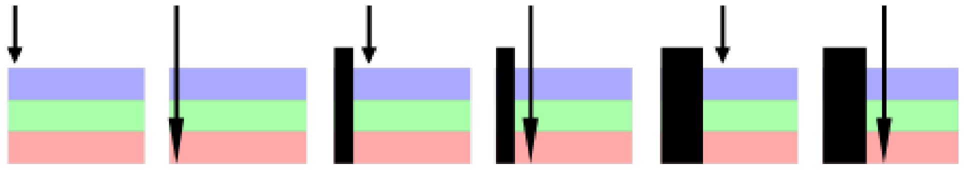
Vertical stacking is simple!
Reading is **top** → **bottom**.

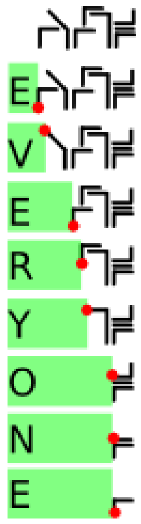


Text reads :

Everyone is entitled to all the rights and freedoms set forth in this Declaration, without distinction of any kind, such as race, colour, sex, language, religion, political or other opinion, national or social origin, property, birth or other status.

Reading Rules





A simple easily programmable reading rule is used.
Vertically descending scan lines, moving left → right search for a Line terminal or dot.

Line Terminal :
A point that has 1 connection is a dot

Every line has 2 terminals, the top-left most terminal is the "start terminal"

**1=Dot,   2=Line,   3=Line with corner*

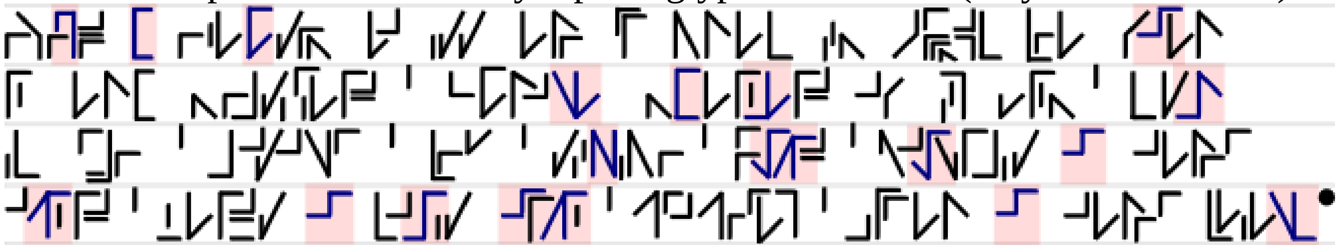
It is best to always ensure each terminal or corner lies in the correct value zone (0,1,2), but there are some cases where this rule can be bent.

eg. if an E or D are under an E they can fall entirely in the 0 zone but still obviously be an E, because if all values were 0 it would be an A, and thus be a line or dot. (refer to the word FREEDOMS in the first line of the example above)

Level 4 – *4 Point Compression*

Each shape so far has been composed of 1 point(dot) 2 points(single line) or three points(line with corner)

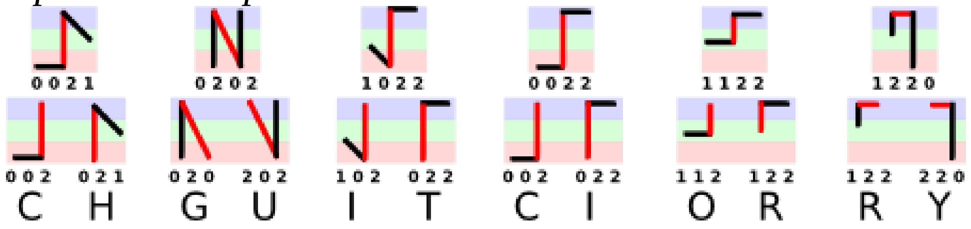
4 Point Compression reads every 4 point glyph as 2 letters. (they share 2 values)



Same text (“ *Everyone is entitled..*”) using 4 point compression. 4 point compression units are highlighted.

There are 4 main types of 4 point glyphs, they vary in how they are decompressed.

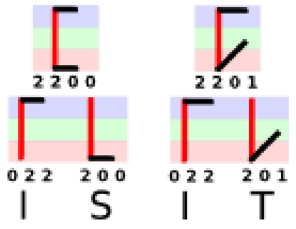
Type 1 – *Simple 4-Point Split*



Simple Splits are all 4 point glyphs that have 2 corners. (many more, these are just a few) If the middle line is vertical then one of the other lines extends to the left.

Simple 4-Point Splits are done by using the first 3 values for the first letter and the last 3 values for the second letter.

Type 2 – Reverse Read *4-Point Split*



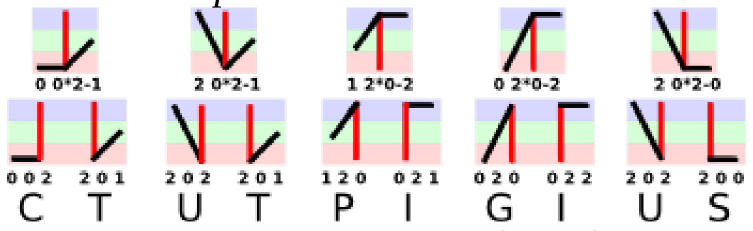
When the middle line is vertical, and both other lines extend to the right the first three values are read in reverse.

This is because both start terminals are “buried” in the vertical line.

Both terminals showing the in 4 point glyph are actually “end terminals” so the software will find the upper most “end terminal” and read the first glyph backwards (*in these examples it finds the top right and then reads to the left*).

***If both are on the left side, reverse the second letter instead**

Type 3 – Intersection *4-Point Split*



Intersection 4-Point Splits are the most complex for software reading.

These have 3 terminals and 1 intersection.

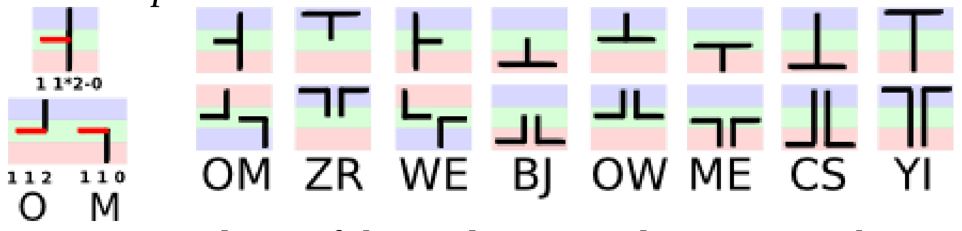
Using normal reading rules, the 3 terminals are organized in sequence.
The first one found by **top→bottom**, **left→right** reading is “terminal 1”, the second one found is “terminal 2” and the third one found is “terminal 3”

The line from “terminal 2” to the intersection is duplicated and the glyph split.

“terminal 1” → “intersection” → “terminal 2” is glyph 1
“terminal 2” → “intersection” → “terminal 3” is glyph 2

The glyphs are then read using proper reading rules, so if the rule above causes the software to read **right** → **left** then it reverses the values.

Type 4 – *4-Point T-Split*



If there is an intersect and one of the angles is 180 degrees, use the T-split rule. There are only 8 possible 4-Point T-splits.

As you can see, some are not useful, this is a good time to bring up re-indexing.

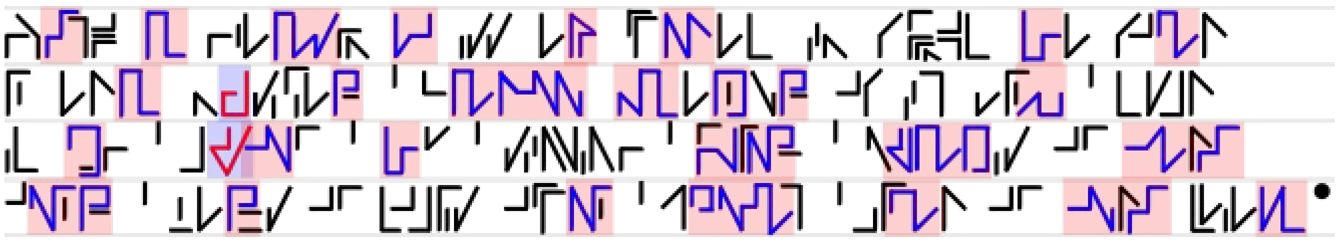
I have simply assigned values based on normal alphabetical sequence. Some letter combinations are efficient.. others not as much. The alphabet itself can probably be optimized based on letter combination frequency.

Being computer friendly means it is possible to analyze this and based on some parameters find an optimized alphabet index through software analysis or even brute force (as opposed to manual trial & error)

Level 5 – 5 Point Compression

5 Point Compression reads every 5 point glyph as 2 letters. (They share one value)

5 Point compression can be repeated as much as desired. Each addition adds 2 corners to the line.

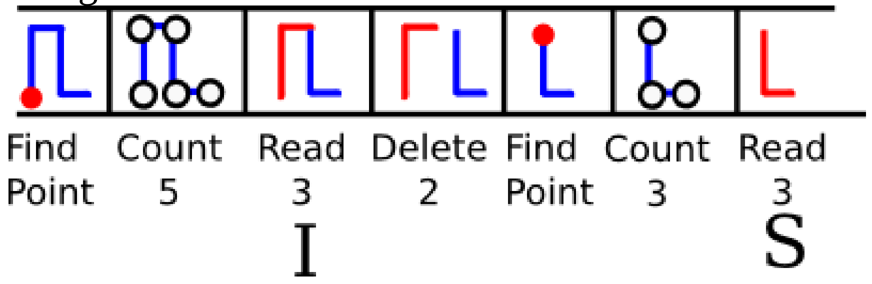


Highlighted are 5 point glyphs

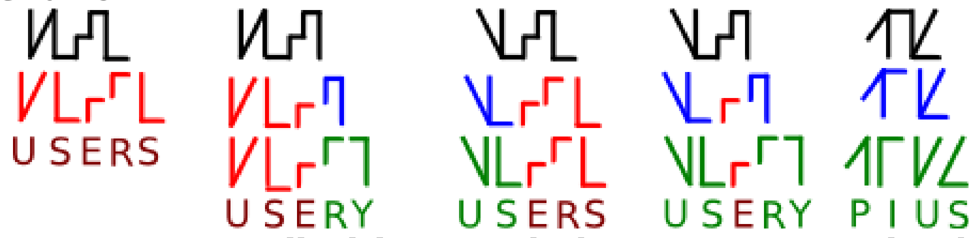
Blue lines are "forward read starts", and red lines are "reverse read starts"

"Reverse read start" simply means the the first letter is connected so that the reading starts **right** → **left** (opposite the reading rules)

Here are the reading rules



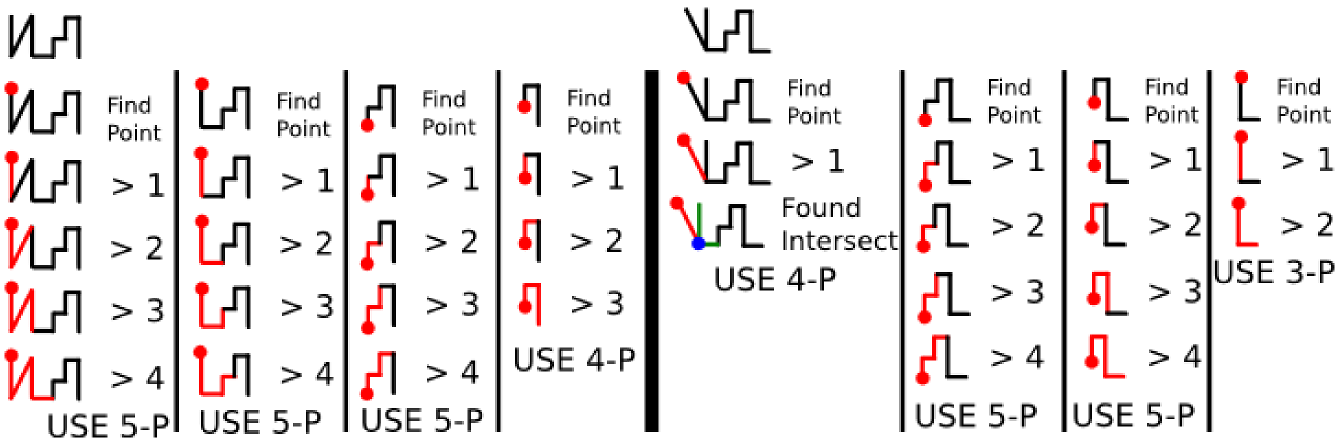
4/5-Point Chains



5-Point Compression "virtually deletes" each character as it is reads. This means that if there are 4 points left at the end it is obviously a 4 point compression.

Regular 4-Point compression glyphs can ONLY be added to a 5-Point Chain at the end of the chain.

Intersection 4-Point Splits can chain just like the 5-Point. They can also be freely chained with 5-Point chains (start, inside, and end)



The reading rules are actually quite simple for all this

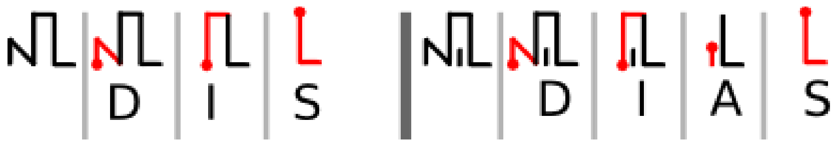
find point		
If length is 1	-	treat as point (read 1, delete 1)
if length is 2	-	treat as dash (read 2, delete 2)
if #2 is an intersect	-	treat as 4-P intersection (read 4, delete 3)
if remaining length is 1	-	delete 1
if length is 3	-	treat as normal glyph (read 3, delete 3)
if length is 4	-	treat as 4-P (read 4, delete 4)
if length is > 4	-	treat as 5-P (read 3, delete 2)
repeat		

The "**if remaining length is 1 - delete 1**" is used because the 4-Point intersection needs a "clean up rule" for when it comes last.

The 5-Point chain has a built in clean up rule, the last character is treated as a 3 point glyph and deletes 3 instead of 2.

Vertical stacking can even be used above and below these chains.

After reading and deleting each segment from a long string, the **find point** function is called again, so if there are any glyphs above or below the string they will be found when enough of the string has been deleted to cause them to be found first.



***Level 6** – Gap Compression*

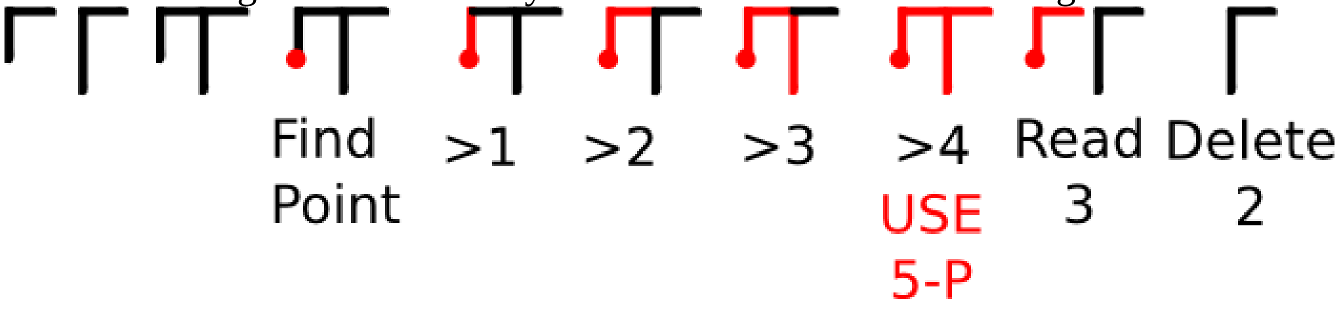
Gap Compression only removes the gaps between 3 point and/or 5 point glyphs (even daisy chained), there are 2 types

1. Intersect Gap Compression
2. Corner Gap Compressions

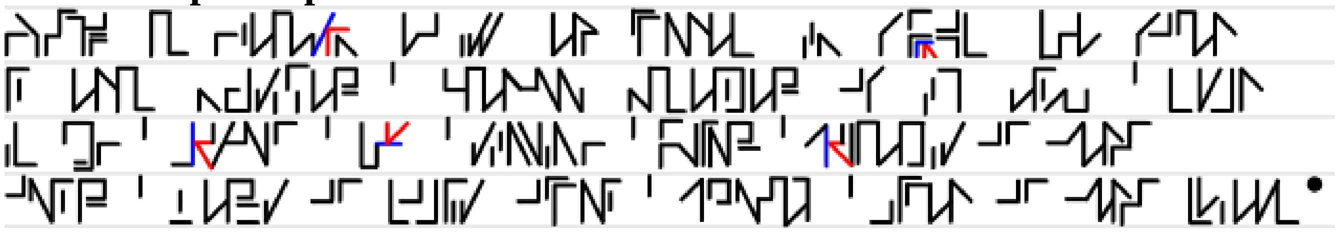
Intersect Gap Compression

For Intersect Gap Compression we attach a terminal to a corner.

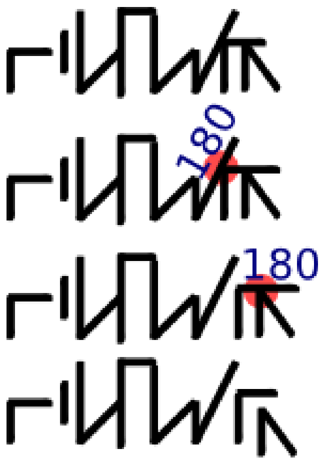
Normal reading rules will already cause these breaks when reading.



Corner Gap Compression



**Corner Gap Compression is still being tested, it may be deprecated later.*



In Corner Gap Compression:

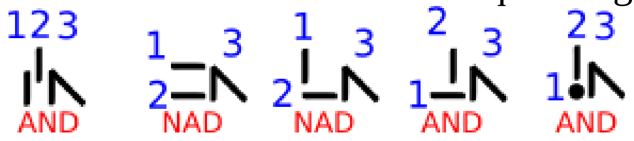
1. 2 lines strike the same wall at the same point on the same side (instead of 1)
2. The wall can be horizontal or vertical

In Other Words : "***All 4 way intersections where one angle is 180 degrees, are broken apart and treated as a corner and a straight line.***"

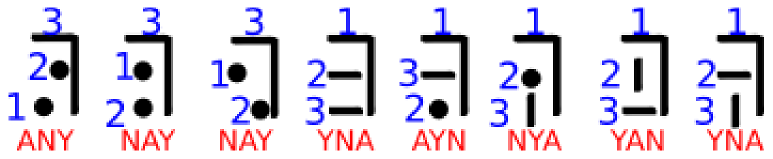
***Level 7** – Vertical Sequencing*

You may have noticed the A and N both have alternate forms that have not yet been used.

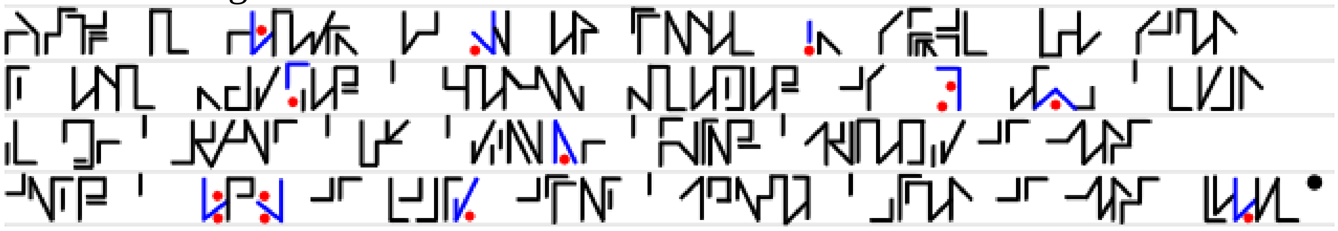
The Dot form for A and N are used for vertical re-sequencing.



A Dot Form will override all letters that exist in its vertical space, and place itself before all of them.



If 2 dots exists in the same vertical column, they both jump ahead of the letters and normal reading rules determine their order.



In the example above the red dots are resequenced to be before the blue letter.

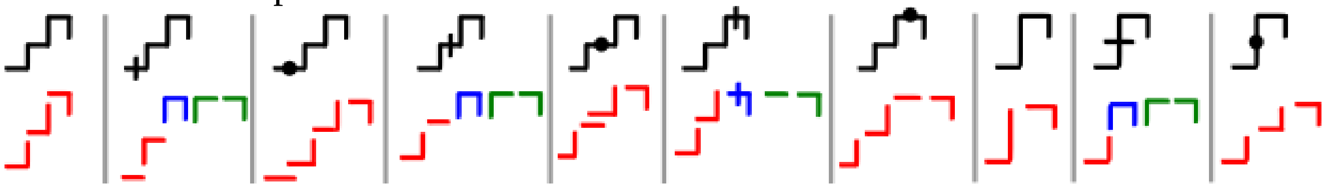
***With Chains :** sequencing dots only override the individual letters which share its space**, not the entire chain.**

***Level 8** – Line Segmentation*

Line Segmentation allows repeating values in 4-Point & 5-Point Compression.

There are 2 kinds of line segmentation:

1. Single Repeater –A dash through the line creating 4 perfect 90 degree angles
2. Double Repeater -A dot drawn in the line



Line broken with 5 point(black → red/blue) and 4 point(blue → green)

Line segmentation also creates more possible forms of 4-Point Compression.

Level 9 *– Custom Shortcut Glyphs*

Customs Glyphs are just that, a customized index of shortcuts.

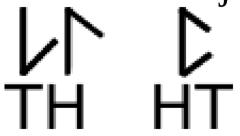
There are 2 types of Shortcut Glyph

1. Resequenced Shortcuts
2. Invalid Symbol Shortcut

Resequenced Shortcut
Any letter glyph or multi-letter compound glyph has a set of letters and a sequence.

Sometimes the rules make useless combination efficient and useful one inefficient.

This is best solved by trying to optimize the alphabet, but that is a great task.
You can also just re-sequence the letters and keep an "index"(physical or mental).



T and H, for example, make a nice 4-point reverse read glyph
One could just define that 4 point glyph as TH instead of HT

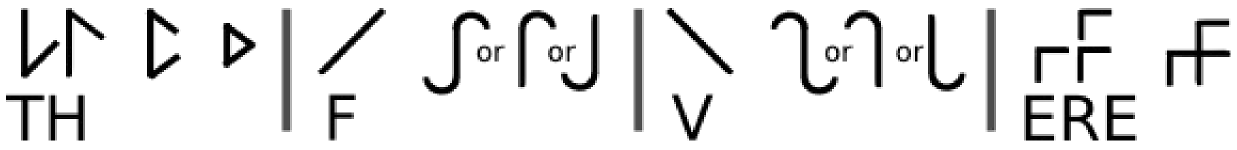
Invalid Symbol Shortcut
These reading rules are very basic and designed disallow ambiguous symbols.

There are in fact tons of possible lines and shapes it does not account for. (loops, circles, triangles, etc..)

These can simply be indexed and defined as certain glyphs.

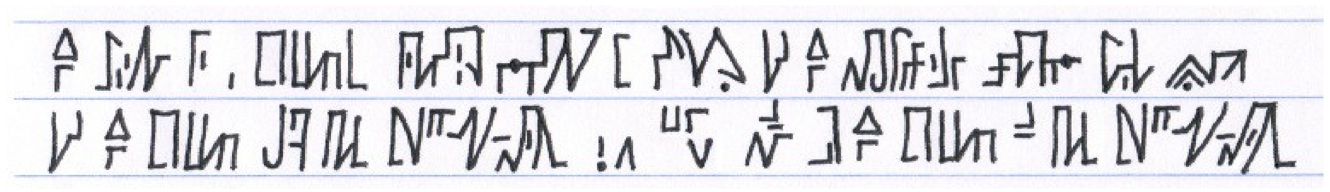
There are even many possible "reading rules" which could be added to allow more shortcuts and chain types.

Here are 2 shortcuts I have adopted so far. (I may want to use curves for something else later and be forced to find another method of doing V & F)



Not Enough?

It never is ;)



The change in a systems internal energy is equal to the difference between heat added to the system from its surroundings and work done by the system on its surroundings

Cscript is the computer friendly sister-script to **Dscript**.

Dscript (Dimensional Script) is a 2D writing system that allows words and phrases to exist as strings that are very flexible and dynamic in 2D.

Dscript intro PDF : http://dscript.org/dscript.pdf

Dscript has been developed much more and even has a notation layer.

Dscript Notation : http://dscript.org/note.pdf
Chemical Calligraphy : http://dscript.org/chem.pdf

Some more fun...

***WireScript**, a 2D/3D writing system that can be written by bending wires. Works great for art, sculptures and jewelry.*
http://dscript.org/wirescript.pdf

***NailScript**, A layered writing system for writing with hammer and nails.*
http://dscript.org/nailscript.pdf

***"Mad Science"/"Technology Art"** inventions and experiments. Great DIY fun.*
http://dscript.org/inventions.pdf