TEXDraw

LaTeX Graphic Mathematical Expressions Input for Unity

Documentation for V2.6

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

Table of Contents

Buyer's General Question to TEXDraw 5

What is TEXDraw? 5

How it does work and why it different? 5

What's the Key Difference between TEXDraw vs. Standard UI Text? 5

OK, how exactly it works? I mean how easily I could draw a single symbol? 5

Is your package depends on Unity's UI? What if I don't want to use it? 6

Inside of TEXDraw Package 7

The TEXDraw Component 7

Other TEXDraw Variant Components 8

Enumeration Choices 9

The TEX Link Component 10

TEX Preference 11

Font Collections 12

Shader Variants 13

Guide to Write in TEXDraw 14

An Introduction 14

The Power of Backslashes 15

Going Deep with Commands 15

Using Custom Font Asset 16

Writing Fractions 17

Writing Roots 17

Introduction to Superscript and Subscript 18

The Special usage of Large Operator Symbols 18

Using Expandable Delimiters 19

Custom Color 20

Custom Size 20

Clickable Link 20

Writing Matrix 21

Writing Table 21

Adding Overlayed Lines (Strikethrough) 22

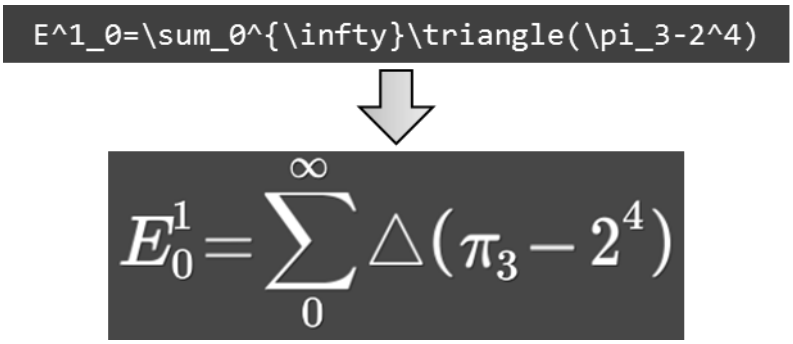
Using & Editing TEX Preference 23
Preamble 23
The Import-Export feature 23
How does TEX Preference saved and be included on build? 23
Tab 1: Symbol & Relation 24
Adding your own font/sprite to the TEXDraw font stacks 25
Configure How Textures are Imported 25
Using & Navigating through Character Map 25
Modify a Character Settings 26
Understanding Symbol Types 27
What is Character Hash, and what's the point of it? 27
The power of Delimiters: Making Character Relations 28
The Real Truth about TEXDraw's Way to Map a Font Characters 29
Tab 2: Real-Time Global configurations 29
Understanding & Using Global Configurations 30
Anatomy of a Character 30
Default Typefaces, what is it? 33
Configuring the Materials 33
Tab 3: Glue Management 34
Symbol Definition Cheatsheet 35
Greek Letters 35
Common Ordinary Symbol 35
Miscellaneous Symbol 36
Astronomical Symbols 36
Block Shapes 37
Geometrical Symbol Shapes 38
Boxed Binary Operators 39
Binary Operators 40
Relation Comparer 42
Miscellaneus Relations 43
Negated Relations 44

Primary Arrows 45
Compound Arrows 46
Expandable Delimiters 47
Open & Closing Delimiter 47
Large Operator 48
Accent 49
Preserved Characters 49
Appendix: A side note to the users 50
Proper Upgrade Step Procedure for further Version of TEXDraw 50
Legacy: Upgrading TEXDraw to 2.4 50
Legacy: Upgrading TEXDraw package from 1.0 to 2.0 in a project 51
TEXDraw Release Notes 53
License notices for Included Fonts 56
Guide to Write in TEXDraw (Runtime Script) 57
Troubleshoot for Common Problems 59
I don’t see where is TEXDraw support NGUI 59
Font Texture is frequently scrambled in the game 59
TEXPreference was fail to Import XML Data 59
Created a TEXDraw Material, but don’t want to input font textures manually 59
Every TEXDraw that generated runtime doesn’t work properly at build 59
Deleting unused fonts/Adding new font in TEXDraw font lists 59
Non-Latin, Cyrillics, Arabics, and Unicode symbols doesn’t appear correctly 59
Each TEXDraw component takes 4 batches at Gameplay which is… Expensive 59
Any chance for use Unity’s new Texture Array? 60
Informing other bugs/Feature request/General Talks 60
About This Package 60

Buyer's General Question to 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.



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:

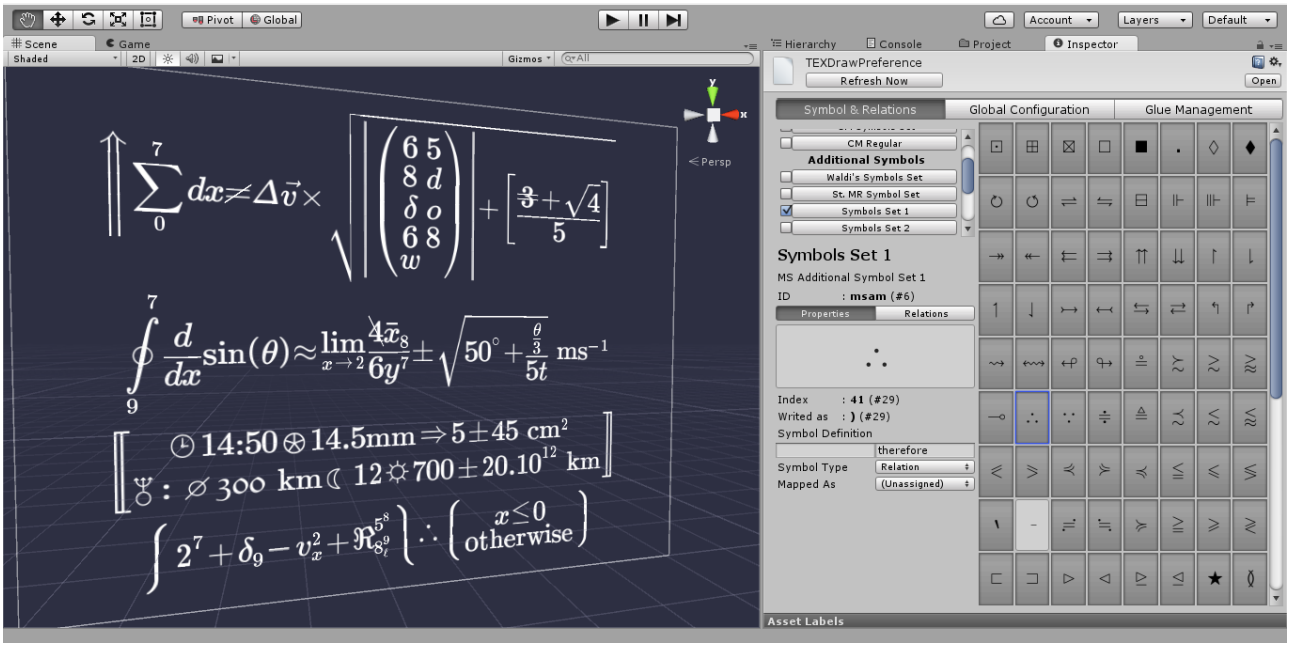
- LaTeX-Based, which is has more expanded features than standard html markup
- Math Expressions, including fractions, root, matrix, scripts, straight lines, etc.
- Resizable delimiters, which is fundamental for creating a complex math expressions.
- +600 symbols included, or built-your-own.
- Use UI-Based component, or Mesh-Based component
- Import and draw Sprites as you were importing fonts.
- Use of multiple fonts in single component
- Word wrap, including justify alignment
- Manage everythings, in single place (by editor preference)
- Customizable kernings, one setup for every characters
- No bundled DLLs, customize everything to suits your need.
- Compatible to All Platform, work fast on mobiles
- And many more...

OK, how exactly it works? I mean how easily I could draw a single symbol?

It's very easy! Any symbols that doesn't exist in your native keyboard can be substituted by a backslash followed by symbol name, for example like image above, if you type `\triangle` then

TEXDraw will easily understand you want a triangle shape symbol, so do the `\pi`, `\infty`, `\sum`, etc. Even if you are trying to draw some sprites, we don't treat them as `<quad>` tag, which used in many text generators, instead it will treated as symbol, so it would be very easy to create many of them at any time.

If you are curious, there are **more than 600** of possible (*defined*) symbols, that's includes Greek alphabets, geometrical shapes, binary operations, binary relations, and arrows. Not limited with these symbols, this package supports fractions, roots, matrix, larger operators, negated notations, extendable delimiters, customizable kerning and gaps, and many more. With the power of custom editors, everything would be easy and you will easily able to find a symbol that you were looking for.



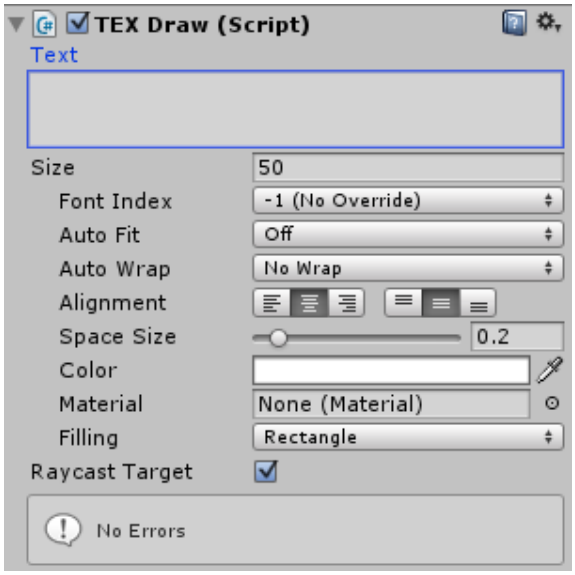
With the Power of custom editor, searching specific symbol can be done in efficient time.

Is your package depends on Unity's UI? What if I don't want to use it?

No Problem! We have an alternative version of TEXDraw which use the classic `MeshFilter` and `MeshRenderer`, it's called as `TEXDraw 3D`. Of course by using this component, you will lose the benefit from Unity's UI System (like defined rectangle bounds, pixels per unit, UI Effect, etc.). But, if you use NGUI for UI system, then good news! We have the NGUI extension for that,

Inside of TEXDraw Package

The TEXDraw Component



Once you have import the package, you can add TEXDraw to your scene by navigate to `GameObject > UI > TEXDraw UI`, or if you want, you can create an empty Game Object inside UI Canvas, and then attach TEXDraw component located in `TEXDraw > TEXDraw UI`. This component will handle rest of rendering process, so you can just type the formula in the **Text** property, and the result will displayed in your scene.

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.
This property support multiline, see here for practical guide to write (and here for runtime script).

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 Runtime script, you enter this as integer value, each number are index fonts that registered in the Font Stack.

Auto Fit How final graphic scaled when it's render is out of the rectangle bound
Turn off this check box if you want to preserve the graphic size even it's out of rectangle box.

Auto Wrap Auto wrap mode
Set text wrapping mode if text are wider than the component rectangle itself horizontally.

Alignment The horizontal and vertical alignment of the text.
In Debug Mode, this property is actually a normalized (ranging from 0 to 1) Vector2 value, with respect to Unity's Axis Direction, for example, value {1,1} means top-right alignment.

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

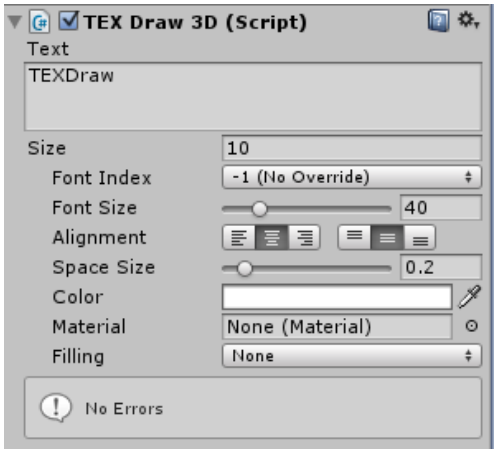
Color The main color for generated graphics
Use \color if you want to write specific color

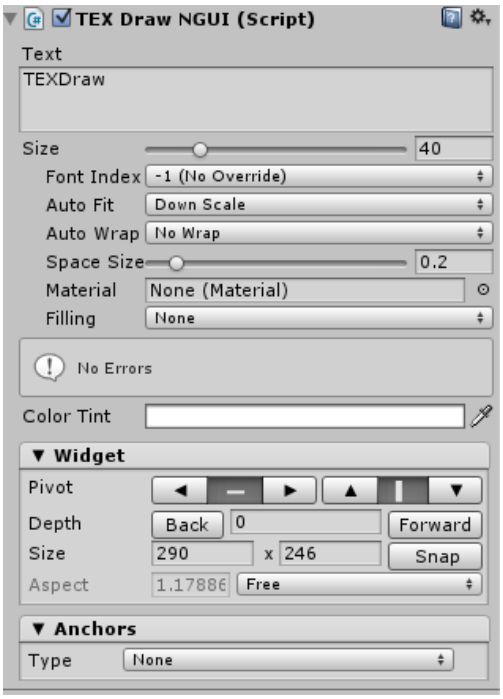
Material Assign a Custom Material for This component
If none assigned, the default material (from TEX Preference) will be used for rendering

Filling Optional options for filling UV3 data
Used in conjunction with custom Material, choose the way when font characters are overlay-ed with some graphics

---

Below of these properties there is a box called **Message Box**. This message box contains information that shows when you incorrectly typing formulas in Text properties. If you want to get this message at runtime then you just simply get a variable called `debugReport`. `debugReport` always return empty string when there is no error happened.





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`.

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.

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	Scaling 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

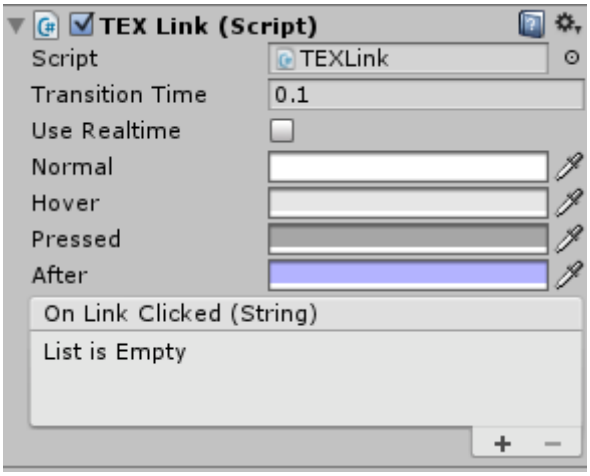
Auto Wrap is used for what happen to each line of generated text when it's horizontal line is beyond than given rectangle width. Auto wrap is always be calculated first before Auto Fit.

NoWrap	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

Auto Fill is useful only if you use a custom material which requires UV3 vectors like Gradient and Texture Overlay shaders. This is about how do texts are UV-mapped, in automatic-way.

None	Don't attempt to fill any UV3 values (make things 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.

The TEX Link Component



TEX Link is a new feature that perform the handling for links that created inside a TEXDraw package. 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 for TEXDraw 3D.

To make this component works, you need to create at least one `\link{}` command to the text, and only can be tested if you are still running the game. TEXLink behavior is works for Mouse and Touch screen (Keyboard is yet supported).

Transition Time How long the time it taken to change a color.
Zero to make it instant.

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

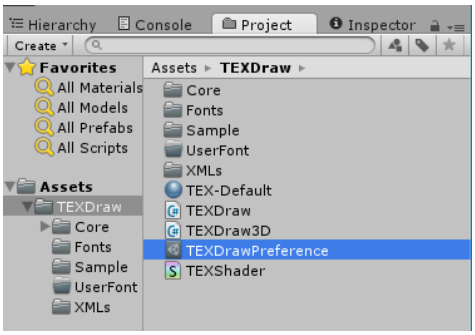
Hover Color tint when the mouse just above the link
This only happens on desktop where mouse devices are exist.

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 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



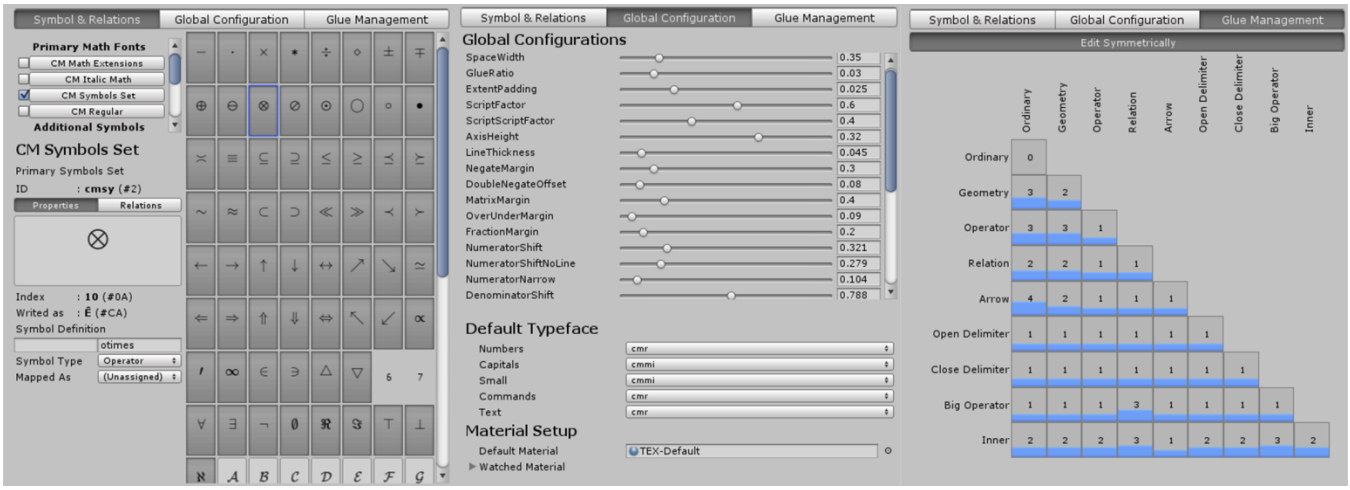
The preference here located inside of TEXDraw root folder as an arbitrary asset.

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 `Edit > Project Settings > TEXDraw Preference`.

TEXDraw Preference holds shared information across one project and saved as arbitrary asset located in: `Assets/TEXDraw/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 created, and any last un-exported preferences will lose and reverted according to what's included in XML Files (which holds the original preference data).

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



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

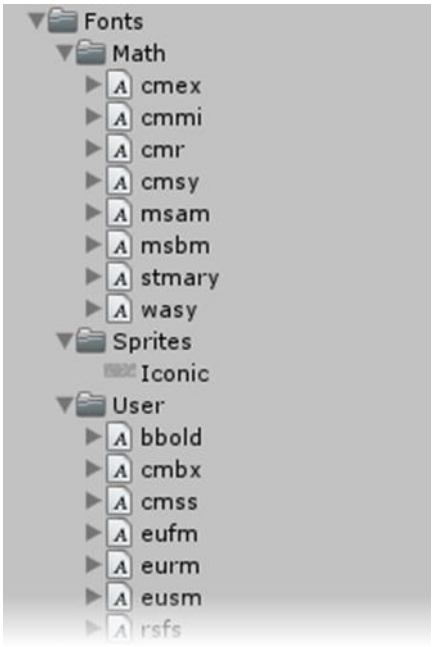
A place for shared (static-like) properties for controlling character sizes, margin, fraction gaps, script drops, etc.

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 are originated (it's not made entirely from us!) from collections of AMS Fonts, and it's now available as TTF Format thanks to `JsMath` team (though `JsMath` itself were processes the fonts from `BaKoMa`). Though we include these fonts, the license still holds back to font creator itself, but you are legally can use and include these fonts to your games both commercial and non-commercial builds. Read more about that here.

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

Math Every mathematical character & symbols will be placed here
It's built-in, so becareful to not remove/modify the content and placing other fonts here.

User Any True-Type and Open-Type fonts placed here
Put any fonts here and it will registered and used for TEXDraw components

Sprites Any Sprite Textures placed here
If you want to register a texture instead, place them here and it will treated as you are importing fonts to TEXDraw Font Stack.

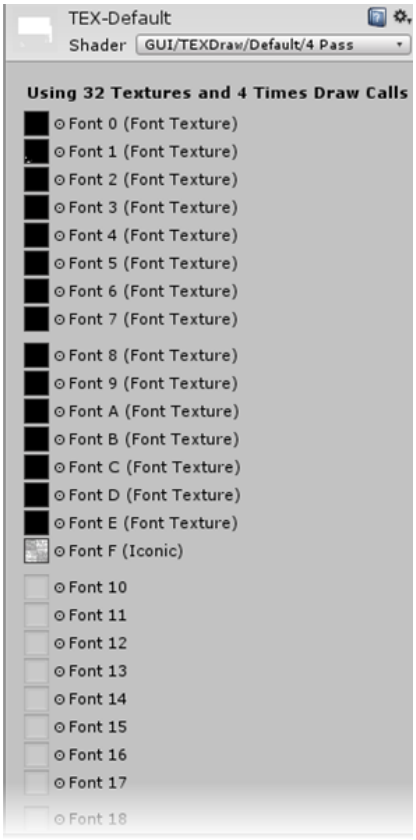
You can place any font/texture to User/Sprites folder, then tell the preference to *re-import* so it can be registered and used in TEXDraw components (detail instruction is in here).

Besides the flexibility when importing any font/texture to these folder. Always note that total fonts combined in Math+User+Sprites has to be **less than 32**. More or equal than 32, then one or more fonts is ignored, won't be used in TEXDraw components and can't be included in build.

For practical purposes, we included some font and texture samples so you can see how things actually working. So it's up to you whether you want to use and keep our custom fonts provided inside the User/Sprites folder, or if want to use your own fonts instead, then you may delete it. It's always safe to remove these fonts as long as you not using it.

In Math fonts, there are 8 fonts that needed for creating math expression inside unity, each of them holds ±50 KB data. It's still possible but not recommended to remove the entire math fonts. It may still safe to remove symbol extensions font (that's it: *msam*, *msbm, stmary, wasy*), but by removing the rest of the fonts (that's it: *cmex*, *cmmi, cmr, cmsy*), some math functionality will not available, even if you still use it, it will fail and throws an error on console. So please only delete the math Folder only if you really doesn't use any math expressions in your project.

Shader Variants



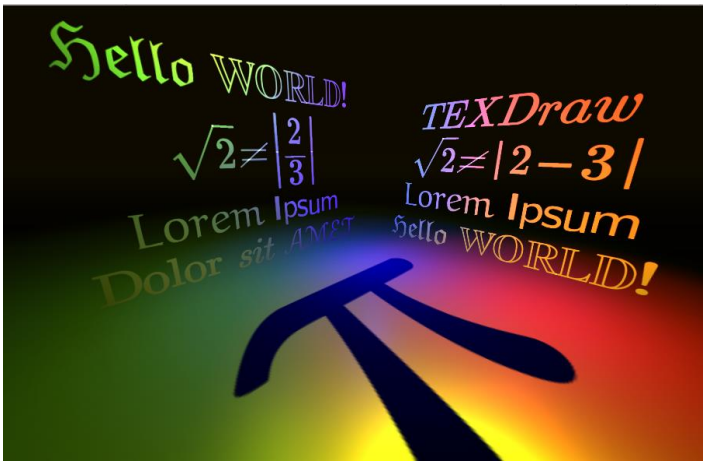
Shader, which is a main part of renderings, included in here is slightly different than other majority of how shader used in most cases. Because we want to renders all fonts, including sprites in one time, we had to write a custom shader that can only be used in TEXDraw component. Here in this package, we built two types of shader:

First, the default shader, located in `GUI/TEXDraw/Default`, is the shader that you'll use in most cases. It's non-lit, cheap to render and comes with stencil support. This shader is used in TEXDraw main material located in `TEXDraw/TEX-Default.mat`. As you see in the shader properties, there's a bunch of texture properties. It filled based on what ordered in the font collections. You don't have to fill them manually, because it automatically be filled within the TEXDraw preference (as described here).

Previously, as we said earlier that we had imposed a limit to 31 fonts, this also means that the shader itself should can hold up to 31 font textures, which is beyond to unity's shader capatibilies (which only holds a maximum of 16 samplers per single batch). To break up this limitation, we chunk up (splitting) the shader's job to multiple passes. In the default shader, we write 4 Passes to render all 31 textures, so each of shader pass have holds 7-8 textures only. (Note: we choose to split the passes to four instead of two, since most old graphic card/mobiles can only have a maximum of 8 textures per passes)

Enough said for passes topic. But here's another problem: since we draw 4 passes for each component, it would be noticeably expensive when used many times, anyway, of course some of you aren't use all of texture slots available. This is why we put some variants over the default shader; these variants ranging from 1 until 4 Passes. Each passes can holds up to 8 textures, so, by using 2 pass variant, the maximum fonts that shader can handle is 2x8=16, etc.

It's your choice about how much fonts that you use. It's always better to only include fonts that really you'd use for your project. After deciding which font do you use, for optimal performance, don't forget to adjust how much passes do you use.



Other than the default shader, there's a lit variant, which is slight expensive but offers an ability to interact with lights.

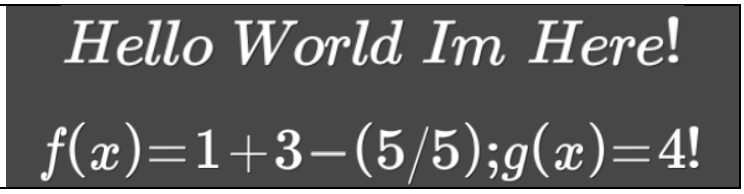
Guide to Write in TEXDraw

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 support multi-lines.

```
Hello World Im Here!

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

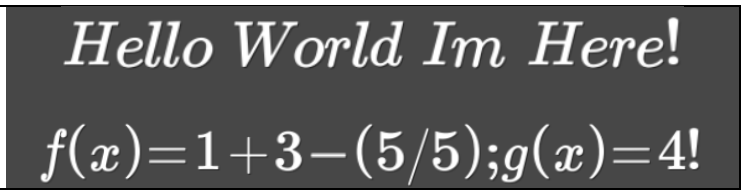


Please note that it doesn't support Tab spaces, some characters also can't be typed and used directly, like `{}`, `\`, `^`, and `_`, because it's used for internal purposes. If you want to write them though, type a backslash \ before it. For example, \} and \^.

*) Unicode is characters that doesn't exist in common keyboard. Characters like é or á is common example. Unicode character is supported **if font that you've selected does support it too.** All of the fonts that provided in this package does **not** have unicode character support**. So, it will results in Unity's default font: Arial. To make the unicode charater works properly, you can import your own font asset into TEXDraw preference.

```
(Font set to -1 (math))
 Hello World Im Here!

(Font set to 15 (opens))
 f(x)=1+3-(5/5);g(x)=4!
```



**) All except opens (which is a shortname for Open Sans). You can use this one for quick demostration for unicode characters.

The Power of Backslashes …

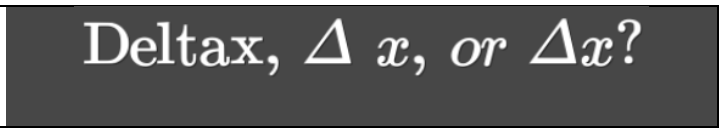
The major power of using this package is actually coming from the use of backslash. As a kick starter, type a backslash followed by symbol name. Symbols like `\alpha`, `\triangle`, `\permil`, `\ell`, and `\rightarrow` is a common one. Also please note that all symbol name is case-sensitive.

```
\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}?
```



Going Deep with Commands

Beside symbols, there are more things that you can do with commands. In short, commands is preserved symbol names that, instead of showing symbols, they'll gain you an access to a number of mathematical styles that used commonly, and other selective text customization snippets that can customize any character, easily.

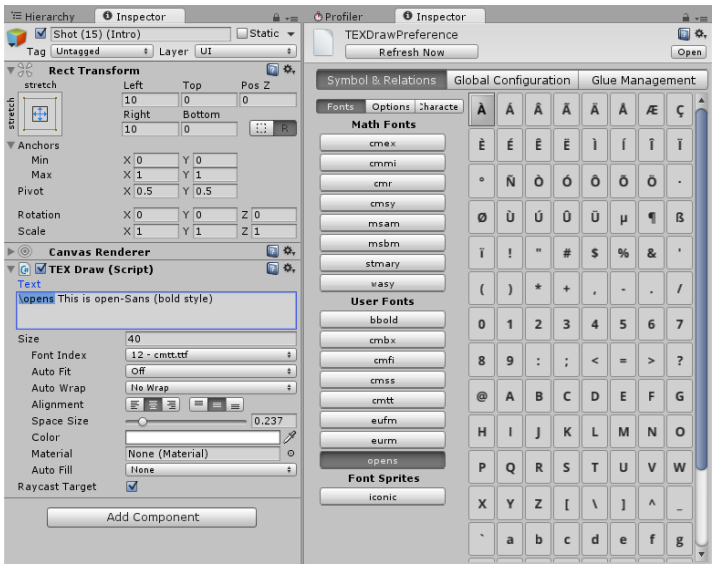
For list of possibe commands, are in the variable within TexFormulaParser.cs, or …

```
private static string[] commands = new string[]
{
    "root", "vmatrix", "matrix", "text", "mclr", "clr", "color", "size", "not",
    "nnot", "hnot", "dnot", "unot", "onot", "hold", "vhold", "bhold", "link",
    "ulink", "frac", "nfrac", "lfrac", "rfrac", "nlfrac", "nrfrac", "under", "over"
};
```

Every commands have braces next to it, like \root{5}, means $\sqrt{5}$, and outside of braces will remains unaffected. In futher reading, we'll show you how to use each commands, with quick examples…

Using Custom Font Asset

As you see in the TEXDraw Preference, there's a list of font assets that you can preview in the editor. The name of these fonts (file name) is what you'll type if you want to use that font.



For the demonstration, type backslash followed by a font name, after that surrounded by braces. There's also an optional options for font styling (bold/italic) by group brackets. To summarize, The formal structure of using a custom font would be similar to:

`\<fontname>[style]{text you want input to}`

Where `<fontname>` is the file name (accourding to the list) of font that you'll use, `[style]` means what font style will be use, the options are `[b]` (bold), `[i]` (italic), `[bi]` (bold-italic, in that order), `[]` (normal style), or no at all (styles remain unchanged). Remember that like unicodes, font styles is ideal if you have your own font imported.

```
\opens[i] Open Sans italic

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

Open Sans italic

Double Inside but still double 'till back again

Since V2.6, all braces is now 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)
```

Solve this test: $\sin(x)+cos(x)$

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

Writing Fractions

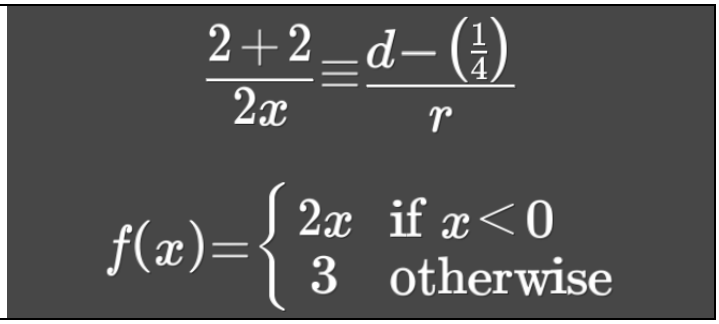
Fractions is common in math, they have a numerator and denominator. It is possible to write them in TEXDraw, to do that, we need to follow on this rule:

`\[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}

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



Now 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

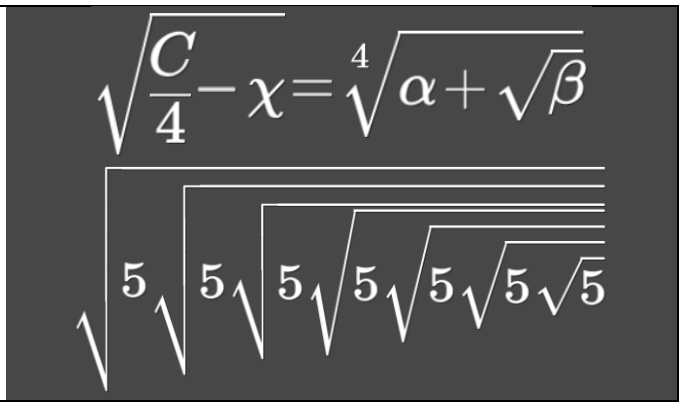
Root is another common math operation in everyday life. It's consisting of expandable surd (radical) sign (√) with a thick line on the root base. Writing Roots is easy, by follow on this format:

`\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, while it is still possible to get a nested root.

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

\root 5\root 5\root 5\root 5\root 5\root 5\root 5
```

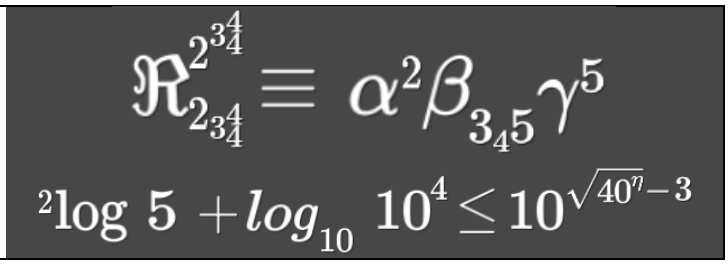


Introduction to Superscript and Subscript

Scripts, is commonly used in many scientific games and simulation apps. It's easy to create one, to create one type ^ after a character to make the next character be superscripted, or _ to make a subscript one. To create both simply type the both character no matter the order. And remember to get these characters surrounded by braces if they are symbols or compounded formula.

```
\Re^{2^{3^4_4}}_{2_{3_4^4}}\equiv
  \alpha^2\beta_{3_45}\gamma^5

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



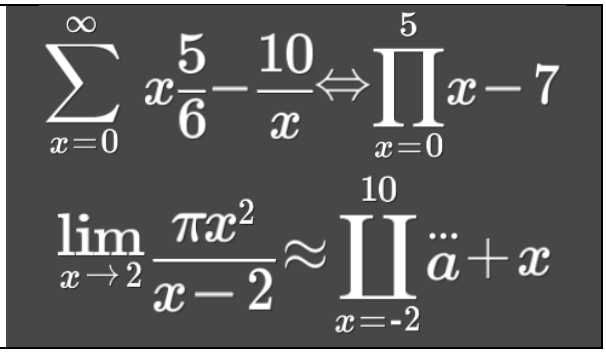
Take a look at the first example, as you see there's a size decreasing in every scripts level, but it stopped at third level, after that It's just shift their vertical position. If you haven't understood this, this is the maximum depth level of scripts size that are possible to make it in a smaller size (which is, three). After the third script, it won't go smaller anymore.

The Special usage of Large Operator Symbols

Now you know how a script works, but there's one exception: if you type a symbol which has type of Large Operator then it's simply goes to over/under it. The example of this feature is like Sum or Integral Operators. This feature also can be applied to other non-big operator type by putting a double script `^^` or `__` instead of one.

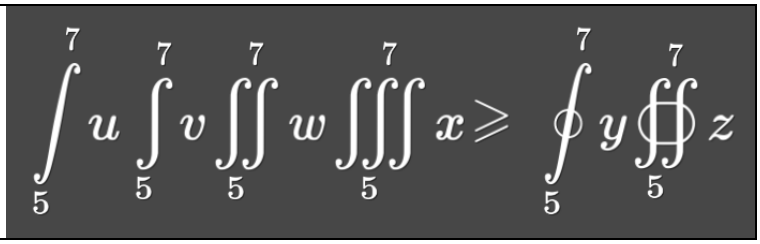
```
\sum^{\infty}_{x=0} x\frac{5}{6}-
\frac{10}{x}\Leftrightarrow\prod^5_{x=0
}x-7

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



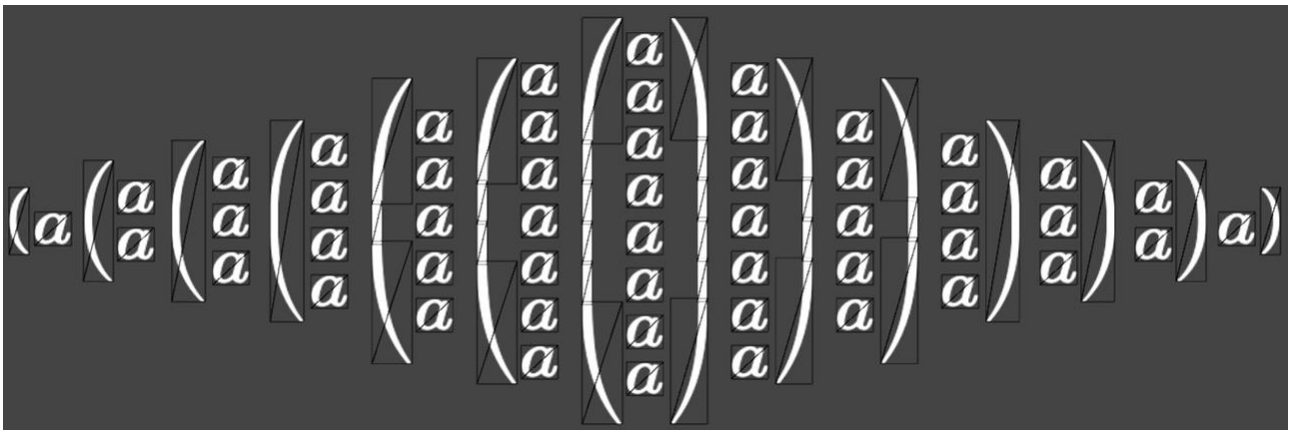
Unfortunately, custom alignment for integrals doesn't supported - however, we have a tricky solution for this: add some invisible spaces!

```
 \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.

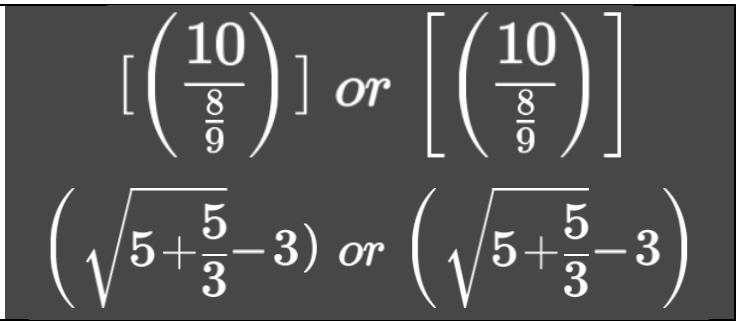


Expandable delimiters, does expand higher until sufficient height achieved, without suffering from stretching.

Curious? Try it here:
(a(\matrix{a|a}(\matrix{a|a|a}(\matrix{a|a|a|a}(\matrix{a|a|a|a|a}(\matrix{a|a|a|a|a|a}(\matrix{a|a|a|a|a|a|a}(\matrix{a|a|a|a|a|a|a|a})\matrix{a|a|a|a|a|a|a})\matrix{a|a|a|a|a|a})\matrix{a|a|a|a|a})\matrix{a|a|a|a})\matrix{a|a|a})\matrix{a|a})a)

The implementation of this behavior is effortless; you can do that by type some tall formula and then surround them by delimiters, sometimes grouping (by braces {}) also needed. To make things clear, take look at these examples:

```
[(\frac{10}{\frac{8}{9}})] or
[{(\frac{10}{\frac{8}{9}})}]
```



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


So, the point is, in some circumstances, groupings will make the parser understand which is actually be inside of delimiter so they can determine which is tallest and make the delimiter around it to match the height with that selected tallest character. If none is grouped, then the delimiters simply match with next or previous character's height.

Custom Color

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

New in V2.2, you can type \color followed by hex code and text inside of braces to give a custom color in the text. The hex code can be either 3 or 6 digit depening on what you like, and optionally with a numbersign (like #aaa) if you prefer. You also can use HTML color name, like "yellow", "magenta", "blue", "cyan", etc. Look at the rich text manual for complete HTML color list.

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

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}`

Also new in V2.5, you can type \size followed by decimal number to specify the multiplier of resulted size of selected character. The options for size is ratio and offset. The ratio is scale (normalized), so if you enter scale to 1 it will unaffected. There's also an offset (optional) to shift their position upward if resizing doesn't looking good.

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

Clickable Link

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

Introduced in V2.5, this command is useless if you don't add a component called TEXLink besides TEXDraw. This command will make selected (the one inside braces) be marked to be clickable and able to respond user feedbacks by changing (tint) it's color based on what TEXLink color configuration says. Later if user clicks on it, `OnLinkClicked(string)` will be triggered with setting the string parameter to what says on `[eventname]` (or `{base}` if there's no eventname specified). Look at the link scene example to see how it works on gameplay. There also a variant \ulink if you want to give an underline styling for it.

Writing Matrix

Matrix is a bunch of formulas that grouped in specific column and row, and sometimes surrounded by delimiters. You can write that according to this formula:

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

Don't understand? To make it work, Type `\matrix` then "some formula" surrounded by braces. Now, that "some formula" is what we talking here, you can type `&` after some character as a sign to put next character in a new column, and type `|` after some column as a sign to put next character in a new row (with column index started back as 1). You can make any number of column and row as much as you want.

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

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

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

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

Additionally, if you want to write matrix column-by-column instead of row-by-row, you can type `\vmatrix{…}`. For example like `\matrix{a&b|c&d}` will give an equal display 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&\Name|001&A&John|002&B&Skeet|003&C&Brow
```

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

Adding Overlayed Lines (Strikethrough)

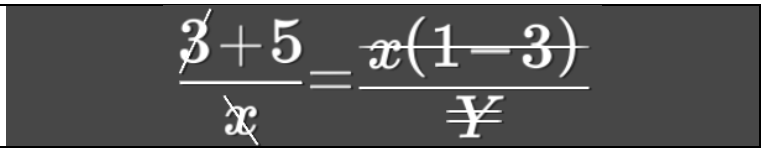
Sometimes, in math, you need a line that crosses some formula either horizontal or diagonally. Here, in TEXDraw, it's possible and simple (and lots of customization option) to implement this behavior, according to this:

```
\[v|n|h|d|u|o]not[offset1-offset2]{base}
```

Means there are many options available,

\not ab		\hnot ab		\unot ab	
\nnot ab		\dnot ab		\onot ab	

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

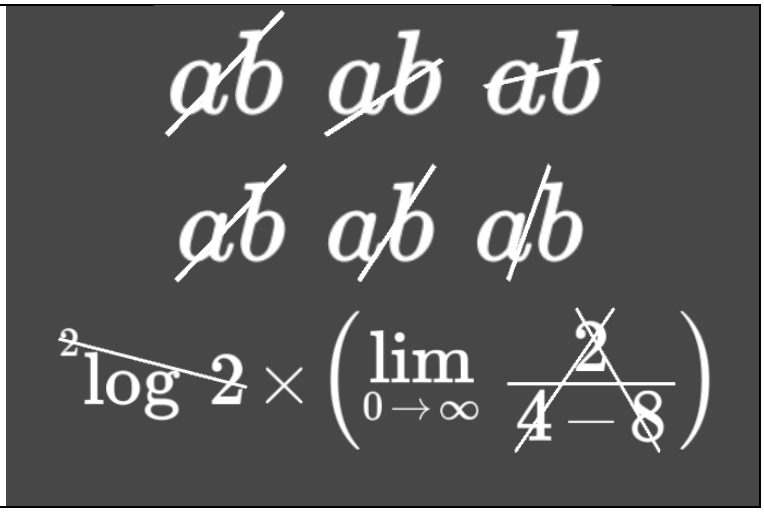


There's also parameters `[offset1-offset2],` which specific for each option. To make this thing much simpler to understood, take a look for these examples,

```
\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\infty y} {\vnot[.1-.8]\vnnot[.9-.2]{\frac{2}{4-8}}})
```

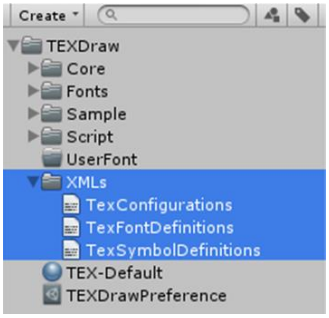


Using & Editing TEX Preference

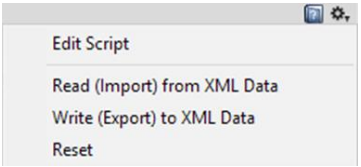
Preamble

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 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)

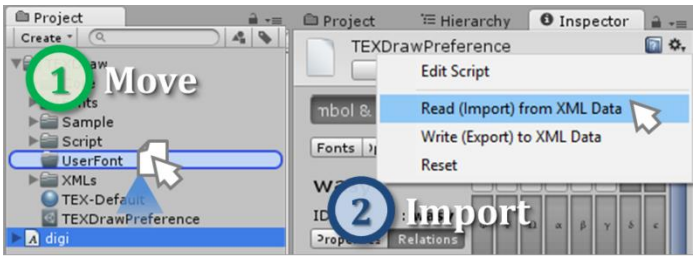
3 Per-Character Configuration
Selected character can be configured here, including it's symbol definition, type, and relation to other characters.

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

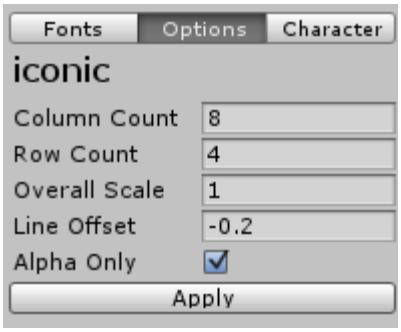
In short, this tab is used to manage how does fonts and textures are imported, including it's each character configuration. You can define your own symbol definition, or configure how it behaves with another character. All of features included inside this tab will be discussed futher later.

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



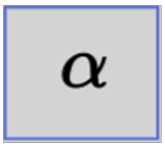
Textures are imported as it is contains bunch of sprites that have an equal dimension (grid-style). You can tell to the importer about how much column and row it has by set-up the column and row count. Optionally, overall scale for how large the texture size, and line offset for adjusting the "vertical 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

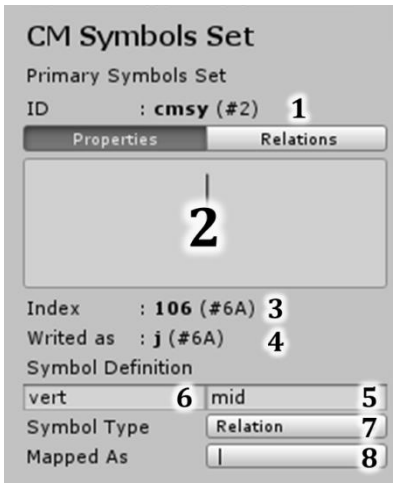


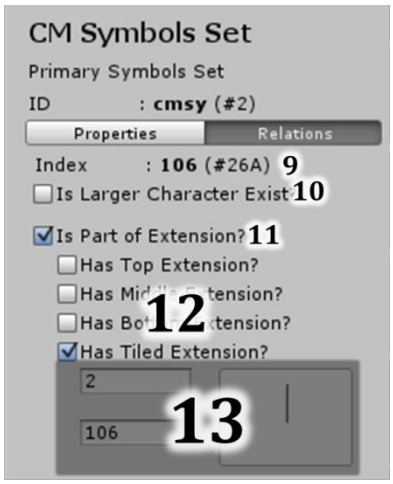
Char is Related
The character doesn't defined but it has relationship with other character

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.

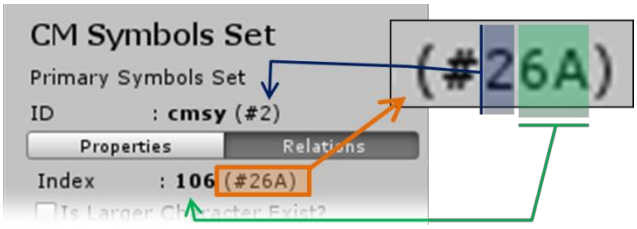
Please note that for custom font you should leave the “Mapped as” option unassigned. This option helps the parser guess common symbol that exist on your keyboard (example like | means \vert; + for \plus; ! for \faculty, etc.). Since all character has been preserved in math fonts, there’s no reason to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

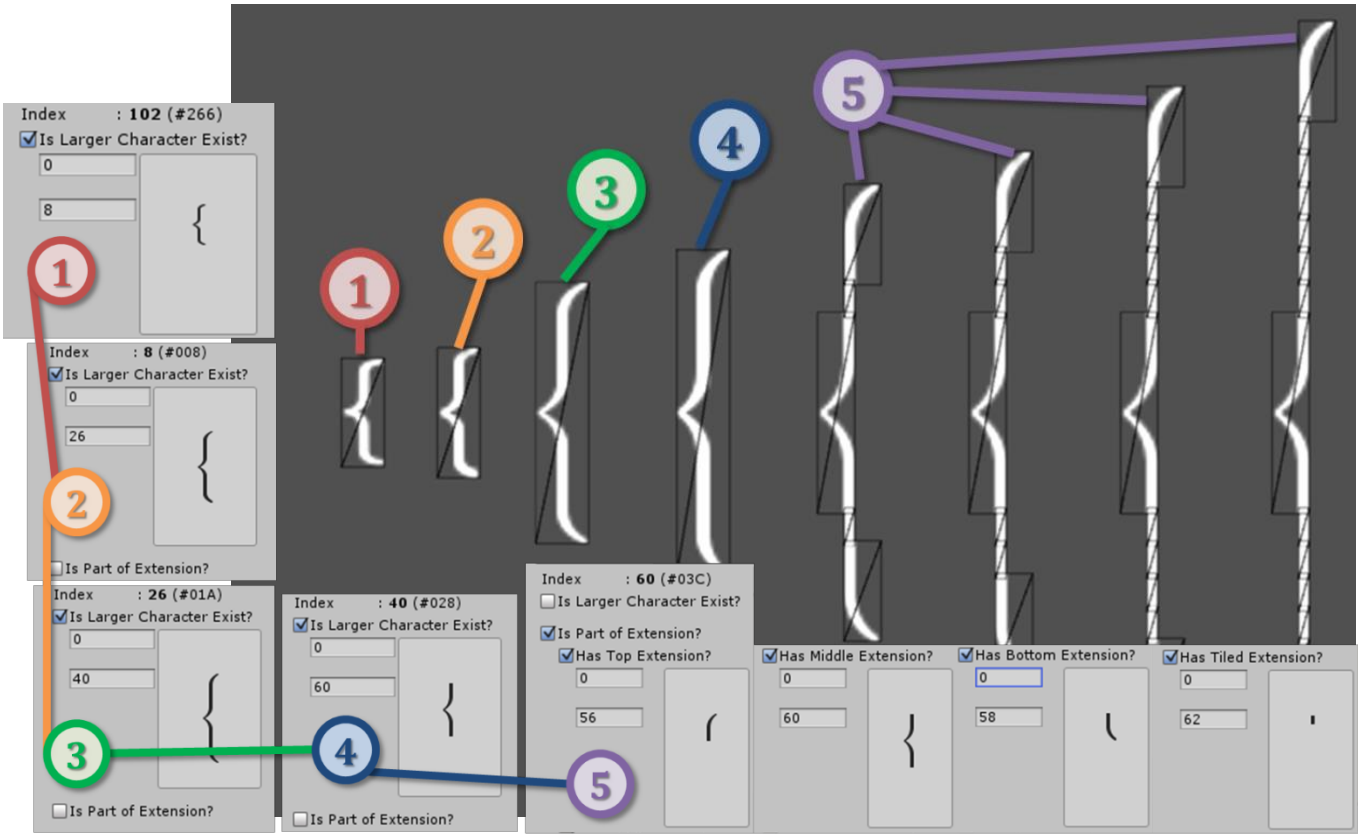


Character hash is a number that given for each character registered in TEXDraw font stack. A hash number that given on a character is unique among the rest. It's easy to read it in Hex Format. For example like in the screenshot in the right, #26A, means the character located in font with index of 2 (cmr), 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.

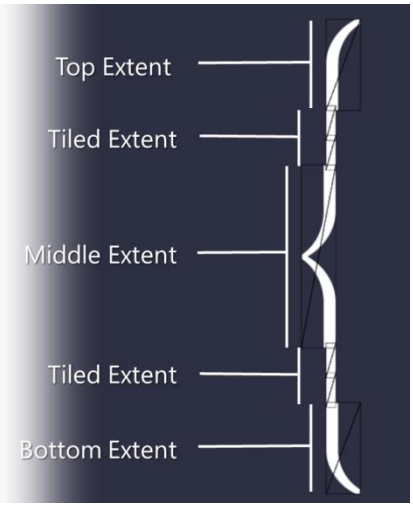
Please note that for validity of character hash, in hex display, the first two digits should be ranging from #0 to #7F (0 to 127), while third digit should be ranging from #0 to #1E (0 to 31), or in short, maximum possible value of a hash is #1E7F.

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!

The Real Truth about TEXDraw's Way to Map a Font Characters

Before you head-on to import your customized font to TEXDraw font stack, please take account to table below, it's contains a map sheet of TEX character (#00 to #7F) to actual character in font file:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
10	B0	D1	D2	D3	D4	D5	D6	B7	D8	D9	DA	DB	DC	B5	B6	DF
20	EF	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	30	31	32	33									3C	3D	3E	3F
40																
50	50															5F
60	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	FF

ASCII Printable Character Area

The font index in this area has the same index within the actual ASCII Font file.

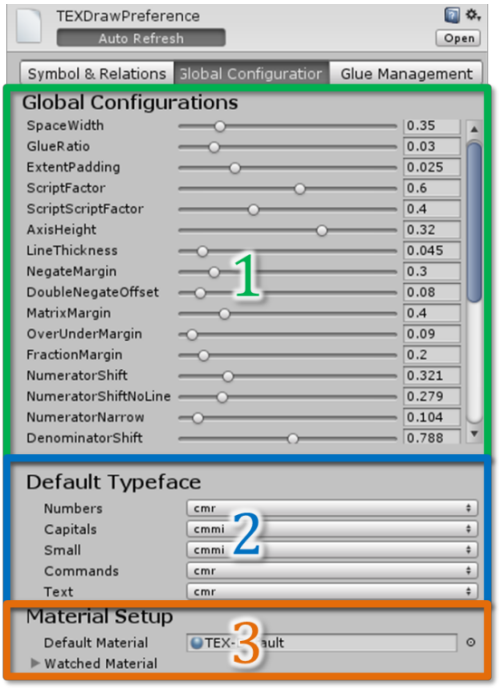
So, things like \text{} will working as long as it's only contains char from this area.

Character Index (0 – 7F) Actual Font Index (Writed As)

As you can see in the character map above, any character index ranging from 0 to 32 (#0 to #20) and 7F will have a different actual character index, which is different from actual character index (in standard font file, it's actually a non-printable (blank spaces)ASCII characters), meaning it won't be available directly (like typing from \text{}, etc.), instead, it mapped through any other character beyond #7F, that are printable. (side note: ASCII character that are printable has a total of 128 characters, which we call as Latin-1 Character.)

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

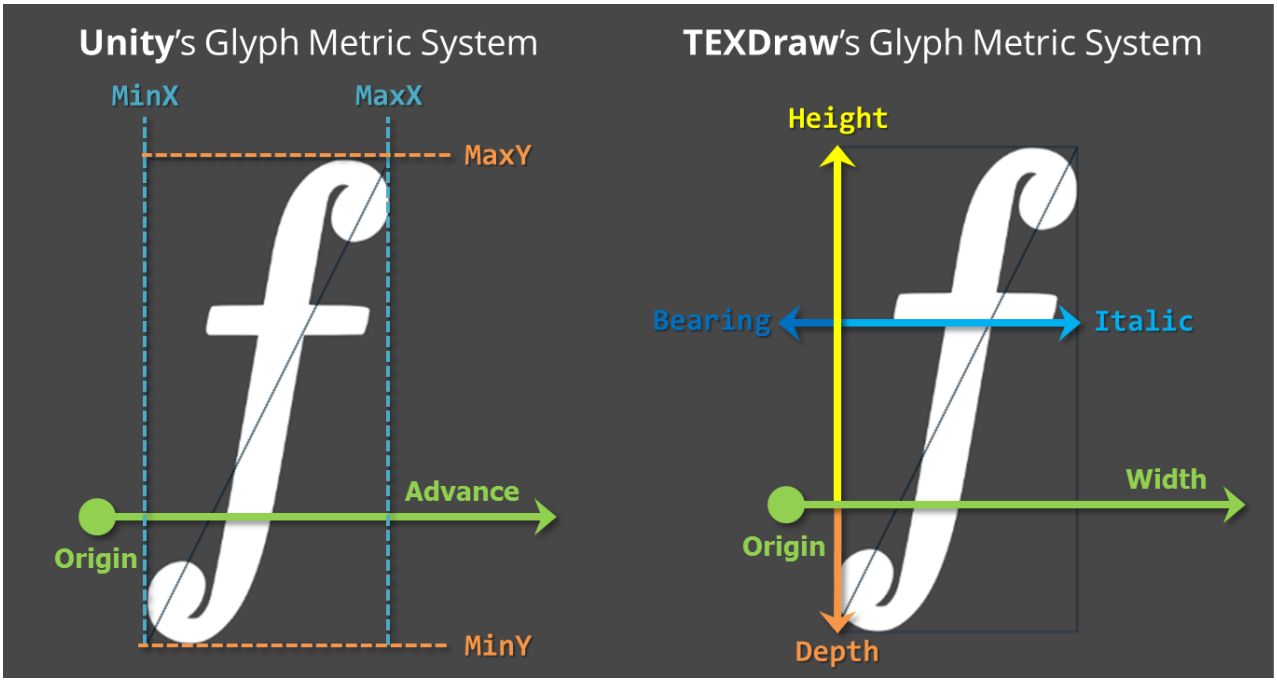
3 Material Setup
Determine what's material be used as default rendering

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:



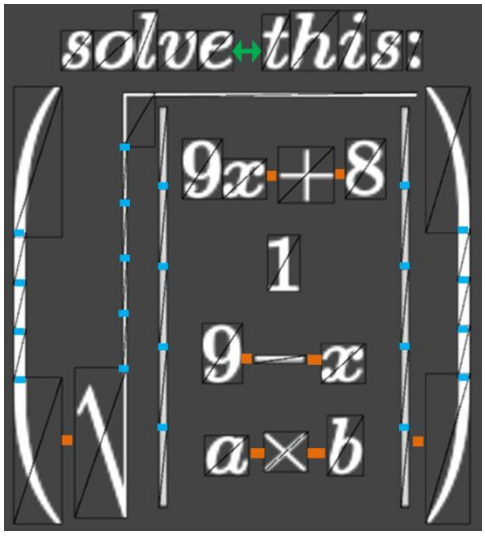
A character has a baseline (marked as green line), top and low bound. The image above will help you to understand each config which we will discuss below:
(We provide each config explanation with relevant image and indicator colors)

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

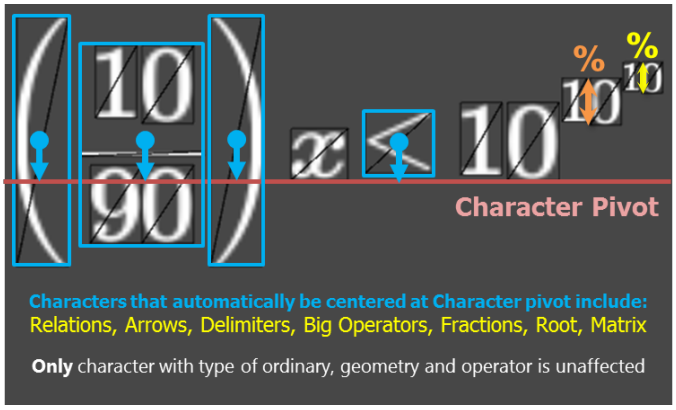
Size Ratio of Nested Script

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

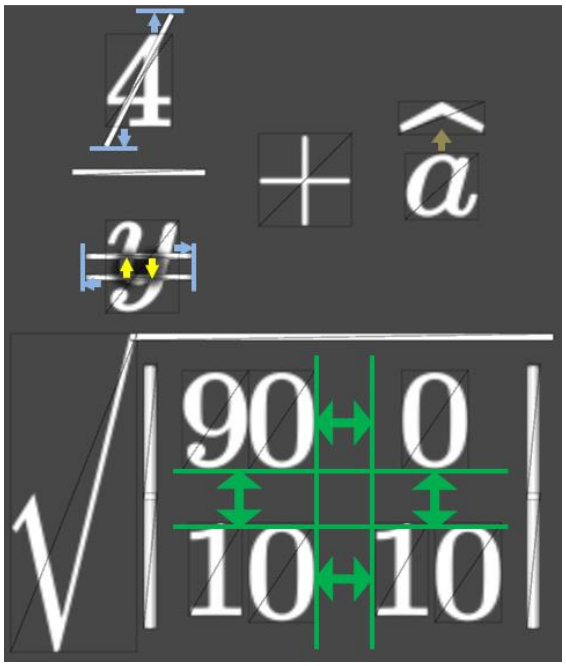
Matrix child-by-child margin

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

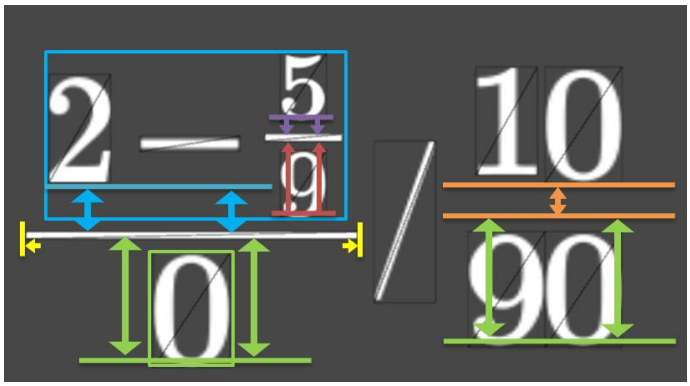
Standard Denominator Margin

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

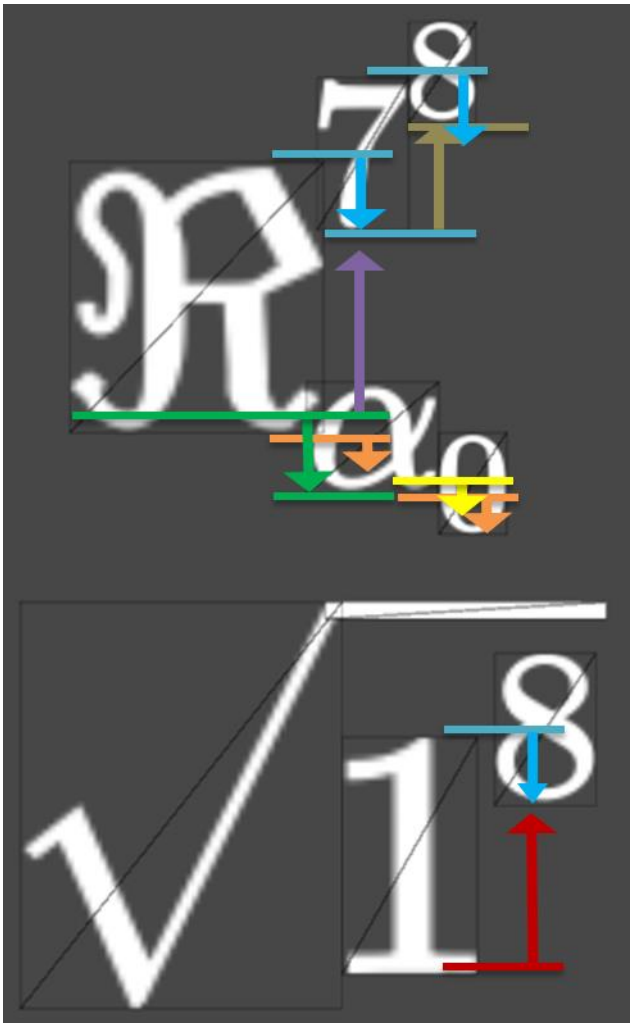
Subscript Minimum Drop (No Supscript Above)

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

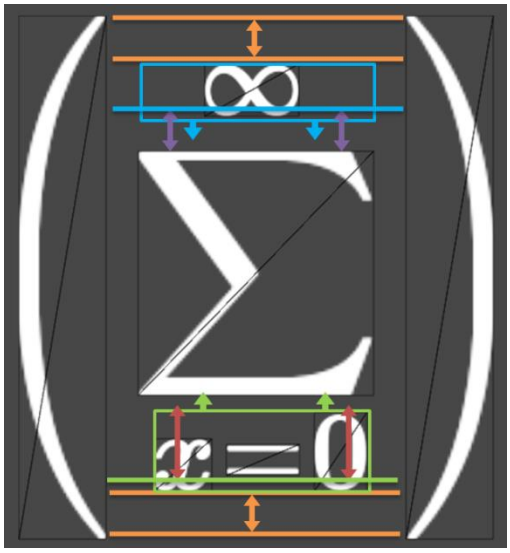
Large Operator Low Shift

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



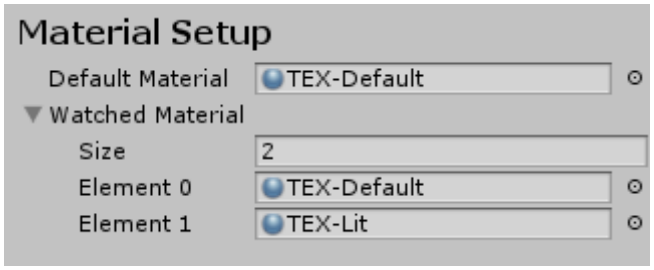
Default Typefaces, what is it?

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).

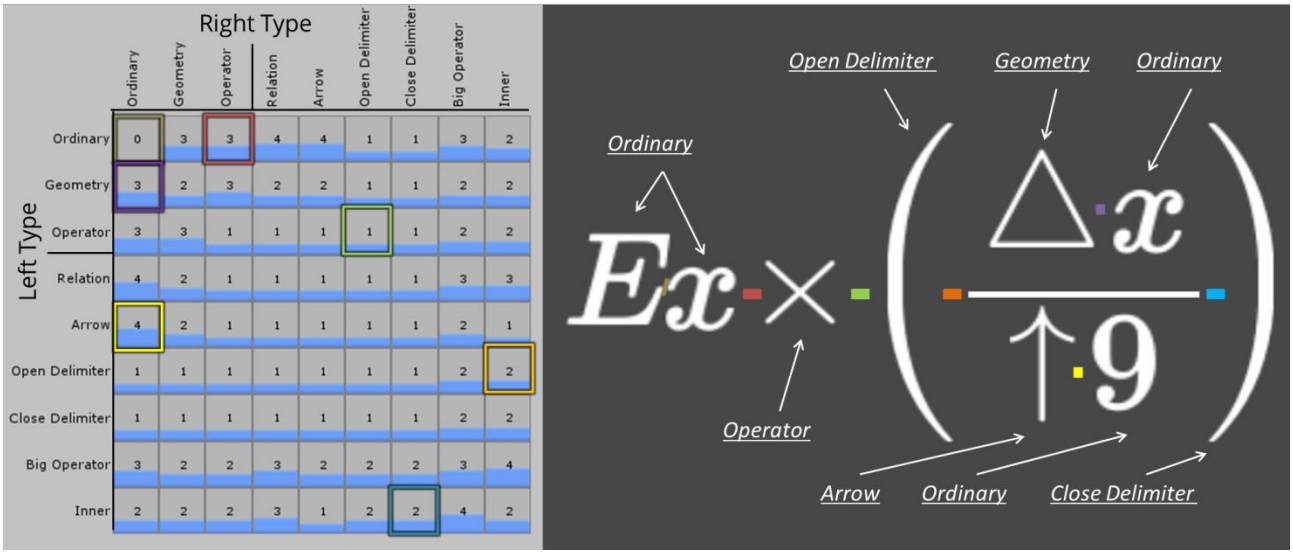
Configuring the Materials



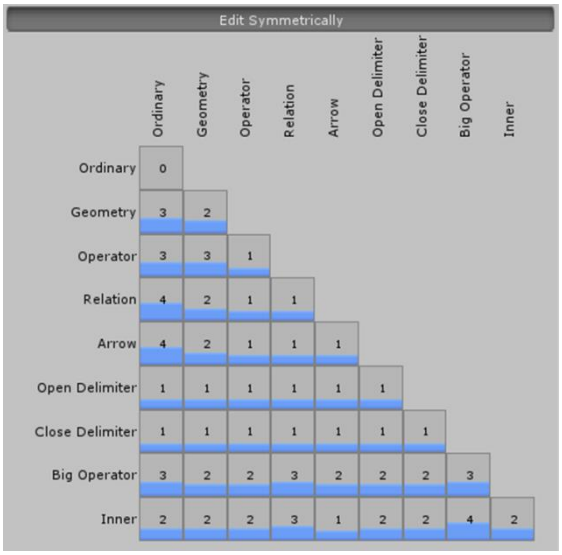
Every TEXDraw shader share the same texture slot depending on what font inside the preference. Of course that's also mean users don't have to plug each texture; Here we'll do it for you automatically. All you need to do is plug all the TEXDraw materials to **Watched Material**. Also, there's a slot for **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	⅁	\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

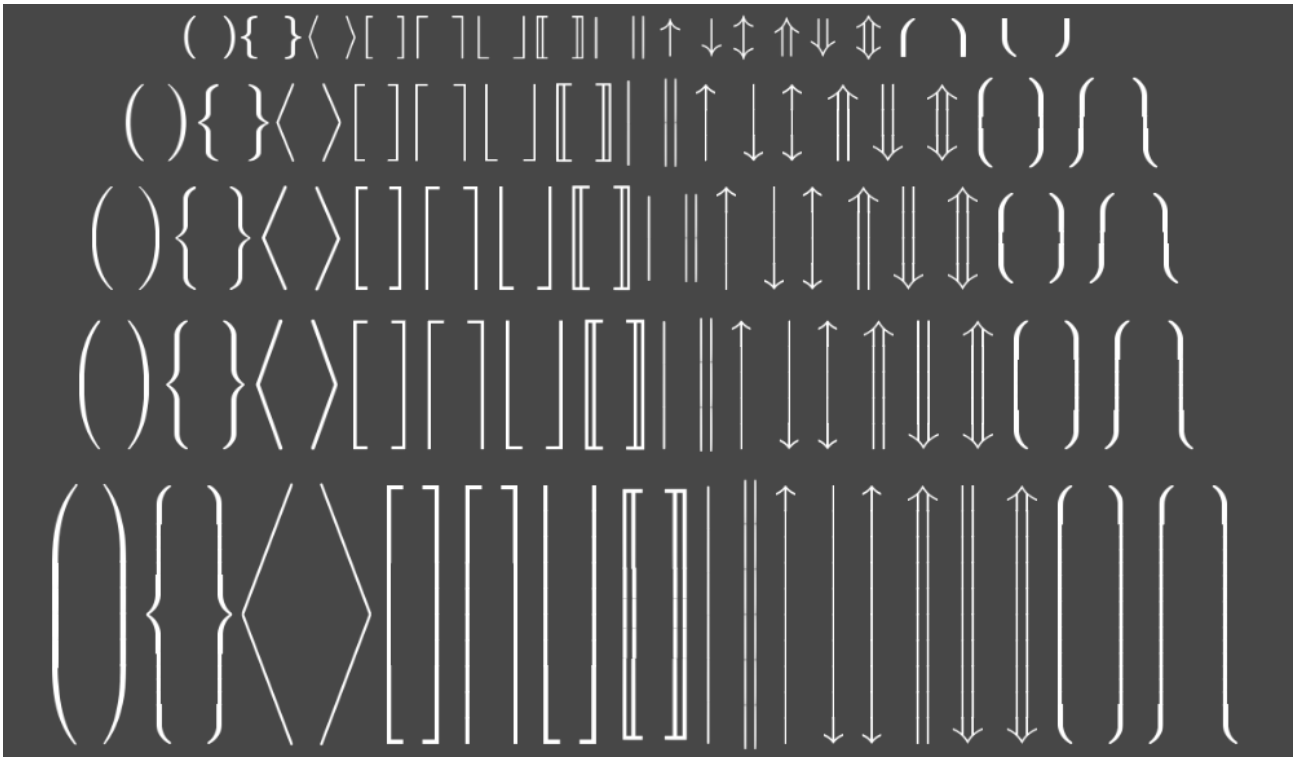
	\uparrow		\Uparrow		\leftharpoonup		\nearrow
	\downarrow		\Downarrow		\leftharpoondown		\searrow
	\leftarrow		\Leftarrow		\rightharpoonup		\swarrow
	\rightarrow		\Rightarrow		\rightharpoondown		\nwarroy
	\leftright arrow		\Leftright arrow		\upharpoonleft		\nnwarrow
	\updownarrow		\Updownarrow		\upharpoonright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

*) New in V2.6, every horizontal arrow can stretch automatically by using ^^ or __ (example: {into}__{\rightarrow}).

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

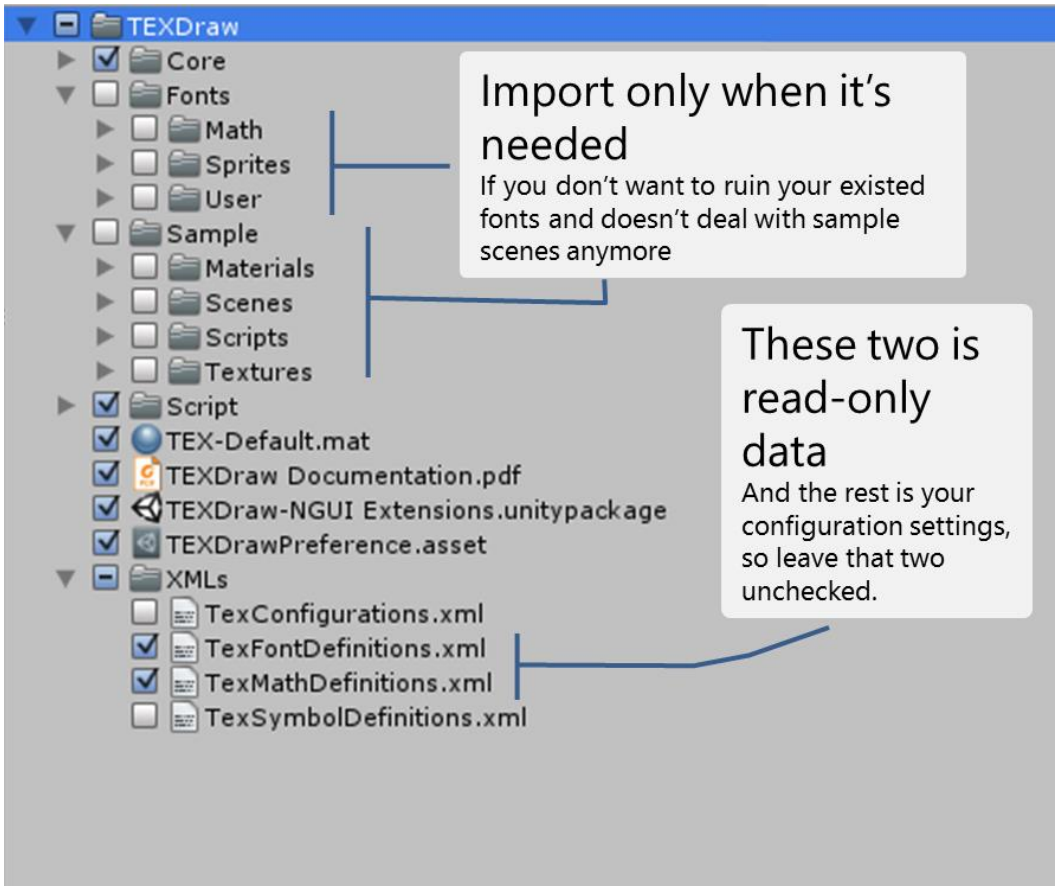
Appendix: A side note to the users

Proper Upgrade Step Procedure for further Version of TEXDraw

Before you apply a TEXDraw upgrade, please **Backup your Project** first.

The procedure to Upgrade an existing TEXDraw package to newer release is slightly different, since some of your data might

- First, **Load** a new, empty scene,
- **Export** your existing preference to XML File,
- **Import** the new package, and wait until import file selection shows,
- Check and uncheck these files:



- After importing done, in TEX Preference, **Import back** your last preferences from XML File.

Legacy: Upgrading TEXDraw to 2.4

In 2.4, we got a feature to importing a sprite as a font, so, to make it possible, we had to break XML database compatibility. And here, we'll show you how to upgrade an existing project to TEXDraw 2.4 without losing preference settings and breaking any existed reference to TEXDraw:

1. **Export** the preference (if needed)
2. in "XMLs/TexSymbolDefinitions.xml" **add** these texts between `<FontIDs>` and `<Font index="8" ...`, so it would be:

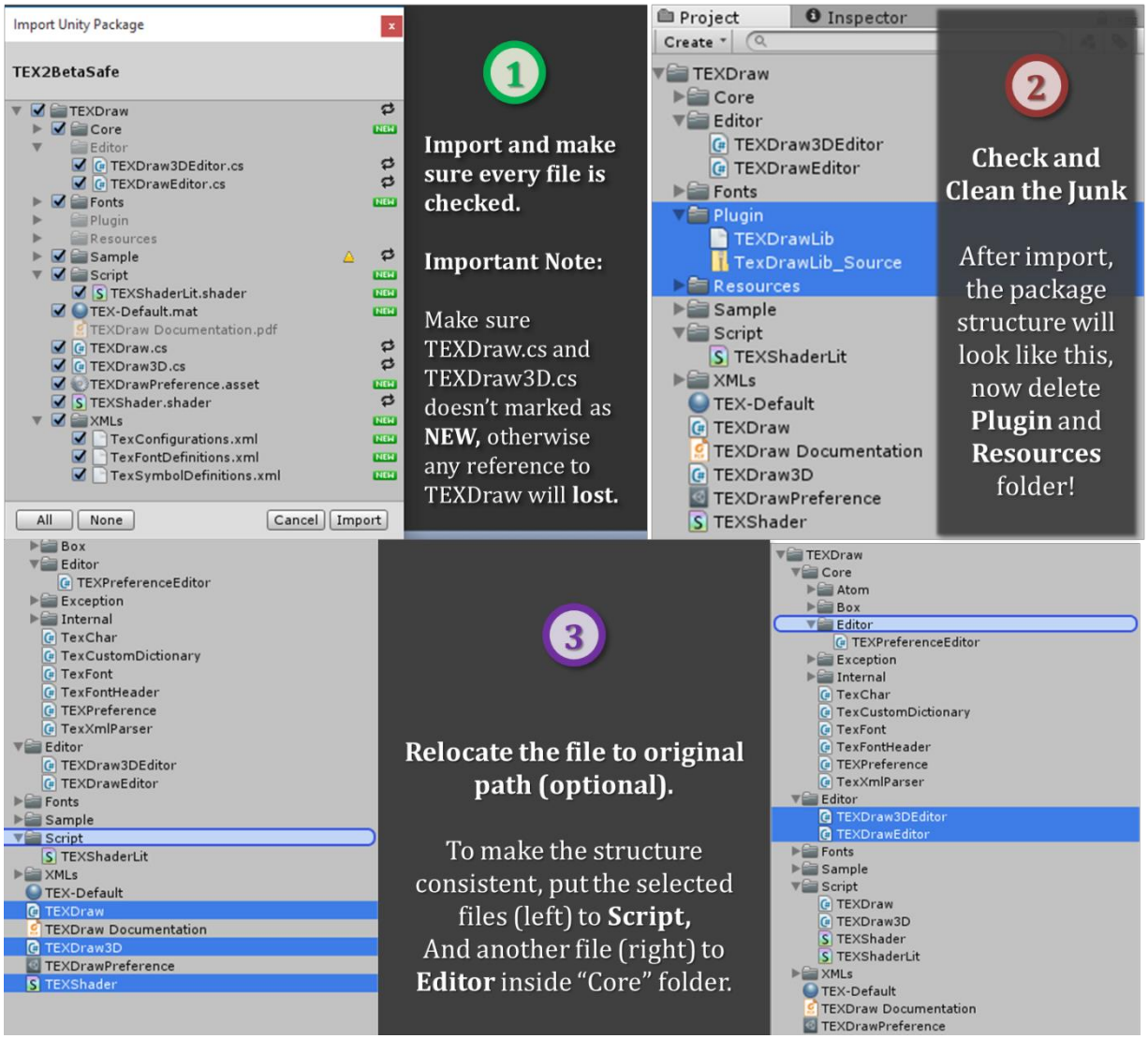
```
    <FontIDs>
        <Font index="0" id="cmex"/>
        <Font index="1" id="cmmi"/>
        <Font index="2" id="cmsy"/>
        <Font index="3" id="cmr"/>
        <Font index="4" id="stmary"/>
        <Font index="5" id="wasy"/>
        <Font index="6" id="msam"/>
        <Font index="7" id="msbm"/>
        <Font index="8" id="cmbx"/>
```

3. **Delete** everythings inside TEXDraw/Fonts
4. **Import** the new package, and wait until import file selection shows,
5. After it has showed, Import (select) everything **Except the XML Files** (TEXDraw/XMLs)
6. After importing done, in TEX Preference, **Import back** your last preferences from XML File.

Legacy: Upgrading TEXDraw package from 1.0 to 2.0 in a project

Here, we will show you how to safely apply an upgrade of TEXDraw Package to existing project that using our previous TEXDraw package version 1.0. The point of following these procedure is prevent break links your project dependency to the TEXDraw script and components:

1. Check if everything is ready (including back-up your project).
2. **Save** your current and load a new (empty) scene
3. **Import** the new package and make sure everything looks fine and imported.
4. **Delete** Plugin and Resources folder after import.
5. **Move** the changed file (the one that have a refresh icon at import) back to their original path (this is optional, the package still working even if you don't do this)


Import Unity Package
TEX2BetaSafe
TEXDraw
Core
Editor
TEXDraw3DEditor.cs
TEXDrawEditor.cs
Fonts
Plugin
Resources
Sample
Script
TEXShaderLit.shader
TEX-Default.mat
TEXDraw Documentation.pdf
TEXDraw.cs
TEXDraw3D.cs
TEXDrawPreference.asset
TEXShader.shader
XMLs
TexConfigurations.xml
TexFontDefinitions.xml
TexSymbolDefinitions.xml
All
None
Cancel
Import
1
Import and make sure every file is checked.
Important Note:
Make sure TEXDraw.cs and TEXDraw3D.cs doesn't marked as NEW, otherwise any reference to TEXDraw will lost.
Project
Inspector
Create
TEXDraw
Core
Editor
TEXDraw3DEditor
TEXDrawEditor
Fonts
Plugin
TEXDrawLib
TexDrawLib_Source
Resources
Sample
Script
TEXShaderLit
XMLs
TEX-Default
TEXDraw
TEXDraw Documentation
TEXDraw3D
TEXDrawPreference
TEXShader
2
Check and Clean the Junk
After import, the package structure will look like this, now delete Plugin and Resources folder!
Box
Editor
TEXPreferenceEditor
Exception
Internal
TexChar
TexCustomDictionary
TexFont
TexFontHeader
TEXPreference
TexXmlParser
Editor
TEXDraw3DEditor
TEXDrawEditor
Fonts
Sample
Script
TEXShaderLit
XMLs
TEX-Default
TEXDraw
TEXDraw Documentation
TEXDraw3D
TEXDrawPreference
TEXShader
3
Relocate the file to original path (optional).
To make the structure consistent, put the selected files (left) to Script,
And another file (right) to Editor inside "Core" folder.
TEXDraw
Core
Atom
Box
Editor
TEXPreferenceEditor
Exception
Internal
TexChar
TexCustomDictionary
TexFont
TexFontHeader
TEXPreference
TexXmlParser
Editor
TEXDraw3DEditor
TEXDrawEditor
Fonts
Sample
Script
TEXDraw
TEXDraw3D
TEXShader
TEXShaderLit
XMLs
TEX-Default
TEXDraw Documentation
TEXDrawPreference

TEXDraw Release Notes

2.6 – Aug 8, 2016: **Parser Stability**

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

2.5 – Jul 28, 2016: Interactive Link

- Official support for NGUI, included as an external .unitypackage file
- Support for UV3 filling, unlocking many shader features and variants
- Gradient, and Texture overlay shader for TEXDraw
- Brand New TEXLink: put links over TEXDraw!
- Underline and overlined text style
- More color command variants (\clr and \mclr)
- \size command for having variant in sizes.
- Improved autofit mode
- Improved performance: Now only few bytes GC inprints on repaint!
- Fixed incorrect glyph sizes on large display characters
- Fixed pixel-perfect behaviour for UI, now it's always shows crisp on screen space Canvas.
- Fixed Shader ordering for straight lines

- Fixed Preference trigger error when displaced, now path is automated
- Fixed Compatibility for Webplayer
- Changed Latin symbol with font override will follow the overrided font, not following the default font rule.
- Changed shader location, now detached from GUI\.. path

2.4 – Jun 20, 2016: Sprite Import

- Now it's possible to have a texture-based characters (a.k.a. importing textures instead of fonts)
- Increasing imported font limit from 15 up to 31 fonts.
- Polished TEXDraw preference
- Shader-Rewriting: Shadow lit removed
- \color now support html text
- Improved Performance, more less GC overhead at render.
- New scene example and improved stress test scene.

2.3 – Apr 20, 2016: Performance Upgrade

- New Autowrap and Justify alignment
- New Resource Pooling System, about 10 times less GC overhead!
- Added Stress Test example scene
- Added functionality to change entire font character in specific component.

2.2 – Apr 13, 2016: Unicodes

- Fixed important bug when Imported fonts showing white boxes.
- Fixed Mobile Shader: Giving two passes instead of one
- Unicode support
- \color command for custom text colors.

2.1 – Apr 4, 2016: Compabilities

- Fixed Shader Compilation for PS 3.0/4.0 and Mobile
- Fixed UI Layout Problem and Functionality
- Fixed Unity 5.3 Compability
- TEXDraw Lit Shader (with Shadows) now available

2.0 – Mar 29, 2016 : Package Rewrite

- Rewritten Package & Docs.
- Brand New TEXDraw Preference Editor
- Configure and customize global prefs.
- Add your own custom Fonts!
- Faster Loading times (be serialized on build)
- No resource dependency
- Unity 5.4 Compatibility
- 15 Fonts Data included

- +600 Symbols Catalog Available
- Expandable Delimiters (No More Stretching!)
- UI Effect Support

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 and 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 don't see where is TEXDraw support NGUI

Please import the included package snippet named 'TEXDraw-NGUI Extension' from TEXDraw root folder, then you can create one by navigate the menu to NGUI/Create/TEXDraw.

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.

TEXPreference was fail to Import XML Data

Importing here means overwrite/update existing data from current database, if it fail, then make sure:

- You are not inside the Webplayer/WP 8.0 platform
- MainFolderPath is correct, (switch to debug inspector when opening the preference)
- No symbol conflicts when editing symbol definitions

Reach us if the problem still persist.

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

Assign your material to Watched Material configuration, and re-import the preference.

Every TEXDraw that generated runtime doesn't work properly at build

Always make sure there at least one TEXDraw component that created in the editor! Otherwise, you will see any generated TEXDraw component will fail to render.

Deleting unused fonts/Adding new font in TEXDraw font lists

Deleting every provided user/sprites font is always safe, simply delete, and re-import the preference. To import a new font, simply move that font into TEXDraw/Font/User, then do reimport. Deleting math is 'safe' as long as you are not using any math features. More information here.

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.

Each TEXDraw component takes 4 batches at Gameplay which is... Expensive

Great power comes great reponsibility, but we will take easy for that. The key is on the Shader. As you see in the Default Shader has path to TEXDraw/Default/4 Pass, this "4 Pass" determines

how much batches are used for each TEXDraw Component. Why 4 Batches? Because of Texture Limit! There's up to 31 Fonts, means there's about 8 texture for each pass (actual limit is 16, but we seen that some mobile device imposed texture limit to 8, instead of 16). It's your job to change each TEXDraw material to the apporiate pass, more details is here.

Any chance for use Unity's new Texture Array?

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

Informing other bugs/Feature request/General Talks

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! Go 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