TEXDraw

LaTeX Graphic Mathematical Expressions Input for Unity

Documentation Manual for V4.1

For list of TEXDraw syntaxes and symbols please refer to Documentation Reference
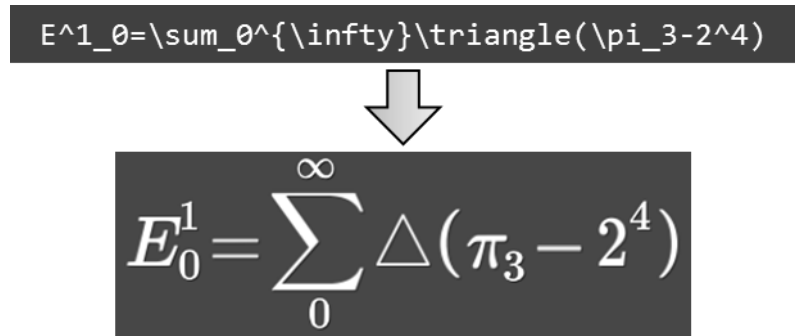
Table of Contents

FAQ About TEXDraw

What is TEXDraw?

TEXDraw is a Component that makes a plain text can be converted into a graphical representation of mathematical formulas. TEXDraw draws mathematical formulas using the similar approach introduced in LaTeX writing system. The results

```
E^1_0=\sum_0^{\infty}\triangle(\pi_3-2^4)
```

$$E_0^1 = \sum_0^{\infty} \triangle(\pi_3 - 2^4)$$

is a 3D mesh that can be used inside Unity's UI or other mesh-based rendering systems.

How it does work and why it different?

TEXDraw, just like many other text generators, 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 are used in various apps like educational software, scientific simulations, and many more. With the power of Unity's built-in UI System and dynamic text support, generating math expressions was never so easier and seamlessly than ever!

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

A lot, including:

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

How easily I can make it work on my project?

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

Will this work on all platform? Any performance issues?

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

Do this package provide support for other External Asset?

Yes, TEXDraw supports drawing text into NGUI environment or using the benefit of SDF Rendering from TextMeshPro. [Click here](#) for instruction setup.

Can TEXDraw receive Text Input?

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

Note that if you want a clickable (button-like) behaviour in portion of text, then we have that.

I have a trouble. Any suggestion for me?

See [troubleshoot page](#) for common problems.

Inside of TEXDraw Package

The TEXDraw Component

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

Aside from **Text**, there are other optional properties that are quite useful for handling display output. Below is Description of each property inside this Component:
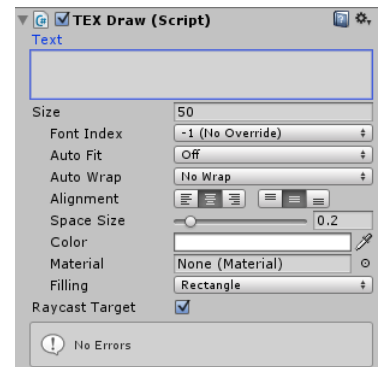
|	
Text	A plain text that you want input to. See here for practical guide to write, and here for scripting instruction.
Size	Size of generated graphics The font texture size will automatically resized to be detail enough to display on screen
Font Index	Index of used default font (-1 to follow default typeface rules) In scripting, you can set this as integer value. Each number represents an index of font that registered in the Font Stack.
Auto Fit	How final graphic is scaled when it's render is out of the rectangle bound See here for available options
Auto Wrap	Horizontal wrapping mode if a line is beyond rectangle's horizontal bound See here for available options
Alignment	The horizontal and vertical alignment of the text. In scripting, this is a Vector2 property. A value of {0, 0} represent left-bottom alignment
Space Size	The space size on each lines The actual space size is proportional to what given in Size property.
Color	The main color for generated graphics Use \color if you want to write specific color
Material	Assign a Custom Material for This component If none assigned, the default material (from Resource) will be used for rendering
Filling	Optional options for manipulating UV2 data Used in conjunction with custom Material effects, choose the way when font characters are overlaid with some shader graphics

---

Other TEXDraw Variant Components

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

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

Those three variants have the similar functionality and properties. What you type inside in either text will yield the same result.
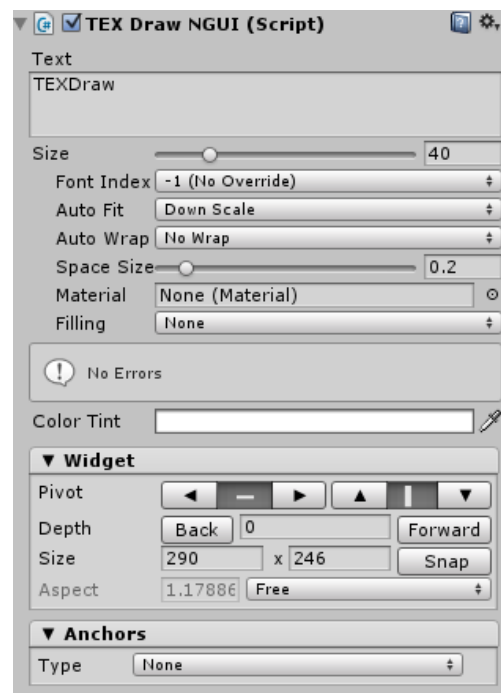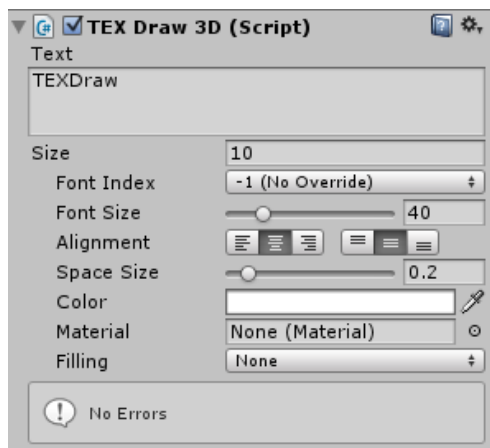
Either these three components, they can be Integrated seamlessly into TextMeshPro's SDF Rendering. If you have TextMeshPro in the project, SDF Rendering in TEXDraw can be activated by Declaring scripting symbol `TEXDRAW_TMP` in Player settings and importing required shaders, which explained in detail [here](#).

Enumeration Choices

Auto Fit is used for what happens to the whole text when generated text is out of the given rectangle layout. The choices are...

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

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

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

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

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

TEX Link Component

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

To make this component works, you need to create at least one `\link{}` command to an expression. TEXLink responds to Mouse and Multi-touchscreen (Keyboard is yet supported).
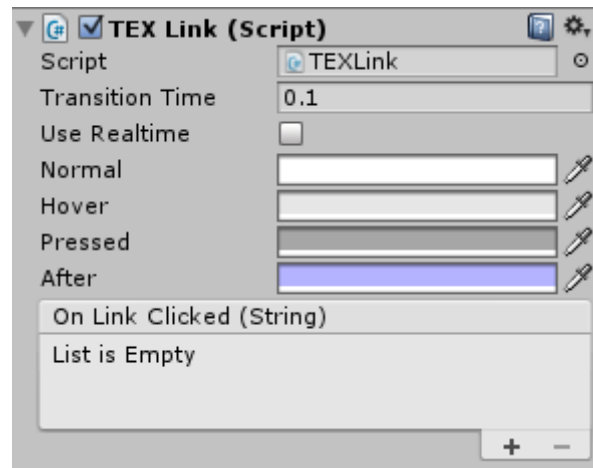
|	
Transition Time	How long the time it taken to change a color.
	Zero to turn off Animation
Use Realtime	Should we ignore Time.timeScale?
	Check this box on if you don't want link freezes when game pauses
Normal	Color tint when the link is yet clicked.
	Tint means the final color is multiplied between this color * color from the character
Hover	Color tint when the mouse just above the link
	This only happens on desktop where mouse devices are exist.
Pressed	Color tint when mouse/touch presses the link
	This one works both mouse/touch (even if the user do multi touches) presses down.
After	Color tine when mouse/touch just been released
	Will reset to normal when this script goes destroyed, or ResetLinks() is invoked.

Below of these properties, there's an UnityEvent class named `OnLinkClicked`. This is where you put your script functions to receive an event when user get clicked the link. There's also a string parameter which will let know which link that user clicks on. For example, if a 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 its [command](#).

TEX Preference

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

TEXDraw Preference holds shared information across one project and saved as arbitrary asset located in `TEXDraw/Resources/TEXDrawPreference.asset`.
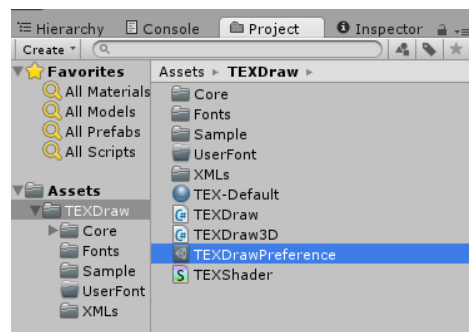
The preference here located inside of TEXDraw root folder as an arbitrary asset.

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

TEXDraw Preference has four main tabs. Each has separate purposes.

|	
Characters	Search, define, manage font stack and symbol definitions. It also can preview characters in single font as a character map.
Configurations	Configure shared (project-wide) parameters for controlling character sizes, margin, fraction gaps, script drops, etc. Each change is applied automatically and each param also has an extra tooltip.
Materials	Manage materials that uses TEXDraw shaders The purpose is so that each material texture slot is synced correctly whenever Font Stack is changed. The management is happens automatically however.
Glue Matrix	controlling custom kerning for each different type of character. The type of individual characters can be configured in 'Characters' Tab

A guide for usability on each tab will be explained in detail in another section of this documentation [here](#).

Font Collections

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

In that folder, there's 3 subfolders in inside of that folder, and each of them has their own purposes:
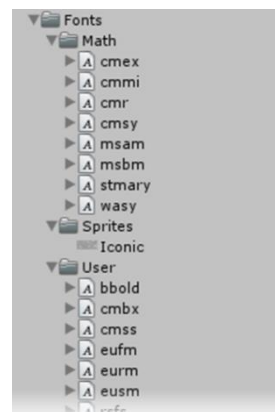
Math	Contain built-in font package for constructing math expression You shouldn't put anything here. It is safe to delete this fonts as long as you don't use it.
User	User font-defined packages Put any fonts here and it will registered and used for TEXDraw components
Sprites	Texture as user font-defined packages Yes! Textures can be imported as a tiled sprite and used just like regular fonts

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

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

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

If you use TextMeshPro integration, then you'll see `TMPro` folder inside them. That is the place where generated TextMeshPro metadata is saved for builds.

Shader Variants

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

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

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

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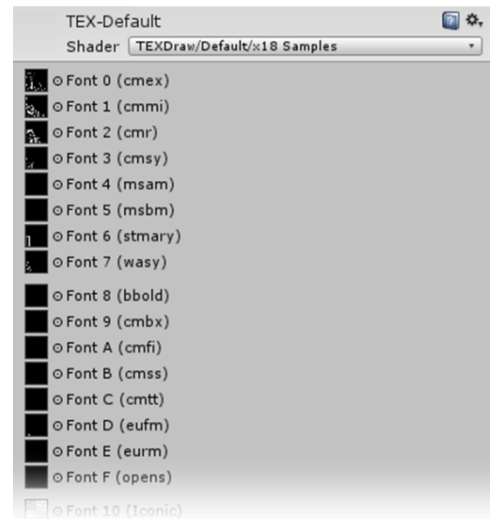
Supplements

A supplement is a component to modify (find & replace) text just before get used in rendering process, automatically. The usage is similar to UI Effects, where you just need to attach a relevant Supplement component beside TEXDraw, and it will change how TEXDraw displayed, instantly.
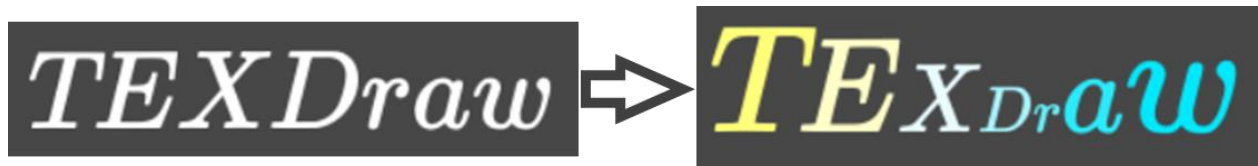
Many supplements were created and included for various user needs.

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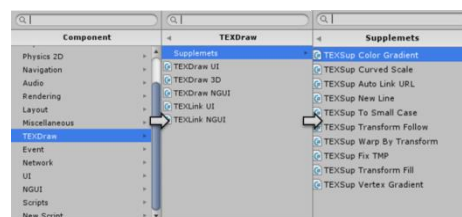
|	
Auto Link URL	Detect and replace any relevant URL or email links
New Line	Treat \n command as a new line
Color Gradient	Attach \color command on every character and interpolate the color based on a Gradient
Curved Scale	Attach \size command on every character and interpolate the scale factor based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

Using & Editing TEX Preference

Preamble

In the previous section, we know how to open TEX Preference and where it's located. Now we'll 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.
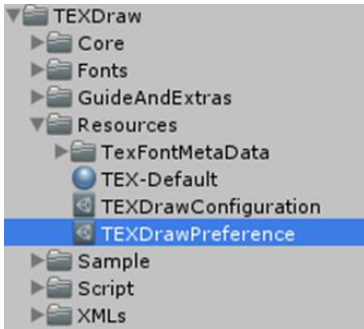

Where it is saved

TEX Preference saves all configuration as a serialized data in the Resources folder. However, to reduce the chance to lose data, TEXDraw split the metadata into several pieces inside the folder. All metadata is organized automatically so you don't have to worry about it.

How does TEX Preference managed and what happen if it lost/corrupt?

In Editor, every time you add a TEXDraw component in your scene, they'll find and locate where the TEX Preference live in your project. In case if missing, it'll create a new one inside Resources and reinitialize themselves.

NOTE: Losing serialized data is still harmful as you'll lose some preconfigured data. It's helpful to track your project changesusing Git or other version control.
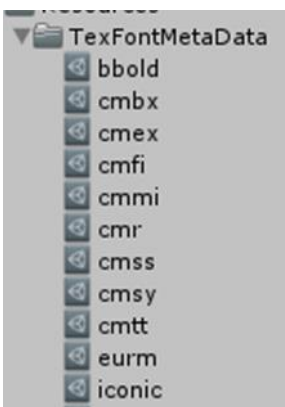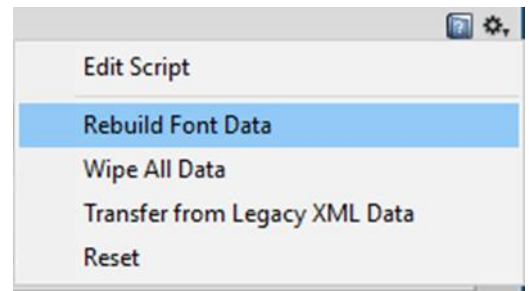
How one Preference can be splitted up to many kinds of Arbitrary asset?

In previous section I have mentioned that there are 4 tabs inside preference.

The first tab is used to manage fonts and displaying each characters. Each font hold separate metadata which contained in TexFontMetaData.

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 further 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 further later.

Add your own font/sprite to the TEXDraw font stack

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. (For sprites only) if you were importing a texture, then you need to split them into multi-sprite pack.
3. Rebuild font data. This will trigger the importer to register the font you've add earlier.
4. Now your font is registered. To use it, simply define your own symbol, or leave as it is.
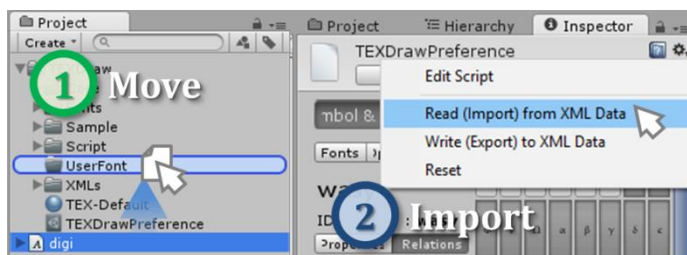
Configure How Textures/Sprite are Imported
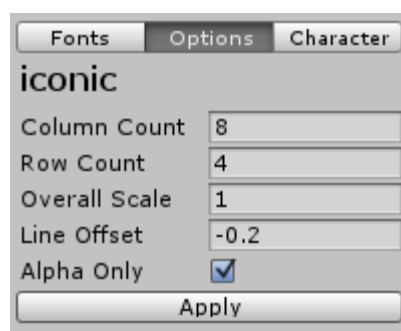
Textures are imported as it is contains bunch of sprites that have an equal dimension (grid-style). You can tell to the importer about how much column and row it has by set-up the column and row count. Optionally, overall scale for how large the texture size, and line offset for adjusting the "vertical 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.
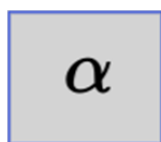
As you can see in the screenshot on the right, there's different box style applied on each character. These different styles tell us about what's state that they're on. Take a look of these previews to make it clearer:
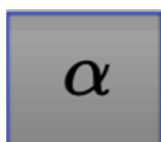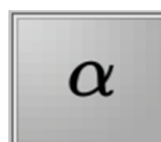
Char is Defined	**Char is Related**	**Char is Available**	**Not Available**
The character has its own symbol definition	The character doesn't defined but it has relationship with other character	The character is yet defined nor related but still available.	The character doesn't exist and can't be used 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 to 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

a font which the index is 2 (cmr), and its character index is #6A (in digits, 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.
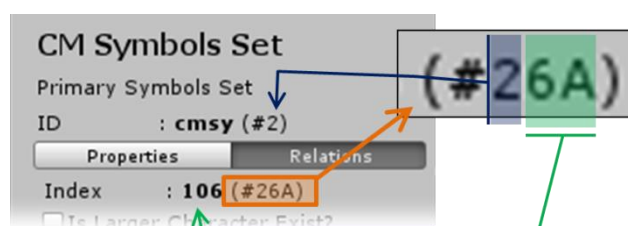
Please note that for validity of character hash, in hex display, the first two digits should be ranging from #0 to #FF (0 to 256), while third digit should be ranging from #0 to #1E (0 to 31), or in short, maximum possible value of a hash is #1EFF.

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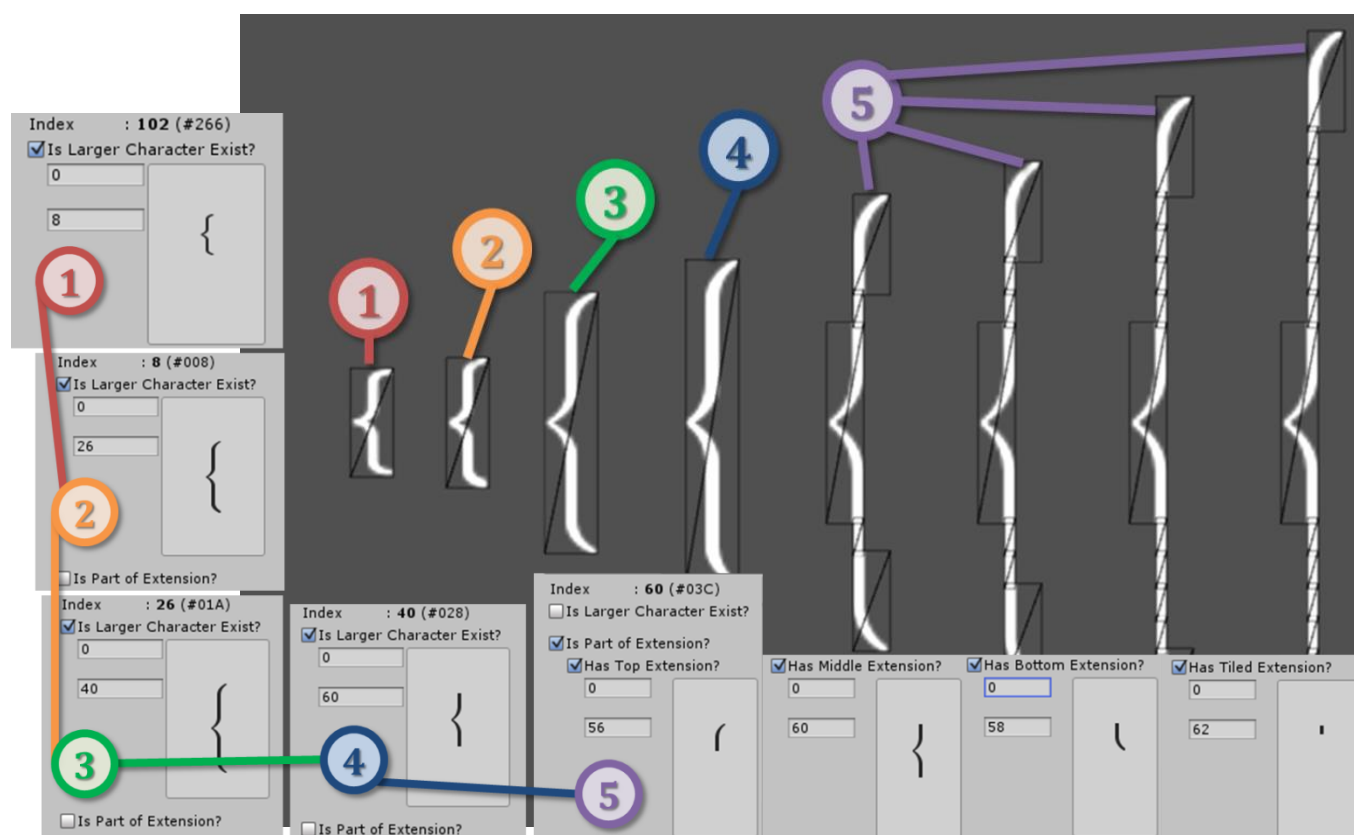
The idea is simple. This image shows every "level" and each "relation" character configuration of a delimiter \lbrace, which we note on each level in a number. \lbrace has up to 5 levels height. In the first level, a character with hash #266 holds the \lbrace symbol definition, and does refer to a larger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

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

 Please **be sure** that only symbols that have type of Relation, Arrow, Open Delimiter, and Close Delimiter can have relations feature like above, so check the character type support this feature!
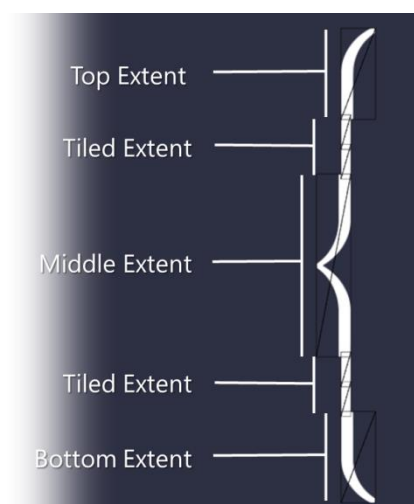
Tab 2: Global Configurations

In this tab, a lot of customizable parameters are provided to control project-wide setups like margin, scale, offset, shift of specific expression. Each parameters has been equipped with tooltip for extra explanation, though also in this section we might helps you more understand it behaviors.

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:
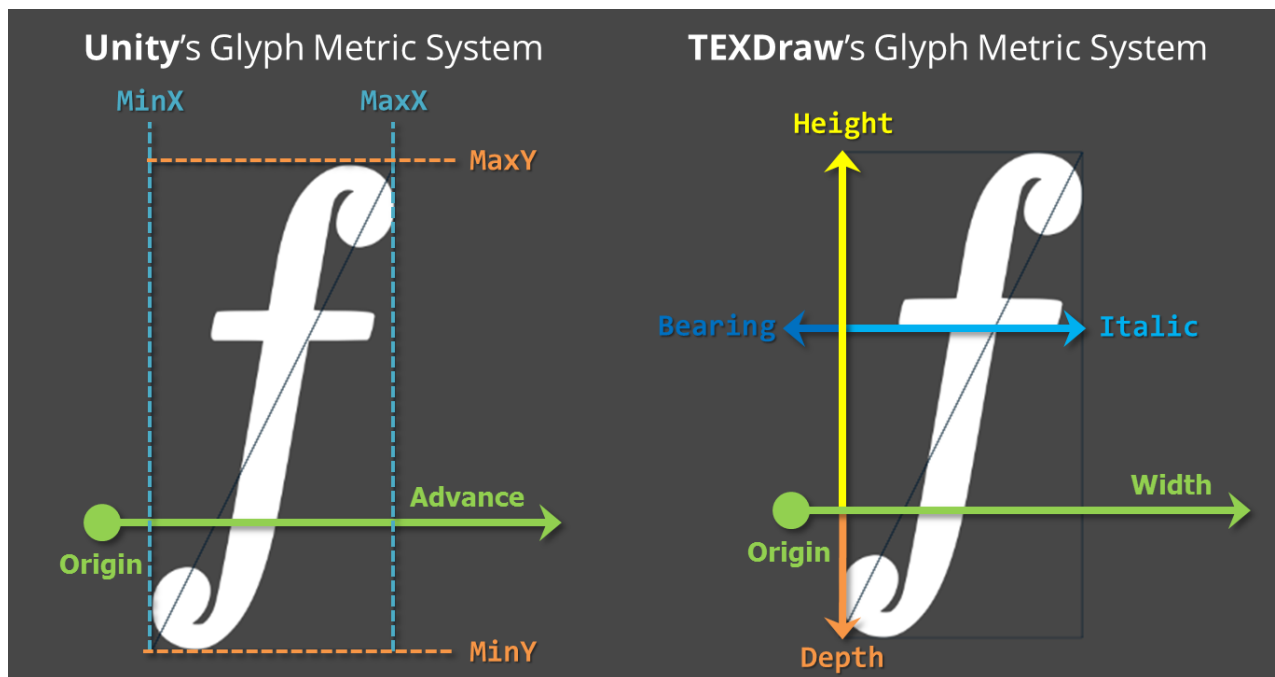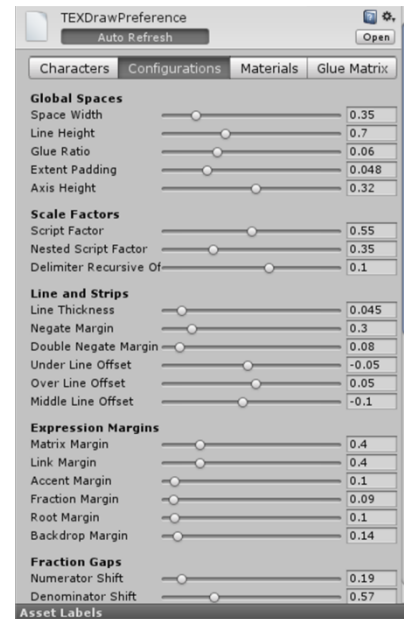

A character has a baseline (marked as green line), top and low bound. The image above will help you to understand each config which we will discuss below:
(We provide each config explanation with relevant image and indicator colors)

Line Height — Minimum height of a line
Height is the distance from top character to baseline. This is the minimum value.

Line Depth — Minimum depth of a line
Just like line height, but distance from bottom.

**Line Space** — Minimum space between line
The fixed space between line
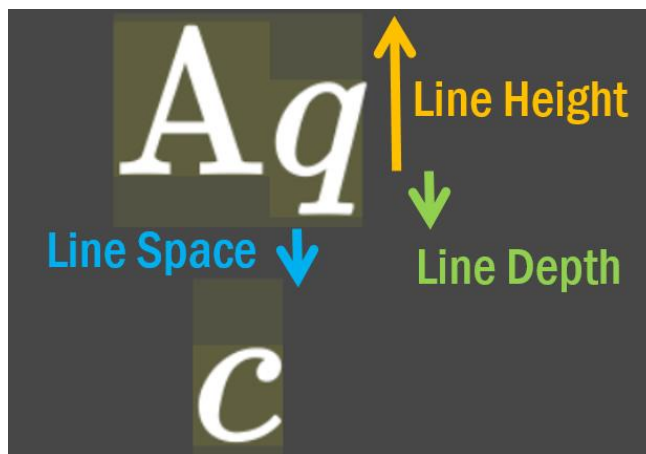
Space Width — The width of a single whitespace
The width is on fixed value.

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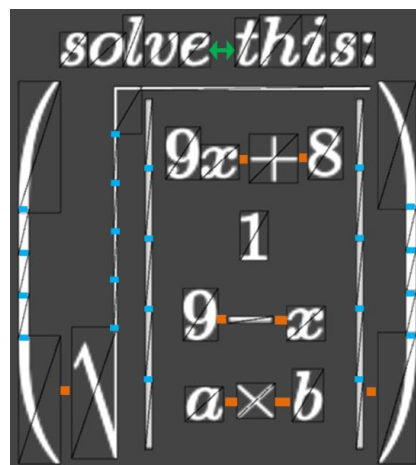
Script Factor — Size Ratio of a Script
The final script total height in percentage compared to standard character size.

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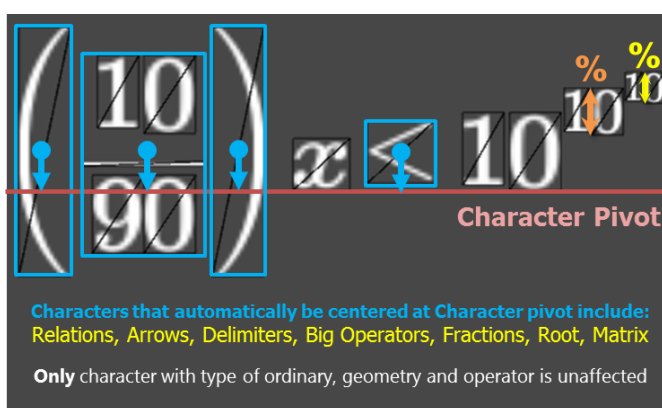
Line Thickness — .Fixed Line Thickness Width
Line thickness for common things (fraction, root, negations, etc.)

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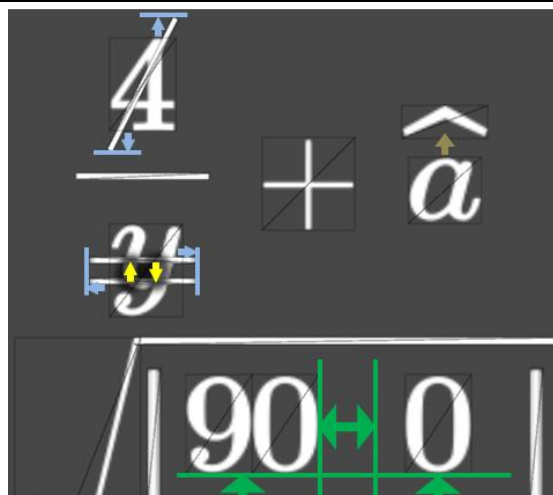
Over Under Margin	Additional Accent gap height Shift Accent position upward until certain height (calculated from character's top bound).

Fraction Margin	**Additional fraction line width** Additional line width for fractions.
Numerator Shift	**Standard Numerator Margin** Numerator's lift amount from line base 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.).
Denominator Shift	**Standard Denominator Margin** Denominator's lift amount from line base to the fraction line. If it less than the char height, it will be "clamped" instead.
Denominator 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 (e.g. 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
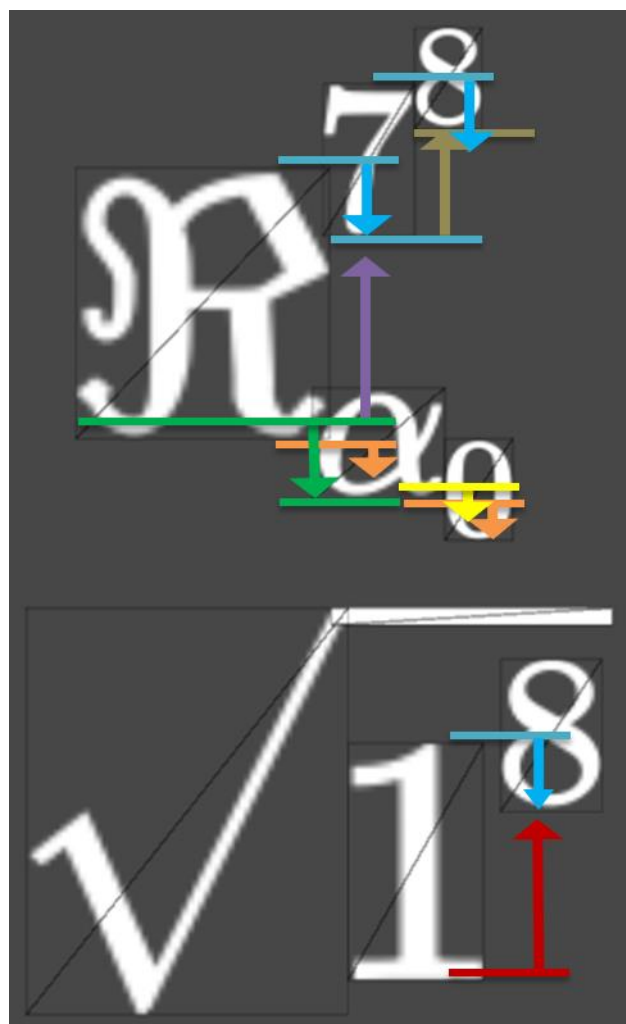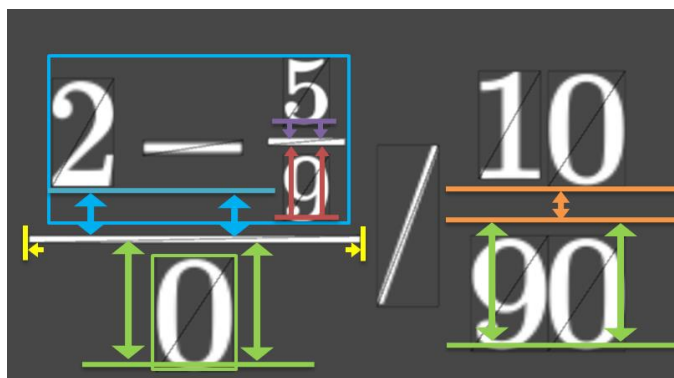
|	
Sub Min On Sup	**Subscript Minimum Drop (With Superscript Above)** Similar like SubMinNoSup, but will used instead if superscript exist on same level.

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	
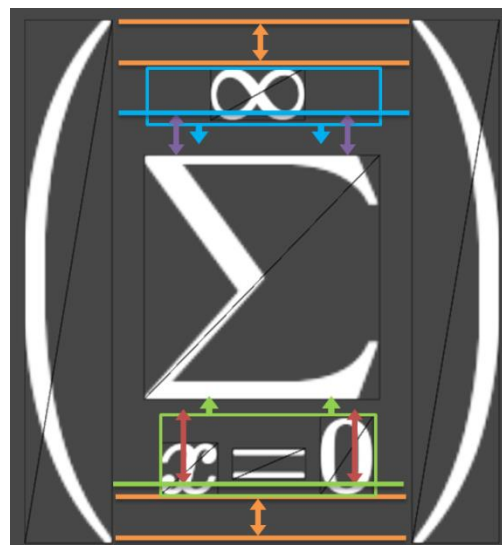
Default Typefaces, what is it?

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).

Tab 3: 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.
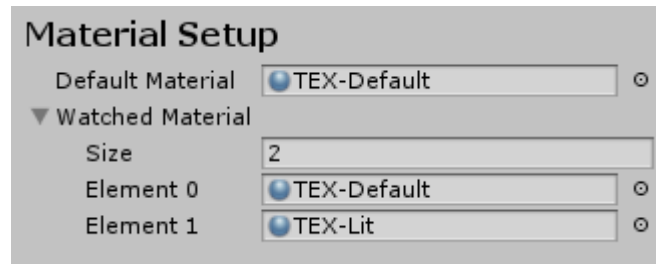
Tab 4: Glue Matrix

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.

Appendix: Additional Info

Upgrading To V4.0 (and Upwards)

Upgrading To V3.0

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

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

1. Save current scene and load new empty scene

2. (if project holds V2.x) → Export current preference to XML files (prevent data lost)

3. Delete All files & directory, except `Fonts`, `Resources` (for 3.x), and `XMLs` directory

4. Now Import latest version of package

5. Select all, except directories that you keep in step 3 (do not replace existing files)

6. Hit OK and wait until import process finished.

7. Rebuild Fonts by Click menu item in `Tools > TEXDraw > Rebuild Font Data`

8. (if you do step 2) → Transfers back properties from XML by selecting Preference (`Tools > TEXDraw > Open Preference`), then in Gear Menu, click `Legacy Transfer from XML Data`

9. Profit! Now Enjoy Upgraded version of TEXDraw.
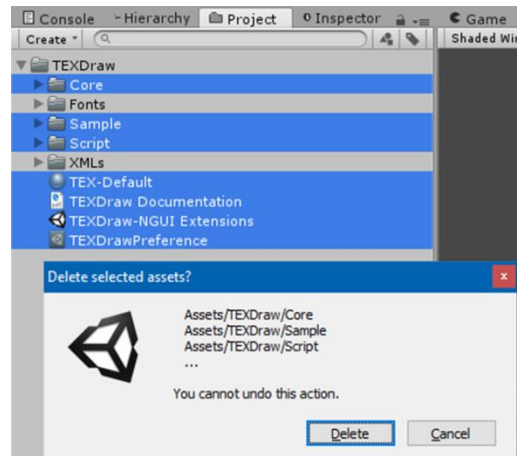
How to make NGUI integration works.
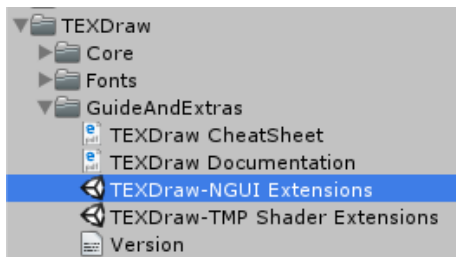
You can navigate to `Tools/TEXDraw/Enable NGUI Integration` menu and it'll ready to use. Otherwise you can do the manual step by extracting the package inside `TEXDraw-NGUI Extension.unitypackage` from TEXDraw `GuideAndExtras` folder.

After that you can add TEXDraw for NGUI by navigating to `NGUI/Create/TEXDraw`

How to make TextMeshPro integration works.

After TMP and TEXDraw is in the project, you need to navigate and click this menu:
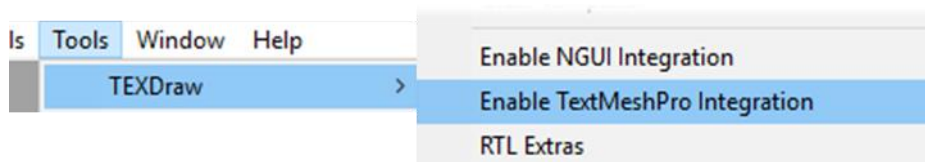
You'll be asked to choose between 'Automatic' and 'Manual'
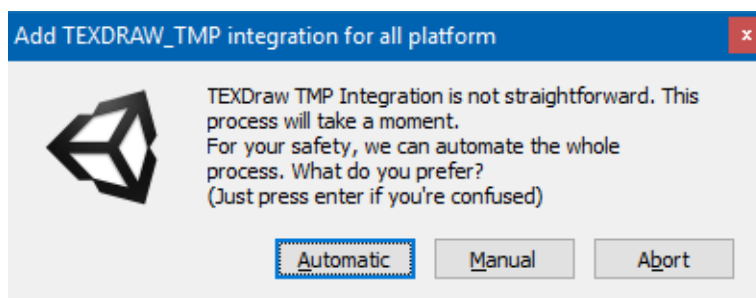
For first time, I'd recommend to follow the automatic process. Otherwise, you can step in the manual process:

1. Click the Manual, then It will give you some hints to these next steps:

2. Extract shader package in `TEXDraw/GuideAndExtras/TEXDraw-TMP Shader Extensions.unitypackage`
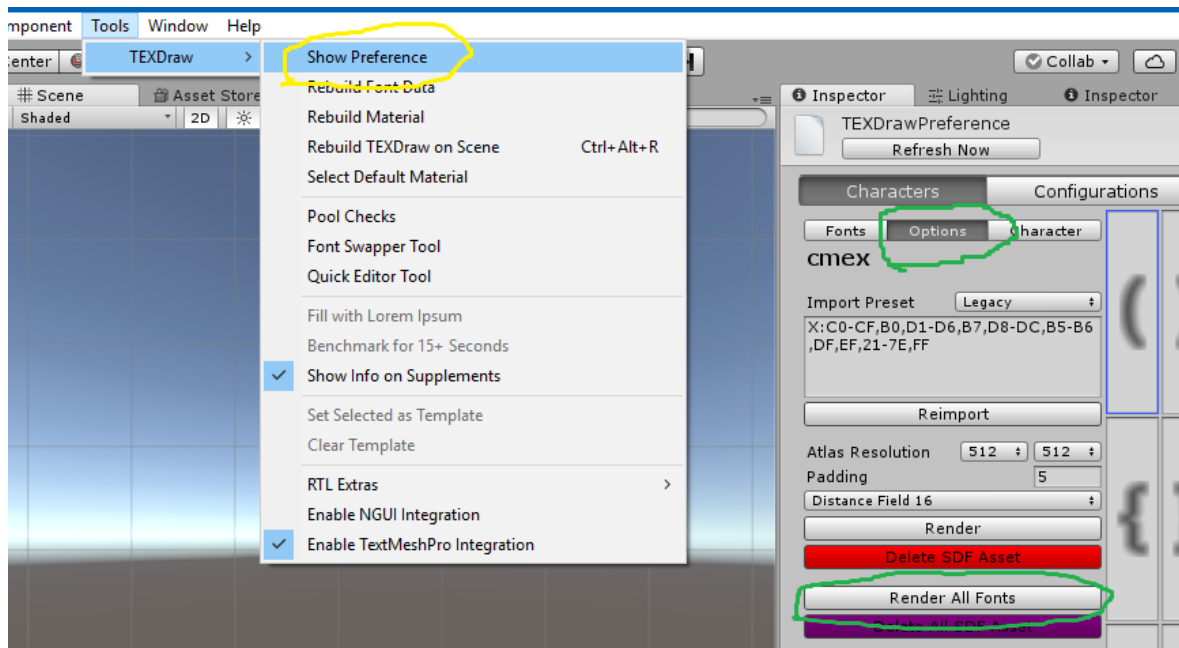
3. Open TexDraw Preference (`Tools > TEXDraw > Show Preference`) and navigate to Symbol & Relation, then options, then Click `Render All Fonts`. This action will render all fonts (in background), so it might take awhile.

(Suggested resolution is 512x256 and padding = 2 for faster rendering)
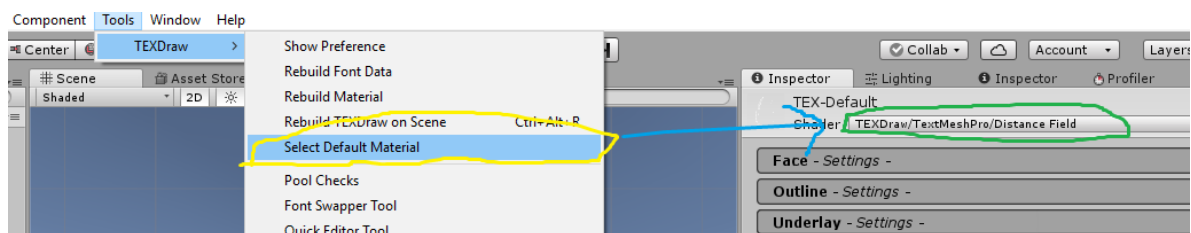
4. After rendering complete, Open Default Material (`Tools > TEXDraw > Select Default Material`) and set the shader to `TEXDraw > TextMeshPro > Distance Field`

5. You are ready. Create one by `UI > TEXDraw` and start type right there.

License notices for Included Fonts

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

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

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

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

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

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

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

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

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

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

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

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

Guide to Write in TEXDraw (Runtime Script)

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

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

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

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

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

Look like simple, but if you write a lot of backslashes you will find this is just not efficient. A better solution is write a verbatim string literals (i.e. 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 (i.e. {0} to num1, {1} to num2, etc.).

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

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

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

Troubleshoot for Common Problems

I want to see the complete list of Commands.

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

Expandable delimiters do not works.

Make sure the delimiter that you'd use is comes from **math**, not other type of font. The solution for this is either change font index to -1, or type `\math` surround it, or even type it symbol name directly, like `)` becomes `\rbrack`.

Best Fit mode is very expensive.

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

Font textures frequently cleared up/blank on Editor.

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

How to make Bold/Italic styling works?

You can bring together the styling into the project, but it won't necessary to add in TEXDraw font list. TEXDraw follow Unity's way to provide styling, and you can do that by typing \opens[i]. If you had problem to referencing these fonts, this good QA might solves your problem.
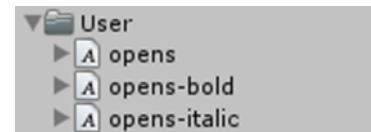
Although these three is in valid folder, those bold/italic is actually ignored because of invalid name (contain '-')

Adding a Font, but nothing happen.

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

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

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

Turning off TextMeshPro integration.

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

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

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

Font Texture is frequently scrambled in the game.

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

Why Shadow and Outline do not work?

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

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

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

Deleting unused fonts/Adding new font in TEXDraw font lists

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

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

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

Why TEXDraw shaders uses too many batches?

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

What is TexFontMetaData in Resources?

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

Any chance for use Unity's new Texture Array?

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

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

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

Informing other bugs/Feature request/General Talks/Misspells

Please refer to forum, or leave us a reply.

About This Package

This package is published by Wello Soft,

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

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

Forum thread available! Head to :

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

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

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

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