Font Setter user manual

Table of Contents

0. Introduction

Unity is capable of displaying text in a rendered scene, via text meshes, using GUIText, or using the UI system. By default, there is no simple way to create fully customised fonts. This editor extension gives the user the freedom to have any created art asset act as a font.

Videos.

For a quick tutorial that covers creation of a Font from start to finish, check this Youtube video:

https://www.youtube.com/watch?v=28Z_4e7S1lE

Limitations.

There are a couple of small limitations to using a bitmap font:

- Fonts created this way cannot be resized in the conventional manner GetComponent(TextMesh).fontSize = 12; By changing the 'font size' parameter in the inspector).
  - TextMeshes have a property 'characterSize', which can be changed, 1.0 is full-size, less than 1.0 is smaller, greater than 1.0 is bigger, Alternatively, you can change the localScale of the TextMesh gameObject transform.
- Fonts created this way will not display fully textured when using UnityGUI Legacy system

  (GUI.Label(Rect(0,0,40,20), "This is a GUI Label", "someGUIStyle").
  - Fonts assigned to UnityGUI via GUIStyles display as silhouette only, so you can still use Font Setter to create the shapes for UnityGUI fonts.
  - There are workarounds for this - sequentially calling DrawTextureWithTexCoords can produce correct results.
  - Unitys Canvas UI system was made to replace UnityGUI, and works fine (with appropriate shader).

1. Terminology

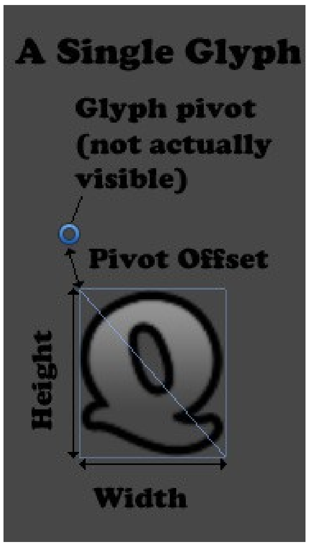
In Unity, fonts can either be displayed by automatically generated textures, or (from Unity 3.0) using the FreeType font rendering engine for 'dynamic' fonts. If you import a font, it will probably be set by default to dynamic. This means that you do not need to provide a texture, and can the font be resized and displayed in large typefaces, without a loss of quality. However, using dynamic fonts, relying on Unity to generate a texture prevents you from using full-color lettering in your Text assets.

It is possible to create your own font textures, of any shape, size and color, and this editor extension will help you define the character parameters.

When Unity draws Text Mesh characters, there are 4 key components in play; they are created according to a 'font settings' object (someFont.fontsettings), drawn using a 'material' (someFont.mat), which in turn references (via a shader, someFontShader.shader) a bitmap texture, or 'font map' (eg. SomeFont.png).

The font settings object defines the measurements shown in [Fig1.].

- Every glyph definition has a height and a width.
- Each glyph has a pivot offset.
- If you imagine a line, like a wire with flags hanging below it, each pivot is positioned on that line, and the glyphs are drawn relative to it.
- Once a glyph is positioned, the next pivot moves the Advance Width distance for the newly drawn glyph.
- Kerning squashes and separates ALL characters in a font.
- Line spacing determines the gap created when using newline characters in strings (pressing return).
- Packing a font map means reducing any gaps in the texture, to enable smaller texture file sizes – this can be important for distribution (download sizes) and graphics memory of the target device.



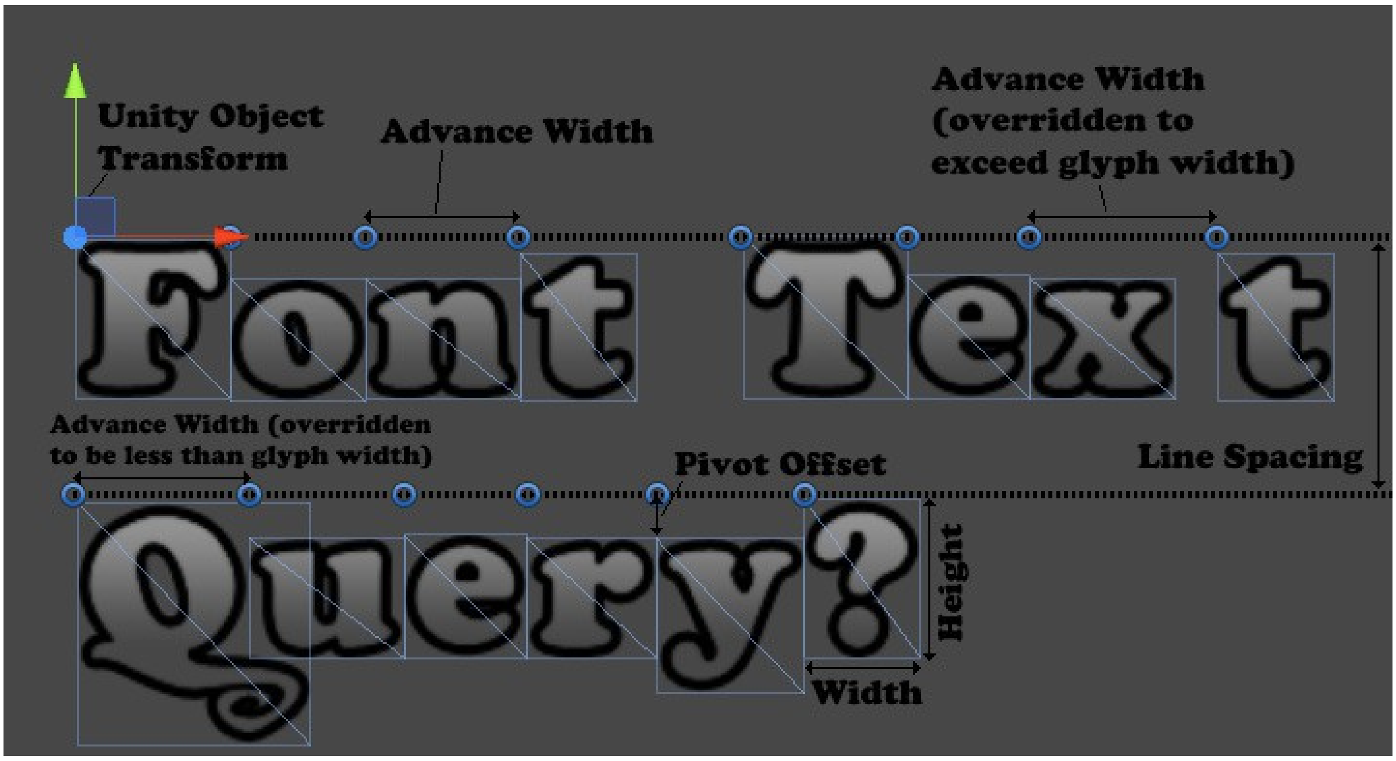
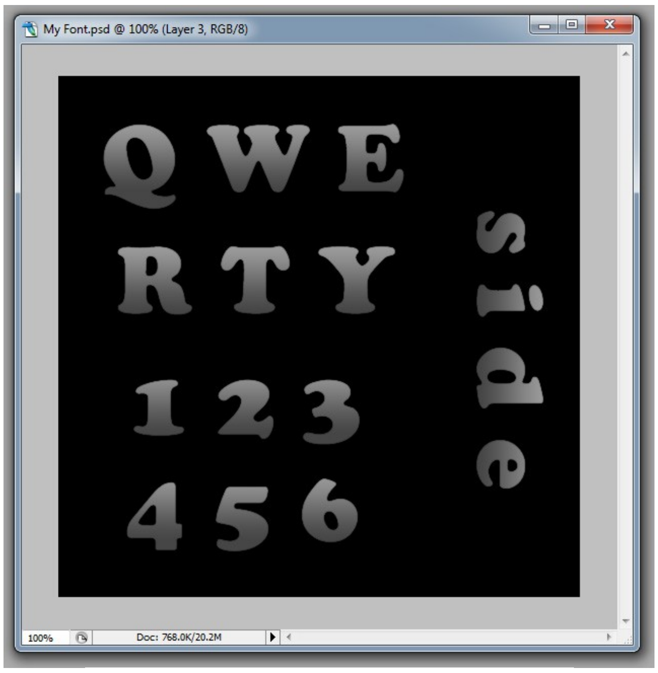

Unity Object Transform
Advance Width
Advance Width (overridden to exceed glyph width)
Advance Width (overridden to be less than glyph width)
Pivot Offset
Line Spacing
Height
Width

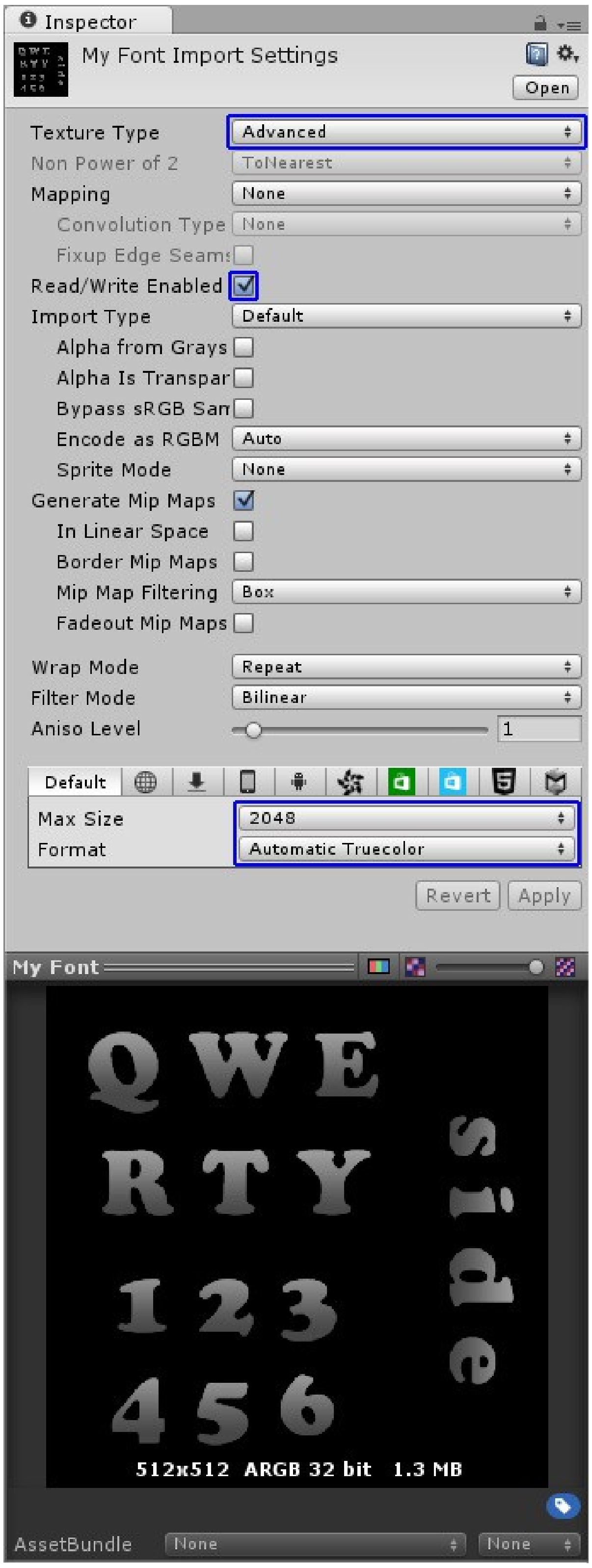
Fig1. Glyph properties and measurements

2. Initial Set-up.

Before using the Font Setter/Packer editor extension, you will need to create a bitmap file (font map) containing your font glyphs. Create this in your preferred image editing software (eg. Photoshop). The file should be either a .png, or something that maintains transparency (eg .psd). In the import settings of the image, you will need to tick 'Read/Write enabled' in the Advanced texture type (to use the advanced features of the editor). You may also need to increase the 'Max Size' to be bigger than the size of the image, or quality will be reduced. Ideally the image should have sizes that are powers of 2 (512,1024,etc), if this is not the case, ensure that the "Non power of 2" setting is "None". Lastly, the image must be in the correct format: Automatic Truecolor works best to ensure no quality loss and so the advanced editor functions correctly read the image; if you need to use a compressed format (eg mobile deployment) wait until the font is finalised to choose a compressed format.



Create font map in image editor (eg. Photoshop)



Inspector

My Font Import Settings

Open

Texture Type	Advanced
Non Power of 2	ToNearest
Mapping	None
Convolution Type	None
Fixup Edge Seams	☐
Read/Write Enabled	✓
Import Type	Default
Alpha from Grays	☐
Alpha Is Transpar	☐
Bypass sRGB San	☐
Encode as RGBM	Auto
Sprite Mode	None
Generate Mip Maps	✓
In Linear Space	☐
Border Mip Maps	☐
Mip Map Filtering	Box
Fadeout Mip Maps	☐
Wrap Mode	Repeat
Filter Mode	Bilinear
Aniso Level	1

Default

Max Size	2048
Format	Automatic Truecolor

Revert Apply

My Font

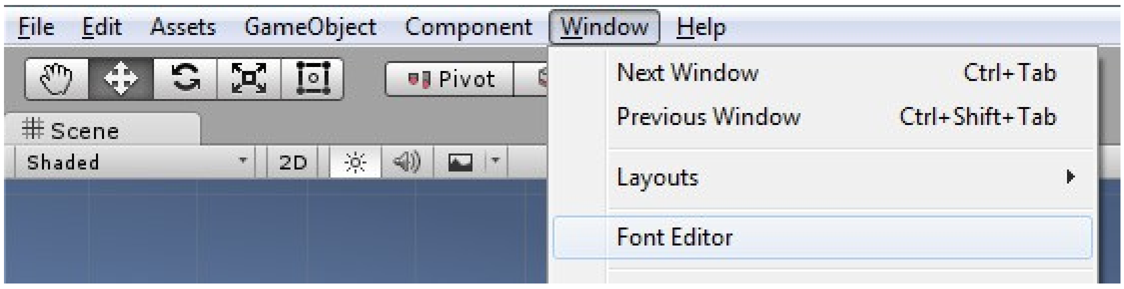
512x512 ARGB 32 bit 1.3 MB

AssetBundle None None

Import settings for font map

3. Operation

To start the Font Setter/Packer editor extension, click on Window > Font Editor in the menu bar of the Unity3D editor.
This should open a new floating, dockable window.



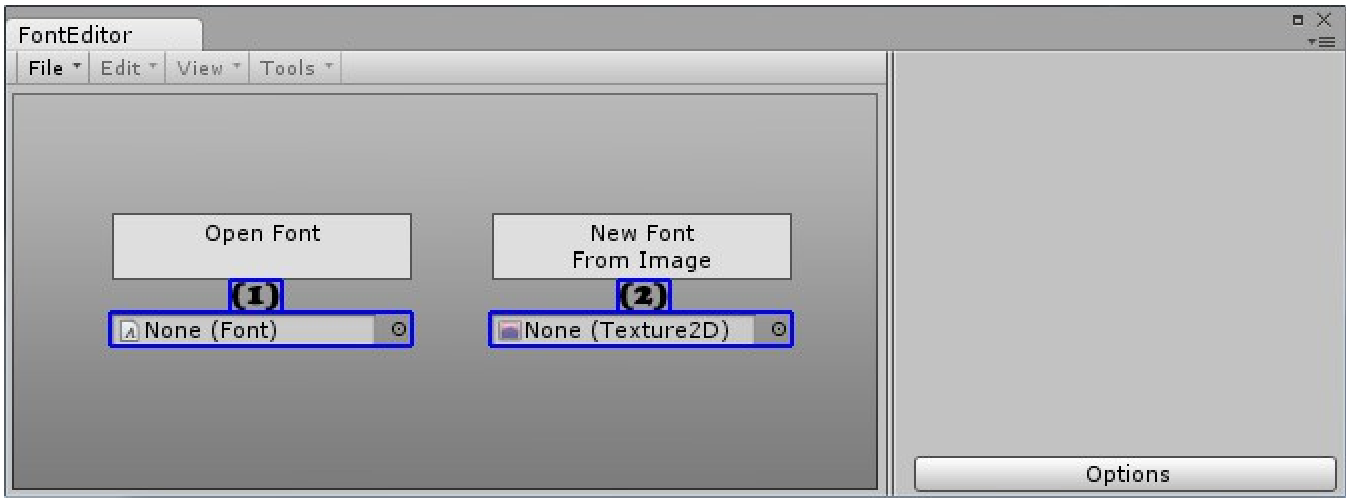


Fig2. The Main Menu

There are 2 slots for project assets.

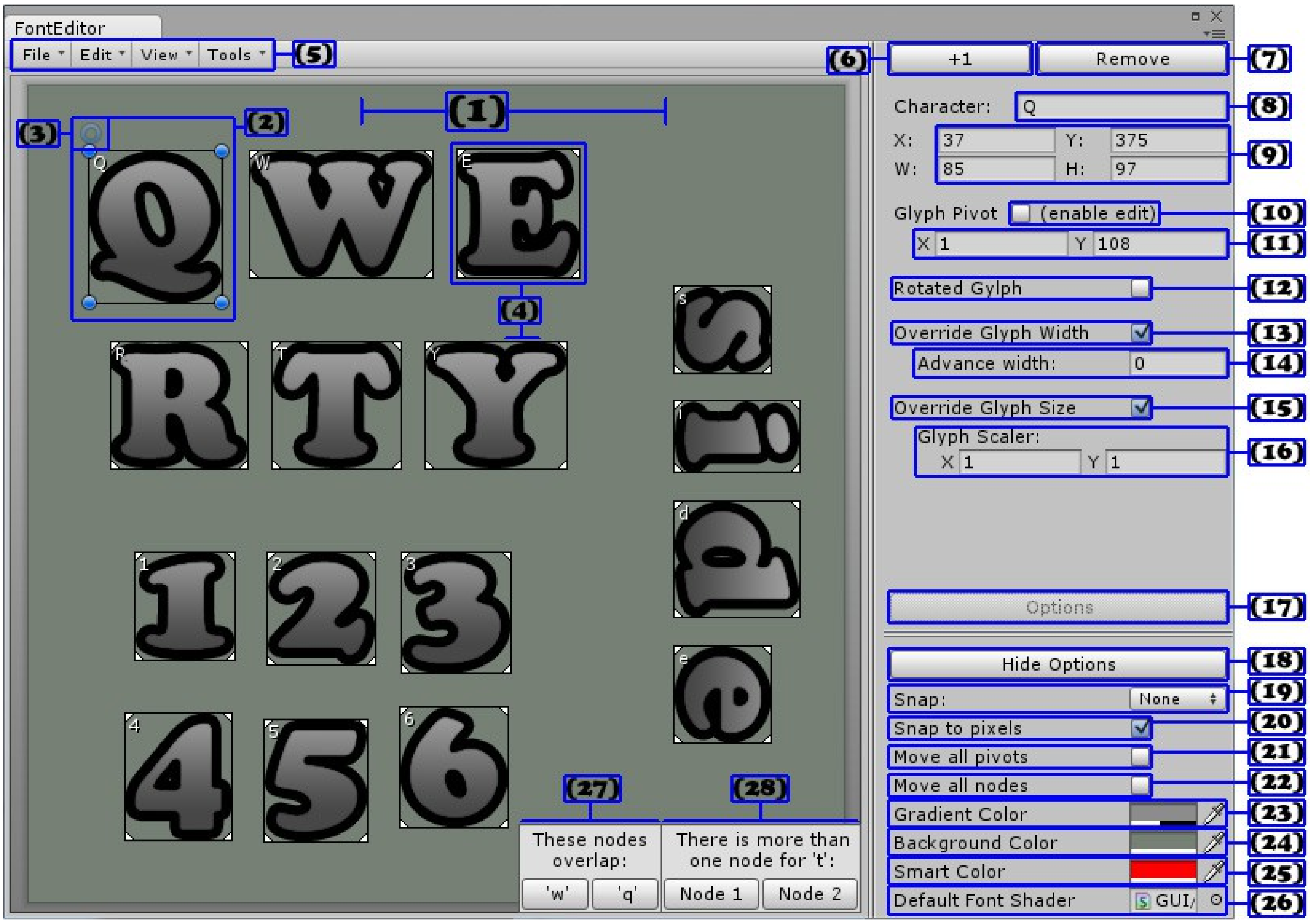
- **(1) Drag a Font object here from the project window (eg. My Font.fontsettings) to open the font and begin editing.**
- **(2) Drag an Image asset from the project window (eg. My Font.png) to create a new Font and Material in the same directory (eg. My Font(Font).fontsettings & My Font(Material))**

You can also use the File menu for the same functions (see Fig4 below)

The editor will then perform some checks on the image (for size and format, etc) and display the main interface.

4. Font Setter Interface.

Fonts are defined by UV coordinate rectangles (Rects), and this screen allows to you create and define the size and position of these Rects by moving Nodes around the screen, and (if required) typing values into the interface.



Character: Q
X: 37 Y: 375
W: 85 H: 97
Glyph Pivot ☐ (enable edit)
X 1 Y 108
Rotated Gylph ☐
Override Glyph Width ✓
Advance width: 0
Override Glyph Size ✓
Glyph Scaler:
X 1 Y 1
Snap: None
Snap to pixels ✓
Move all pivots ☐
Move all nodes ☐
Gradient Color
Background Color
Smart Color 🟥
Default Font Shader GUI/

These nodes overlap: 'w' 'q'

There is more than one node for 't': Node 1 Node 2

Fig3. The Font Setter Interface

(1) Your font bitmap displays in the main portion of the window, at full resolution. Click and drag with the mouse button to pan, and use the scroll wheel to zoom.
(2) This is the currently active/selected Node. Click and drag with the left mouse button to change its position. Use the corner handles to resize.
(3) The Glyph pivot (see Terminology).
(4) This is an unselected Rect (white corners).
(5) The Toolbar (See Fig4)
(6) Click this button to add a Node. If no Nodes are selected, a new one will appear towards the bottom left of your bitmap. If a Node is selected, the new Node will duplicate its size, and appear next to it.
The currently assigned character is displayed inside the Rect.
(7) This button removes the currently active Node, alternatively, press delete on the keyboard.
(8) Type into this field to define the character for the currently active Node. You can paste into this field if you want special characters
(9)This group of fields allows you to type in the X & Y position, width and height of the current Nodes UV Rect. The units are measured in pixels, in relation to the font bitmap, with X=0, Y=0 at the bottom left corner.

(10) Enable this toggle to allow click+drag editing of the glyph pivot. This also makes all glyph pivots visible, not just the currently selected Node.
(11) Type-in interface for the glyph pivot.
(12)If the current glyph is rotated 90 degrees clockwise (see the 'side' characters in Fig3) this must be toggled on.
(13) Enable this if you wish to specify the Glyphs' Advance Width (by default it uses the Rect width)
(14) [Only visible with (13) enabled] Input Glyph Advance Width here.
(15) Enable this if you wish to adjust the drawn size of the glyph.
(16) [Only visible with (15) enabled] Input Glyph Scale factors (Less than 1 = smaller. Greater than 1 = bigger)
(17) Click do display the options.
(18) Click to hide the options.
(19) Select a snap mode:

- **None: No snapping.**
- **Snap: Snaps the corners of Node together, for both moving and resizing.**
  Glyph pivots can be snapped to the 9 major axis points on their own glyphs, or when snapped to another glyph, the pivot values are set relative to the glyphs position
- **Axis Constraint X: Stops you from moving or resizing vertically.**
- **Axis Constraint Y: Stops you from moving or resizing horizontally**

(20) By default the UV Rects and Nodes are aligned to pixel boundaries. If you want to edit down to sub-pixel values, toggle this option.
(21) This toggle allows you to move all glyph pivots simultaneously. Pivots will move relative to glyph orientation [Fig3. (12)]
(22) This toggle allows you to move all Nodes simultaneously. No effect on orientation.
(23) Change the color/transparency of the gradient background.
(24) Change the color/transparency of the editor area.
(25) Change the color used by the smart auto-set (See Section 7)
(26) Any shader assigned here will be used when the editor creates materials for new fonts. The editor-included shader is located in "Editor Default Resources/FontSetter"
(27) If there are any Nodes overlapping, this info box will appear, you can click the buttons to highlight the nodes that are overlapped. Overlapping UV Rects is not recommended, because stray pixels from glyphs could appear inside others when drawn.
(28) If there is more than 1 Node defining any character, this info box will appear. You can click the buttons to highlight the offending nodes.

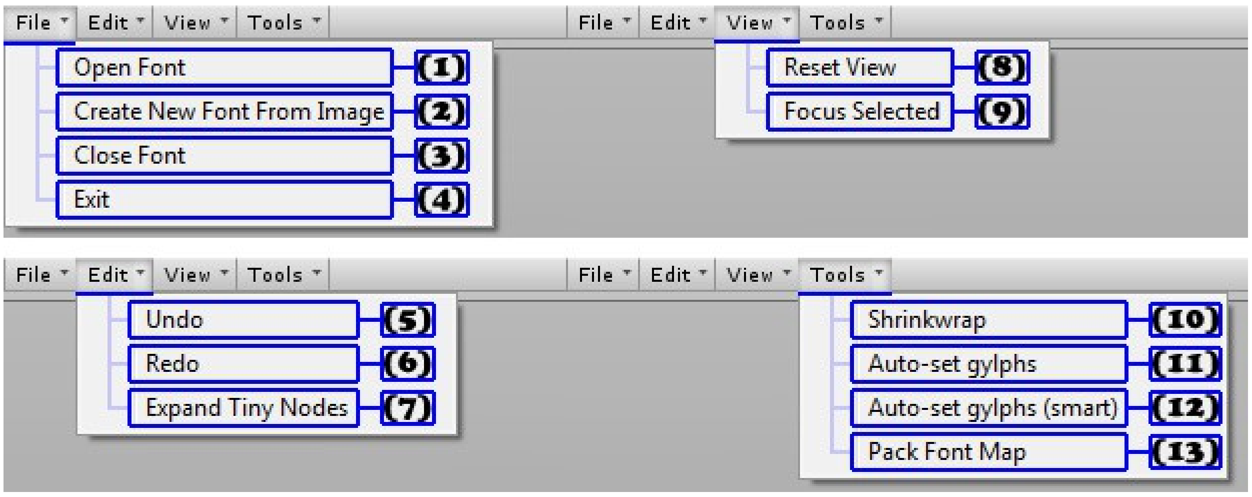
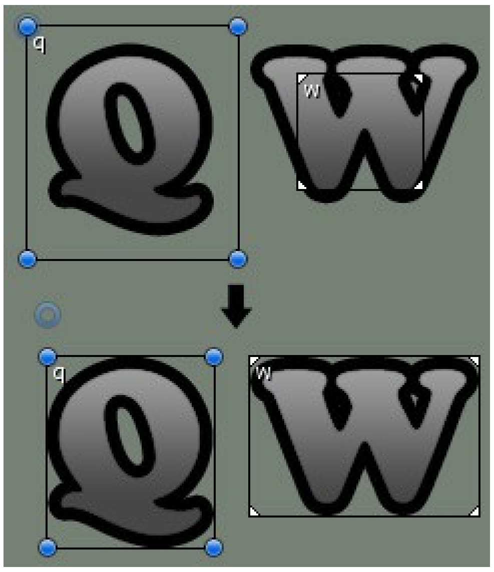


Fig4. The Toolbar buttons expanded

The Toolbar buttons [Fig 3 (5)] contain a number of useful functions.
(1) Open a Font Asset for editing [same as Fig 2(1)]
(2) Create a new Font from an image asset [same as Fig 2(2)]
(3) Close the opened font and return the front menu (Fig 2)
(4) Close the editor window.

(5) Undo the last edit (Same as Ctrl+Z).
(6) Redo the last un-done edit (Same as Ctrl+Y).
(7) If a Font has any zero-width or zero-height Rects, they are un-selecteable in the editor. Use this to expand them and make them editable.
(8) Reset the view to be centered and at default zoom.
(9) Zoom in as required and center view on the currently selected Node.
(10) Run the shrinkwrap function. This automatically resizes all Nodes to the minimum size needed to encompass the bitmap shape they are covering. Rects that cover no shapes are not affected.
(11) Run the auto-set function (See Section 6).
(12) Run the auto-set function while accounting for smart colors in the font bitmap (See Section 7).
(13) Run the Font Packer, to atlas the font glyphs into a smaller, more compact map.



Result of Shrinkwrapping

5. Procedure for manual setting.

If you choose to manually set the font Rects, this is roughly the sequence to follow.

- **Assign your font bitmap asset into the slot on the initial menu [Fig2. (2)].**
- **Click “+1” [Fig3. (6)] to add a Node. It will appear towards the bottom left.**
- **Click and drag the Node with the left mouse button to an appropriate position and size for a character, eg. over the 'a' character. Adjust using the type-in fields [Fig3. (9)] if needed (or wait and shrinkwrap all Nodes at the end).**
- **Type the character eg. 'a' into the field [Fig3. (8)].**
- **If the character is oriented 90 degrees, indicate as such using [Fig3. (12)].**
- **Press “+1” again. The editor will add a new Node of the same size adjacent to the one just set. The character displayed in the field [Fig3. (8)] will increment to 'b'.**
- **Move and resize the Node to the next character eg. 'b'.**
- **Repeat until all characters are defined by a Rect.**
- **If desired, click “Shrinkwrap” [Fig4. (10)] to automatically adjust all the Rects to the 'ideal' size.**
- **Don't forget to add a Node for the ' ' (space) character. This can be made small (as there is no image data to be used for a space) and the width of the space controlled using [Fig3. (14)].**
- **Don't forget to adjust the 'Line Spacing' in the font settings.**
- **You may also want to set the kerning up or down.**

6. Auto Set.

When you click "Auto-set Glyphs" [Fig4. (11)], the font is first reset, removing all defined Rects – make sure you want to Auto Set first! The Font Setter then automatically identifies shapes in your font map, and brings up the dialog shown on the right, asking you to identify the shapes, one at a time.

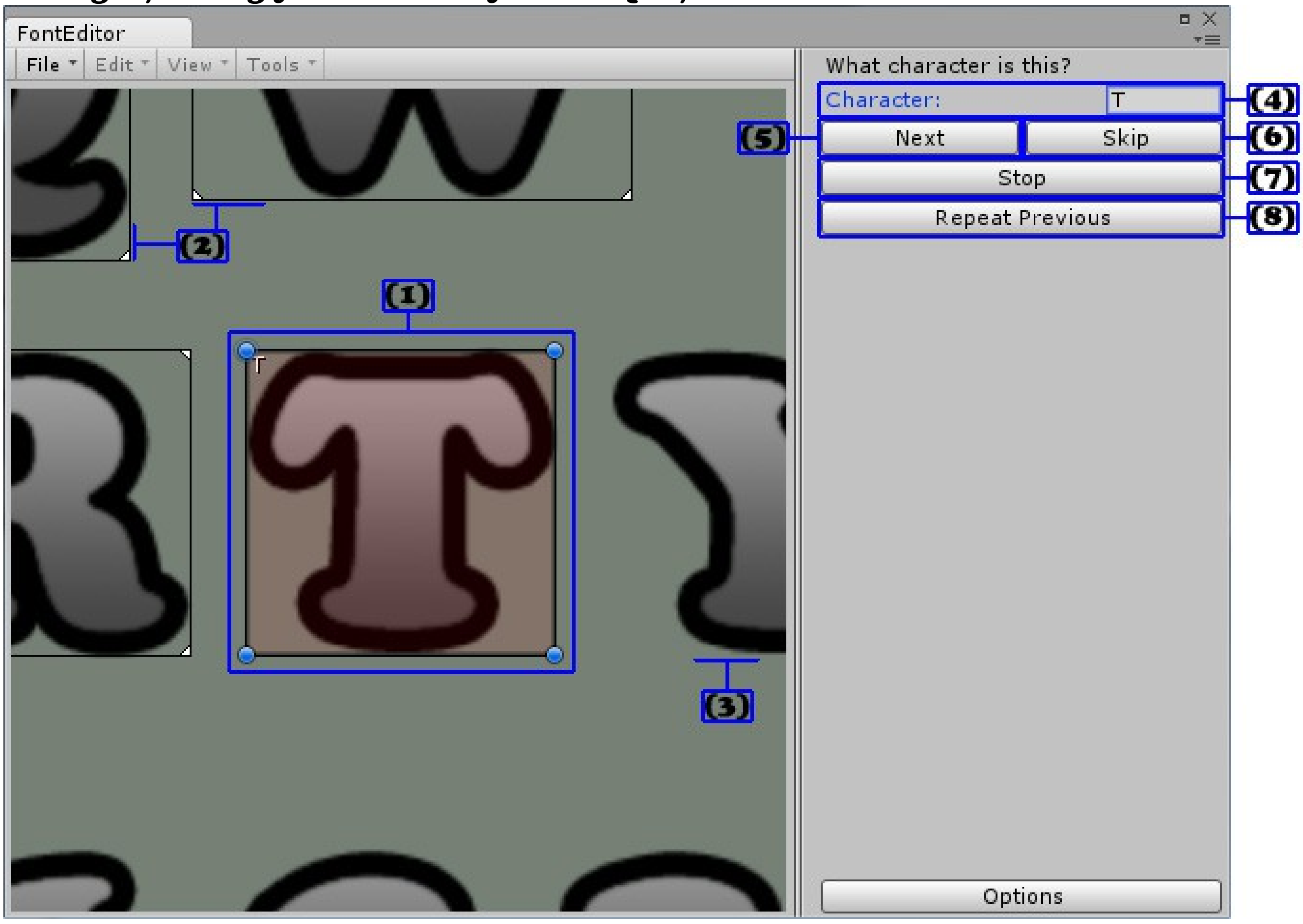


Fig4. Auto Set dialog

(1) This is the currently queried shape (highlighted red).
(2) These are shapes already identified.
(3) These shapes are yet to be identified.
(4) Type or paste into this field which character the editor is querying.
(5) Click "Next" after typing a character to identify the current shape, and move to the next. Alternatively, press Enter on the keyboard.
(6) Click "Skip" to skip to the next shape without identifying the current one. Press Escape to skip, without the need to click.
(7) Clicking "Stop" will end the Auto Set procedure.
(8) If you have previously attempted to Auto Set a font during this session, you can click "Repeat Previous" to automatically identify all the characters in the same order as before.
(9) After the Auto Set is complete, some information may be shown in popups inside the editor window.

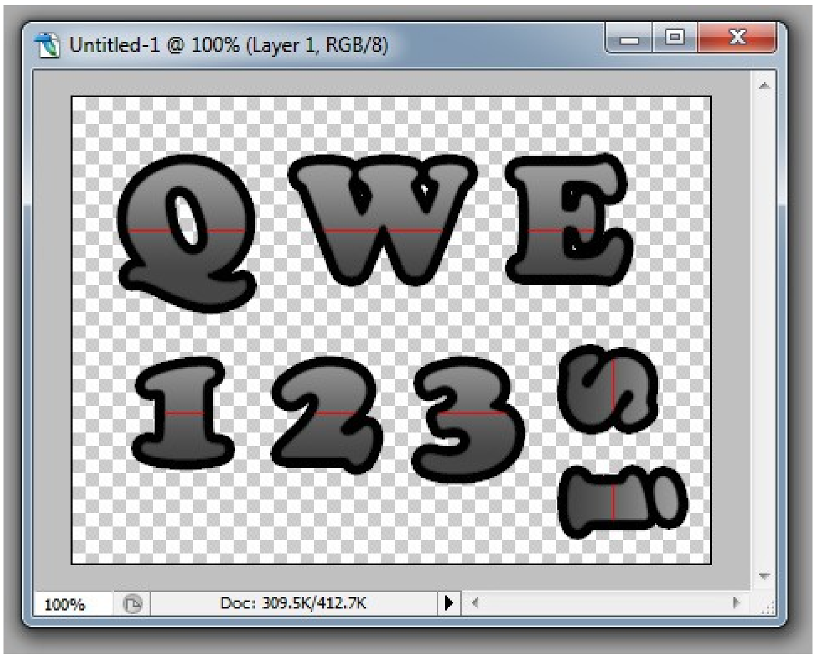
After the Auto Set has finished, your font is now ready to roll. Create a 3DText Object, and assign your new Font into the Text Meshes Font field. Add some example text (eg. "the quick brown fox jumped over the lazy dog". Carefully check this to see if any of the Rects need adjusting, any of the letters are missing, etc.

While this Auto Set mode is probably sufficient for simple fonts, like pixel fonts, or fonts with a consistent character height...

...it does not finely control the glyph pivots for each character (see Terminology). When your font is more complicated with tall and short characters, you will probably want to use the Smart set.

7. Smart Set.

For Smart Set to work, you need to make an extra edit to your font map, draw a single pixel line over each character, such that the line is the same vertical height on every letter and number. Set the Smart Color [Fig3. (25)] to this same color, and the Auto Set will detect those pixels, and automatically assign the glyph pivot. Additionally, it will detect if any of the characters are rotated, and set the relevant properties.



Ensure that this line does not extend past the bounding rectangle of the character, or the Auto Set will assume the character is bigger than intended. If you mask the line with the shapes of the characters, ensure that the masks are sufficiently opaque that the color is 100% itself, not semi-transparent, or the Auto Setter will not treat it as special. For the same reason, make sure there are no layers above the line layer that may change its color (eg. transparent effect layers).

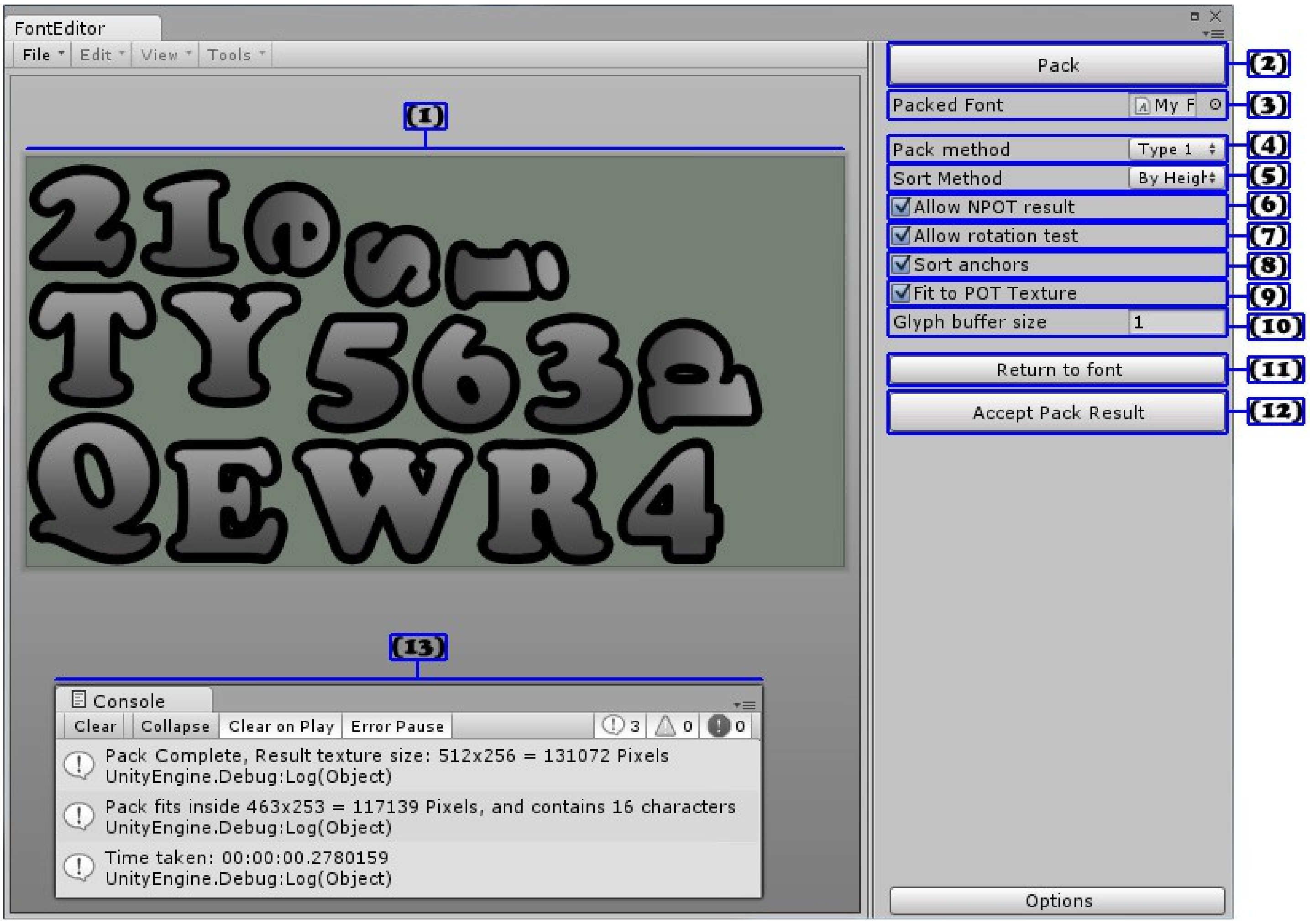
When you click the Smart button [Fig4. (12)], the program goes straight to the Auto Set procedure shown in (Section 6), it has no additional steps.

Once the Auto Set is complete, and you are happy with the Rects (and confirmed with a 3DText), you should then go back to your image editing software and remove the line layer(s).

You should then consider Packing the font bitmap into a more efficient atlas.

8. Font Packer.

Re-arrange bitmap glyphs into a more efficient atlas in one easy step.



```
Pack Complete, Result texture size: 512x256 = 131072 Pixels
UnityEngine.Debug:Log(Object)
Pack fits inside 463x253 = 117139 Pixels, and contains 16 characters
UnityEngine.Debug:Log(Object)
Time taken: 00:00:00.2780159
UnityEngine.Debug:Log(Object)
```

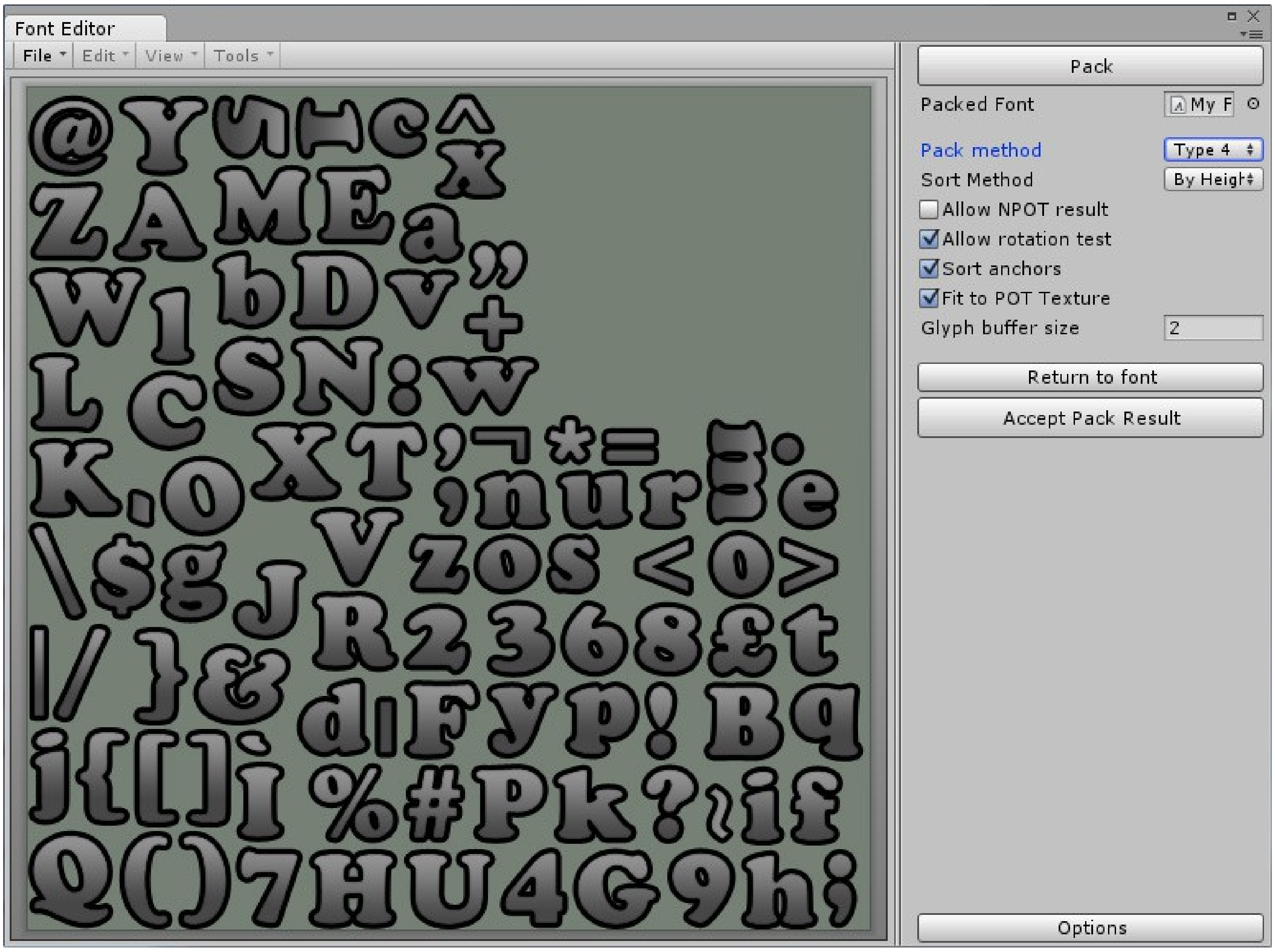
Fig5. The Font Packer Interface

(1) The finished packed font map is displayed here (after running the packer). Pan with mouse button and zoom with the scrollwheel.
(2) Press "Pack" to run the packer according to the settings defined below.
(3) The output Font settings Asset. Leave this blank to have the editor create new assets in the same directory as the original Font (My font(Packed).fontsettings, etc)
(4) The packer has 4 different custom packing algorithms, plus Unitys own Texture2D.packTextures method, each producing different results, choose the method here (see below).
(5) The packer sorts the Rects before inserting them into the result. Choose here which property to sort by
(6) Tick this box to dis/allow Non-Power-Of-Two results. If unchecked, the resulting bitmap will always be POT; if checked, the resulting bitmap size is determined by the packer, or forced to POT by [Fig5. (8)].
(7) Tick this box to dis/allow the packer rotating characters after it has sorted them (different sort modes will rotate characters before any packing is done).
(8) Tick this box to change how Type 3 [Fig (4) packs Rects (no effect on the other modes).
(9) Tick this box to force the resulting bitmap to be POT. If [Fig5. (5)] is ticked, this can result in blank areas (in some cases this may be beneficial, see [Fig6.]).
(10) Set a value here to space the Rects apart. This can prevent texture bleeding, but will result in a larger bitmap.

(11) Clicking "Return to Font" will return you to the main editor screen. Results of the pack are not discarded.
(12) Clicking "Accept Pack Result" will return you to the main editor screen with the newly packed font loaded.
(12) When the pack is complete, it writes a few lines to the console, to inform you how large the texture is, how large the actual pack area is, how many characters were drawn, and how long the pack took (larger, complicated packs with many characters could take longer).

Recommendations.
Depending on your input font map, different sort and pack modes will give varying results. For a font with many different sized characters, Type 3 seems to give the most efficient and aesthetically pleasing results. For a font with mostly similar sized Rects, Type 1 pack will probably give a better result. Sorting by shortest length generally seems better. While allowing NPOT results will produce a smaller image, (Obsolete?) Unity will automatically resample NPOT images up to the nearest power-of-two if you use the texture on a non-GUI object (ie. A TextMesh), regardless of the texture import settings.

http://answers.unity3d.com/questions/386490/how-can-i-get-unity-to-stop-resampling-my-sprites.html



Packed Font: My F
Pack method: Type 4
Sort Method: By Height
☐ Allow NPOT result
✓ Allow rotation test
✓ Sort anchors
✓ Fit to POT Texture
Glyph buffer size: 2

Fig6. An Example POT Pack

9. Font Creation Procedure Overview.

From start to finish, this is an outline of the process you can follow to create a full-color font ready for use in your project.

For a video tutorial, visit:
https://www.youtube.com/watch?v=28Z_4e7S1lE

- **Using Photoshop, start with an image with a very large width.**
- **If using an existing font as a starting point, type in all the letters, numbers, and symbols you need in the font. If making the font entirely from scratch, this step will take a bit longer.**
- **Now you have the shape of your font, you can use it as a mask for other layers.**
- **Once you have achieved artistic satisfaction, add a layer to the top, and draw a single pixel line horizontally over the whole image, so that it is at the same vertical position on every character. Mask this line to the same shape as the characters.**
- **Save in a format that respects transparency.**
- **Ideally the images should respect power-of-two sizes (256, 512, 1024, 2048).**
- **In Unity, adjust the import settings of the image so that Read/Write is enabled, and no compression is applied.**
- **Start the editor extension, assign image to [Fig2. (2)].**
- **Start the Smart set [Fig4. (12)], with the correct color assigned in [Fig3. (25)].**
- **Identify every character as prompted.**
- **Fix any issues as directed by the prompts.**
- **Add a new Node for the 'space' character.**
- **Return to Photoshop and turn off the Smart line layer(s), and re-save.**
- **Start the Font Packer, via the editor extension toolbar menu [Fig4. (13)].**
- **Try different settings to achieve the most efficient pack.**
- **Close the editor extension.**
- **In the font settings object (in the project browser), adjust the line spacing.**

The font object can now be used in any object that requires a font,
3DText TextMeshes, GUIText objects, Canvas UI Text (with appropriate shader), etc.

Bee Striped
Horizontal Split
DIAMOND STENCIL
Comic Sans Cutouts