TEXDraw

LaTeX Graphic Mathematical Expressions Input for Unity

Documentation for V3.3

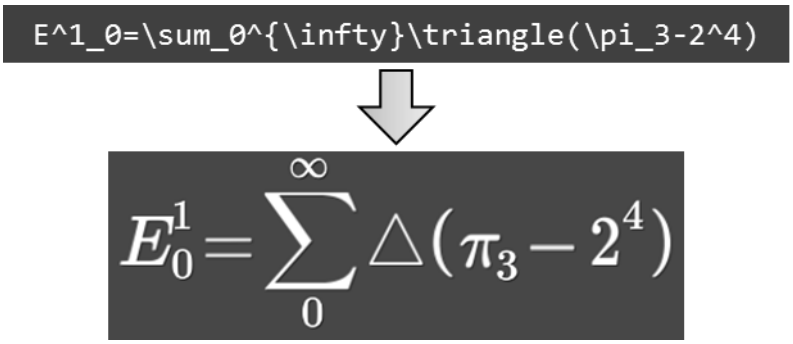
PS : If you like this asset, don't forget to leave a review at the asset store page ;)

Table of Contents

FAQ About TEXDraw

What is TEXDraw?

TEXDraw is a Component that makes a plain text can be converted into graphical representation of mathematical formulas. TEXDraw draws mathematical formulas using the similar approach introduced in LaTeX writing system. The resulted output is a 3D mesh that can be used inside Unity's UI or other mesh-based rendering system.



How it does work and why it different?

TEXDraw, just like many other text generator, is designed for rendering some kind of text (or *string*, exactly) to show in your display screen. In TEXDraw, however, adds some functionality to render any kind of mathematical expressions that is used in various apps like educational software, scientific simulations, and many more. With the power of Unity's built-in UI System and dynamic text support, generating math expressions was never so easier and seamlessly than ever!

What's the Key Difference between TEXDraw vs. Standard UI Text?

A lot, including:

- LaTeX-Based, which is has more clean typos and flexible features than standard html markup
- Complete Math Expressions, including fractions, root, matrix, scripts, straight lines, tables, etc.
- Resizable delimiters, which is fundamental for creating a complex math expressions.
- Dedicated Layout system (No need external Horizontal/Vertical Layout).
- +600 symbols included, or built-your-own.
- +40 different instructions, or known as *commands*.
- Import and use Sprites as you were importing fonts (aka. Graphic inline).
- Can output into Unity's UI rendering, MeshFilter, or even NGUI.
- Built-in integration with TextMeshPro™ Rendering
- Have Standard Layout features, like Word wrap, justify alignment, and best fit mode.
- Manage symbols, kernings, everything, in dedicated editor preference.
- No bundled DLLs, customize everything to suits your need.
- Compatible to All platform, work fast on mobiles
- And much more...

How easily I can make it work on my project?

Soon after you download and importing it into your project, You can give it a try by creating a TEXDraw Object in `GameObject → UI → TEXDraw`, and start to typing on it. Also, you may want to check Sample scenes in `Assets/TEXDraw/Sample/Scenes`. After you satisfied with the result (I hope so), you can erase our sample fonts, and fill-it with your own (read more here).

Will this work on all platform? Any performance backdown?

This package have been tested on runtime builds, even mobiles. We always considering to performance when we write new features. This package blows up your game performance? Reach us.

Do this package provide support for other External Asset?

Yes, TEXDraw support drawing into NGUI environment, or using the benefit of SDF Rendering from TextMeshPro. Click here for instruction setup.

Can TEXDraw replacing Text Input in uGUI?

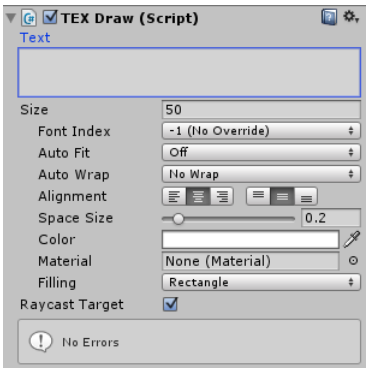
No, TEXDraw's goal is for providing read-only information. So far there's no way to provide support for such feature because the complexity of TEXDraw rendering system.

I have troubles. Any suggestion for me?

See troubleshoot page for common problems.

Inside of TEXDraw Package

The TEXDraw Component



This component is available under Unity's UI Canvas object. The components have been made simple, so you can keep focus on what's you'll type into.

Aside from **Text**, there are other optional properties that are quite useful for handling display output. Below is Description of each property inside this Component:

Text A plain text that you want input to.
See here for practical guide to write, and here for scripting instruction.

Size Size of generated graphics
The font texture size will automatically resized to be detail enough to display on screen

Font Index Index of used default font (-1 to follow default typeface rules)
In scripting, you can set this as integer value. Each number represents an index of font that registered in the Font Stack.

Auto Fit How final graphic is scaled when it's render is out of the rectangle bound
See here for available options

Auto Wrap Horizontal wrapping mode if a line is beyond rectangle's horizontal bound
See here for available options

Alignment The horizontal and vertical alignment of the text.
In scripting, this is a Vector2 property. A value of {0, 0} represent left-bottom alignment

Space Size The space size on each lines
The actual space size is proportional to what given in Size property.

Color The main color for generated graphics
Use \color if you want to write specific color

Material Assign a Custom Material for This component
If none assigned, the default material (from Resource) will be used for rendering

Filling Optional options for filling UV1 data
Used in conjunction with custom Material effects, choose the way when font characters are overlay-ed with some graphics

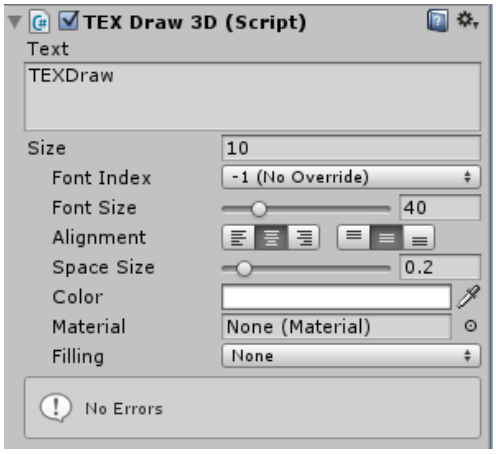
---

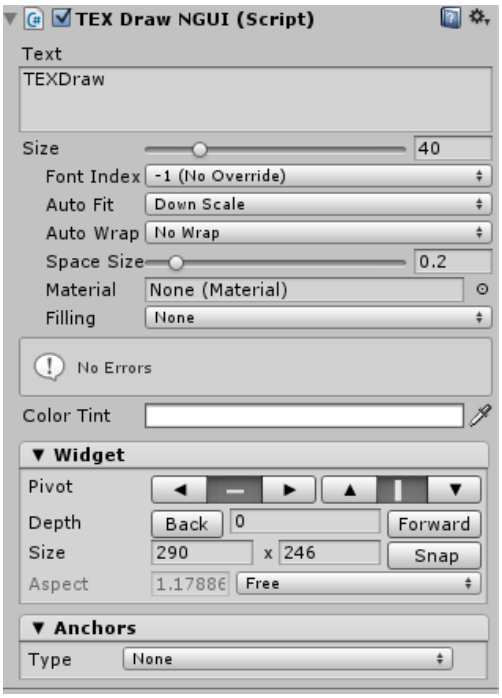
Other TEXDraw Variant Components

For any non-UI user, or those who don't want UI Canvas dependency, may use TEXDraw 3D. It's a great alternative to using this component rather than standard one. You can add TEXDraw 3D to your scene by navigate to `GameObject > 3D Object > TEXDraw 3D` or attach this to your Game Object located in `TEXDraw > TEXDraw 3D`. You also can attach RectTransform (yes, it is still work outside Canvas) if you prefer.

Alternatively, for those who already use NGUI can use the NGUI variant. This kind of variant is doesn't available without importing the NGUI extension for TEXDraw first. This extension is packed as a .unitypackage file that can be found inside TEXDraw root folder. To import it, simply open the package. To add this component to your scene, hit the NGUI menu located in `NGUI > Create > TEXDraw`

Those three variants have the similar functionality and properties. What you type inside in either text will yield the same result.





Either these three component, they can be Integrated seamlessly into TextMeshPro's SDF Rendering. If you have TextMeshPro in the project, SDF Rendering in TEXDraw can be activated by Declaring scripting symbol `TEXDRAW_TMP` in Player settings and importing required shaders, which explained in detail here.

Enumeration Choices

Auto Fit is used for what happen for whole text when generated text is out of the given rectange layout. The choices are...

Off	Turn off rescaling. Text can generated beyond it's rectangle
Down Size	Scale text down if it oversized
Rect Size	Force the rectangle to follow the generated text size
Height Only	Adjust the height of the rectangle automatically
Scale	Scale the generated text until fit on the rectangle
Best Fit	Attempt to Find the largest possible size (caution: potentially expensive)

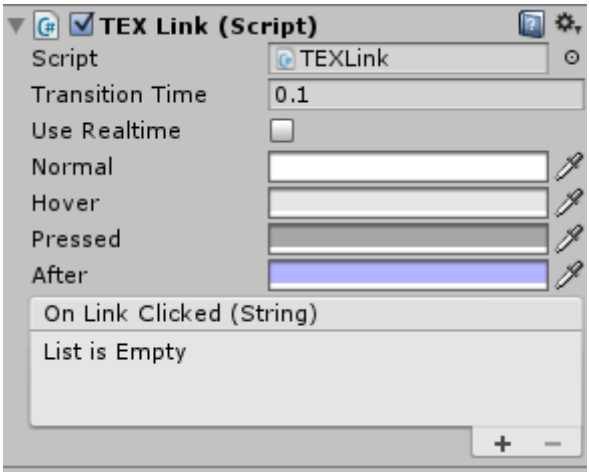
Auto Wrap is used for what happen to each line of generated text when it's horizontal line is beyond than given rectangle width. Auto wrap is always be calculated first before Auto Fit.

No Wrap	No wrapping applied
Letter Wrap	Wrap (Move to below) any character if it oversized
Word Wrap	Wrap any word if it oversized
Word Wrap Justified	Wrap any word, then stretch space sizes to rectamgle edges

Auto Fill is useful only if you use a custom material which requires UV1 vectors like Gradient and Texture Overlay shaders. This is about how texts are UV-mapped, in automatic-way.

None	Don't attempt to fill any UV1 values (Faster)
Rectangle	Interpolate according to Rectangle Bound (Scaling will take effect)
Whole Text	Interpolate to the generated text rectangle (Scaling isn't taken into effect)
Whole Text Squared	Interpolate like Whole text, but keep at ratio size of 1:1 (prevent stretches)
Per Line	Interpolate text line-by-line
Per Character	Generated Character Quads will always either have (0,0) or (1,1) coordinate.
Per Character Squared	Like per-character, but keep it's aspect ratio at 1:1.
Local Continous	All characters UV is mapped based on their local position
World Continous	All characters is mapped based on their world position, interactively.

TEX Link Component



TEX Link is a feature that takes the handling for links that created inside a TEXDraw component. To create one, simply add the component in the same GameObject within TEXDraw. This component located in `TEXDraw > TEXLink UI`. For NGUI variant can use TEXLink NGUI instead. So far there's no TEXLink available for TEXDraw 3D.

To make this component works, you need to create at least one `\link{}` command to an expression. TEXLink responds to Mouse and Multi-Touch screen (Keyboard is yet supported).

Transition Time How long the time it taken to change a color.
Zero to turn off Animation

Use Realtime Should we ignore Time.timeScale?
Check this box on if you don't want link freezes when game pauses

Normal Color tint when the link is yet clicked.
Tint means the final color is multiplied between this color * color from the character

Hover Color tint when the mouse just above the link
This only happens on desktop where mouse devices are exist.

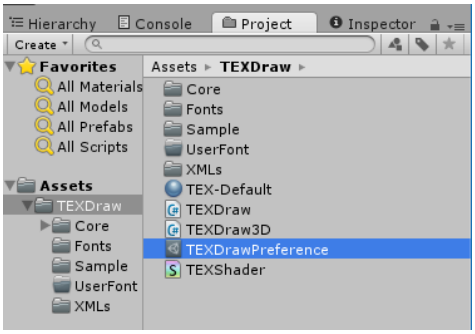
Pressed Color tint when mouse/touch presses the link
This one works both mouse/touch (even if the user do multi touches) presses down.

After Color tine when mouse/touch just been released
Will reset to normal when this script goes destroyed, or ResetLinks() is invoked.

Below of these properties there's an UnityEvent class named `OnLinkClicked`. This is where you put your script functions to receive an event when user get clicked the link. There's also a string parameter which will let know which link that user clicks on. For example if user clicks on `\link{\root[3]{3}}` then that string will yield as what goes inside the braces (ie, `\root[3]{3}`). Optionally, more functionality is described in it's command.

TEX Preference

Beside of powerful TEXDraw Component, it would be never works without databases, and it holds inside the powerful TEX Preference. You can open the Preference by navigate the menu bar placed in `Tools > TEXDraw > Show Preference`.

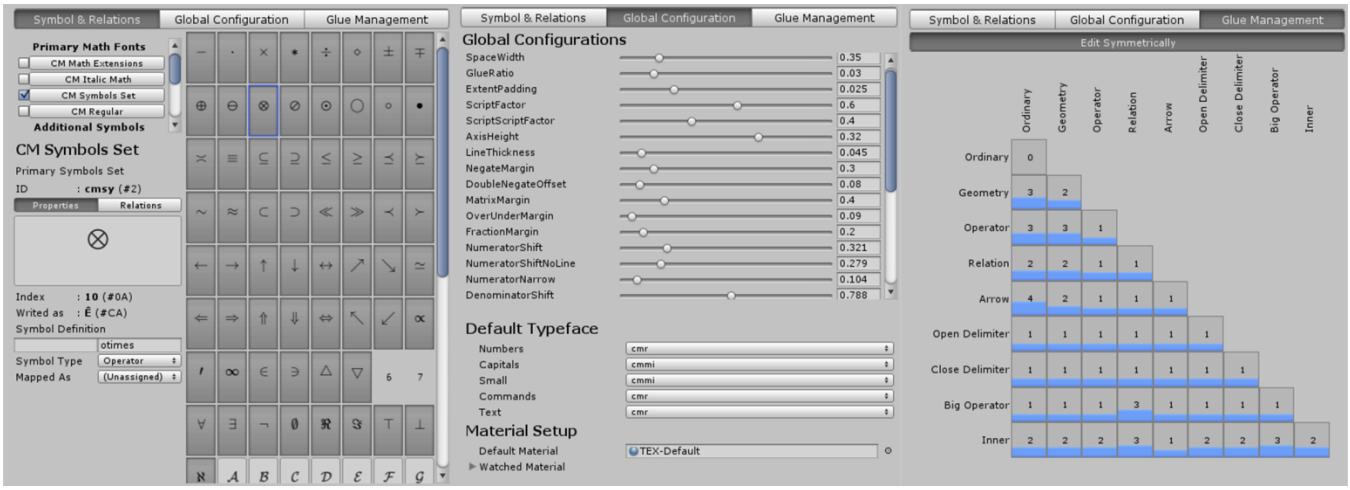


The preference here located inside of TEXDraw root folder as an arbitrary asset.

TEXDraw Preference holds shared information across one project and saved as arbitrary asset located in: `TEXDraw/Resources/TEXDrawPreference.asset`.

Please note that **only one** TEX Preference allowed in single project. If this file is missing, a new copy of asset will be automatically generated.

TEXDraw Preference has three main tabs. Each has separate purposes, take a look of these pics:



Symbol & Relations

A place for finding, defining, managing used fonts and symbol definitions. It also can preview characters in single font as a character map.

Global Configuration

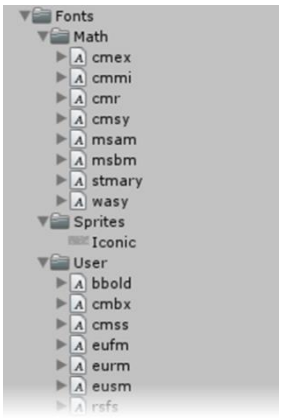
A place for shared (static-like) properties for controlling character sizes, margin, fraction gaps, script drops, etc.

Glue Management

A place for controlling custom kerning for each different type of character.

A guide for usability on each tab will be explained in detail in another section of this documentation here.

Font Collections



Inside of the `TEXDraw` package, 15 fonts are packed (+1 with sprite texture) and included inside of a folder located in `TEXDraw/Fonts`. These 15 fonts (except opens) are collections from JSMath and it's allowed for commercial use. You can put any fonts/texture in this folder so they can be used in TEXDraw.

In that folder, there's 3 subfolder in inside of that folder, and each of them have their own purposes:

Math Contain built-in font package for constructing math expression
You shouldn't put anything here. It is safe to delete this fonts as long as you doesn't use it.

User User font-defined packages
Put any fonts here and it will registered and used for TEXDraw components

Sprites Texture as user font-defined packages
Yes! Textures can be imported as a tiled sprite and used just like regular fonts

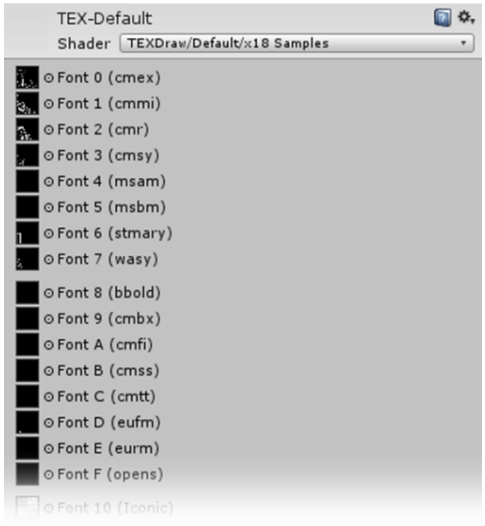
---

You can place any font/texture to User/Sprites folder, then hit menu item on `Tools > TEXDraw > Rebuild Font Data` so it can be registered and used in TEXDraw component.

You can put as much as you like, but keep note that maximum font (Math+User+Sprites) that TEXDraw can handle is 31, and anything beyond that will simply not included in TEXDraw components. You also have to take care about font's file name to not having any other than letters so they can be called from commands by their name.

TEXDraw also support alternative styling like bold/italic, so it isn't necessary to put it on stacks. Unity will do it for you automatically. See here for more instruction.

Shader Variants



Shader, which is a main part of renderings, included in here is slightly different than other majority of how shader used in most cases.

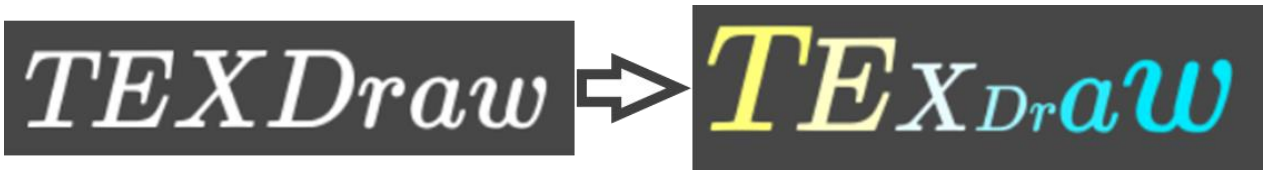
All TEXDraw shader is put on TEXDraw/. Here you can choose which type of shader that you want to use. However, each shader also have subcategories of *Number of Samples*. You can select 'Full' as a starting point, as in later time TEXDraw will choose it automatically.

TEXDraw have 5 built-in shader variants which can be used for additional effect like gradients, normal bumps, or texture overlay.

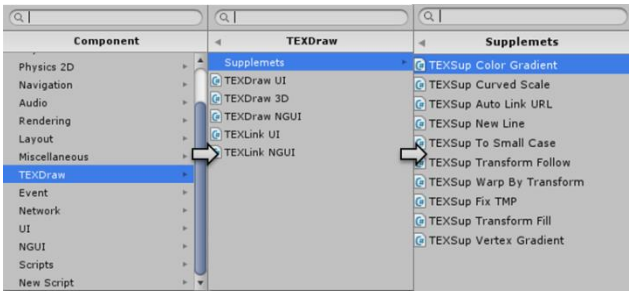
More samples mean more draw calls (batches/passes). Internally we are not combine all font textures, instead, we load them all in shader program. This comes to a problem where texture (sampler) count (31) exceeds more than maximum allowed sampler count (16). TEXDraw solve the problem by splitting samplers in different passes so each of pass only handle up to 8 samples. But what if I only used fewer than 31? You can choose fewer samples, which is handled automatically.

Supplements

Supplement is the way to modify (find & replace) text just before used in rendering process, automatically. The usage is similar to UI Effects, where you just need to attach a relevant Supplement component beside TEXDraw, and it will change how TEXDraw displayed, instantly.



Right now, we have 4 types of Supplements included in the package. You can add these components beside TEXDraw located in TEXDraw/Supplements.



The specific usage of these supplements are:

Auto Link URL	Detect and replace any relevant URL or email links
New Line	Treat \n command as a new line
Color Gradient	Attach \color command on every character and interpolate the color based on a Gradient
Curved Scale	Attach \size command on every character and interpolate the scale factor based on a Curve.
To Small Case	Turn lower case characters to upper case with given scale ratio.
Warp by Transform	Elevate (and rotate) on every character with given curve
Transform Follow	Used in conjunction with World Continous, make TEXDraw revalidate when transform changes
Transform Fill	Apply post-effect UV1 Transformation (Offset and Scale)
Vertex Gradient	Apply post-effect color tint based on character edges.
TMP Fix	Apply post-effect fixes when using TextMeshPro shaders. Read here.

Built-in effects like Shadow and Outline still works, through need more bit setup.

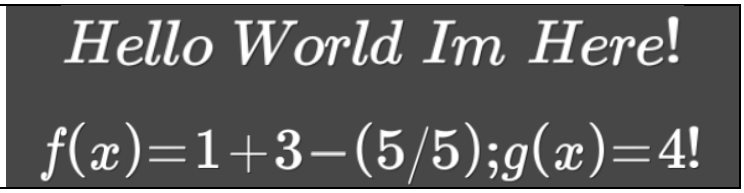
Guide to Write in TEXDraw

An Introduction ...

As a basic feature, you can write anything regularly just like standard text generator, it accepts letters, digits, popular symbols (that exist on physical keyboard), whitespaces, unicode characters, and also multi-lines.

```
Hello World Im Here!

f(x)=1+3-(5/5);g(x)=4!
```



Though they mostly work for all characters, please keep a note that Tab spaces do not work (use \hold as subtitute). Characters `{`, `}`, `\`, `^`, and `_` is can't be used directly, instead type a backslash `\` before it. For example, `\}` and `\^`.

The Power of Backslashes ...

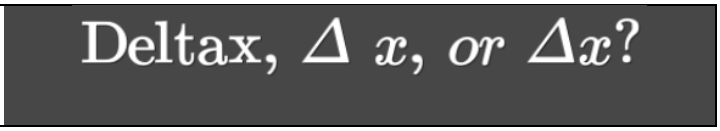
The major power of using this package is coming from the use of backslash. Backslashes can be used for either declaring a command or symbol. Symbol in TEXDraw is created by typing a backlash after character name. Navigate here for list of symbols used in TEXDraw.

```
\Delta\theta\approx2t\times(3\pi+4\omega)

\diamondsuit\cup\spadesuit=\diamondsuit+\spadesuit-(\diamondsuit\cap\spadesuit)
```

Sometimes you might find problem when joining a symbol with letter character, to do that you need to group the letter using braces `{}` so the parser can separate it.

```
\Deltax, \Delta x, or \Delta{x}?
```

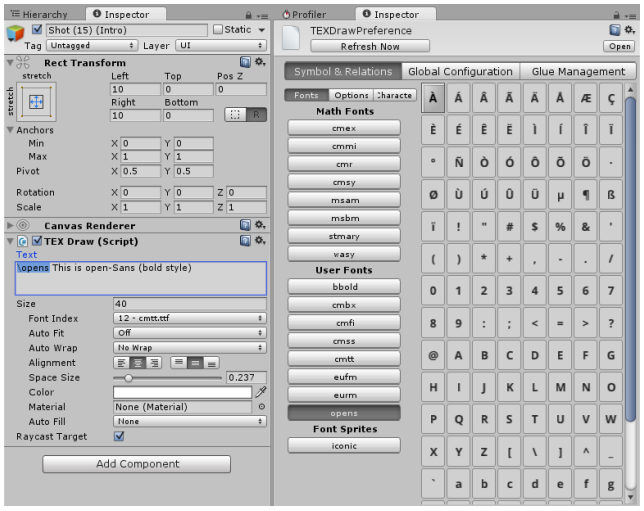


Using Custom Font Asset

The first common usage of commands is change which font is used in rendering. The complete format of this is:

```
\<fontname>[style]{text here}
```

Where `<fontname>` is the file name (according to the list) of font that you'll use. `[style]` means what font style will be use with options `[b]` (bold), `[i]` (italic), `[bi]` (bold-italic), `[]` (normal style), or no at all (styles remain unchanged).




```
\opens[i] Open Sans italic

\bbold Double \eufm{Inside but}
still double 'till} back again
```

Since V2.6, all braces is optional. This makes typing slightly cleaner without dying with lots of braces. Like second example above, this typing...

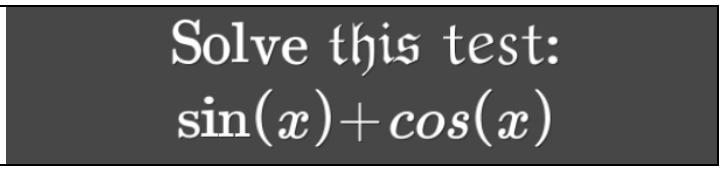
```
\size[2]{R\color[ff0]{e\cmtt{d\size[1]{d\color[f11]{e\cmss{r}}}}}}
```

Will exactly equivalent to...

```
\size[2] R\color[ff0] e\cmtt d\size[1] d\color[f11] e\cmss r
```

Another easy implementation for this is by undefined symbols. Type backslash followed by a non-symbol-defined word will generate a text with different styling. This behaviour is mostly used for differentiate between math function and variable.

```
\text Solve \eufm this \eurm test:
\sin(x)+cos(x)
```



For turning off font styling (similar to selected -1 in inspector), you can use `\math` instead.

Writing Fractions

Fractions is common in math, they have a numerator and denominator. It is possible to write them in TEXDraw, to do that, we need to follow on this rule:

`\[n|l|r]frac{`*`numerator`*`}{`*`denominator`*`}`

Don't understand? At very basic usage, type `\frac` followed by numerator surrounded by braces and then denominator with also surrounded by braces will generate a fractions. Nested fraction (ie, fraction inside a fraction) also supported here.

```
\frac{2+2}{2x}\equiv\frac{d-(\frac{1}{4})}{r}
```

$$\frac{2+2}{2x}\equiv\frac{d-\left(\frac{1}{4}\right)}{r}$$

```
f(x)=\lbrace\nfrac{2x}{3}
\nlfrac{\if x<0}{\otherwise}
```

$$f(x)=\begin{cases}2x & \text{if } x<0\\ 3 & \text{otherwise}\end{cases}$$

If you look at the second example, `\nfrac` is another variation of fraction where it doesn't render a line. So do the `l` and `r` attribute, which is aligning the position either numerator or denominator to the left or right. The combination of `n` and `l` or `r` attribute like example above (`\nlfrac`) is also supported.

Writing Roots

Root is another common math operation in everyday life. It's consisting of expandable surd (radical) sign (√) with a thick line on the root base. Writing Roots is easy, by follow on this format:

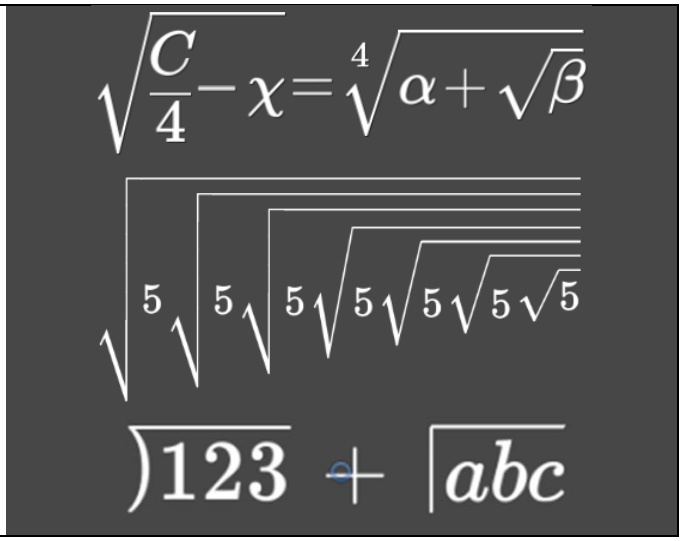
```
\root[degree]{base}
```

Here, type \root followed by base root surrounded by braces. The degree symbol is optional, but if you need it, simply type it before root base and surrounded by square bracket. Unlike fraction, root doesn't have any variations, but the root sign (√) can be replaced by typing a delimiter in `[degree]`

```
\root \frac{C}{4}-\chi=\root[4]
\alpha+\root{\beta}

\root 5\root 5\root 5\root 5\root
5\root 5\root 5

\root[)]{123} + \root[|]{abc}
```

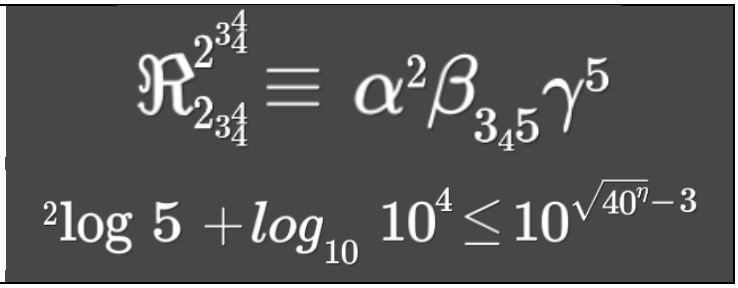


Superscript and Subscript

Scripts in TEXDraw can be archieved by typing `^` for superscript, or `_` for subscript. Optionally you can put braces `{}` after it so it is clear which character are taken into account

```
\Re^{2^3^4_4}_{2_3_4^4}\equiv
\alpha^2\beta_3_4}5\gamma^5

^2\log 5 +\log_10
10^4\leq10^{\root{40^\eta}-3}
```



Note the first example. Scripts have depth level, and it is limited to three, beyond that, they won't go smaller again.

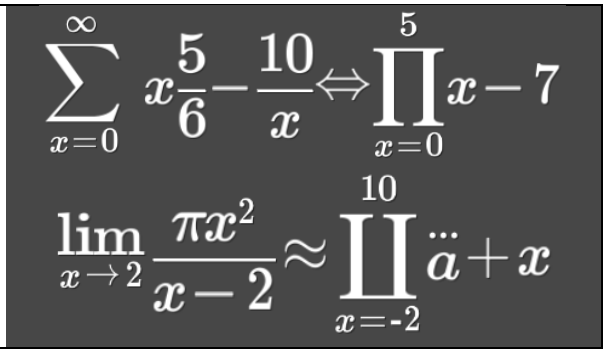
NOTE: Scripts have issues when used in conjuction with TexSupPerCharacterBase. Make sure *always* make braces `{}` after script, otherwise it'll mess up final rendering.

Expression Over/Under another Expression

Scripts put expressions in front their base expression, but to put it directly over/under them, they need to declare scripts for two times. That's mean `^^` to put it over, and `__` to put in under. Double script isn't necessary if base expression is member of Big Operator.

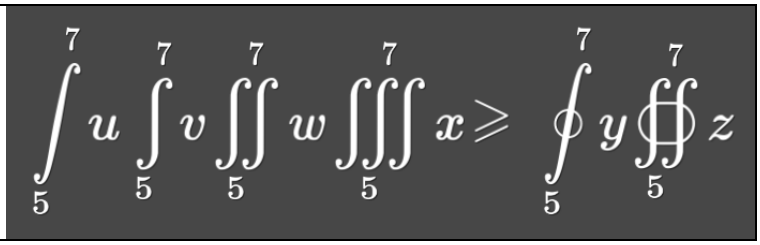
```
\sum^\infty_{x=0} x\frac{5}{6}-
\frac{10}{x}\Leftrightarrow\prod^5_{x=0
}x-7

\lim__{x\rightarrow2}\frac{\pi{x}^2}{x-
2}\approx\coprod^10_{x=\min2}{a^^{...}+
x}
```

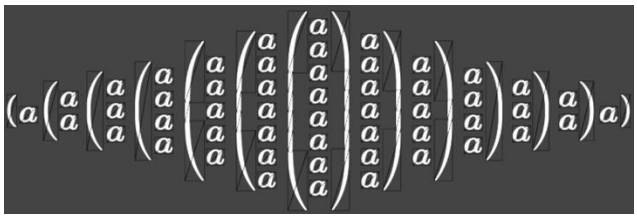


For integrals, they'll automatically aligned to 'code-implemented' aligment.

```
\int^7_5}u
\varint^7_5}v
\iint^7_5} w
\iiint^7_5} x
\geqslant
\oint^7_5}y
\oiint^7_5}z
```



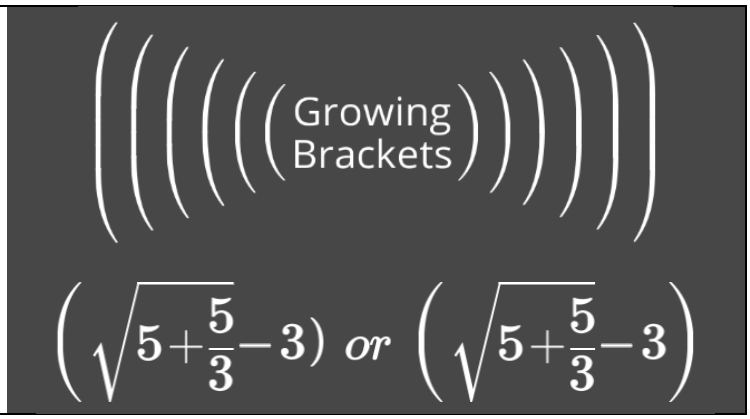
Using Expandable Delimiters



Delimiters like brackets `()`, or any other variations like `[]`, `{}`, `||` can expand higher or equal than their neightbours, automatically. This feature called Expandable delimiter and they can expand either vertically or horizontally depending on the specific character itself.

Growing brackets determining it's minimum height by comparing on other character in either left or right side of it. This behavior mostly result in equal height on pairs, except on specific case, and therefore, optional braces `{}` can be given to make both equal in height

```
((((((\opens\nfrac{Growing}
    {Brackets}}))))))
```

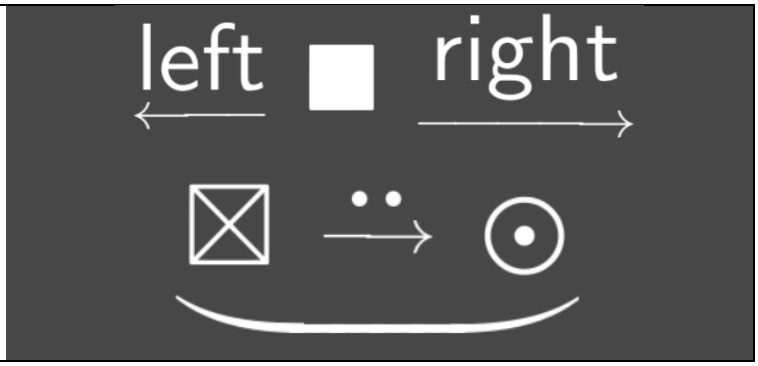



```
(\root{5+\frac{5}{3}}-3) or
({\root{5+\frac{5}{3}}-3})
```

Using Horizontal Extension

Horizontal extension is something like expandable delimiter... but expand horizontally. This situation can be used for something like very long horizontal Arrow, or if you want to create some horizontal arrow with text/graphic placed above/under it. This feature can be used by putting double script before a horizontal extension. This is also works for vertical extension, but they'll rotated clockwise (so it is still a horizontal extension).

```
{left}__{\leftarrow}
    \blacksquare
{right}__{\rightarrow}
```




```
{\boxtimes
{\ldot\ldot}__{\rightarrow}
    \odot}__\rbrack
```

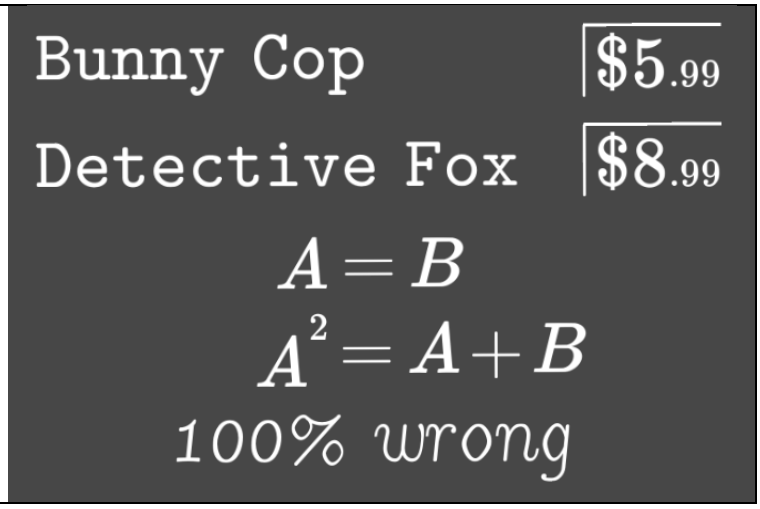
Preserving Fixed amount of Horizontal Space

`\[l|r]hold[width]{base}`

`\hold` command preserves a relative amount of `[width]`, and then use the reserved space to fill with `{base}`, optionally. If `{base}` is an Expandable delimiter, it'll expand automatically. Also optionally you can choose the alignment either left or right using `\lhold` or `\rhold`. Much likely you'll use this to align expressions correctly without splitting game objects.

```
\lhold[7]{Bunny Cop}
\math\root[|]$5\size[.].99
\lhold[7]{Detective Fox}
\math\root[|]$8\size[.].99

\rhold[4]{A}=\lhold[4]{B}
\rhold[4]{A^2}=\lhold[4]{A+B}
\cmfi 100% wrong
```

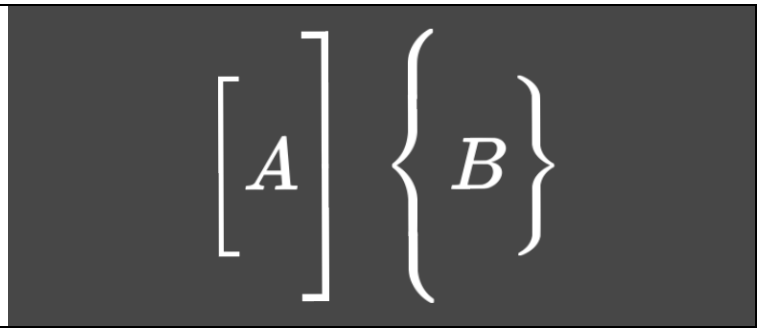


Preserving Fixed amount of Vertical Space

`\[t|v|b]hold[height]{base}`

This version of \hold command reserves expression vertically. Use this if you want a fixed tall of expandable delimiters.

```
\vhold[2]{[}A\vhold[3]{]}
\vhold[3]{\{}B\vhold[2]{\}}
```



Custom Color

`\color[hex-color]{base}`

Rendered color can be configured by \color and specifying by `[hex-color]`. Supported schemes for `[hex-color]` is `[#rgb]`, `[#rgba]`, `[#rrggbb]`, `[#rrggbbaa]`. It also accept without hashtag `[rgb]` , or unity's html name `[yellow]`, or even customized 4-bit console color.

```
\color[f52] a\color[#bf1]
b\color[#2f9] c\color[cyan]
d\color[46f] e

\clr[0]0\clr[1]1\clr[2]2\clr[3]3
\clr[4]4\clr[5]5\clr[6]6\clr[7]7
\clr[8]8\clr[9]9\clr[a]a\clr[b]b
\clr[c]c\clr[d]d\clr[e]e\clr[f]f
```

Beside `\color`, there's also `\clr` and `\mclr`. The difference between these three is located in how they mix existing color. `\color` will overwrite RGB, but A will be multiplied, `\clr` overwrites all RGBA channel, while `\mclr` (abbreviate for *mix-color*) will multiply all RGBA channel.

Custom Size

`\size[ratio-offset]{base}`

The `\size` command resize characters relatively, optionally offset can be given for shift character upward. Unlike other commands, size work independenly each other, so they can't be nested. There also special typos like `\size[.]` to make it smaller as script, and `\size[..]` to make it smaller as size of nested scripts.

```
\eufm{Station} 9\size[.45-
.15]\frac{3}{4}

This{\size[..]is ridicolously
small as}__{\Rightarrow}Hell
```

Writing Matrix

Matrix is a bunch of expression that grouped in specific column and row. Matrix is separated in column by `&`, then in row by `|`. By default, matrix is filled row-by-row.

```
\[v]matrix{n11&n12&n13|n21&n22&n23|n31&n32&n33 … }
```

```
[\matrix{x|y}]\times|\matrix{2&8|
3&\min1}|=(\matrix{\min9&8|9&\alp
ha})
```

$$\begin{bmatrix} x \\ y \end{bmatrix} \times \begin{vmatrix} 2 & 8 \\ 3 & -1 \end{vmatrix} = \begin{pmatrix} -9 & 8 \\ 9 & \alpha \end{pmatrix}$$

```
n_{xy}=\{\matrix{n_0& ...
&n_x||{:\dot}&&{:\dot}||n_y& ...
&n_{xy}}\}
```

$$n_{xy} = \left\{ \begin{matrix} n_0 & \cdots & n_x \\ \vdots & & \vdots \\ n_y & \cdots & n_{xy} \end{matrix} \right\}$$

To write matrix column-by-column you can type `\vmatrix{…}` instead, so `\matrix{a&b|c&d}` is equal to `\vmatrix{a|c&b|d}`.

Writing Table

Writing Table in TEXDraw is similar to Matrix, the only difference is that they added some lines between and outside of each child. In this table, you can also set-up cell alignment and line widths.

```
\[v|r|l]table[line-widths]{n11&n12&n13|n21&n22&n23|n31&n32&n33 … }
```

You can type \rtable for alignment to the right, or \vtable if you want column-by-column table (like matrix above). You can also change each cell line thickness by modifying the line-widths section. In Line-width options, type 6 digits that defines their thickness of (correspond to) Horizontal lines in outside, first, and secondary cell, while last 3 digits represent the thickness for vertical lines in outside, first, and secondary cell. Maximum allowed line thickness is 2, while you still can type them zero if you doesn't want to.

```
\ltable[111121]\Number&\Class&\Na
me|001&A&John|002&B&Skeet|003&C&B
row
```

Number	Class	Name
001	*A*	*John*
002	*B*	*Skeet*
003	*C*	*Brow*

Adding Diagonal Overlay Lines

Sometimes, in math, you need a line that crosses some formula either horizontal or diagonally. TEXDraw made them simpler.

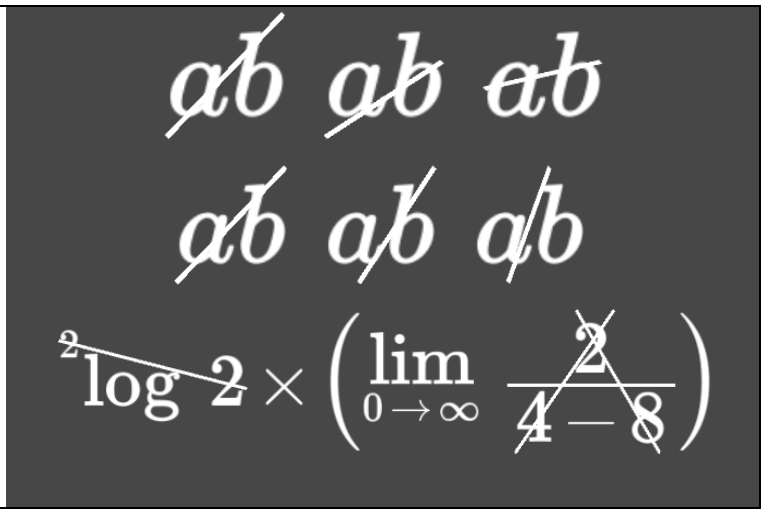
```
\[v|n]not[offset1-offset2]{base}
```

Formula above creates a diagonal line across base. Default direction is from bottom-left to top-right, and you can inverse it by using \nnot. Additionally, `[offset1-offset2]` determine distances between corners (horizontally), while \vnot giving distances from corner vertically. Both also can be combined.

```
\not[0-0]{ab} \not[0-.4]{ab}
\not[.4-.4]{ab}

\vnot[0-0]{ab} \vnot[.3-0]{ab}
\vnot[.3-.3]{ab}

\nnot[.6-.3]{^2\log
2}\times(\lim__{0\rightarrow\inft
y} {\vnot[.1-.8]\vnnot[.9-
.2]{\frac{2}{4-8}}})
```



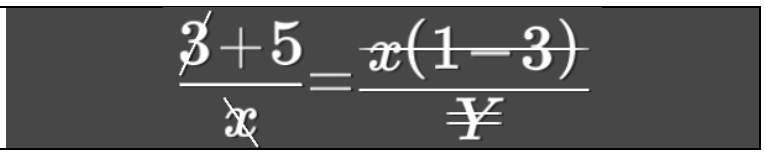
Adding Horizontal Overlay Lines

To give horizontal line across base, you can instead choose one of four choices below.

```
\[h|d|u|o]not[offset]{base}
```

`\hnot` means strikethrough, while `\dnot` means double strikethrough. `\unot` and `\under` can be used as underline, while `\onot` and `\over` means overline. All of them are matter of placing and can be shifted vertically using `[offset]`, optionally.

```
\frac{\not{3}+5}{\nnot{x}}=\frac{
\hnot{x(1-3)}}{\dnot{Y}}
```



Clickable Link

`\[u]link[eventname]{base}`

This command requires TEXLink to be added besides TEXDraw, otherwise it is never functional at all. This command make {base}'s colour goes interactable through user interaction.

When user clicks on {base}, TEXLink's event `OnLinkClicked(string)` are triggered, where `(string)` is what `[eventname]` says, or `{base}`, if it omitted.

There's also \ulink to get a hyperlink-like by giving underline beneath it.

Meta (Paragraph-wide) configuration

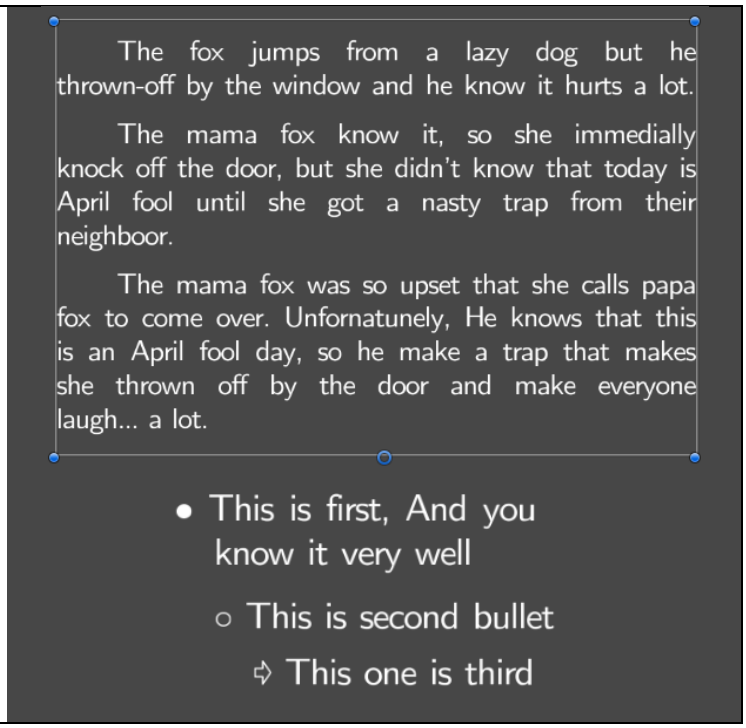
Meta is a special command that instead of make the effect on specific block, it's affect the whole paragraph, and any paragraph beneath it. The options of using Meta are:

`font`	`f`	Select font by index	`kern`	`k`	Additional character kerning
`size`	`s`	Override an absolute size	`lead`	`l`	Left margin of first line in paragraph
`align`	`x`	Align paragraph by l, c, or r	`line`	`h`	Set a fixed line height
`left`	`b`	Left paragraph margin	`space`	`n`	Additional line spaces at every line
`right`	`r`	Right paragraph margin	`para`	`p`	Additional line spaces at end of paragraph

You can combine multiple options into one like: `\meta[left=2 align=r para=1]` or make it shorter: `\meta[@b2xrp1]`. Meta is really useful if you want to create paragraph-based text or bulleted list of things. Also if you put meta on empty paragraph, the paragraph will have zero height. You can reset the properties back by entering empty `\meta[]`

```
\meta[lead=2 para=.5 kern=-.05]
The fox jumps from a lazy dog but he thrown-off by the window and he know it hurts a lot.
The mama fox know it, so she immedially knock off the door, but she didn't know that today is April fool until she got a nasty trap from their neighboor.
The mama fox was so upset that she calls papa fox to come over. Unfornatunely, He knows that this is an April fool day, so he make a trap that makes she thrown off by the door and make everyone laugh... a lot.
```

```
\meta[lead=-1.5 left=1.5 para=.5 kern=-.05]
\rhold[1]\bullet This is first, And you know it very well
\rhold[2]\circ This is second bullet
\rhold[3]\pointer This one is third
```



Apply 3D Transformation to Character

`\[m]trs[transformation]{base}`

Using this command, characters can be translated, rotated, and scaled either individually (`\trs`) or by median (`\mtrs`). Please note that this command only doing the transformation, after *boxing* process, so this mean other character won't be recalculated anymore. The rules for `[transformation]` is like:

Example	Means	Example	Means
[T1.0]	Move by 1 unit at Z direction	[R20,30]	Rotate by (X, Y) = (20,30)
[T1,2]	Move at (X,Y) = (1,2)	[S2]	Scale by factor of 2, uniformly
[T3,2,-1]	Move at (X,Y,Z) = (3,2,-1)	[S1,3]	Scale by (X,Y) = (1,3)
[R20]	Rotate by 20 degree at Z	[T2R30]	Move (Z) = 2, then Rotate (Z) = 30
[RX-20]	Rotate by -20 degree at X	[S2TZ1]	Scale by factor of 2 then move Z by 1

```
\trs[R10]Slanted \trs[S1.6]\cmss text

\trs[R10]B}\trs[Y.15R14]i}\trs[Y.3R13]n}\trs[Y.45R10]d}\trs[Y.5R6]i}\trs[Y.5R2]n}\trs[Y.5R-1]g}\trs[Y.45R-9]I}\trs[Y.4R-8]t}
```

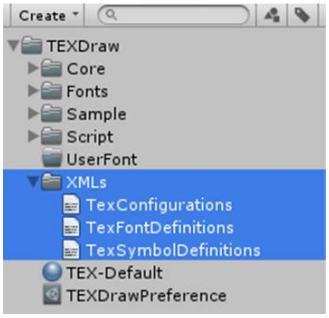
3D Transformation like this are super useful if the calculation is automated, and one of supplement that using this is TexSupWarpByTransform.

Using & Editing TEX Preference

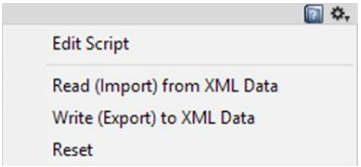
Preamble

In the previous section, we know how to open TEX Preference and where it's located. Now we will talking about what's inside of this preference and how to customize it to suit your project need. Configuring TEX Preference is optional and you can skip this section if you are OK with default configurations that already provided in the package.

The Import-Export feature



TEX Preference save all configurations as a serialized data (so, it will saved inside `TEXDrawPreference.asset` placed in the TEXDraw root folder). Prior to V2.0, all preference saved as read-only data as a XML Files, which is slower (but stable). While we save all data as a serialized data, there is no guarantee that any changes will be safe and secure (not stable) (for example, when unity crashes, or when updating the TEXDraw package), so we need a separate saving method, and XML Import-Export feature is good solution for this issue.



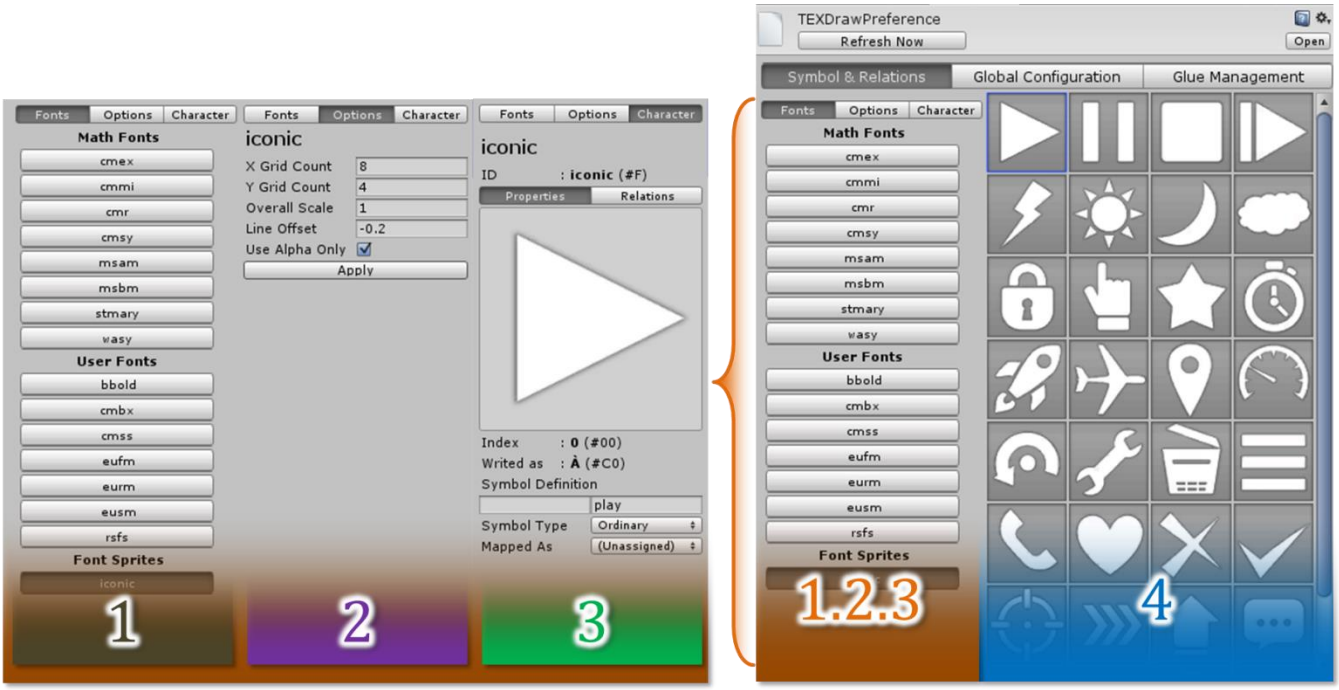
The XML Data (located in TEXDraw/XMLs) holds original data to the preference. In TEX Preference, You can read/write XML Data to the Preference itself with a single click. Remember that XML Files will **not** be included at build time (only the Preference asset itself).

In the "Gear Button" placed in top-right of preference, two main functions available, "Import" will read the XML Data, and override all modified preferences back to related XML Data, while "Export" will overwrite XML Data from TEX Preference. Do "Export" when you are done and OK with your changes and "Import" only if the preference was somehow broken / corrupt, or you made changes to the font data.

How does TEX Preference saved and be included on build?

In Editor time, every time you add a TEXDraw component in your scene, they'll find and locate where the TEX Preference live in your project, then serializing it to the component so later in real build, each component have a copy (reference) to the preference itself. If TEXDraw component we're added runtimely, they'll pick the preference from another TEXDraw component in the same scene (so make sure at least one active TEXDraw **exist** and enabled in your scene!).

Tab 1: Symbol & Relation



See the image above, there are 2 main sections, and in first section, there are three panels:

---

1 Font "Stack" Selections
Select a Font that will be previewed and configured in futher section

2 Importer Options
The options about how the selected font be imported (so far only customizable for textures)

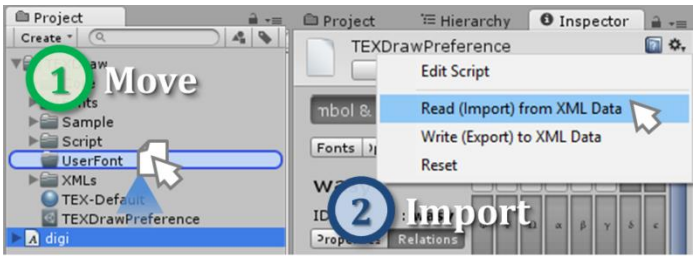
3 Per-Character Configuration
Selected character can be configured here, including it's symbol definition, type, and relation to other characters.

4 Character Map
Displays available characters that can be configured in section 3.

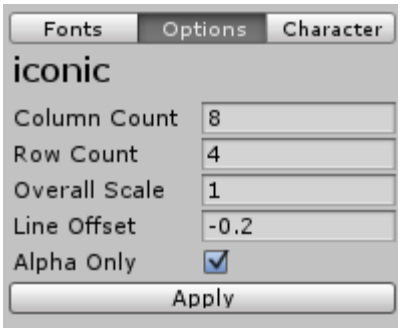
In short, this tab is used to manage how does fonts and textures are imported, including it's each character configuration. You can define your own symbol definition, or configure how it behaves with another character. All of features included inside this tab will be discussed futher later.

Adding your own font/sprite to the TEXDraw font stacks

1. Add your font (*.TTF or *.OTF) to TEXDraw/Fonts/User, or TEXDraw/Fonts/Sprites if it texture. The name of your font will be used as Their ID Name. (Be warned that the name must be only letters, unique, and case-insensitive)
2. Re-Import XML Data (you may want to export your preferences first). This will trigger the importer to register the font you've add earlier.
3. (Optional) if you were importing a texture, then you can adjust how it imported, in importer options.
4. Now your font is registered. To use it, simply define your own symbol, or leave as it is.



Configure How Textures are Imported



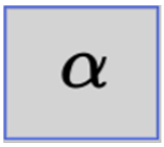
Textures are imported as it is contains bunch of sprites that have an equal dimension (grid-style). You can tell to the importer about how much column and row it has by set-up the column and row count. Optionally, overall scale for how large the texture size, and line offset for adjusting the "vertical offset" over the character's baseline. The alpha only check box determines whether your texture is colorable (by \color command) or not. If it unchecked, then the sprite color will be preserved.

Using & Navigating through Character Map

The characters that available in the selected font will be displayed here. If your keyboard is focused on this table, you can navigate what's selected by arrow keys.

As you can see in the screenshot on the right, there's different box style applied on each character. These different styles tell us about what's state that they're on. Take a look of these previews to make it clearer:

Char is Defined
The character has its own symbol definition

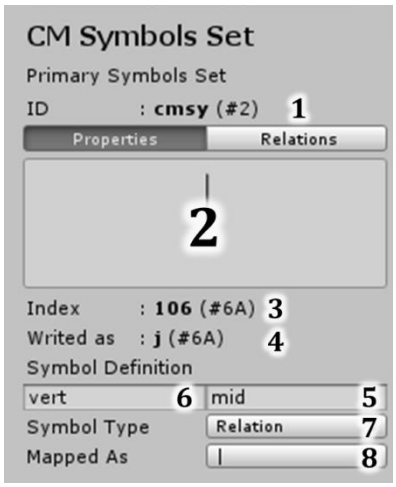


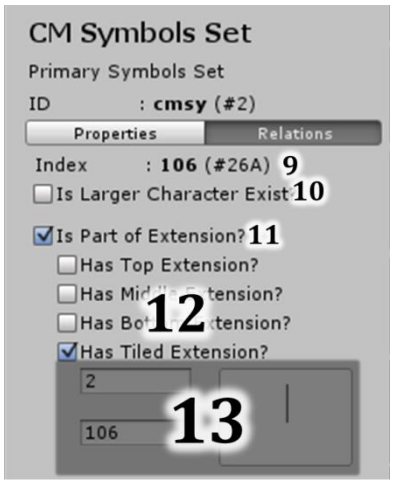
Char is Related
The character doesn't defined but it has relationship with other character

Char is Available
The character is yet defined nor related but still available.

Not Available
The character doesn't exist and can't be used or defined

Modify a Character Settings





The character configuration has 2 main tabs. The first tab contains some information and configurable properties.

1 ID of Selected font (for overriding font style like \cmr{}, etc.)
2 Preview of Selected Character
3 Character index (in TEX-Space)
4 Actual character index, Also see here.
5 Primary symbol definition
6 Secondary (alternative) symbol definitions
7 The Type of symbol, discarded if symbol definition still blank
8 Default Character Map, see the note below
9 Character Index (the Hex value display Hash index)
10 Does the similar but larger edition exist? See here.
11 Is the character refers to a group of extension? See here.
12 What part of extension exists? See here.
13 Index of Font (top) and Character (bottom) which is refer to.

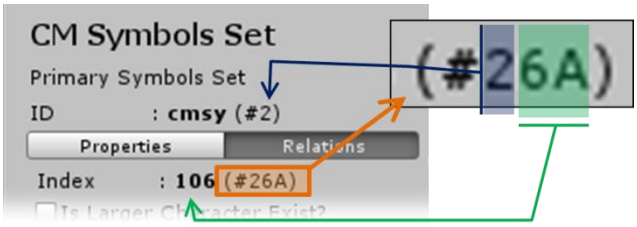
Please note that for custom font you should leave the “Mapped as” option unassigned. This option helps the parser guess common symbol that exist on your keyboard (example like | means \vert; + for \plus; ! for \faculty, etc.). Since all character has been preserved in math fonts, there’s no reason to assign it to another symbols.

Understanding Symbol Types

Symbol type is crucial (especially when dealing with glues) and it has to be in the right choice. Below is the detail of every available choice:

Ordinary	Character is used in conjunction with variables or letters. Example: \min \alpha \beta \epsilon \vartheta \gamma \hbar
Geometry	Character is in Geometrical Shapes Example: \triangle \lozenge \circ \blacksquare \smiley \leftmoon
Operator	Character is likely be used for alphabetical (binary) operators Example: \plus \cup \wedge \times \oslash \boxdot \circledcirc
Relation	Character is mostly used for comparing between two kind of formula Example: \leq \Less \gtrsim \leqslant \approx \equiv \risingdotseq \ncong
Arrow	Character's Shape is pointing Arrow Example: \rightarrow \Leftarrow \Updownarrow \curlywedgedownarrow
Open Delimiter	Character is used as delimiter with face directing to the right side Example: \lbracket \lsqbrack \lbrace \lfloor \lceil \lgroup \llbracket
Close Delimiter	Character is used as delimiter with face directing to the left side Example: \rbracket \rsqbrack \rbrace \rmoustache \rangle \rrbracket
Large Operator	Character usually used in its larger size Example: \sum \prod \int \oiint \bigcup \bigsqcup \bigotimes \bigvee
Accent	Character usually be put over previous symbol Example: \dot \vec \hat \widehat \tilde \widetilde \Dot \breve \tip
Inner	(Not available) used for internal types like fractions, root, matrix, etc.

What is Character Hash, and what's the point of it?

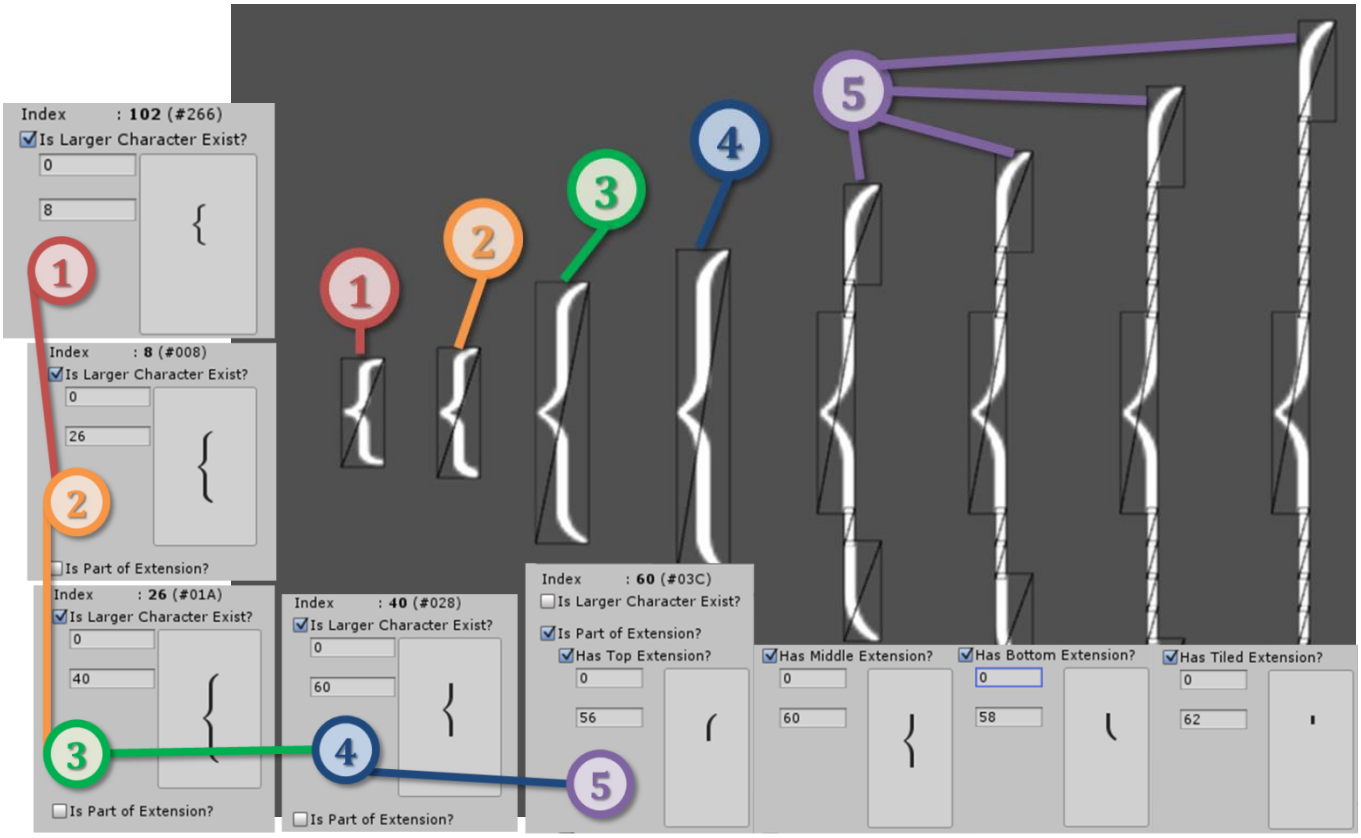


Character hash is a number that given for each character registered in TEXDraw font stack. A hash number that given on a character is unique among the rest. It's easy to read it in Hex Format. For example like in the screenshot in the right, #26A, means the character located in font with index of 2 (cmr), and it's character index is #6A (in digit, it reads 106). This feature is useful for cases where you want to check if something wrong in XML data or debugging where duplicate symbol exist.

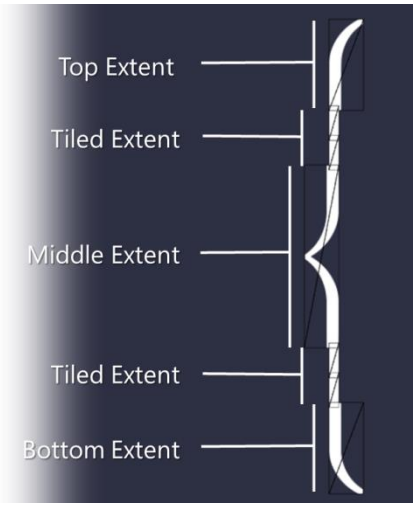
Please note that for validity of character hash, in hex display, the first two digits should be ranging from #0 to #7F (0 to 127), while third digit should be ranging from #0 to #1E (0 to 31), or in short, maximum possible value of a hash is #1E7F.

The power of Delimiters: Making Character Relations

If you have read previous section about delimiters, you should know what happened when delimiter expand to achieve certain height. Now in this section we will reveal the background process of how delimiters can expand their height. Take a look on image below:



Don't get the Idea? This image shows every "level" and each "relation" character configuration of a delimiter \lbrace, which we note on each level in a number. \lbrace has up to 5 levels height. In the first level, a character with hash #266 holds the \lbrace symbol definition, and does refer to a larger character located in #008. So if #266 doesn't have sufficient height, the character will be replaced by #008. This also happens on second, third, and fourth level. But what happen on fifth level?



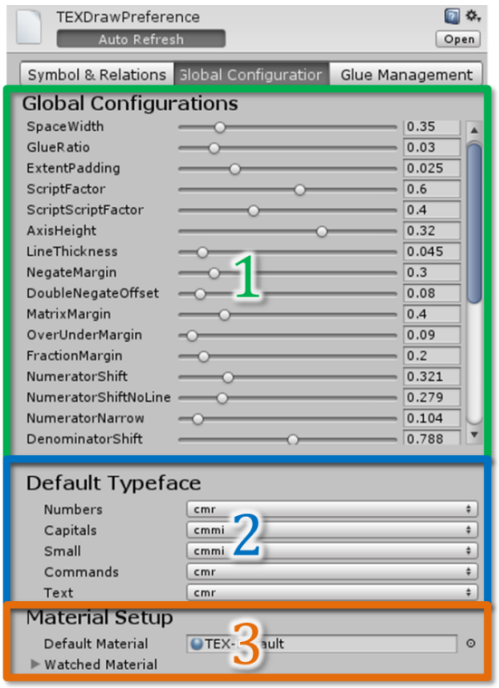
In the fifth level, the character doesn't refer to a larger one. Instead, it's marked as part of an extension character. An Extension character is a group of multiple characters that stacked one-by-one vertically so they can reach any certain height. One extension character contains 4 extent types: Top, Middle, Bottom, and Tiled extent. Every extension should have tiled extent. While top, middle and bottom extent is optional.

Please **be sure** that only symbols that have type of Relation, Arrow, Open Delimiter, and Close Delimiter can have relations feature like above, so check the character type support this feature!

Tab 2: Real-Time Global configurations

In this tab, a lot of customizable settings provided so it can suit on your need. Similar like previous tab, it has three main sections:

1. Global Configurations
   Control gaps and sizes using this section
2. Default Typeface
   Determine what's font are used for common things
3. Material Setup
   Determine what's material be used as default rendering

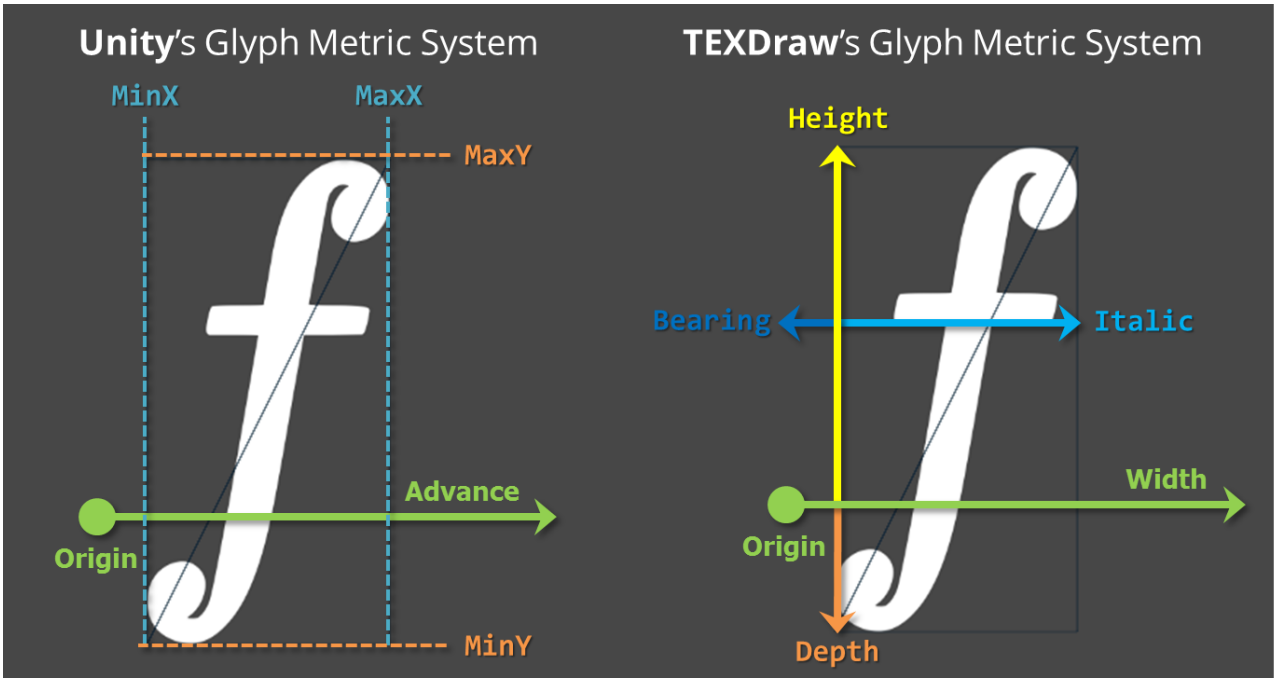


Understanding & Using Global Configurations

Each "config" has its unique purposes. You can tune each config with our example scene named "PreferenceSetUp" until it looks perfect for your project. Here in this section we provide some useful information for each config with relevant color on images for quick guidance.

Anatomy of a Character

Before you can understand the key of how a config works, you need to understand how a character behaves. Every character has a character rectangle (bound), and this rectangle sometimes can be called as glyph metric. This glyph metric data is already saved within TEX Preference with help from Unity's built-in glyph metric system... with some modification:



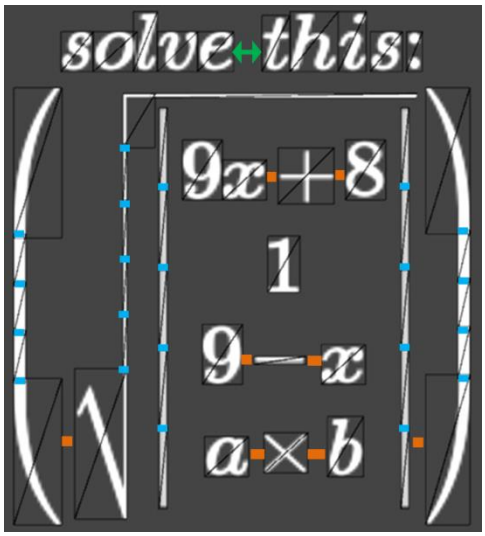
A character has a baseline (marked as green line), top and low bound. The image above will help you to understand each config which we will discuss below:
(We provide each config explanation with relevant image and indicator colors)

Space Width — The width of a single whitespace
The width is on fixed value.

Glue Ratio — Fixed Width of one "Glue" unit
Glue is additional gaps (kerning) of different symbol type. You can control individual Glue on the next tab.

Extent Padding — Extension's Additional Stretch width
Control's the extension padding. This config is existed to keep part of extension looks "connected" each other.

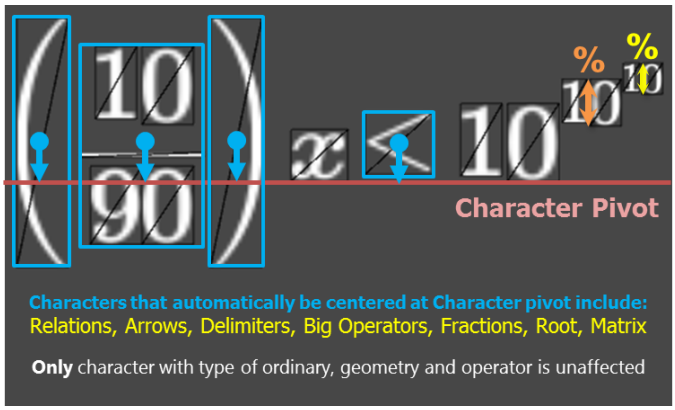
Line Height — Minimum line height
Minimum height of a single line.



Script Factor — Size Ratio of a Script
The final script total height in percentage compared to standard character size.

Nested Script Factor — Size Ratio of Nested Script
Similar to script factor, but applied for a nested script.

Axis Height — Centre Axis Pivot Offset
For a centered character (see the image note), this config will shift their position upward (to match with standard character height).



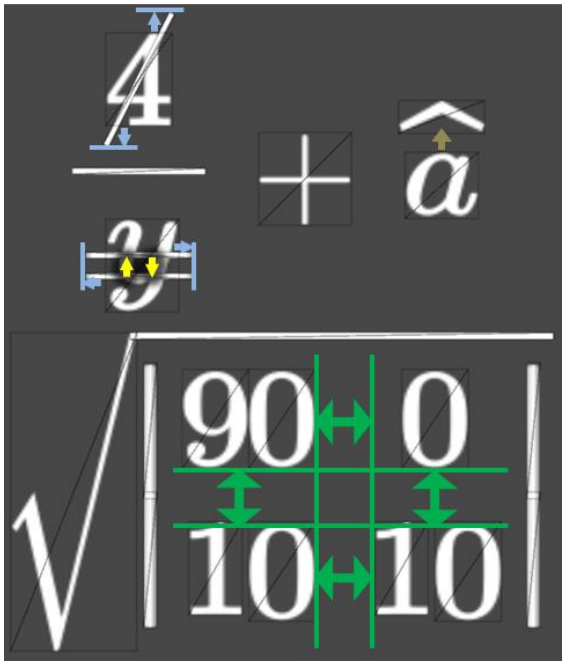
Line Thickness — .Fixed Line Thickness Width
Line thickness for common things (fraction, root, negations, etc.)

Negate Margin — Negation's Line Margin
Negation line with stretch beyond negated character bound until certain value.

Double Negate Offset — Double Negation Line Gap
Adjust to match the best gap height between top and bottom line.

Matrix Margin — Matrix child-by-child margin
The matrix boxes space gap on each column-by-column and row-by-row.

Over Under Margin — Additional Accent gap height
Shift Accent position upward until certain height (calculated from character's top bound).



Fraction Margin

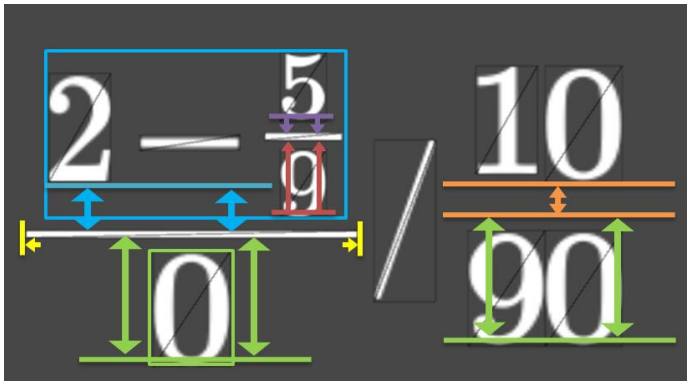
Additional fraction line width

Additional line width for fractions.

Numerator Shift

Standard Numerator Margin

Numerator's lift amount from linebase to the fraction line. If it less than the char depth, it will be "clamped" instead.



Numerator Shift no Line

Numerator Margin (no line)

Similar like Numerator Shift, but specialized for fraction with no line (\nfrac)

Numerator Narrow

Numerator Margin (narrow)

Similar like Numerator Shift, but specialized for a narrowed situation (eg. Inside a fraction, script, etc.).

Denomi-nator Shift

Standard Denominator Margin

Denominator's lift amount from linebase to the fraction line. If it less than the char height, it will be "clamped" instead.

Denomina-tor Narrow

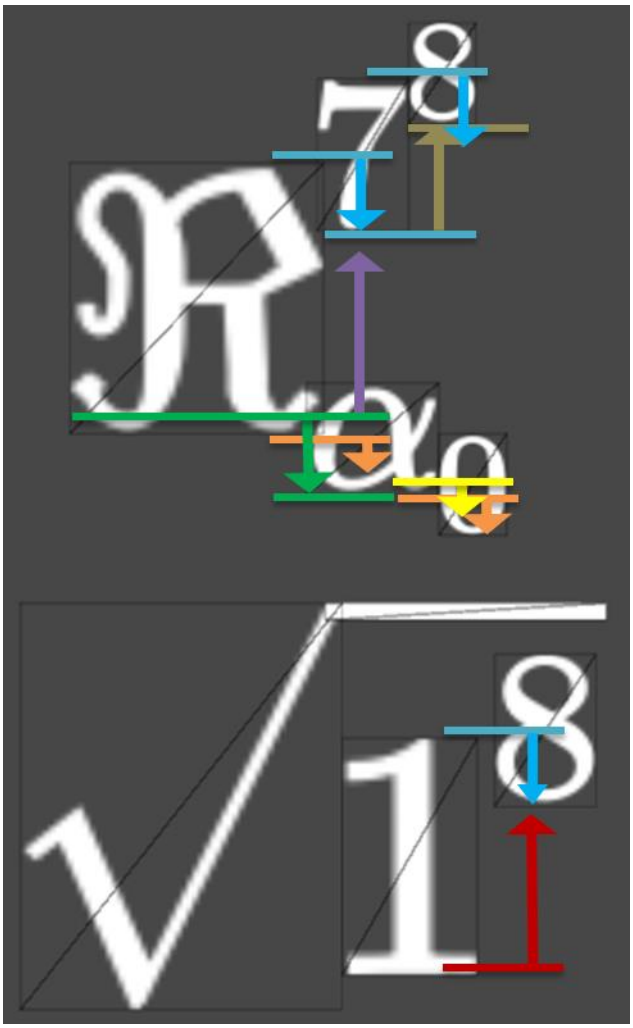
Denominator Margin (narrow)

Similar like Denominator Shift, but specialized for a narrowed situation.

Sup Drop

Superscript Drop Value

Superscript will be shifted downward until certain value. If value is zero, superscript baseline is in the same height as base script's top bound.



Sub Drop

Subscript Drop Value

Subscript will be shifted downward until certain value. If value is zero, subscript baseline is in the same height as base script's low bound.

Sup Min

Superscript Standard Minimum Low Bound

Minimum distance allowed between superscript's baseline to base script's baseline

Sup Min Cramped

Superscript Minimum Low Bound (Cramped)

Similar like Sup Min, but specialized for cramped situation (eg. Inside root, matrix, etc.)

Sup Min Narrowed

Superscript Minimum Low Bound (Narrowed)

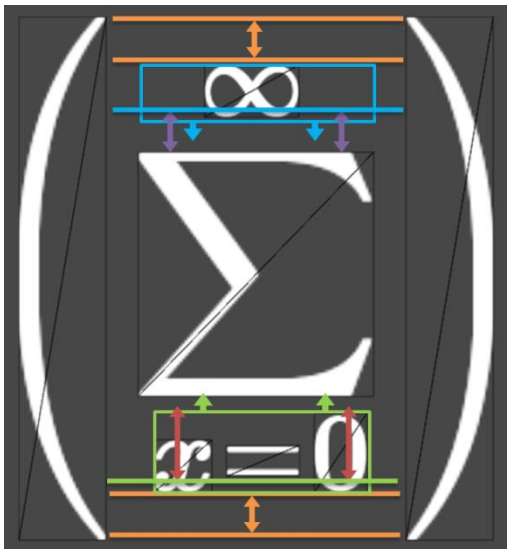
Similar like Sup Min, but specialized for narrowed situation

Sub Min No Sup

Subscript Minimum Drop (No Supscript Above)

Minimum distance allowed between subscript's baseline to base script's baseline

Sub Min On Sup	Subscript Minimum Drop (With Superscript Above) Similar like SubMinNoSup, but will used instead if superscript exist on same level.
Big Op Margin	Large Operator Top-Low Margin Big operator's additional height size.
Big Op Up Shift	Large Operator Up Shift Distance between top baseline to big operator's top bound
Big Op Upper Gap	Big Op Minimum Upper Gap Minimum distance allowed between top's low bound to big operator's top bound
Big Op Low Shift	Large Operator Low Shift Distance between bottom baseline to big operator's low bound.
Big Op Lower Gap	Big Op Minimum Lower Gap Minimum distance allowed between bottom's low bound to big operator's low bound



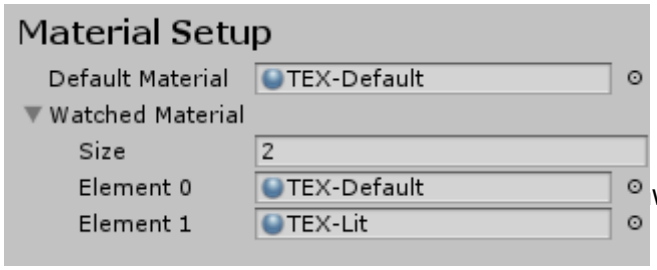
Default Typefaces, what is it?

In the section 2 of Global Configuration, you can configure what's font is used when you type something like number or letters. There are 6 different typefaces. Take a look on this example with customized settings and some indicators to make you easy to understand:

These different typefaces are: Number, Capital, Small, Command, Text, and Unicode. You can select a font that will be used as default renderings for each typeface.

Note: Unicode character *is* actually not supported nor listed in our font database, which is the reason why you can't have a symbol that refers to unicode characer. However, you can select any font you want, because any dynamic font is supported to render any unicode characters (ie. Never select a sprite font as default Unicode typeface).

Configuring the Materials

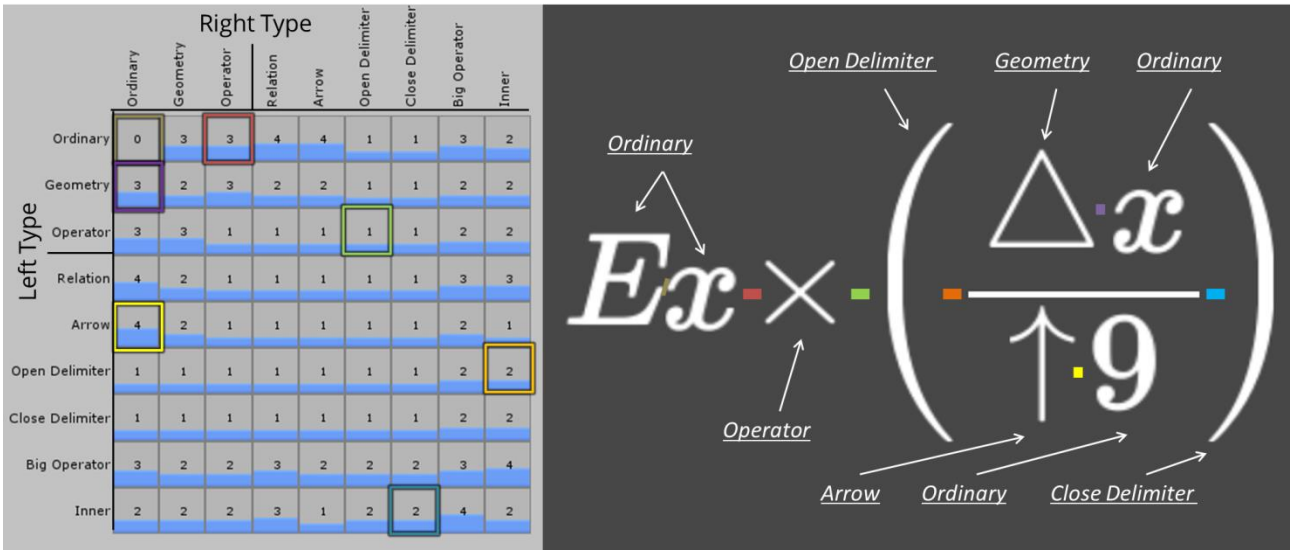


Every TEXDraw shader share the same texture slot depending on what font inside the preference. Of course that's also mean users don't have to plug each texture; Here we'll do it

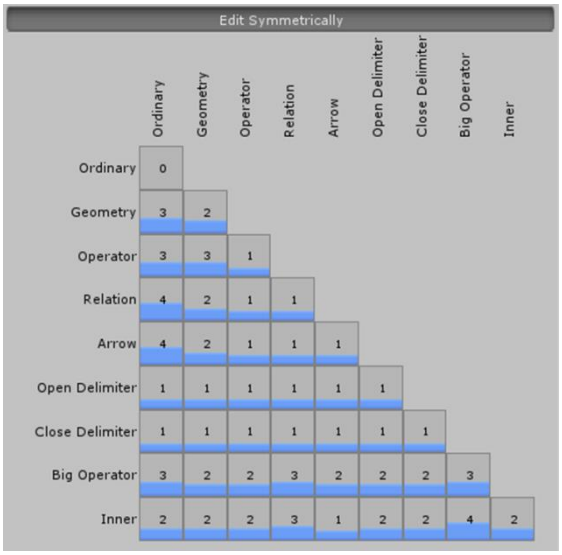
for you automatically. All you need to do is plug all the TEXDraw materials to **Watched Material**. Also, there's a slot for **Default Material** which is default to our built-in TEX-Default material. If you create a custom shader for TEXDraw, then you can put the material here, and it's texture slots will be filled automatically.

Tab 3: Glue Management

In this last tab, we can manage custom “kerning” spaces that applied on each character. We call this kerning space as “Glue”. This “Glue” can be customized quickly by manage per-character type (like a relation by geometry, etc.) instead of individual character. Take a look of this image:



As you see in the table, each row is left side type while each column is right side type. For example like in the green strip, operator \times meets delimiter \lbracket, to change how much it’s space between, simply adjust it in the glue table in column “operator” and row “open delimiter”, so do in another strip, and so on.



Sometimes to help avoid headaches, you can turn on the “edit symmetrically” button. This will make the table hide the half of its cells and “wrap and merge” around its transpose cell, or in another word, there’s no more left and right type difference (just like when you edit the physics collision matrix).

Please note that we can’t adjust Accent glue because it’s simply goes over previous character, while it’s possible to change inner types.

Symbol Definition Cheatsheet

Here, we display of all defined symbol in math fonts included in the package. Although you might find easier to find a symbol in the preference itself, it's not a bad things to display all of them in some groups:

Greek Letters

α	\alpha	η	\eta	ν	\nu	υ	\upsilon
β	\beta	θ	\theta	ξ	\xi	ϕ	\phi
γ	\gamma	ι	\iota	π	\pi	χ	\chi
δ	\delta	κ	\kappa	ρ	\rho	ψ	\psi
ϵ	\epsilon	λ	\lambda	σ	\sigma	ω	\omega
ζ	\zeta	μ	\mu	τ	\tau		
ε	\varepsilon	ϱ	\varrho	ϖ	\varpi	$\ni$	\backepsilon
ϑ	\vartheta	ς	\varsigma	φ	\varphi		
Γ	\Gamma	Λ	\Lambda	Σ	\Sigma	Ψ	\Psi
Δ	\Delta	Ξ	\Xi	Υ	\Upsilon	Ω	\Omega
Θ	\Theta	Π	\Pi	Φ	\Phi		

Common Ordinary Symbol

/	\forwardslash \slash	¿	\invquestion	¡	\invfaculty	-	\min \varminus
#	\numbersign	?	\question	!	\faculty	&	\ampersand
%	\percent	$	\dollar	”	\cdqot \doublequote	’	\semiquote
‰	\permil	¢	\cent	“	\odqot \vardoublequote	,	\comma
@	\commercialat	:	\colon	;	\semicolon	.	\ldot \ldotp

Miscellaneous Symbol

∂	\partial	$\prime$	\prime	þ	\thorn	℧	\mho
ℓ	\ell	$\backprime$	\backprime	Þ	\Thorn	ð	\eth
$\imath$	\imath	∞	\infty	ð	\dh	ב	\beth
$\jmath$	\jmath	$\varnothing$	\varnothing	ɔ	\openo	ג	\gimel
℘	\wp	$\emptyset$	\emptyset	ⅎ	\Finv	ד	\daleth
ℜ	\Re	$\forall$	\forall	⅁	\Game	Ϝ	\digamma
ℑ	\Im	$\exists$	\exists	$\surd$	\surd	$\varkappa$	\varkappa
ℵ	\aleph	$\nexists$	\nexists	$\amalg$	\amalg	𝕜	\Bbbk
®	\circledR	¬	\neg \lnot	∇	\nabla	ħ	\hslash
Ⓢ	\circledS	⌐	\rnot	$\smallint$	\smallint	ħ	\hbar
∁	\complement	¥	\yen	/	\diagup	\	\diagdown
⋈	\bowtie	¦	\brokenvert	ə	\inve	\	\backslash

Astronomical Symbols

☊	\ascnode	☿	\mercury	♉	\taurus	♐	\sagittarius

☋	\descnode	♃	\jupiter	♊	\gemini	♑	\capricornus
♀	\male	♄	\saturn	♋	\cancer	♒	\aquarius
♂	\female	♅	\uranus	♍	\virgo	♓	\pisces
♁	\earth	♆	\neptune	♎	\libra	☌	\conjunction
☼	\sun	♇	\pluto	♏	\scorpio	☍	\opposition

Block Shapes

▲	\blacktriangle	○	\circ	△	\bigtriangleup \triangle
▼	\blacktriangledown	●	\bullet	▽	\bigtriangledown
◀	\blacktriangleleft	◯	\bigcirc	△	\vartriangle
▶	\blacktriangleright		\star	▽	\triangledown
◖	\halfleftcirc	★	\blackstar	◀	\leftblacktriangle
◗	\halfrightcirc	□	\square	▶	\rightblacktriangle
◐	\blackhalfleftcirc	■	\blacksquare	◊	\lozenge
◑	\blackhalfrightcirc	◇	\diamond	♦	\blacklozenge

	\leftmoon		\bendsquare		\ataribox
	\rightmoon		\pentagon		\clubsuit
	\smiley		\hexagon		\spadesuit
	\blacksmiley		\varhexagon		\diamondsuit
	\frownie		\octagon		\heartsuit

Geometrical Symbol Shapes

	\flat		\angle		\smile		\smallsmile
	\natural		\varangle		\frown		\smallfrown
	\sharp		\measuredangle		\top		\dagger
	\eighthnote		\sphericalangle		\bot \perp		\ddagger
	\quarternote		\diameter		\clock		\phone
	\halfnote		\invdiameter		\pointer		\recorder
	\fullnote		\rightturn		\bell		\ball

	\twonote		\leftturn		\check		\lightning
	\AC \photon		\penstar		\checkmark		\varlightning
	\gluon		\hexstar		\pilcrow		\currency
	\VHF		\varhexstar		\kreuz		\comment
	\vernal		\davidstar				\maltese

Boxed Binary Operators

	\oplus		\boxarrowup		\varotimes		\boxplus
	\ominus		\boxarrowdown		\varoast		\boxminus
	\otimes		\boxarrowleft		\varobar		\boxtimes
	\oslash		\boxarrowright		\varodot		\boxdot

	\odot		\varolessthan		\varoslash		\varboxast
	\obar		\varogreaterthan		\varobslash		\varboxbar
	\obslash		\varovee		\varocirc		\varboxdot
	\olessthan		\varowedge		\varoplus		\varboxslash
	\ogreaterthan		\Yup		\varominus		\varboxbslash
	\ovee		\Ydown		\circledcirc		\varboxcirc
	\owedge		\Yleft		\circledast		\varboxbox
			\Yright		\circleddast		\varboxempty

Binary Operators

	\plus		\pm		\mp		\cdot

	\minus		\cup		\cap		\centerdot
	\times		\ucup		\nplus		\wr
	\ast		\sqcup		\sqcap		\moo
	\div		\wedge \land		\vee \lor		\merge
	\vartimes		\varcurlywee		\varcurlywedge		\varbigcirc
	\dotplus		\minuso		\baro		\talloblong
	\intercal		\sslash		\bbslash		\oblong
	\fatsemi		\fatslash		\fatbslash		\pointleft
	\divideontimes		\binampersand		\bindnasrepma		\pointright

	\doublebarwedge \Barwedge		\barwedge		\veebar		\pointup
	\leftthreetimes		\Cap \doublecap		\Cup \doublecup		\pointdown
	\rightthreetimes		\curlywedge		\curlyvee		
			\ltimes		\rtimes		

Relation Comparer

	\less \l		\gtr \g		\prec		\succ
	\leq		\geq		\preceq		\succeq
	\leqq		\geqq		\precsim		\succsim
	\leqslant		\geqslant		\precapprox		\succapprox
	\lesssim		\gtrsim		\preccurlyeq		\succcurlyeq
	\lessapprox		\gtrapprox		\curlyeqprec		\curlyeqsucc
	\eqslantless		\eqslantgtr		\subset		\supset
	\lessgtr		\gtrless		\subseteq		\supseteq
	\lesseqgtr		\gtreqless		\subseteqq		\supseteqq
	\lesseqqgtr		\gtreqqless		\Subset \doublesubset		\Supset \doublesupset
	\ll \Less		\gg \Gtr		\sqsubset		\sqsupset
	\lll \llless		\ggg \gggtr		\sqsubseteq		\sqsupseteq

$\lvertneqq$	\lvertneqq	$\gvertneqq$	\gvertneqq	$\subsetplus$	\subsetplus	$\supsetplus$	\supsetplus
$\lessdot$	\lessdot	$\gtrdot$	\gtrdot	$\subsetpluseq$	\subsetpluseq	$\supsetpluseq$	\supsetpluseq

Miscellaneus Relations

$\triangleleft$	\triangleleft	$\triangleright$	\triangleright	$=$	\equal \eq	$\equiv$	\equiv
$\vartriangleleft$	\vartriangleleft	$\vartriangleright$	\vartriangleright	$\doteqdot$	\doteqdot \Doteq	$\triangleq$	\triangleq
$\trianglelefteq$	\trianglelefteq	$\trianglerighteq$	\trianglerighteq	$\risingdotseq$	\rising dotseq	$\fallingdotseq$	\falling dotseq
$\trianglelefteqslant$	\trianglelefteq slant	$\trianglerighteqslant$	\trianglerighteq slant	$\asymp$	\asymp	$\propto$	\propto
$\leftslice$	\leftslice	$\rightslice$	\rightslice	$\varsmallpropto$	\varsmall propto	$\varpropto$	\varpropto
$\vdash$	\vdash	$\dashv$	\dashv	$\sim$	\sim	$\approx$	\approx
$\Vdash$	\Vdash	$\vDash$	\vDash	$\thicksim$	\thicksim	$\thickapprox$	\thickapprox
$\Vvdash$	\Vvdash	$\approxeq$	\approxeq	$\simeq$	\simeq	$\eqsim$	\eqsim
$\in$	\in	$\ni$	\ni	$\backsim$	\backsim	$\backsimeq$	\backsimeq

⟥	\inplus	⟤	\niplus	≏	\bumpeq	≎	\Bumpeq
∴	\therefore	∵	\because	≑	\eqcirc	≗	\circeq
∣	\mid	∥	\parallel	⫴	\interleave	⋔	\pitchfork
∣	\shortmid	∥	\shortparallel			≬	\between

Negated Relations

≮	\nless	≯	\ngtr	⊀	\nprec	⊁	\nsucc
≰	\nleq	≱	\ngeq	⋠	\npreceq	⋡	\nsucceq
⪇	\lneq	⪈	\gneq	⪵	\precneqq	⪶	\succneqq
≰	\nleqslant	≱	\ngeqslant	⋨	\precnsim	⋩	\succcnsim
≨	\lneqq	≩	\gneqq	⪹	\precnapprox	⪺	\succnapprox
≰	\nleqq	≱	\ngeqq	⊈	\nsubseteq	⊉	\nsupseteq
⋦	\lnsim	⋧	\gnsim	⊊	\subsetneq	⊋	\supsetneq
⪉	\lnapprox	⪊	\gnapprox	⫋	\varsubsetneq	⫌	\varsupsetneq
≁	\nsim	≆	\ncong	⫅̸	\nsubseteqq	⫆̸	\nsupseteqq
∤	\nmid	∦	\nparallel	⫋	\subsetneqq	⫌	\supsetneqq
∤	\nshortmid	∦	\nshort parallel	⫋	\varsubsetneqq	⫌	\varsupsetneqq
⊬	\nvdash	⊮	\nVdash	⋪	\ntriangleleft	⋫	\ntriangleright
⊭	\nvDash	⊯	\nVDash	⋬	\ntrianglelefteq	⋭	\ntrianglerighteq

					\ntriangleleft eqslant		\ntriangleleft eqslant

Primary Arrows

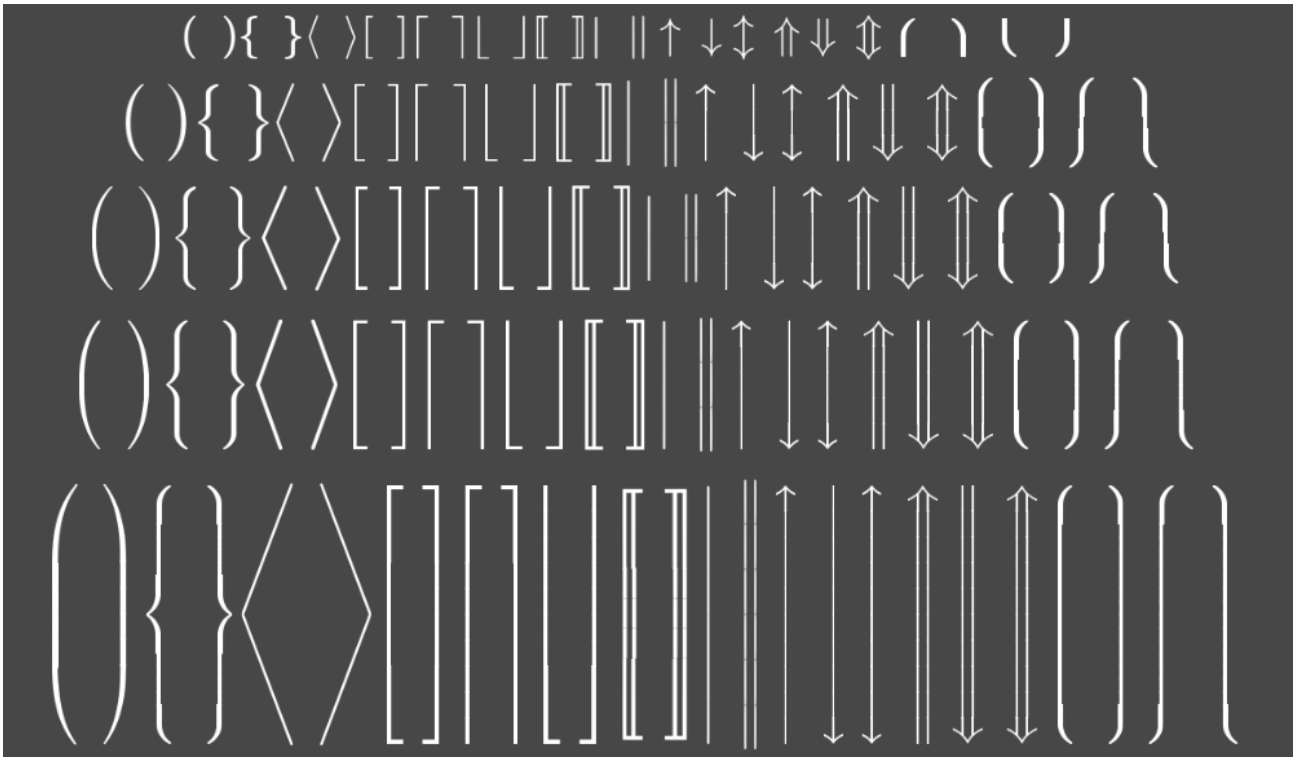
	\uparrow		\Uparrow		\leftharpoonup		\nearrow
	\downarrow		\Downarrow		\leftharpoondown		\searrow
	\leftarrow		\Leftarrow		\rightharpoonup		\swarrow
	\rightarrow		\Rightarrow		\rightharpoondown		\nwarroy
	\leftright arrow		\Leftright arrow		\upharpoonleft		\nnwarrow
	\updownarrow		\Updownarrow		\upharpoonright		\nnearrow
	\circle arrowright		\curve arrowleft		\downharpoonleft		\sswarrow
	\circle arrowleft		\curve arrowright		\downharpoonright		\ssearrow

*) New in V2.6, every horizontal arrow can stretch automatically using ^^ or __ (example: {into}__{\rightarrow}). In 2.7, Vertical delimiters also accepted with rotating the character clockwise.

Compound Arrows

	\shortup arrow		\upuparrows		\leftright harpoons		\curlyvee uparrow
	\shortdown arrow		\downdown arrows		\rightleft harpoons		\curlyvee downarrow
	\shortleft arrow		\leftleft arrows		\leftright arrows		\curlywedge uparrow
	\shortright arrow		\rightright arrows		\rightleft arrows		\curlywedge downarrow
	\nleftarrow		\Lsh		\twohead leftarrow		\Rrightarrow
	\nrightarrow		\Rsh		\twohead rightarrow		\Lleftarrow
	\nLeftarrow		\looparrowleft		\rightsquig arrow		\leftright arrowtriangle
	\nRightarrow		\looparrowright		\leftright squigarrow		\leftarrow triangle
	\nleftright arrow		\leftarrowtail		\leftright arroweq		\rightarrow triangle
	\nLeftright arrow		\rightarrowtail				\multimap

Expandable Delimiters



From left to right (read column-by-column):

\lbrack	\lsqbrack	\rrbracket	\Downarrow
\rbrack	\rsqbrack	\vert	\Updownarrow
\lbrace	\lceil	\Vert	\lgroup
\rbrace	\rceil	\uparrow	\rgroup
\langle	\lfloor	\downarrow	\lmoustache
\rangle	\rfloor	\updownarrow	\rmoustache
	\llbracket	\Uparrow	

Open & Closing Delimiter

(	\lbrack	)	\rbrack	[	\lsqbrack	]	\rsqbrack
{	\lbrace	}	\rbrace	⟨	\langle	⟩	\rangle
⌈	\lceil	⌉	\rceil	⌊	\lfloor	⌋	\rfloor
⎛	\lgroup	⎞	\rgroup	⎝	\lmoustache	⎠	\rmoustache

⟅	\lbag	⟆	\rbag	⦇	\Lbag	⦈	\Rbag
⟦	\llbracket	⟧	\rrbracket	⦅	\llparenthesis	⦆	\rrparenthesis
⦋	\llfloor	⦌	\rrfloor	⦍	\llceil	⦎	\rrceil

Large Operator

∫	\int	∫	\varint	∬	\iint	∭	\iiint
∮	\oint	∮	\varoint	∯	\oiint	⫼	\bigparallel
∑	\sum	∏	\prod	∐	\coprod	⫼	\biginterleave
⋃	\bigcup	⋂	\bigcap	⨁	\bigoplus	□	\bigbox
⊔	\bigsqcup	⊓	\bigsqcap	⨂	\bigotimes	▽	\big triangledown

⨄	\biguplus	⨇	\bignplus	⨀	\bigodot	△	\big triangleup
⋀	\bigwedge \bigland	⋁	\bigvee \biglor	⨘	\bigcurlyvee	⋎	\bigcurlywedge

Accent

These accents can be applied after a digit or symbol (widehat and widetilde can support more than one character as their base.

IMPORTANT: Always put accents in a braces inside (eg: {e\acute})

´	\acute	~	\tilde	ˇ	\check	˙	\dot
`	\grave	‾	\bar	^	\hat	→	\vec
”	\floatquote	⏜	\floatband	°	\Dot	⃛	\ddot
				⏞		⃩	
		˘	\breve		\widehat		\widetilde

Preserved Characters

These character defines char map data that included in the preference.

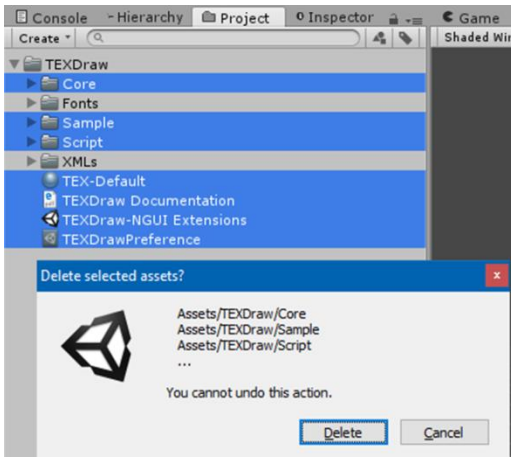
Char	Defined As	Char	Defined As	Char	Defined As	Char	Defined As
+	\plus	[	\lsqbrack	;	\semicolon	?	\question
-	\minus	]	\rsqbrack	:	\colon	!	\ldotp
*	\ast	<	\lt	`	\vert	@	\commercialat
/	\slash	>	\gt	~	\question	#	\numbersign
=	\equals	|	\vert	‘	\faculty	$	\dollar
(	\lbrack	.	\ldot	“	\ampersand	%	\percent
)	\rbrack	,	\comma	\^	--	&	\ampersand
\{	\lbrace	\}	\rbrace	_	--	\\	\backslash

Appendix: A side note to the users

Upgrading To V3.0

The proper steps to upgrade TEXDraw to version 3.0 follows conventional way to upgrade assets in Unity... with additional steps (to prevent user data lost)

NOTE: It is recommended to backup your project first, If you don't exactly know the conventional steps for upgrading an asset.



1. Save current scene and load new empty scene
2. (if project holds V2.x) → Export current preference to XML files (prevent data lost)
3. Delete All files & directory, except `Fonts`, `Resources` (for 3.x), and `XMLs` directory
4. Now Import latest version of package
5. Select all, except directories that you keep in step 3 (do not replace existing files)
6. Hit OK and wait until import process finished.
7. Rebuild Fonts by Click menu item in `Tools > TEXDraw > Rebuild Font Data`
8. (if you do step 2) → Transfers back properties from XML by selecting Preference (`Tools > TEXDraw > Open Preference`), then in Gear Menu, click `Legacy Transfer from XML Data`
9. Profit! Now Enjoy Upgraded version of TEXDraw.

TEXDraw Release Notes

3.3 - March 16, 2017: Supplement Upgrade

- NEW: A new example scene for keeping formulas in the game
- NEW: Reset button in preference context menu (suitable if you really stuck in a problem)
- NEW: Editor and additional notes for each supplement
- NEW: Editor toggle for showing supplement info, optionally (show is default)
- NEW: TEXSupDepthEffect for giving a depth effect for 3D Based UI
- NEW: TEXSupTrimSpaces for cleaning unneeded spaces in each paragraph
- NEW: (Undocumented yet) Modifiable template for creating TEXDraw objects (can be accessed via menu)
- FIX: Dirty isn't flagged at import which result in Font meta data won't be saved
- FIX: Dirty isn't flagged at editor change which result in Font meta data won't be saved
- FIX: (Only in editor) Shadow & Other built-in UI effect is not picked up automatically
- FIX: console errors isn't readable enough
- FIX: Possible error on internal part of SerializedDictionary

3.2 – Feb 18, 2017: Demo Released

- NEW: Demo (Windows build) is released for asset demonstration
- CHANGE: Integral over/underscript now aligned automatically
- CHANGE: Script without base will be right-aligned (useful for script-before-base cases)
- FIX: Infinite stack overflow when backslash typed after scripts
- FIX: ‘Iconic’ asset compression setting

3.1 - Feb 2, 2017: Bug Sweeping

- NEW: Basic Editor for TEXFont arbitrary assets
- NEW: Extra editor info for font import setup
- CHANGE: Importer will throw warn in console, if '-Regular' is used in font name
- FIX: Delimiter is not growing in particular cases
- FIX: Updated NGUI Scripts & Mask Shaders
- FIX: Build compilation exception & warning
- FIX: EOL consistency issue in script
- FIX: Preference will marked as 'dirty' after reimporting fonts.

3.0 – Dec 31, 2016: Repolished API

- NEW: Built-in TextMeshPro Integration
- NEW: Built-in SDF Importer and Shader for TMP Integration
- NEW: \meta command to apply paragraph-wide styling setting
- NEW: Rootsign of \root can be changed
- NEW: \hold command to reserve a fixed amount of space
- NEW: \trs and \mtrs to apply 3D transformation on character
- NEW: Best Fit mode in Autofit
- NEW: Local Continous and World Continus in Autofill Options
- NEW: 7 Additional supplements in package
- NEW: TEXDraw 3D accepts Rect Transform
- NEW: TEXDraw Menu items in Tools/TEXDraw
- NEW: Customizable character sets
- NEW: Benchmark tool, Font Swapper tool, and many else
- NEW: Material Assistant for quick management of different Materials
- NEW: Rebuild material now automatically choose best samples if available
- CHANGE: Improved TexDrawPerCharacterBase regex filter
- CHANGE: UV2 and tangents data is swapped for consistency
- CHANGE: Redefinition of \clr console color choices for consistency
- CHANGE: Deprecation of XML Data, through it is still supported to read from
- CHANGE: FontData now separated from Preference for data consistency
- CHANGE: TEXDraw no longer checks for UI mesh effect every rebuild time
- CHANGE: TexDraw Preference now saved in Resource
- CHANGE: Improved TEXDraw 3D Editor
- CHANGE: Tables now only add outer border if necessary
- CHANGE: Renamed ‘4 Passes’ to ‘Full’, and ‘X Passes’ to ‘X Samples’
- CHANGE: Fonts that have incompatible character will be ignored
- CHANGE: Font texture will resized if rescaled to keep it crisp on screen
- FIX: Unity 5.6 Compatibility
- FIX: TEXDraw 3D double render issue in some frequent cases
- FIX: Delimiters now can consistently grow bigger if necessary
- FIX: Scripts now consistent with unbraced expressions
- FIX: Accents now only wrap more than a character if it braced
- FIX: Preference now do null checks before previewing fonts
- FIX: Small performance improvement on rebuild time

2.7 – Oct 1, 2016: Improved Editor & API

- NEW: TEXDrawSupplement API to modify text similar to UI effects
- NEW: 4 Built-in Supplements component
- NEW: Editor Highlight system for text property
- NEW: \size[.] will decrease font size into scripts level
- CHANGE: Splitting of parser script for ease of customization
- CHANGE: Debug Information now hiding if there is no problem
- CHANGE: One hex letter \color code now follows modified 4-bit colorful index
- CHANGE: Vertical delimiter now be rotated if used as horizontal delimiter
- CHANGE: font size texture is limited to 1024, prevent memory crashes
- FIX: Eliminating use of System.Linq for all runtime scripts
- FIX: Optimized Parser, minimum usage for StringBuilder
- FIX: Nested script no longer need to braced separately
- FIX: Editor improvements for NGUI extension
- FIX: Unity 5.5 compatibility
- FIX: \color glitch when resizing UI
- FIX: symbol definition for \angle
- FIX: Horizontal Extension's width shows less than actual requirement
- FIX: Preference font preview sometimes not match with inspected font

2.6 – Aug 8, 2016: Parser Stability

- NEW: Parser now more tolerant to incomplete typos
- NEW: Braces after commands is now optional
- NEW: Font styling (Bold/Italic) included in custom font tags (\font)
- NEW: Accents can hold by more than one character
- NEW: Bump Lit shader
- NEW: Support for Horizontal Extension
- NEW: \math for turning off modified custom font tag
- NEW: offset control for \size
- NEW: Showcase example scene
- NEW: Added (Bonus) Editor pool check for checking pooled resources
- CHANGE: 2 user fonts now changed into more useful one
- CHANGE: behavior when imports configuration in XML files (and upgrading process)
- CHANGE: \not now have a very extensive customization
- CHANGE: Characters now used pixel-perfect font size according to their actual size
- CHANGE: Shader now wrapped in one .cginc file, improves shader readability
- FIX: Faster and less GC overhead when parsing string
- FIX: NGUI Extension Glitch 'IM HIT'
- FIX: \text and \font doesn't parsing backslashes
- FIX: radical top line floats in incorrect position
- FIX: issue with Unicode characters
- FIX: \font and \color isn’t able to wrap
- FIX: nested delimiter doesn't expand it's height
- FIX: Unicode characters still do incorrect glyph sizes
- FIX: Unicode not working in inside \font block
- FIX: Progress bar doesn’t update when Preference imports

2.5 – Jul 28, 2016: Interactive Link

- NEW: Official support for NGUI, included as an external .unitypackage file
- NEW: UV3 filling, unlocking many shader features and variants
- NEW: Gradient and Texture overlay shader for TEXDraw
- NEW: TEXLink component, put links over TEXDraw!
- NEW: Underline and overlined text style
- NEW: Color command variants (\clr and \mclr)
- NEW: \size command for having variant in sizes.
- CHANGE: Improved autofit mode
- CHANGE: straight lines now rendered last
- CHANGE: shader paths now detached from GUI\..
- CHANGE: Latin symbol now follow their override font, not the default math rule.
- FIX (critical): incorrect glyph sizes on large characters
- FIX: Improved performance, only few bytes GC inprints on repaint.
- FIX: Pixel-perfect behaviour for UI
- FIX: Pathfinding of Preference is now automated
- FIX: Compatibility for Webplayer

2.4 – Jun 20, 2016: Sprite Import

- NEW: Import sprites as grid-based characters
- NEW: HTML names for \color
- NEW: more scene example and improved stress test scene.
- CHANGE: Shaders now split into 4 passes
- CHANGE: Increasing font limit from 15 to 31 fonts.
- CHANGE: Tabbed Symbols tab on preference
- CHANGE: Removal of Shadow lit
- FIX: Improved Performance, more less GC overhead at render.

2.3 – Apr 20, 2016: Performance Upgrade

- NEW: Autowrap and Justify alignment
- NEW: Stress Test example scene
- NEW: Font Index selection in Components
- CHANGE: API now using Resource Pooling
- FIX: Performance problem with Garbage Collector

2.2 – Apr 13, 2016: Unicodes

- NEW: Unicode Support
- NEW: \color command for custom text colors.
- FIX (critical): Imported fonts showing only white boxes.
- FIX: Mobile Shader now using two passes instead of one

2.1 – Apr 4, 2016: Compabilities

- NEW: TEXDraw Lit Shader (with Shadows)
- FIX: Shader Compilation for PS 3.0/4.0 and Mobile
- FIX: Problem in UI Layout and their Functionality
- FIX: Unity 5.3 Compability

2.0 – Mar 29, 2016: Package Rewrite

- NEW: 15 fonts data included in Package
- NEW: Symbols catalog is increased with total of +600 symbols
- NEW: TEXDraw Preference Editor
- NEW: Dynamic Global Preference
- NEW: Ability to change/import custom Fonts
- NEW: Expandable Delimiters (previously just scale the character)
- NEW: Accent support
- NEW: Lines above characters (aka. Negations)
- NEW: TEXDraw 3D Component
- CHANGE: Internal scripts now released from DLL
- CHANGE: Assets now released from Resources
- CHANGE: Rewriting of Documentation
- CHANGE: Spaces now used instead of \w
- FIX: Unity 5.4 Compatibility
- FIX: Support for UI Effects
- FIX: Assets now loading much faster (serialized on build)

1.0 – Jan 9, 2016: First Release

License notices for Included Fonts

These 15 fonts, included in this package, is a copy from JsMath website. You can use and include these fonts in commercial and non-commercial builds and don't have to notice the final users about the source of the fonts.

Link to source font (JsMath): (Apache License)
http://www.math.union.edu/~dpvc/jsmath/download/jsMath-fonts.html
JsMath does modify the fonts from BaKoMa: (Distribution limits apply, see below)
https://www.ctan.org/tex-archive/fonts/cm/ps-type1/bakoma/
BaKoMa created these fonts by converting the original glyph data from AMS Fonts:
http://www.ams.org/publications/authors/tex/amsfonts
We checked every license requirements and you (as the user) can use these fonts according to this license (this license provides a clear and major agreement):

```
                    BaKoMa Fonts Licence
                    --------------------

  This licence covers two font packs (known as BaKoMa Fonts Colelction,
  which is available at `CTAN:fonts/cm/ps-type1/bakoma/'):

    1) BaKoMa-CM (1.1/12-Nov-94)
       Computer Modern Fonts in PostScript Type 1 and TrueType font formats.

    2) BaKoMa-AMS (1.2/19-Jan-95)
       AMS TeX fonts in PostScript Type 1 and TrueType font formats.

  Copyright (C) 1994, 1995, Basil K. Malyshev. All Rights Reserved.

  Permission to copy and distribute these fonts for any purpose is
  hereby granted without fee, provided that the above copyright notice,
  author statement and this permission notice appear in all copies of
  these fonts and related documentation.

  Permission to modify and distribute modified fonts for any purpose is
  hereby granted without fee, provided that the copyright notice,
  author statement, this permission notice and location of original
  fonts (http://www.ctan.org/tex-archive/fonts/cm/ps-type1/bakoma)
  appear in all copies of modified fonts and related documentation.

  Permission to use these fonts (embedding into PostScript, PDF, SVG
  and printing by using any software) is hereby granted without fee.
  It is not required to provide any notices about using these fonts.

Basil K. Malyshev
INSTITUTE FOR HIGH ENERGY PHYSICS
IHEP, OMVT
Moscow Region
142281 PROTVINO
RUSSIA
E-Mail:       bakoma@mail.ru
     or       malyshev@mail.ihep.ru
```

The point of this section is to make sure you are correctly taking the fact that **we do not owns even sell the fonts**. You can download and import these fonts from provided link above and get those +600 symbols without using this package with no problem (so we just provide an easy implementation with maximum benefits of using fonts above inside Unity Engine).

Guide to Write in TEXDraw (Runtime Script)

You can write a TEXDraw formula inside a script by modify the text property. However, some problem may occur in writing on a script (especially when dealing with backslashes). In this section we will provide a quick guide to write a TEXDraw formula inside a script efficiently.

As a first, you will know that this will generate a compiler-time error:

```
// Compiler-time Error
string formula = "Solve: \sin(30)+\root{\frac{5}{1}}";
```

This is because backslashes (in C#) is preserved as semantic character (for something like \n stand for new line, etc.). The solution for this is type a double backslashes so it will tell the compiler to write a single backslash:

```
//Correct approach
//Resulting "Solve: \sin(30)+\root{\frac{5}{1}}"
string formula = "Solve: \\sin(30)+\\root{\\frac{5}{1}}";
```

Look like simple, but if you write a lot of backslashes you will find this is just not efficient. A better solution is write a verbatim string literals (ie. Add @ before string) so the compiler just ignore any semantics.

```
//Better approach (same result)
string formula = @"Solve: \sin(30)+\root{\frac{5}{1}}";
```

For a plus note, usually for less experienced devs, want to put something in middle of string will have to close the string and add + in middle of it:

```
//Still Correct approach (same result)
int num1 = 30, num2 = 5, num3 = 1;
string formula = @"Solve: \sin(" + num1.ToString() +
        @")+\root{\frac{" + num2.ToString() + @"}{" + num3.ToString() + @"}}";
```

This is somewhat slower, and cases that similar like above should use string.Format instead:

```
//Better and Efficient Approach (again it's return the same result)
int num1 = 30, num2 = 5, num3 = 1;
string formula = string.Format(
        @"Solve: \sin({0})+\root{{\frac{{{1}}}{{{2}}}",
        num1, num2, num3);
```

See the example above, the number inside of braces will be replaced according to arguments order (ie. {0} to num1, {1} to num2, etc.).

(Note: when formatting string, the double braces {{ will treated as single brace {. This is needed to make it less confusing within the use argument number).

Another problem is when you want to type a new line, since the compiler ignores semantics in verbatim literals, it does also ignore \n (which stands for new line). The solution for this is add a new line string to our format (although may other solution exist):

```
//Better Multi-lined Approach (after Solve:, it gets a new line).
string formula = string.Format(
        @"Solve:{3} \sin({0})+\root{{\frac{{{1}}}{{{2}}}",
        num1, num2, num3, "\n");
```

Troubleshoot for Common Problems

I want to see the complete list of Commands.

Open script in `TEXDraw/Core/Parser/TexFormulaParser_Command.cs`, you'll see there is array of constant. There it is.

Expandable delimiters do not works.

Make sure the delimiter that you'd use is comes from **math**. The solution for this is either change font index to -1, or type `\math` surround it, or even type it symbol name directly, like `)` becomes `\rbrack`.

Best Fit mode is very expensive.

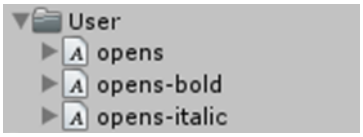
Remember that Best fit mode means they try to box in given size recursively over and over until the suitable size is found, so make sure the size doesn't start too big.

Font textures frequently cleared up/blank on Editor.

This issue comes from Unity Editor itself, you can fix it by hitting Ctrl/Cmd+Alt+R. Note that the issue has been resolved in newer version of unity.

How to make Bold/Italic styling works?

You can bring together the styling into the project, but it won't necessary to add in TEXDraw font list. TEXDraw follow Unity's way to provide styling, and you can do that by typing \opens[i]. If you had problem to referencing these fonts, this good QA might solves your problem.



Although these three is in valid folder, those bold/italic is actually ignored because of invalid name (contain '-')

Adding a Font, but nothing happen.

Make sure it is in appropriate folder then Hit the menu bar in `Tools > TEXDraw > Rebuild Font Data`.

Deleting a Font, now all selected font index in scene messed up?

That's how TEXDraw do it internally. We safe the 'Font Index' by index, that's mean the integer number of given list. If you delete just one font, the font index below it will stack up.. without reconfiguring any component who use that fonts (mean it'll mistarget). You can use the Tool in `Tools/TEXDraw/Font Swapper` tool to replace font index for all opened scenes.

Trying to Make NGUI integration works.

Just import package named `TEXDraw-NGUI Extension.unitypackage` from TEXDraw `GuideAndExtras` folder, then you can create one by navigate the menu to `NGUI/Create/TEXDraw`.

Trying to Make TextMeshPro integration works.

There are many steps to do this, though the whole process is just automated now:

1. Import TexDraw and TMP into Project
2. Extract shader package in `TEXDraw/GuideAndExtras/TEXDraw-TMP Shader Extensions.unitypackage`
3. Open player settings (`Edit > Project Settings > Player`) and in current platform, write `TEXDRAW_TMP` in "scripting define symbols", then hit enter. this action make scripts recompiling again.
4. Open TexDraw Preference (`Tools > TEXDraw > Show Preference`) and navigate to Symbol & Relation, then options, then Click `Render All Fonts`. This action will render all fonts (in background), so it might take awhile.

   (INFO: You can setup the resolution there, The default resolution is 512x512, but 256x512 is good enough for TEXDraw, and smaller is faster)

   (INFO: DO NOT recompile while rendering is in progress, otherwise it'll stop)
5. After rendering complete, Open Default Material (`Tools > TEXDraw > Select Default Material`) and set the shader to `TEXDraw/TextMeshPro/Distance Field`
6. You are ready. Just use standard `UI/TEXDraw` and start type right there.

Turning off TextMeshPro integration.

Just Clear `TEXDRAW_TMP` that you done in step 3, and then hit `Tools/TEXDraw/Rebuild Font Data`.

I have SDF Asset, why Font data still included on Build?

It is required to reference the font data in editor, but it is actually not used in Game Build. There is no way to dereference it when build, but you can uncheck Include Font Data in font importer option so it won't waste the build size.

Font Texture is frequently scrambled in the game.

We aware with this, and to prevent this behavior, make sure you are using the **same font size** across all TEXDraw in one scene. This will optimize the font rendering, and prevent fonts be scrambled by many large-sized characters that taken up most space in font textures.

Why Shadow and Outline do not work?

It is actually work, but TEXDraw doesn't catch it. As of V3.0, user/component itself have to call `SetSupplementDirty()` or disable-then-re-enable TEXDraw component so it can be recognized and used.

Created a TEXDraw Material, but don't want to input font textures manually.

Assign that material to TEXDraw component, then hit 'Fix' button beside it. Now their texture slot should filled properly.

Deleting unused fonts/Adding new font in TEXDraw font lists

All built-in font in `Fonts/User` or `Fonts/Sprites` are safe to delete (and please note that sample scene willn't work). But if you want to delete maths font, it is safe to delete anyway.

Non-Latin, Cyrillics, Arabics, and Unicode symbols doesn't appear correctly

Please import your own font that does support these characters (well, mostly it do). None of our built-in font does have unicode support since it 'light-weight' size. And also don't forget (but optional) to set-up the default typeface unicode to your font. Detail for that is here.

Why TEXDraw shaders uses too many batches?

This is needed to breakdown the sampler limit that internally Unity has. If you are using full 31 fonts, the shader job will be splitted into four, means there will be 4 batches for each component. But you shouldn't worry this much, since newer version of Unity will batch all of UI component that using same material, internally.

What is TexFontMetaData in Resources?

It is a place for putting individual font data like symbols or relations for a specific font. It is automatically created, but manually cleaned. Note that if font are deleted it'll not deleted, because in case if you changed your mind you won't sad of losing customized data.

Any chance for use Unity's new Texture Array?

We wondering that exactly, but that's feature it's not mobile friendly. But in other case, if this feature is used, there's a big advantage that the shader job can be much cheaper (down into one pass per component). If you need this feature, simply inform us.

I'm benchmarking, and it say potentially breakup game performance, it is?

Actually no. The thing that benchmarked is the time taken to completely rebuild TEXDraw component, and TEXDraw will do it only if necessary. So unless you made changes on every frame, your game is still smooth like usual.

I want a nasty surprise. Is there any internal/hidden feature?

Uhh... we have one. Uncomment line number 46 of file `Core/Box/Box.cs`.

Informing other bugs/Feature request/General Talks/Misspells

Please refer to forum, or leave us a reply.

About This Package

This package is provided by Wello Soft,

Check out other assets: http://u3d.as/cco

any question? contact us by email: wildanmubarok22@gmail.com

Forum thread available! Head to :

http://forum.unity3d.com/threads/released-texdraw-create-math-expressions-in-unity.379305/

If there's a good stuff, there's a good reason to share the joy with others

If there's a bad one... Then we fix that for a good reason

Don't forget to leave a review to this package =D