

TEXDraw

LaTeX Graphic Mathematical Expressions Input for Unity

Documentation for V2.3

Table of Contents

General Question to TEXDraw	5
What is TEXDraw?	5
How it does work and why it different?	5
How about Platform Dependencies? Is it works on mobile?	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
TEXDraw 3D Component.....	8
TEX Preference.....	9
Font Collections	10
Guide to Write in TEXDraw	11
An Introduction	11
The Power of Backslashes	11
Writing Fractions.....	12
Writing Roots.....	12
Introduction to Superscript and Subscript	13
The Special usage of Big Operator Symbols.....	13
Using Expandable Delimiters.....	14
Custom Color	14
Writing Matrix	15
Adding Negation Line (Strikethrough).....	15
Using & Editing TEX Preference	16
Preamble	16
The Import-Export feature	16
How does TEX Preference saved and be included on build?	16
Tab 1: Symbol & Relation	17
Navigating and Selecting Font	17
Adding your own font to the TEXDraw font stacks	17
Using & Navigating through Character Map	18

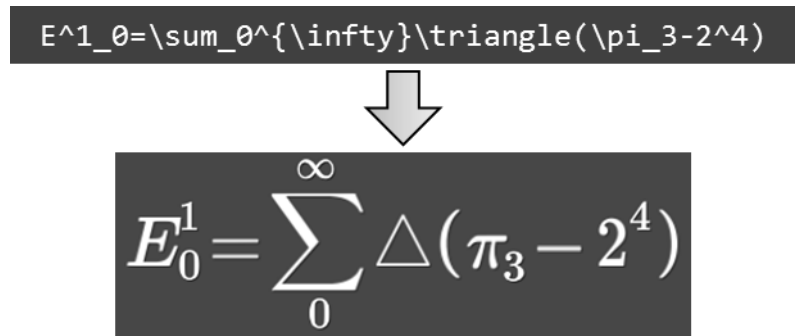
Modify a Character Settings.....	18
Understanding Symbol Types.....	19
What is Character Hash, and what's the point of it?	19
The power of Delimiters: Making Character Relations	20
The Real Truth about TEXDraw's Way to Map a Font Characters	21
Tab 2: Real-Time Global configurations	21
Understanding & Using Global Configurations.....	22
Anatomy of a Character	22
Default Typefaces, what is it?	25
Set-up and create custom TEXDraw Material.	25
Tab 3: Glue Management	26
Escape Character Definition Cheatsheet	27
Greek Letters.....	27
Common Ordinary Symbol	27
Miscellaneous Symbol	28
Astronomical Symbols	28
Block Shapes	29
Geometrical Symbol Shapes.....	30
Boxed Binary Operators.....	31
Binary Operators	32
Relation Comparer	34
Miscellaneous Relations	35
Negated Relations	36
Primary Arrows.....	37
Compound Arrows	38
Expandable Delimiters.....	39
Open & Closing Delimiter.....	39
Big Operator.....	40
Accent.....	41
Preserved Characters	41
Appendix: A side note to the users.....	42

Upgrading TEXDraw package in a project.....	42
Proper Upgrade Step Procedure for further Version of TEXDraw	43
Compatibility Breaks for new Update (2.0).....	Error! Bookmark not defined.
TEXDraw Release Notes.....	43
License notices for Included Fonts	44
Guide to Write in TEXDraw (Runtime Script)	45
About This Package	46

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.



The diagram illustrates the process of rendering a mathematical formula. At the top, a dark gray box contains the LaTeX source code: `E^1_{\theta}=\sum_{\infty}\triangle(\pi_3-2^4)`. A large, light gray downward-pointing arrow connects this box to a second dark gray box below it. This second box displays the rendered mathematical formula: $E^1_{\theta}=\sum_0^{\infty}\triangle(\pi_3-2^4)$.

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!

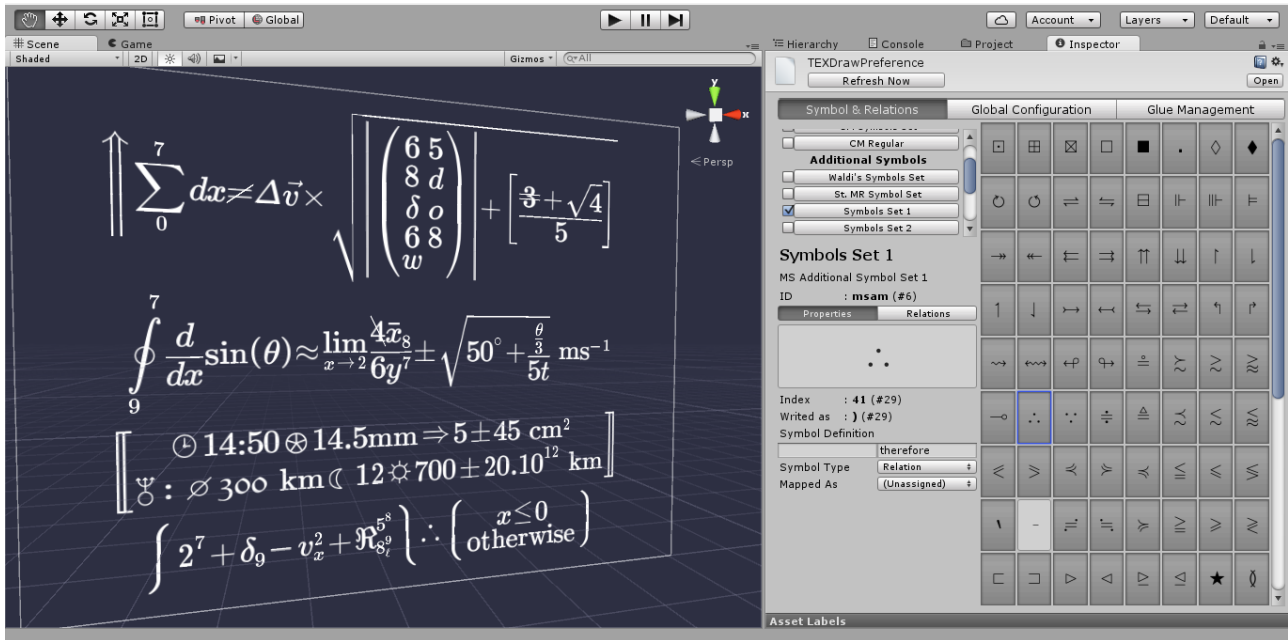
How about Platform Dependencies? Is it works on mobile?

Yes, we don't include any native plug-in in this package, and the only job they do is draw some beautiful renders on math in your screen. The fonts that needed for renders are already included in this package (we're not made the fonts itself, but you are legally can use and include these fonts to builds both commercial and non-commercial, read more [here](#) for source information)

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.

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.

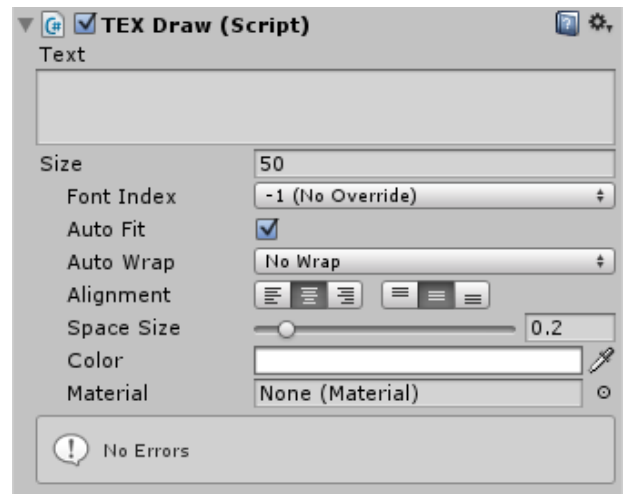
Is your package depends on Unity's UI? What if I don't want to use it?

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 this component is a great alternative for those who use another GUI System like NGUI, etc.

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 `UI > TEXDraw`. This component will handle rest of rendering process, so you 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).

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.

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

Auto Fit Should graphics have to be fit in rectangle bound if it oversized?
Turn off this check box if you want to preserve the graphic size even it's out of rectangle box.
NOTE: Turn **off** this check box when you're using UI Auto-Layout system.

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.

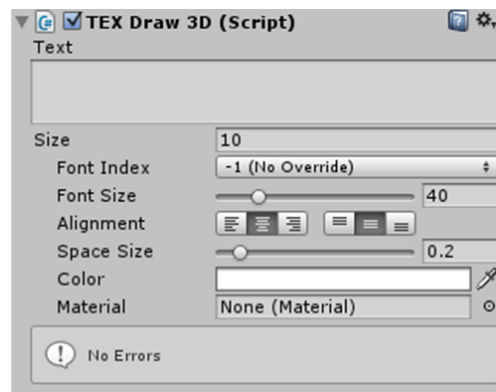
Color The main color for generated graphics
Use [\color](#) if you want to write specific color

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

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.

TEXDraw 3D Component

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 `Mesh > TEXDraw 3D`.



Text A plain text that you want input to.
Similar like TEXDraw, this property support multiline.

Size Size of generated graphics
The size here will be assumed as one unit world size instead of one pixel

Font Size Used Font Texture size
Larger the value, finer the display (with the cons of larger used memory).

Alignment The horizontal and vertical alignment of the text.
The Alignment will limited to mesh boundary only since transform doesn't have defined rect

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

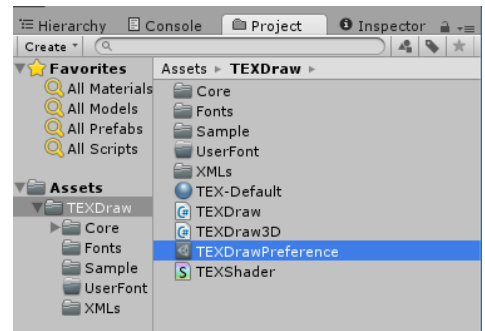
Material Assign a Custom Material for This component
If none assigned, the default material (from TEX Preference) will be used for rendering

TEX Preference

Beside of powerful TEXDraw Component, it would be never works without databases, and it holds inside the powerful TEX Preference. You can open the Preference by navigate into `Edit > Project Settings > TEXDraw Preference`.

TEXDraw Preference holds shared information across one project and saved as arbitrary asset located in:

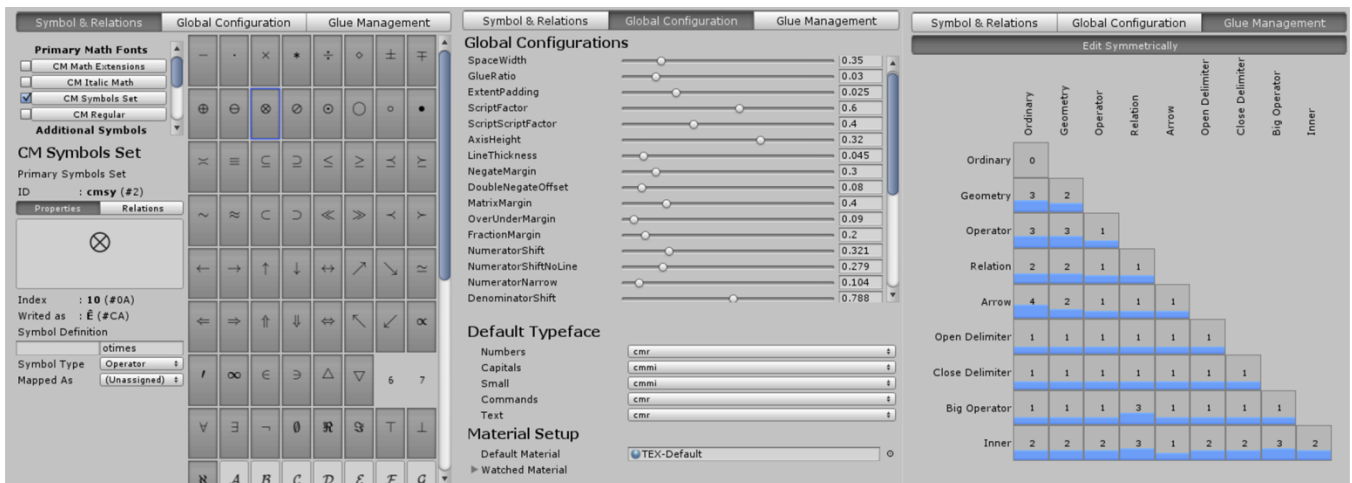
`Assets/TEXDraw/TEXDrawPreference.asset`.



The preference here located inside of TEXDraw root folder as an arbitrary 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 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, fraction gaps, script drops, etc.

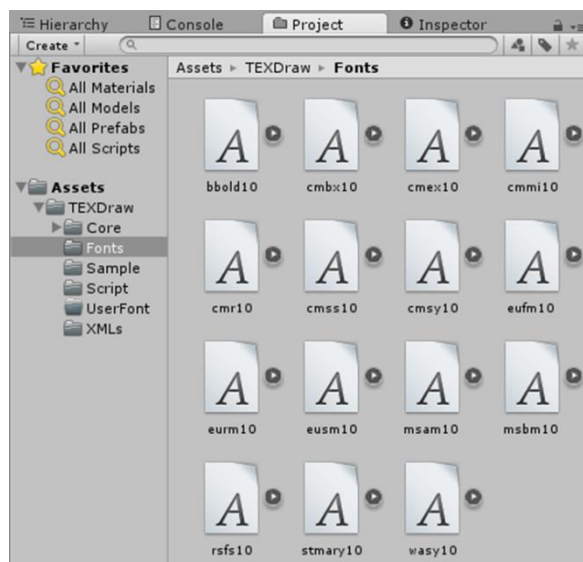
Glue Management

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 and included in inside of a folder located in `TEXDraw/Fonts`. These 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](#).



These 15 fonts each add about 50 KB of size build, which is pretty small for mobile builds. We will discuss the functionality of each font below:

-
- | | |
|---------------|---|
| CMEX | Computer Modern Math Extensions
Primary characters that support extensions are maintained here |
| CMMI | Computer Modern Math Italic
Italic-styled characters along with some Greek characters are belongs to here. |
| CMSY | Computer Modern Symbols Set
Primary (commonly) used symbols are located here. |
| CMR | Computer Modern Regular
A place for standard regular-styled characters. |
| WASY | Roland Waldi Symbol Set
Additional Symbols for Geometric and Astronomic character set. |
| STMARY | St Mary's Road Symbol Set
Provides Additional Symbol set for Complex Usage |
| MSAM | MS Additional Symbol Set Group 1
Additional Symbol Set Group 1, with lots of relation symbols |
| MSBM | MS Additional Symbol Set Group 2
Additional Symbol Set Group 2, with lots of negated relation symbols |
-

The rest of other fonts, like eusm, bbold, are optional. You can override/change these optional fonts to anything described in detail [here](#).

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!
```

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 another internal purposes (we will discuss it later).

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 symbol name is case-sensitive.

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

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

$\Delta\theta\approx 2t\times(3\pi+4\omega)$

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

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 will understand it.

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

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

Deltax, Δx , or Δx ?

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

See the second example above, when you type `\sin`, the symbol doesn't exist, so it returns a plain word (but different font variation). This is good usage for writing short functions like trigonometry, log, limit, etc. Also take a look at "Solve this test" - this is a usage example of command string, type `\text` followed by any string surrounded by braces will apply default font style, while typing backlash by defined font ID (like that `\eufm` and `\eurm`) will resulting

custom font applied inside of text. If you want to control what font will be marked as default `\text` font, jump to [here](#).

Writing Fractions

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

$$\frac{[numerator]}{[denominator]}$$

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.

$$\frac{2+2}{2x} \equiv \frac{d}{(\frac{1}{4})} \{r\}$$
$$f(x) = \begin{cases} \frac{2x}{3} \\ \frac{1}{x} \end{cases} \text{ if } x < 0 \quad \begin{cases} \frac{2x}{3} \\ \frac{1}{x} \end{cases} \text{ otherwise}$$

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

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

Now look at the second example, `\nfrac` is another variation of fraction where it doesn't render 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 ($\sqrt{}$) with a thick line on the root base. Writing Roots is easy, by follow on this format:

$$\sqrt[n]{\text{degree}}\{\text{base}\}$$

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 possible to get a nested root.

$$\sqrt[4]{\frac{C}{4}} - \chi = \sqrt[4]{\alpha + \sqrt{\beta}}$$
[illegible]

$$\sqrt{\frac{C}{4}-\chi}=\sqrt[4]{\alpha+\sqrt{\beta}}$$

[illegible]

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\leq 10^{\sqrt[4]{40^{\eta}}-3}
```

$$\Re_{2_{3_4^4}}^{2_{3_4^4}} \equiv \alpha^2 \beta_{3_45} \gamma^5$$

$$^2\log 5 + \log_{10} 10^4 \leq 10^{\sqrt[4]{40^{\eta}}-3}$$

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 have understood this, this is the minimum size of how small a script could be, through the size still adjustable in preference.

The Special usage of Big Operator Symbols

Now you know how a script works, but there's one exception: if you type a symbol which has type of Big Operator then it's simply goes 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\rightarrow 2}\frac{\pi{x}^2}{x-2}\approx\coprod^{10}_{x=\min 2}\{a^{...}
\}+x
```

$$\sum_{x=0}^{\infty} x \frac{5}{6} - \frac{10}{x} \Leftrightarrow \prod_{x=0}^5 x - 7$$

$$\lim_{x \rightarrow 2} \frac{\pi x^2}{x-2} \approx \coprod_{x=\min 2}^{10} \{a^{...}\} + x$$

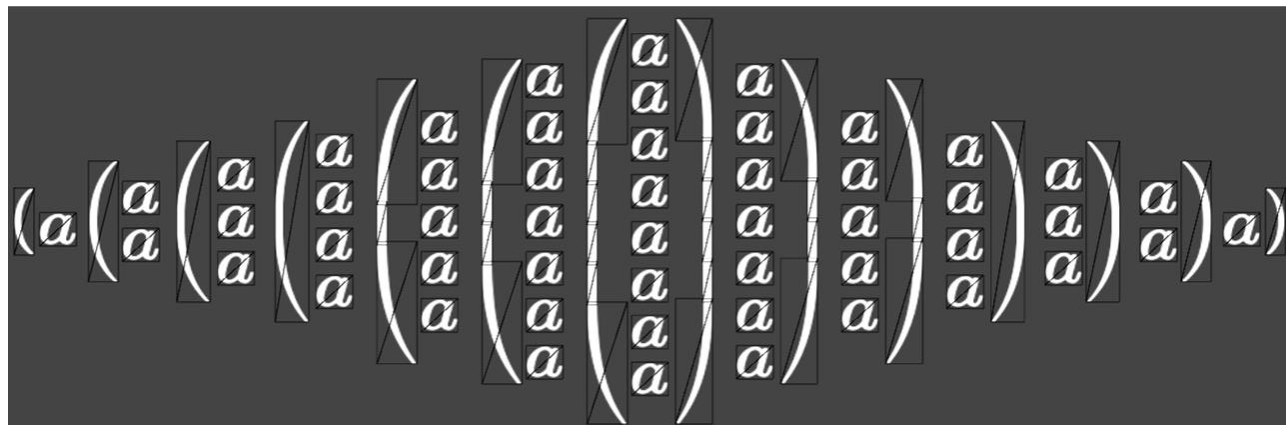
Unfortunately, custom alignment for integrals doesn't supported - however, we have a tricky solution for this: add some invisible spaces!

```
\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
```

$$\int_5^7 u \int_5^7 v \iint_5^7 w \iiint_5^7 x \geq \oint_5^7 y \oiint_5^7 z$$

Using Expandable Delimiters

Delimiters like brackets (), or any other variations like [], {}, | |, should can expand higher or equal than their neighbours. This new feature has been implemented correctly and now you can type formula as tall as possible without having to worry about stretching that does exist in previous release.



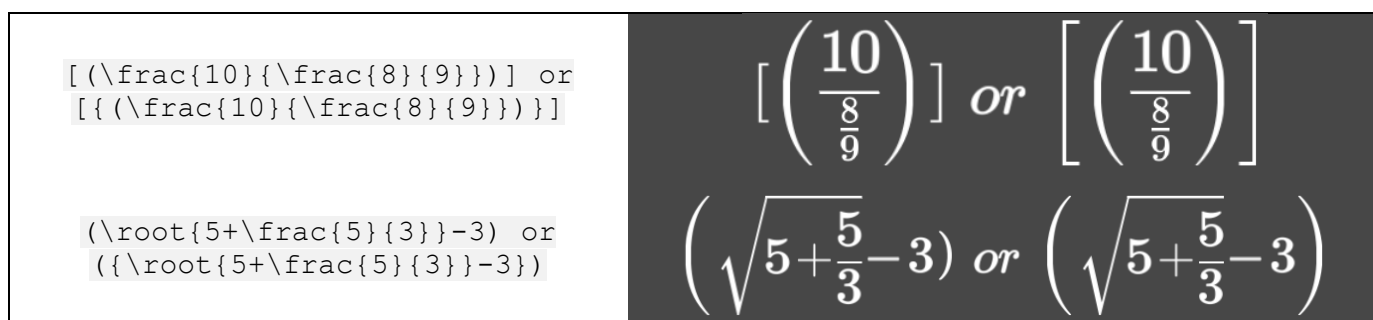
Expandable delimiters, does expand higher until sufficient height achieved, without suffering from stretching.

Curious? Try it here:

Curious? Try it here:

($a \backslash \operatorname{matrix}[a]{a} \backslash \operatorname{matrix}[a]{a}{a} \backslash \operatorname{matrix}[a]{a}{a}{a} \backslash \operatorname{matrix}[a]{a}{a}{a}{a} \backslash \operatorname{matrix}[a]{a}{a}{a}{a}{a} \backslash \operatorname{matrix}[a]{a}{a}{a}{a}{a}{a} \backslash \operatorname{matrix}[a]{a}{a}{a}{a}{a}{a}{a}$)

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:



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

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.



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&\alp  
ha})
```

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

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

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

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}`.

Adding Negation Line (Strikethrough)

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

```
\[n|h|d]not{base}
```

There are 4 types of lines, `\not` gives you a diagonal line, while `\nnot` gives you the inverse version from `\not`, so do `\hnot` which gives you a horizontal line, and finally `\dnot` will give you double-lined horizontal line. Take a look of these examples:

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

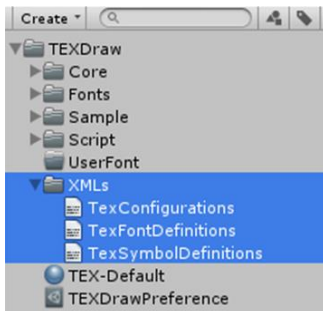
$$\frac{\not{3}+5}{\not{x}} = \frac{x(1-3)}{\nnot{Y}}$$

Using & Editing TEX Preference

Preamble

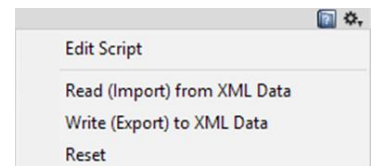
In the previous section, we know how to open TEX Preference and where it's located. Now we will talk about what's inside of this preference and how to customize it to suit your project need. Configuring TEX Preference is optional and you can skip this section if you are OK with default configurations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

The Import-Export feature



TEX Preference save all configurations as a serialized data (so, it will saved inside *.asset itself). 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, we are not guarantee any changes will be safe and secure (it's not stable) (for example, when unity crashes, or you accidentally delete the preference itself), so we need a separate saving method, and XML Import-Export 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 include at build time (but Preference still).



In the "Gear Button", 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.

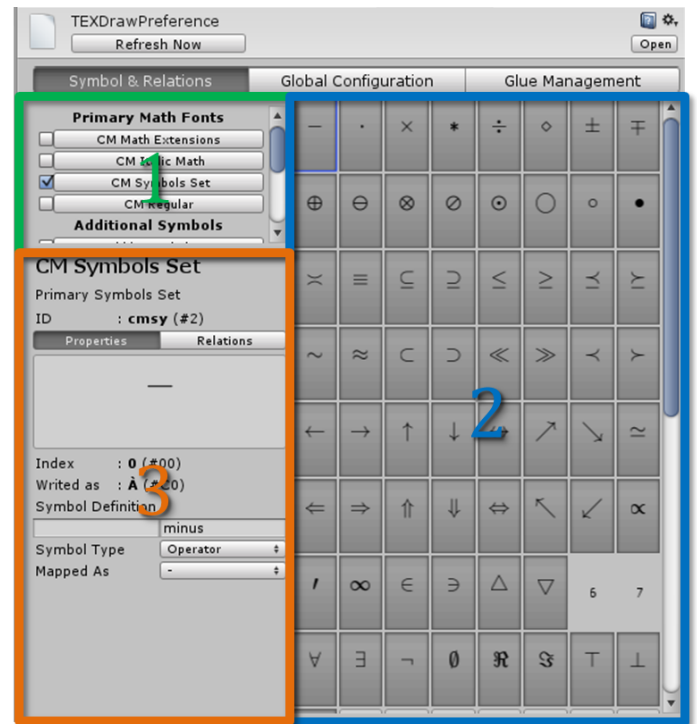
How does TEX Preference saved and be included on build?

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 runtime, 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

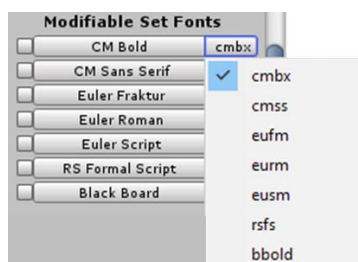
See the right image, there are 3 main sections:

- 1 Font "Stack" Selections
Select a Font that will be previewed in section 2
- 2 Character Map
Characters that are contained in selected font and it's available to be defined will appear here. Select a Character that will be previewed in section 3
- 3 Per-Character Configuration
Selected character can be configured here.



Navigating and Selecting Font

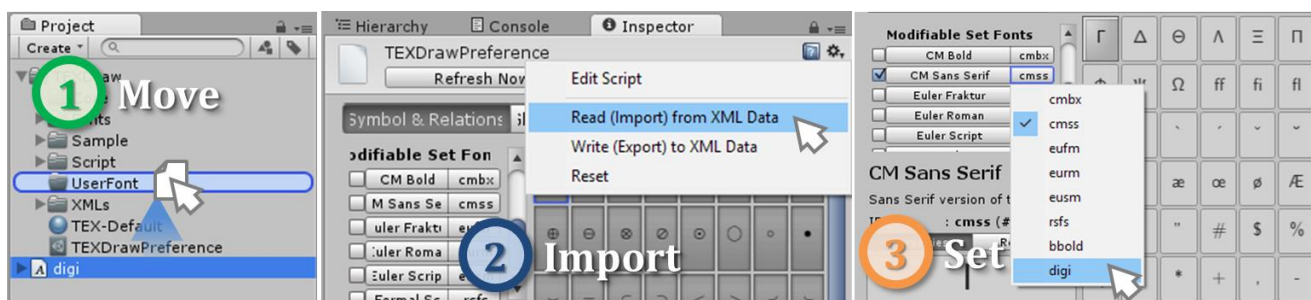
Here, you can determine which font is selected and be previewed in the section 2. However, when you deal with some fonts in the category "Modifiable Set Fonts":



There's a pop up button on right side. This pop up button lets you determine which font is available to be rendered in every TEXDraw component in your scene. If you want to include your custom font then you can use this feature to include your own font to these font stacks (and it's explained below). Please note that only limited set of characters (ranging from #21 to #7E) can be guaranteed to be imported and mapped correctly within TEXDraw component.

Adding your own font to the TEXDraw font stacks

1. Add your font (*.TTF or *.OTF) to TEXDraw/UserFont. The name of your font will be used as Their ID Name. (Be warned that the name must be only letters and case-sensitive)
2. Re-Import XML Data (you may want to export your preferences first)
3. After re-import process done, select (click the pop-up button) in one of the seven *Modifiable* font stack.
4. Now your font is registered (shown) in the pop-up selection, select to the name of your font.

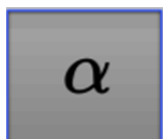


Using & Navigating through Character Map

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.

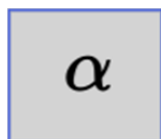
As you can see in the screenshot on the right, there's different box style applied on each character. These different styles tell us about what's state that they're on. Take a look of these previews to make it clearer:

Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ
Φ	Ψ	Ω	ff	fi	fl	ffi	ffl
ı	j	`	'	˘	˙	-	°
˚	ß	æ	œ	ø	Æ	Œ	Ø
-	!	"	#	\$	%	&	'
()	*	+	,	-	.	/



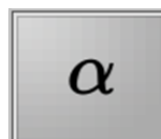
Char is Defined

The character has its own symbol definition



Char is Related

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



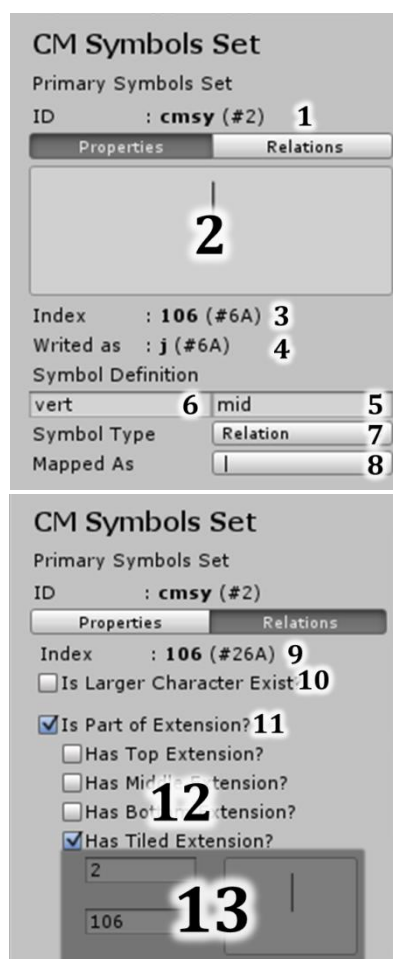
Char is Available

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, there's no reason to assign it to another symbols.

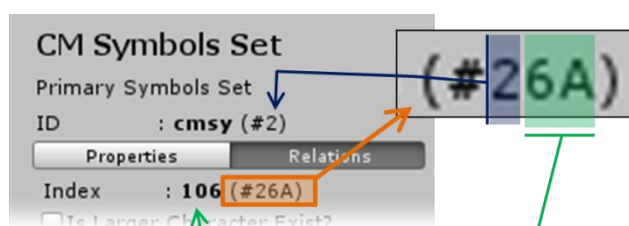
Understanding Symbol Types

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

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

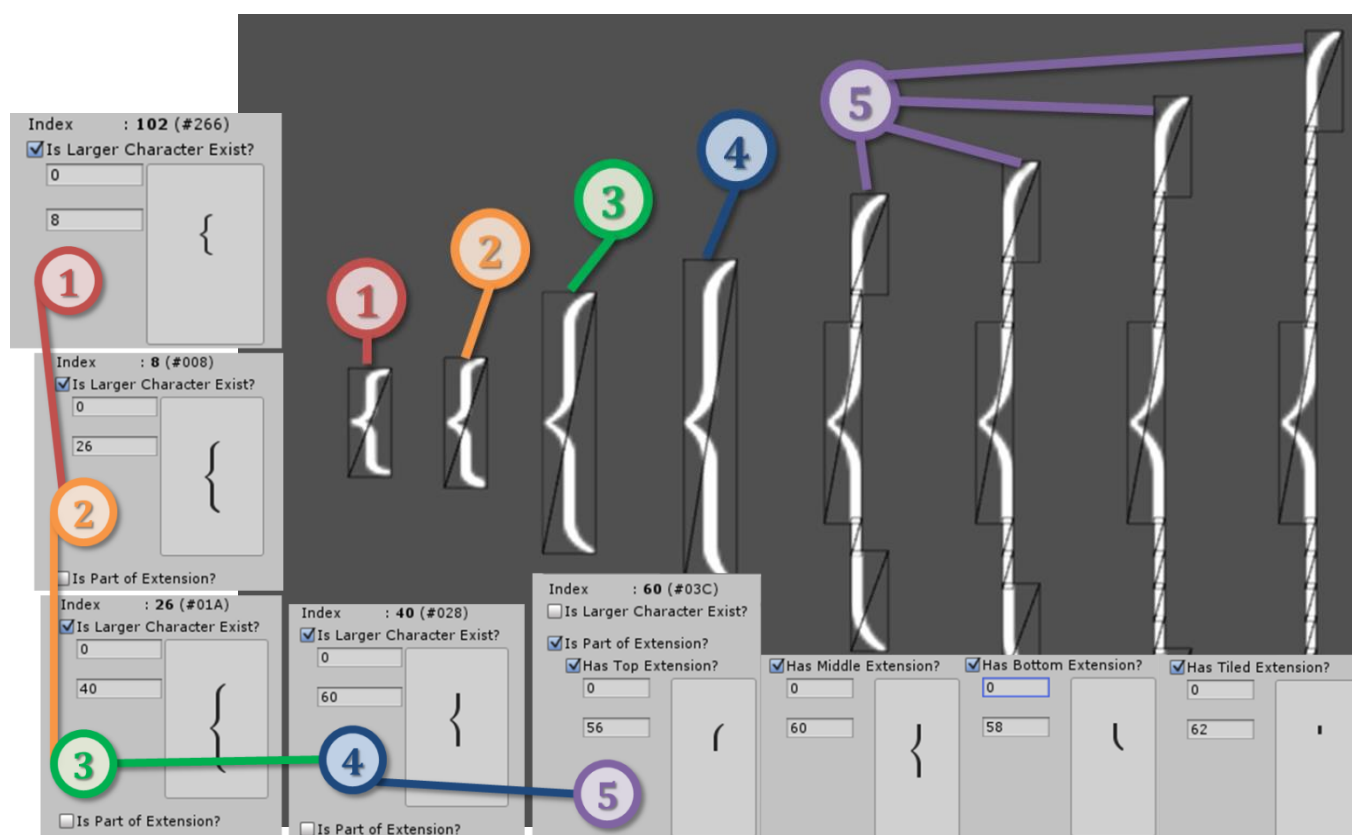
Character hash is a number that given for each character registered in TEXDraw font stack. A hash number that given on a character is unique among the rest. It's easy to read it in Hex Format. For example like in the screenshot in the right, #26A, means the character located in font with index of 2 (cmsy), 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 in 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 #E (0 to 14), or in short, maximum possible value of a hash is #E7F.

The power of Delimiters: Making Character Relations

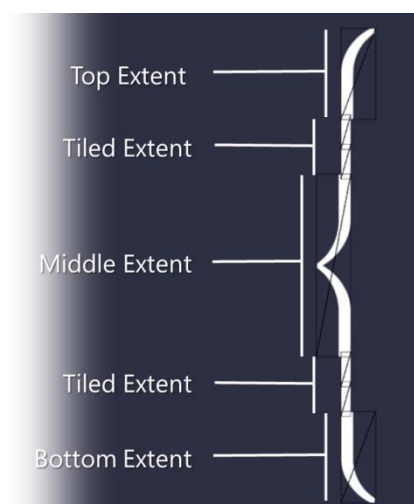
If you have read previous section about delimiters, you should know what happened when delimiter expand to achieve certain height. Now in this section we will reveal the background process of how delimiters can expand their height. Take a look on image below:



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:

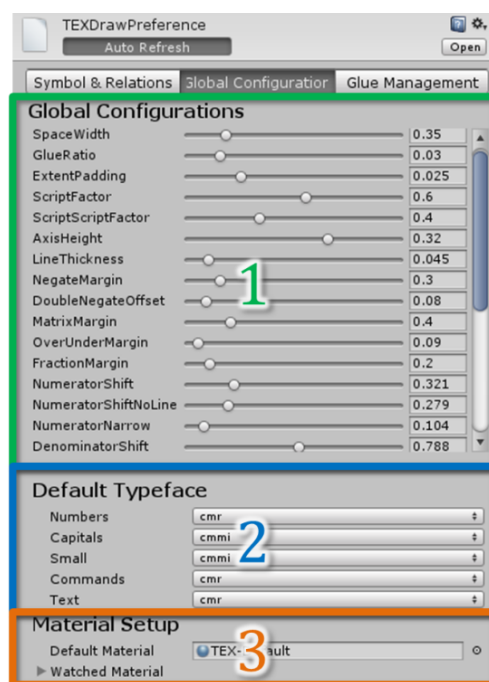
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
10	B0	D1	D2	D3	D4	D5	D6	B7	D8	D9	DA	DB	DC	B5	B6	DF
20	EF	<div>ASCII Printable Character Area</div> <p>The font index in this area has the same index within the ASCII Font file. So things like \text{} will working as long as it's only contains char from this area.</p>														
30																
40																
50																
60																
70																FF

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](#) ASCII characters), meaning it won't be available directly (like typing from `\text{}`, etc.), but still able to be used by defining a 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

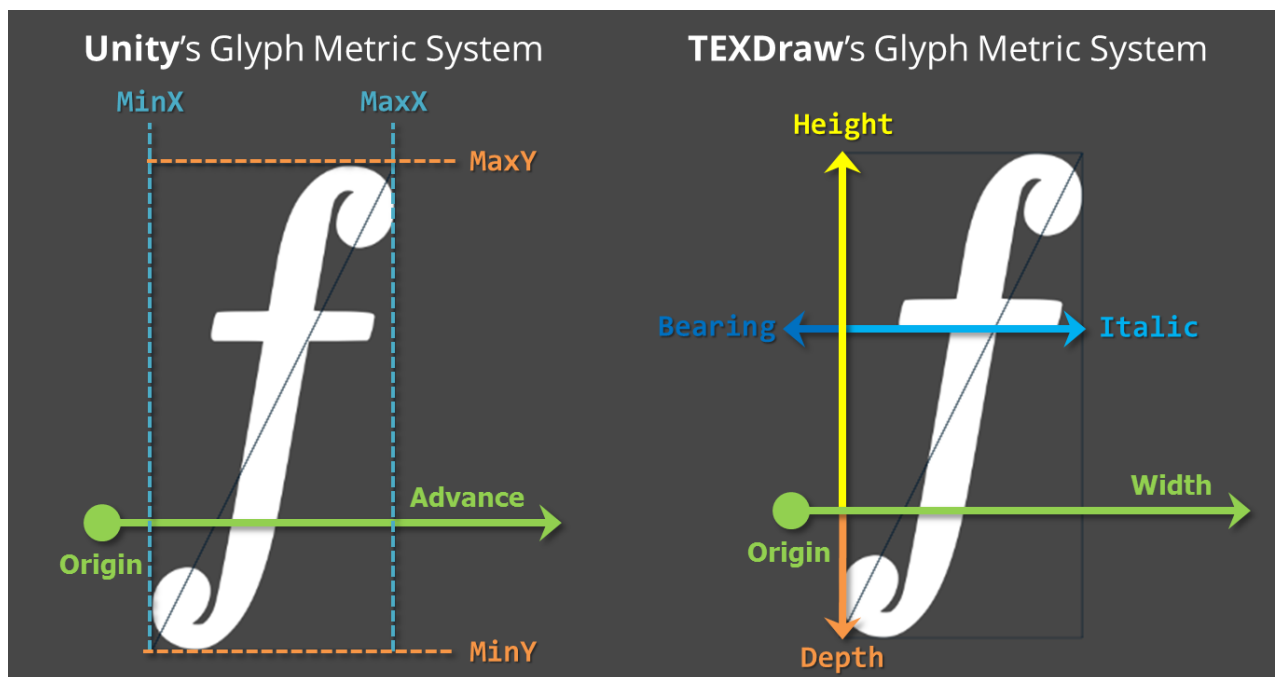


Understanding & Using Global Configurations

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:



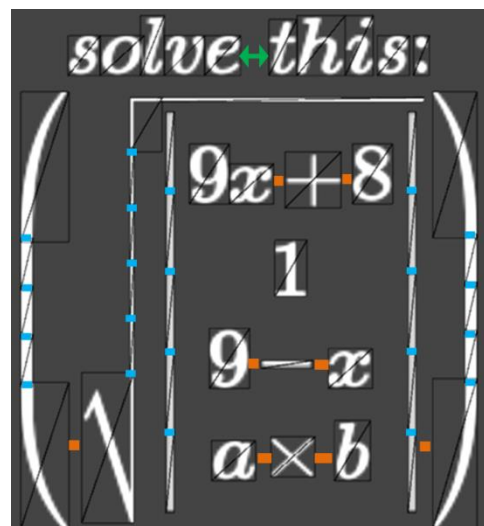
A character has a baseline (marked as green line), top and low bound. The image above will help you to understand each config which we will discuss below:
(We provide each config explanation with relevant image and indicator colors)

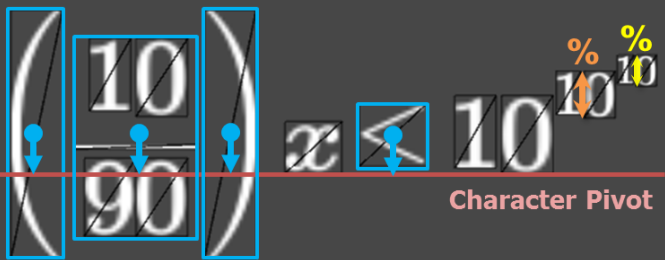
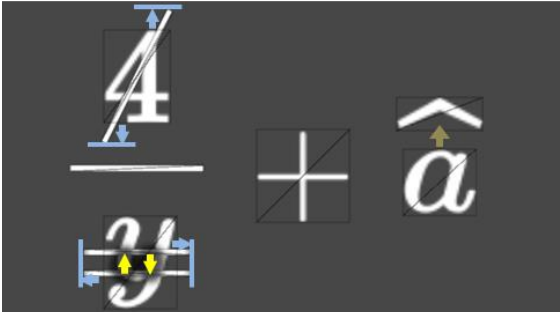
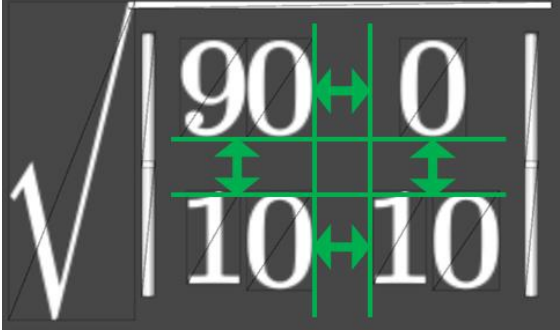
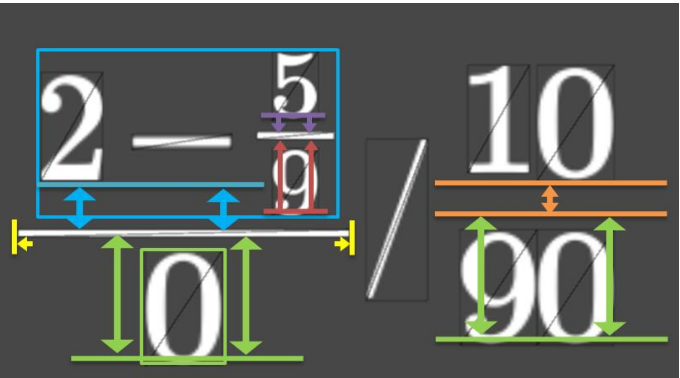
Space Width The width of a single whitespace
The width is on fixed value.

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	<p>Size Ratio of a Script The final script total height in percentage compared to standard character size.</p>	 <p>Character Pivot</p> <p>Characters that automatically be centered at Character pivot include: Relations, Arrows, Delimiters, Big Operators, Fractions, Root, Matrix</p> <p>Only character with type of ordinary, geometry and operator is unaffected</p>
Nested Script Factor	<p>Size Ratio of Nested Script Similar to script factor, but applied for a nested script.</p>	
Axis Height	<p>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).</p>	
Line Thickness	<p>Fixed Line Thickness Width Line thickness for common things (fraction, root, negations, etc.)</p>	
Negate Margin	<p>Negation's Line Margin Negation line with stretch beyond negated character bound until certain value.</p>	
Double Negate Offset	<p>Double Negation Line Gap Adjust to match the best gap height between top and bottom line.</p>	
Matrix Margin	<p>Matrix child-by-child margin The matrix boxes space gap on each column-by-column and row-by-row.</p>	
Over Under Margin	<p>Additional Accent gap height Shift Accent position upward until certain height (calculated from character's top bound).</p>	
Fraction Margin	<p>Additional fraction line width Additional line width for fractions.</p>	
Numerator Shift	<p>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.</p>	
Numerator Shift no Line	<p>Numerator Margin (no line) Similar like Numerator Shift, but specialized for fraction with no line (\nfrac)</p>	
Numerator Narrow	<p>Numerator Margin (narrow) Similar like Numerator Shift, but specialized for a narrowed situation (eg. Inside a fraction, script, etc.).</p>	
Denominator Shift	<p>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.</p>	
Denominator Narrow	<p>Denominator Margin (narrow) Similar like Denominator Shift, but specialized for a narrowed situation.</p>	

Superscript Drop Value

Sup Drop 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.

Subscript Drop Value

Sub Drop 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.

Superscript Standard Minimum Low Bound

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

Superscript Minimum Low Bound (Cramped)

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

Superscript Minimum Low Bound (Narrowed)

Sup Min Narrowed Similar like Sup Min, but specialized for narrowed situation

Subscript Minimum Drop (No Superscript Above)

Sub Min No Sup Minimum distance allowed between subscript's baseline to base script's baseline

Subscript Minimum Drop (With Superscript Above)

Sub Min On Sup Similar like SubMinNoSup, but will used instead if superscript exist on same level.

Big Operator Top-Low Margin

Big operator's additional height size.

Big Operator Up Shift

Distance between top baseline to big operator's top bound

Big Op Minimum Upper Gap

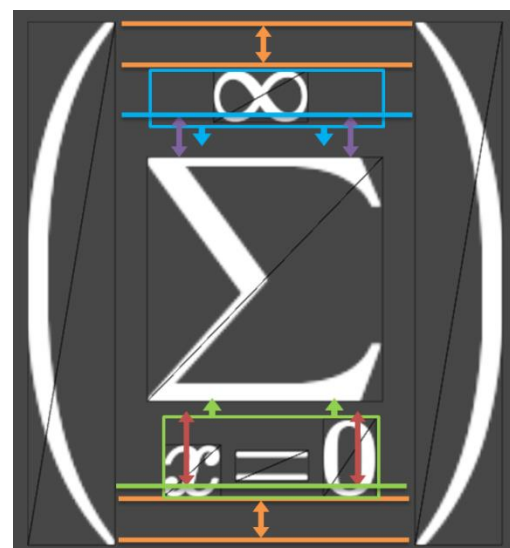
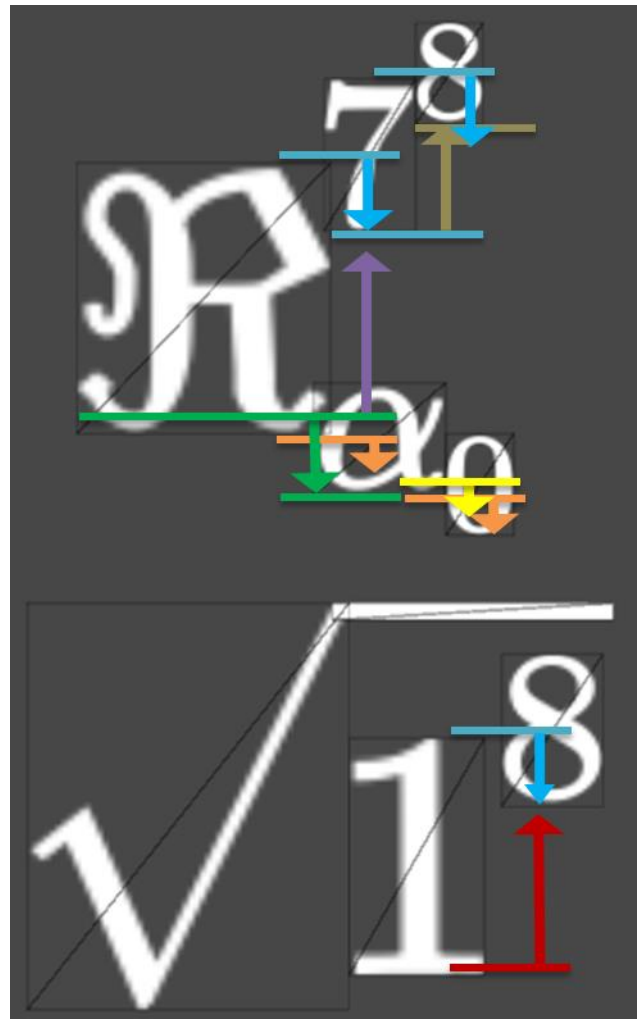
Big Op Upper Gap Minimum distance allowed between top's low bound to big operator's top bound

Big Operator Low Shift

Big Op Low Shift Distance between bottom baseline to big operator's low bound.

Big Op Minimum Lower Gap

Big Op 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 5 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 and Text. You can select a font that will be used as default renderings for each typeface.

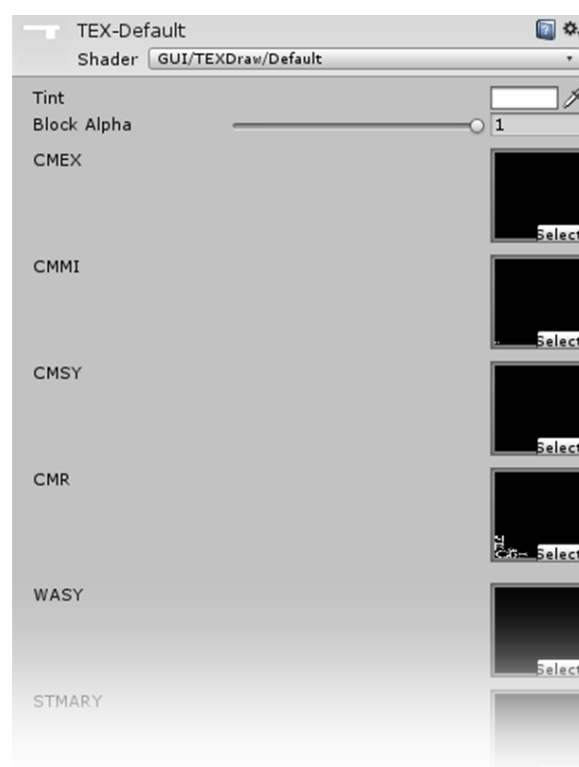
Set-up and create custom TEXDraw Material.

Now, in the section 3, you can assign the default material for all TEXDraw components. A material has been created inside the package. You also can create your own TEXDraw Material then assign it to individual component. There's also **Watched Material** property, this property contains list of material that will refreshes automatically when you change something in the font stack. More detail see at this paragraph:

New feature of TEXDraw 2.1 includes 4 shader variants specialized for use within TEXDraw component.

The TEXDraw Shaders all are located *GUI > TEXDraw* These shaders also support stencil. As you can see in each of the shader properties, there are a total of 15 textures that has to be assigned with matching font texture in TEX Preference font stack. Instead of manually filling the textures, you can assign the material to the **Watched Material** property so it can be filled automatically.

For practical usage of these 4 variant shaders, you can open sample scene *TEXDraw-LitTest* to get a quick reference to apply it in your projects.



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:

	Ordinary	Geometry	Operator	Relation	Arrow	Open Delimiter	Close Delimiter	Big Operator	Inner
Ordinary	0	3	3	4	4	1	1	3	2
Geometry	3	2	3	2	2	1	1	2	2
Operator	3	3	1	1	1	1	1	2	2
Relation	4	2	1	1	1	1	1	3	3
Arrow	4	2	1	1	1	1	1	2	1
Open Delimiter	1	1	1	1	1	1	1	2	2
Close Delimiter	1	1	1	1	1	1	1	2	2
Big Operator	3	2	2	3	2	2	2	3	4
Inner	2	2	2	3	1	2	2	4	2

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.

	Ordinary	Geometry	Operator	Relation	Arrow	Open Delimiter	Close Delimiter	Big Operator	Inner
Ordinary	0								
Geometry	3	2							
Operator	3	3	1						
Relation	4	2	1	1					
Arrow	4	2	1	1	1				
Open Delimiter	1	1	1	1	1	1			
Close Delimiter	1	1	1	1	1	1	1		
Big Operator	3	2	2	3	2	2	2	3	
Inner	2	2	2	3	1	2	2	4	2

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.

Escape Character Definition Cheatsheet

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

Common Ordinary Symbol

















$/$	<code>\forwardslash</code> <code>\slash</code>	?	<code>\invquestion</code>	!	<code>\invfaculty</code>	$-$	<code>\min</code> <code>\varminus</code>
$\#$	<code>\numbersign</code>	?	<code>\question</code>	!	<code>\faculty</code>	$\&$	<code>\ampersand</code>
$\%$	<code>\percent</code>	$\text{\$}$	<code>\dollar</code>	”	<code>\cdot</code> <code>\doublequote</code>	'	<code>\semiquote</code>
\%	<code>\permil</code>	\c	<code>\cent</code>	“	<code>\odot</code> <code>\vardoublequote</code>	,	<code>\comma</code>
@	<code>\commercialat</code>	:	<code>\colon</code>	;	<code>\semicolon</code>	\cdot	<code>\ldot</code> <code>\ldotp</code>

Miscellaneous Symbol























∂	<code>\partial</code>	$'$	<code>\prime</code>	\mathfrak{b}	<code>\thorn</code>	\mathfrak{U}	<code>\mho</code>
ℓ	<code>\ell</code>	\backslash	<code>\backprime</code>	\mathfrak{P}	<code>\Thorn</code>	\eth	<code>\eth</code>
\imath	<code>\imath</code>	∞	<code>\infty</code>	δ	<code>\dh</code>	\beth	<code>\beth</code>
\jmath	<code>\jmath</code>	\emptyset	<code>\varnothing</code>	\circ	<code>\openo</code>	\gimel	<code>\gimel</code>
\wp	<code>\wp</code>	\emptyset	<code>\emptyset</code>	\Finv	<code>\Finv</code>	\daleth	<code>\daleth</code>
\Re	<code>\Re</code>	\forall	<code>\forall</code>	\Game	<code>\Game</code>	\digamma	<code>\digamma</code>
\Im	<code>\Im</code>	\exists	<code>\exists</code>	\surd	<code>\surd</code>	\varkappa	<code>\varkappa</code>
\aleph	<code>\aleph</code>	\nexists	<code>\nexists</code>	\amalg	<code>\amalg</code>	\Bbbk	<code>\Bbbk</code>
\textcircled{R}	<code>\textcircled{R}</code>	\neg	<code>\neg</code>	∇	<code>\nabla</code>	\hslash	<code>\hslash</code>
\textcircled{S}	<code>\textcircled{S}</code>	\lnot	<code>\lnot</code>	\smallint	<code>\smallint</code>	\hbar	<code>\hbar</code>
\complement	<code>\complement</code>	\yen	<code>\yen</code>	\diagup	<code>\diagup</code>	\diagdown	<code>\diagdown</code>
\bowtie	<code>\bowtie</code>	\brokenvert	<code>\brokenvert</code>	\eth	<code>\eth</code>	\backslash	<code>\backslash</code>
















Astronomical Symbols

\ascnode	<code>\ascnode</code>	\mercury	<code>\mercury</code>	τrus	<code>\taurus</code>	\sagittarius	<code>\sagittarius</code>
------------	-----------------------	------------	-----------------------	-----------	----------------------	----------------	---------------------------





























	<code>\descnode</code>		<code>\jupiter</code>		<code>\gemini</code>		<code>\capricornus</code>
	<code>\male</code>		<code>\saturn</code>		<code>\cancer</code>		<code>\aquarius</code>
	<code>\female</code>		<code>\uranus</code>		<code>\virgo</code>		<code>\pisces</code>
	<code>\earth</code>		<code>\neptune</code>		<code>\libra</code>		<code>\conjunction</code>
	<code>\sun</code>		<code>\pluto</code>		<code>\scorpio</code>		<code>\opposition</code>




















Block Shapes

	<code>\blacktriangle</code>		<code>\circ</code>		<code>\bigtriangleup</code> <code>\triangle</code>
	<code>\blacktriangledown</code>		<code>\bullet</code>		<code>\bigtriangledown</code>
	<code>\blacktriangleleft</code>		<code>\bigcirc</code>		<code>\vartriangle</code>
	<code>\blacktriangleright</code>		<code>\star</code>		<code>\triangledown</code>
	<code>\halfleftcirc</code>		<code>\blackstar</code>		<code>\leftblacktriangle</code>
	<code>\halfrightcirc</code>		<code>\square</code>		<code>\rightblacktriangle</code>
	<code>\blackhalfleftcirc</code>		<code>\blacksquare</code>		<code>\lozenge</code>
	<code>\blackhalfrightcirc</code>		<code>\diamond</code>		<code>\blacklozenge</code>

















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
































Geometrical Symbol Shapes

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

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

Boxed Binary Operators

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

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

Binary Operators

$+$	<code>\plus</code>	\pm	<code>\pm</code>	\mp	<code>\mp</code>	\cdot	<code>\cdot</code>
-----	--------------------	-------	------------------	-------	------------------	---------	--------------------

$-$	<code>\minus</code>	\cup	<code>\cup</code>	\cap	<code>\cap</code>	\cdot	<code>\centerdot</code>
\times	<code>\times</code>	\uplus	<code>\ucup</code>	\oplus	<code>\nplus</code>	\wr	<code>\wr</code>
$*$	<code>\ast</code>	\sqcup	<code>\sqcup</code>	\sqcap	<code>\sqcap</code>	\moo	<code>\moo</code>
\div	<code>\div</code>	\wedge	<code>\wedge</code> <code>\land</code>	\vee	<code>\vee</code> <code>\lor</code>	\merge	<code>\merge</code>
\vartimes	<code>\vartimes</code>	\curlywedge	<code>\varcurlywedge</code>	\curlywedge	<code>\varcurlywedge</code>	\bigcirc	<code>\varbigcirc</code>
$\dot{+}$	<code>\dotplus</code>	\ominus	<code>\minuso</code>	ϕ	<code>\baro</code>	\talloblong	<code>\talloblong</code>
\intercal	<code>\intercal</code>	$//$	<code>\sslash</code>	\bbslash	<code>\bbslash</code>	\oblong	<code>\oblong</code>
\circ \circ	<code>\fatsemi</code>	\fatslash	<code>\fatslash</code>	\fatbslash	<code>\fatbslash</code>	\pointleft	<code>\pointleft</code>
\divotimes	<code>\divideontimes</code>	$\&$	<code>\binampersand</code>	\bindnasrepma	<code>\bindnasrepma</code>	\pointright	<code>\pointright</code>

$\overline{\wedge}$	<code>\doublebarwedge</code> <code>\Barwedge</code>	$\overline{\wedge}$	<code>\barwedge</code>	\veebar	<code>\veebar</code>	\wedge	<code>\pointup</code>
\leftthreetimes	<code>\leftthreetimes</code>	\cap	<code>\Cap</code> <code>\doublecap</code>	\cup	<code>\Cup</code> <code>\doublecup</code>	\vee	<code>\pointdown</code>
\rightthreetimes	<code>\rightthreetimes</code>	\curlywedge	<code>\curlywedge</code>	\curlyvee	<code>\curlyvee</code>		
		\ltimes	<code>\ltimes</code>	\rtimes	<code>\rtimes</code>		

Relation Comparer

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

\nlessdot	<code>\lvertneqq</code>	\ngtrdot	<code>\gvertneqq</code>	\subsetplus	<code>\subsetplus</code>	\supsetplus	<code>\supsetplus</code>
\lessdot	<code>\lessdot</code>	\gtrdot	<code>\gtrdot</code>	\subsetpluseq	<code>\subsetpluseq</code>	\supsetpluseq	<code>\supsetpluseq</code>

Miscellaneous Relations

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

































\oplus	<code>\inplus</code>	\niplus	\bumpeq	\Bumpeq
\therefore	<code>\therefore</code>	\because	\eqcirc	\circeq
$ $	<code>\mid</code>	\parallel	\interleave	\pitchfork
$ $	<code>\shortmid</code>	\parallel		\between

Negated Relations








































\nless	\ngtr	\nprec	\nsucc
\nleq	\ngeq	\npreceq	\nsucceq
\lneq	\gneq	\precneqq	\succneqq
\nleqslant	\ngeqslant	\precnsim	\succnsim
\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	\nshortparallel	\varsubsetneqq	\varsupsetneqq
\nvdash	\nVdash	\ntriangleleft	\ntriangleright
\nvDash	\nVDash	\ntrianglelefteq	\ntrianglerighteq

					<code>\ntriangleleft eqslant</code>		<code>\ntriangleleft eqslant</code>
--	--	--	--	---	---	---	---

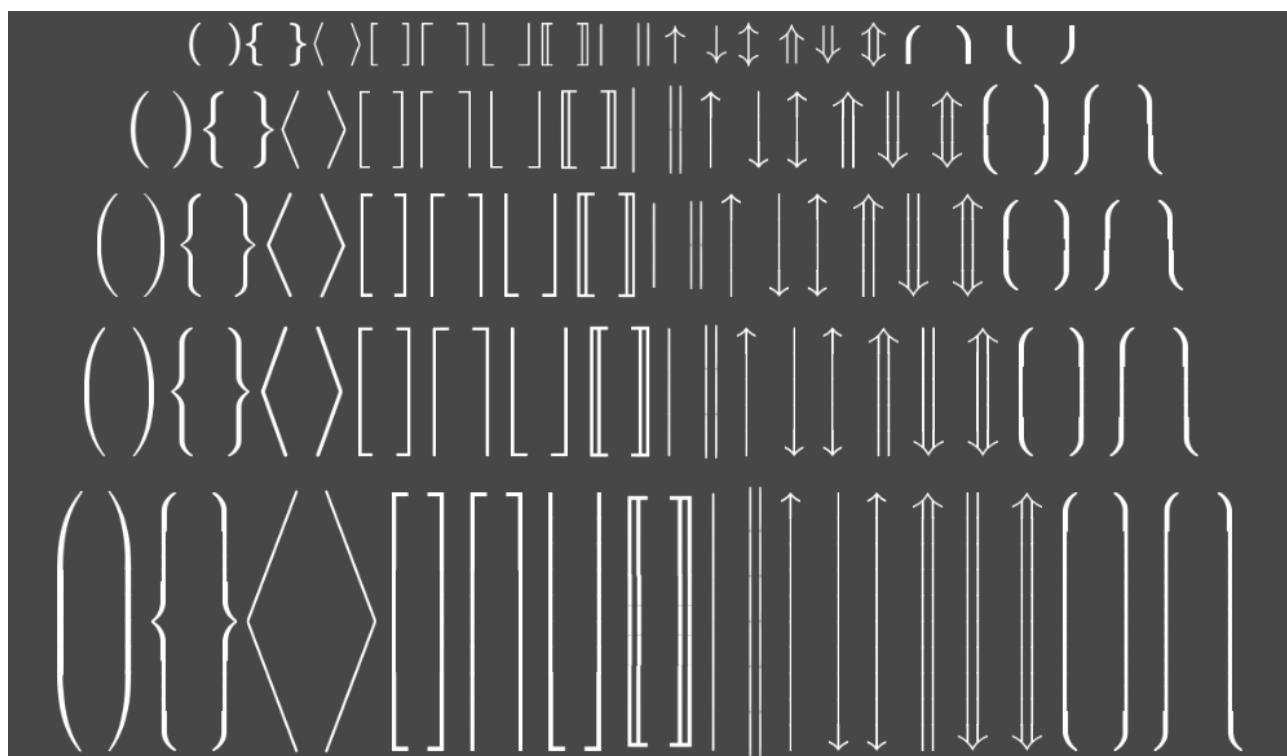
Primary Arrows

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

Compound Arrows

	<code>\shortup arrow</code>		<code>\upuparrows</code>		<code>\leftright harpoons</code>		<code>\curlyvee uparrow</code>
	<code>\shortdown arrow</code>		<code>\downdown arrows</code>		<code>\rightleft harpoons</code>		<code>\curlyvee downarrow</code>
	<code>\shortleft arrow</code>		<code>\leftleft arrows</code>		<code>\leftright arrows</code>		<code>\curlywedge uparrow</code>
	<code>\shortright arrow</code>		<code>\rightright arrows</code>		<code>\rightleft arrows</code>		<code>\curlywedge downarrow</code>
	<code>\nleftarrow</code>		<code>\Lsh</code>		<code>\twohead leftarrow</code>		<code>\Rrightarrow</code>
	<code>\nrightharrow</code>		<code>\Rsh</code>		<code>\twohead rightharrow</code>		<code>\Lleftarrow</code>
	<code>\nLeftarrow</code>		<code>\looparrowleft</code>		<code>\rightsquig arrow</code>		<code>\leftright arrowtriangle</code>
	<code>\nRrightarrow</code>		<code>\looparrowright</code>		<code>\leftright squigarrow</code>		<code>\leftarrow triangle</code>
	<code>\nleftrightarrow</code>		<code>\leftarrowtail</code>		<code>\leftright arroweq</code>		<code>\rightarrow triangle</code>
	<code>\nLeftrightarrow</code>		<code>\rightarrowtail</code>				<code>\multimap</code>

Expandable Delimiters



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

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









Open & Closing Delimiter

<code>(</code>	<code>\lbrack</code>	<code>)</code>	<code>\rbrack</code>	<code>[</code>	<code>\lsqbrack</code>	<code>]</code>	<code>\rsqbrack</code>
<code>{</code>	<code>\lbrace</code>	<code>}</code>	<code>\rbrace</code>	<code><</code>	<code>\langle</code>	<code>></code>	<code>\rangle</code>
<code>⌈</code>	<code>\lceil</code>	<code>⌋</code>	<code>\rceil</code>	<code>⌊</code>	<code>\lfloor</code>	<code>⌋</code>	<code>\rfloor</code>
<code>⌈</code>	<code>\lgroup</code>	<code>⌋</code>	<code>\rgroup</code>	<code>⌈</code>	<code>\lmoustache</code>	<code>⌋</code>	<code>\rmoustache</code>

$\{$	<code>\lbag</code>	$\}$	<code>\rbag</code>	$\{$	<code>\Lbag</code>	$\}$	<code>\Rbag</code>
\llbracket	<code>\llbracket</code>	\rrbracket	<code>\rrbracket</code>	\llparenthesis	<code>\llparenthesis</code>	\rrparenthesis	<code>\rrparenthesis</code>
\llfloor	<code>\llfloor</code>	\llrfloor	<code>\llrfloor</code>	\llceil	<code>\llceil</code>	\llrceil	<code>\llrceil</code>
















Big Operator

\int	<code>\int</code>	\int	<code>\varint</code>	\iint	<code>\iint</code>	\iiint	<code>\iiint</code>
\oint	<code>\oint</code>	\oint	<code>\varoint</code>	\oiint	<code>\oiint</code>	\parallel	<code>\bigparallel</code>
\sum	<code>\sum</code>	\prod	<code>\prod</code>	\coprod	<code>\coprod</code>	\biginterleave	<code>\biginterleave</code>
\bigcup	<code>\bigcup</code>	\bigcap	<code>\bigcap</code>	\bigoplus	<code>\bigoplus</code>	\bigboxplus	<code>\bigboxplus</code>
\bigsqcup	<code>\bigsqcup</code>	\bigsqcap	<code>\bigsqcap</code>	\bigotimes	<code>\bigotimes</code>	\bigtriangledown	<code>\bigtriangledown</code>

	<code>\biguplus</code>		<code>\bignplus</code>		<code>\bigodot</code>		<code>\bigtriangleup</code>
	<code>\bigwedge</code> <code>\bigland</code>		<code>\bigvee</code> <code>\biglor</code>		<code>\bigcurlyvee</code>		<code>\bigcurlywedge</code>

Accent

These accents can be applied after a digit or symbol

	<code>\acute</code>		<code>\tilde</code>		<code>\check</code>		<code>\dot</code>
	<code>\grave</code>		<code>\bar</code>		<code>\hat</code>		<code>\vec</code>
	<code>\floatquote</code>		<code>\floatband</code>		<code>\Dot</code>		<code>\ddot</code>
			<code>\breve</code>		<code>\widehat</code>		<code>\widetilde</code>

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
<code>+</code>	<code>\plus</code>	<code>[</code>	<code>\lsqbrack</code>	<code>;</code>	<code>\semicolon</code>	<code>?</code>	<code>\question</code>
<code>-</code>	<code>\minus</code>	<code>]</code>	<code>\rsqbrack</code>	<code>:</code>	<code>\colon</code>	<code>!</code>	<code>\ldotp</code>
<code>*</code>	<code>\ast</code>	<code><</code>	<code>\lt</code>	<code>`</code>	<code>\vert</code>	<code>@</code>	<code>\commercialat</code>
<code>/</code>	<code>\slash</code>	<code>></code>	<code>\gt</code>	<code>~</code>	<code>\question</code>	<code>#</code>	<code>\numbersign</code>
<code>=</code>	<code>\equals</code>	<code> </code>	<code>\vert</code>	<code>'</code>	<code>\faculty</code>	<code>\$</code>	<code>\dollar</code>
<code>(</code>	<code>\lbrack</code>	<code>.</code>	<code>\ldot</code>	<code>"</code>	<code>\ampersand</code>	<code>%</code>	<code>\percent</code>
<code>)</code>	<code>\rbrack</code>	<code>,</code>	<code>\comma</code>			<code>&</code>	<code>\ampersand</code>

Appendix: A side note to the users

Upgrading TEXDraw package in a project

Before you apply a TEXDraw upgrade from 1.0 to 2.0, please **Backup your Project** first.

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)

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**.

2

Check and Clean the Junk

After import, the package structure will look like this, now delete **Plugin** and **Resources** folder!

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.

Proper Upgrade Step Procedure for further Version of TEXDraw

The upgrade step here is intended for upgrading current TEXDraw package with version 2.0 or above to new further version, before you do, always **backup** your project first.

- First, **Export** your existing preference to XML File,
- **Import** the new package, and wait until import file selection shows,
- After it has showed, Import (select) everything **Except the XML Files** (TEXDraw/XMLs)
- After importing done, in TEX Preference, **Import back** your last preferences from XML File.

TEXDraw Release Notes

2.3 – Apr 20, 2016: Performance Upgrade

- New Autowrap and Justify alignment
- New Resource Pooling System, up to 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> (SIL Open License, prevent selling)

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 a constant string formula (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 less efficient, 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.).

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");
```

About This Package

This package is provided by Wello Soft,

Check out other assets: <http://u3d.as/cc0>

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/>