Text Style Form Macro

By Shayanna Gatchalian, Technical Services Engineer, 4D Inc.

Technical Note 23-20

Table of Contents

Abstract

It is without a doubt that text styling serves as an integral part of user interface design. From choosing the perfect font to applying the most-fitting color scheme, these minute details all come together to create a certain theme and coherence for an application. Between the Windows and Mac platforms, there are dozens of fonts to choose from; however, it can be difficult to quickly preview a font in the 4D Form Editor, as the interface only shows a font's name in the Properties List window. This technical note introduces a 4D component that allows the developer to not only solve this issue, but also generate and export an entire style through a compact graphical interface.

Introduction

The *Text Style Editor* component included in this technical note is a form editor macro that opens a modal dialog window (see below). This dialog includes all you need to create, edit, preview, and export a text style either directly to selected form elements in the form editor or as JSON text to inject into hard disk files (i.e., the form definition or a CSS stylesheet). The next sections of this technical note will cover the minor issues associated with 4D's current form editor interface in regard to fonts, how this component solves these inconveniences, the steps needed to both set up and use the component, and how this component can be customized using the also-included matrix database.

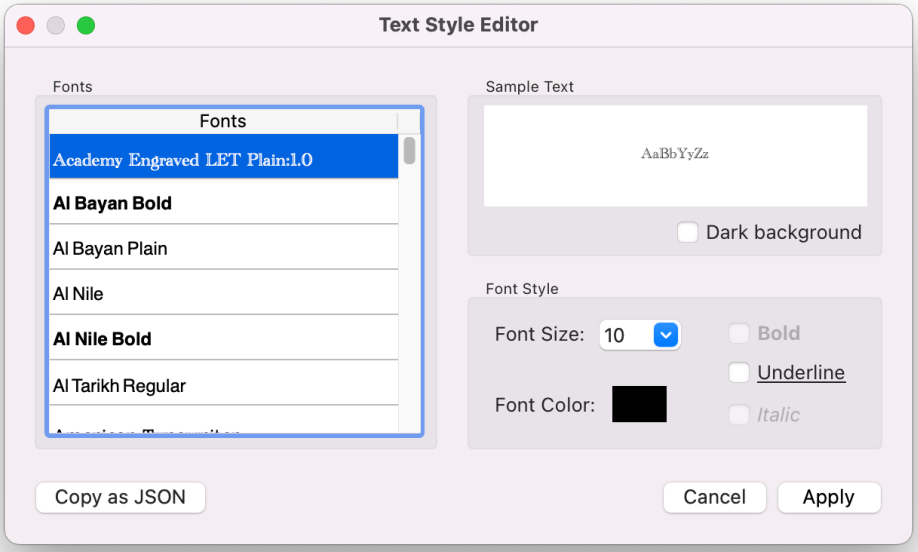

Image 1. The user interface of the *Text Style Editor* modal dialog window

The problem and how this component serves as a solution

Before diving into the component itself, it is important to understand the problem with how 4D's current form editor handles fonts. Without doing so, it would be difficult to see how much benefit the component offers.

The problem

With 4D's current form editor interface, the font attribute of the Properties List window is displayed as a dropdown menu that lists all the available fonts on the machine (see below).

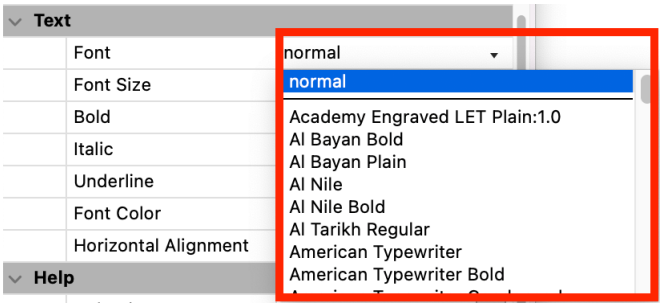

Image 2. The dropdown menu of the *Font* property in the *Properties List* window

Since the list only shows the font name, it can be difficult to visualize what the font looks like. It is not until the font is manually applied to form elements containing text, that the developer can directly see the regular typeface. And although the menu options can be traversed using the up and down arrow keys, the currently selected font is not applied until it is clicked on. The lack of these two features presents an inconvenient scenario where the developer would have to manually click through each and every single font just to preview it.

How this component serves as a solution

Many other state-of-the-art user interface editors and graphic design suites often show a preview of a font directly before the user chooses their desired font. The *Text Style Editor* component directly takes inspiration from this and lists the machine's available fonts in a list box, styled in its corresponding default typeface (see below). Moreover, the list box's *On Selection Change* event allows the developer to quickly scroll through each and every font via the up and down arrow keys on the keyboard. While these are small changes to the Properties List window's font property, they serve to substantially enhance this function.

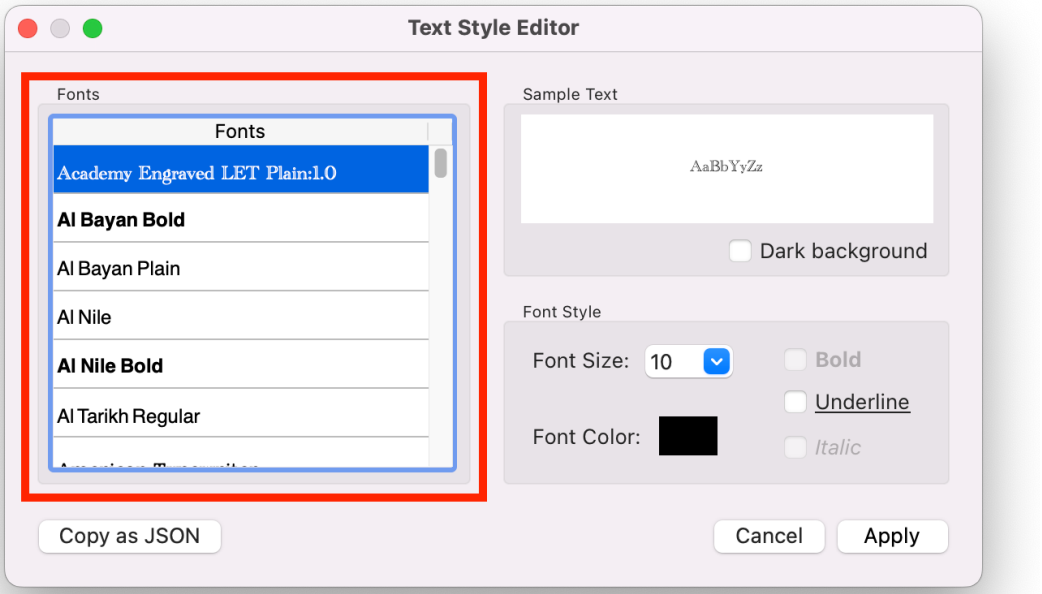

Image 3. The *Fonts* list box of the *Text Style Editor* modal dialog

Before Using the Component

Before installing and using the *Text Style Editor* component, there are a couple things to keep in mind. The next three sections will discuss the differences in font libraries between Mac and Windows platforms, the requirement to export to project mode, and lastly, how to install the component.

Fonts Across the Mac and Windows Platforms

In general, the Mac and Windows platforms each have their own set of fonts when the operating system is first installed on the computer. Some fonts that exist on one platform may not exist on the other; for example, Apple is known to have its own set of exclusive fonts, such as *Apple SD Gothic Neo* and *Apple Braille*, which are not part of the Windows fonts set by default. Furthermore, fonts can also be added or deleted by the user, adding more volatility to the situation. Before deploying an application in production, it is important to ensure that any chosen fonts are compatible across all platforms.

Export to project mode

Form editor macros take advantage of the class feature in the 4D language; because of this, **if you would like to use the *Text Style Editor* component, your database must be in project mode**. Unfortunately, there is no workaround for binary databases as they do not have classes.

If your database is still already in project mode, you can skip to the next section; if not, you can use the following steps to export it into project mode:

1. Open your database in 4D.
2. In the menu bar of the 4D application, go to File > Export > select **Structure to project**.

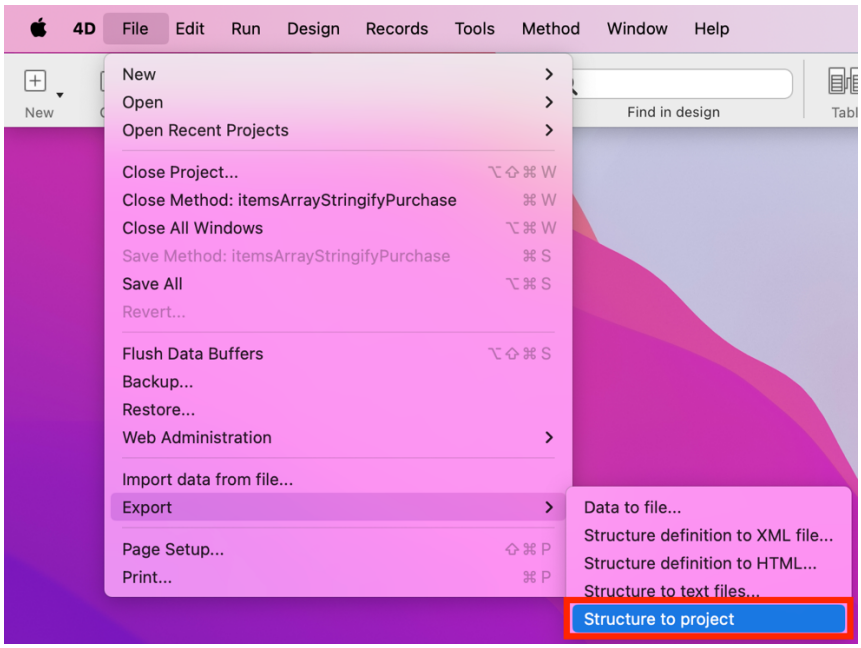

Image 4. The *Structure to project* option found under the *File* menu

3. Once the database has been exported, the original .4dbase file will still be saved in the same directory, but if you were to look at its contents, you will see how the original structure is now neatly organized into different folder and files instead of a single .4DB file.

With the database now in project mode, you can now install the component.

More information on converting to project mode can be found here: https://doc.4d.com/4Dv20R2/4D/20-R2/Converting-databases-to-projects.300-6392885.en.html

Installing the component on your database

Installing the *Text Style Editor* component is very easy. You may follow the steps below to do so:

1. If your database does not have one already, create a **Components** folder on the same level as the **Project** folder of your database.
2. Copy the **TextStyleFormMacro.4dbase** file from the technical note folder and paste it into the **Components** folder of your database.
3. If needed, restart the 4D database in designer mode.
4. Open a form editor, and right-click anywhere in the canvas area. A **Macros** option should pop up as the last option in the context menu. When you hover over this, another hierarchical list should display the **Open Text Style Editor** option (see below).

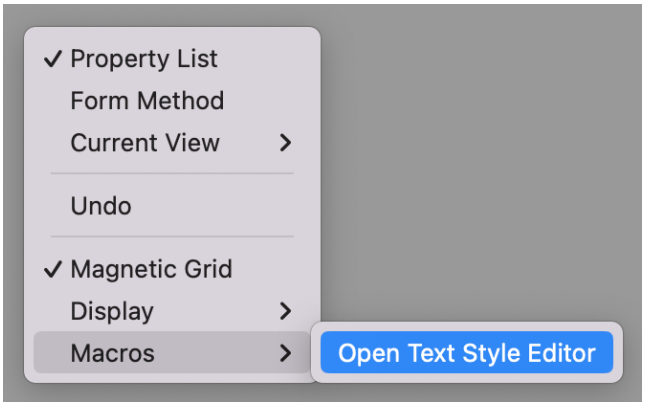

Image 5. The *Open Text Style Editor* macro option in the context menu

Congratulations! You have successfully installed the component. The next section will go over how to use this form macro in development.

More information on component installation can be found here:
https://developer.4d.com/docs/Concepts/components/

How to use the component

This section will go over how to execute the *Text Style Editor* form macro within 4D's design mode and the different parts of its modal dialog window.

How to execute the form macro

As mentioned in Step #4 of *Installing the component on your database*, all you need to do to execute the form macro is open the context menu in the 4D Form Editor canvas, select

Macros, then click on the **Open Text Style Editor** option. There are three ways in which the form macro behaves depending on how many form elements are selected beforehand.

Selecting 0 Form Elements

When no form elements are selected, an alert dialog pops up, warning the developer that they may only export the text style as JSON text (see below).

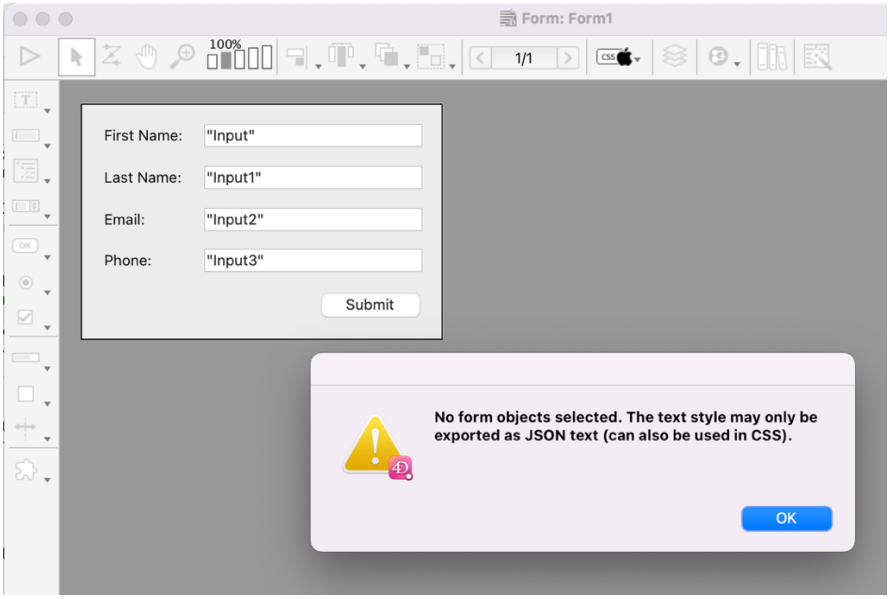

Image 6. The alert dialog that pops up when no form elements are selected

Once the developer clicks on *OK* and the modal dialog is displayed, the **Apply** button is disabled (see below). The user still has the option to generate a text style and export this data as JSON text using the **Copy as JSON** button (see below).

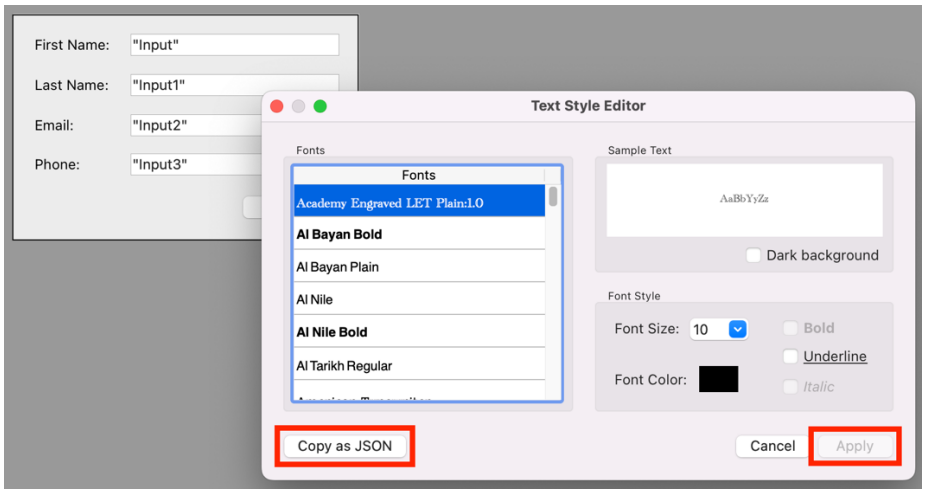

Image 7. The *Text Style Editor* modal dialog when no form elements are selected

Selecting a single form element

When a single form element is selected before executing the form macro, its current text style is loaded and displayed onto the modal dialog (see below). The **Apply** button is also enabled to apply any changes to the current text style.

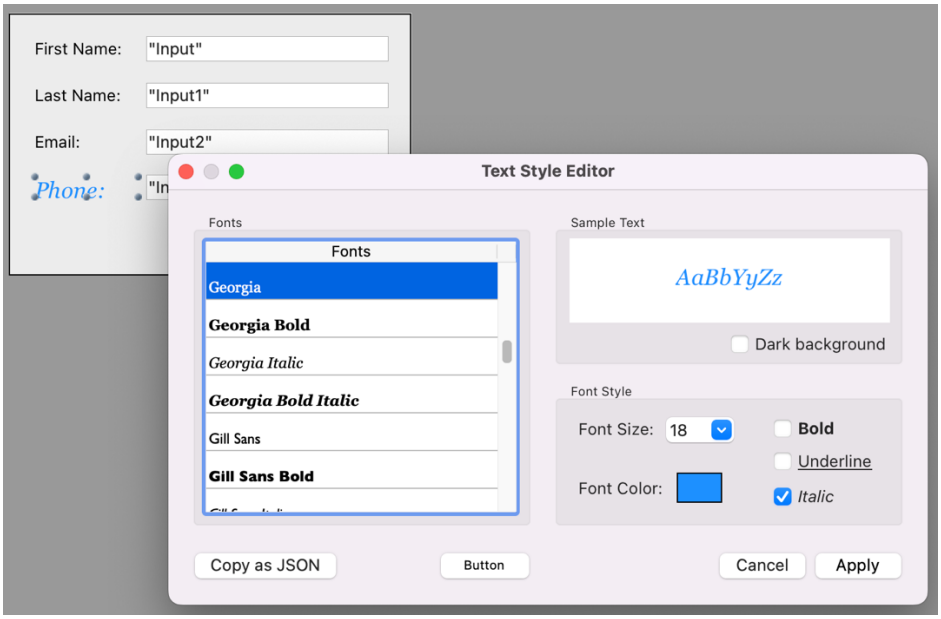

Image 7. The *Text Style Editor* modal dialog when 1 form element is selected

Selecting multiple form elements

When multiple form elements are selected, there comes the possibility that you may deal with multiple text styles as well. Because of this, the form macro defaults the current text style to the following properties:

- The first font in the *Fonts* list box
- Font size of 10
- Black font color
- All font styles set to “normal”/turned off

Once the developer creates a custom text style, it will be applied to *all* form elements that are currently selected.

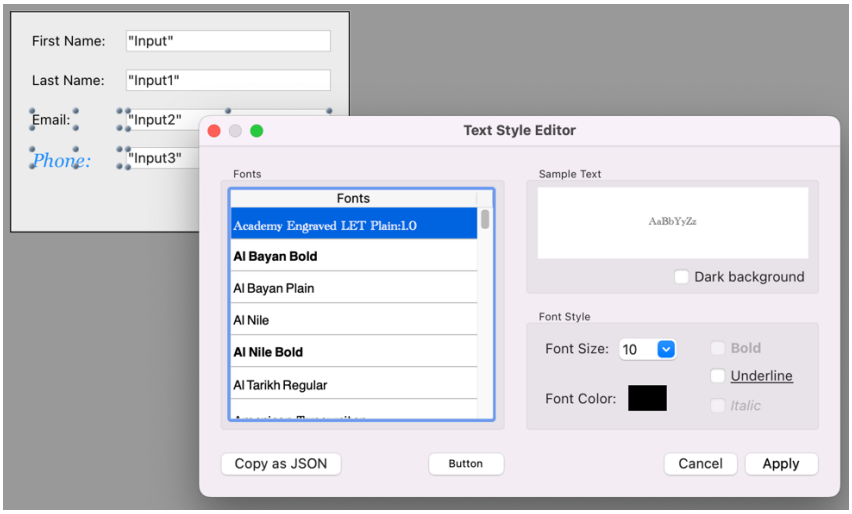

Image 7. The *Text Style Editor* modal dialog when multiple form elements are selected

The *Text Style Editor* modal dialog window

There are four main sections of the *Text Style Editor* modal dialog window: sample text preview, fonts list box, font style options, and application options.

Sample text preview

This section shows what the entire text style will look like when applied to a form element. The **Dark background** option allows the developer to toggle between a white and black background to test readability with certain font colors.

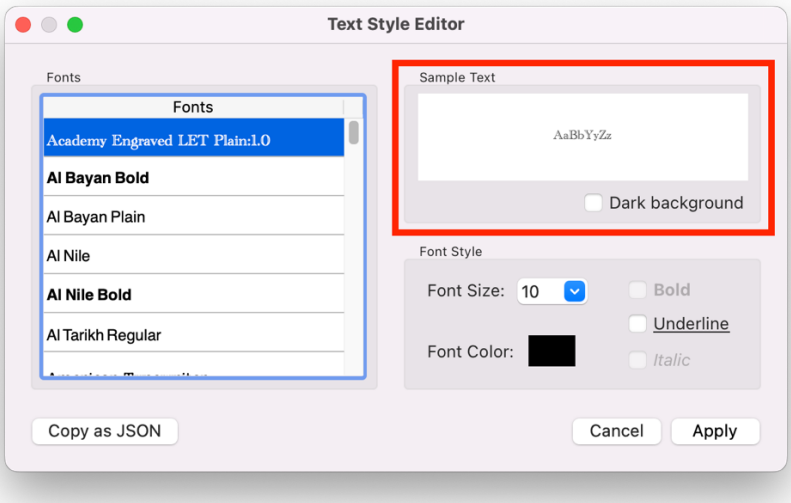

Image 8. The *Sample Text* section of the *Text Style Editor* modal dialog

Fonts list box

The fonts list box serves as the star of the show for this entire component. This is where the developer can quickly browse through the entire list of available fonts and their font previews. The selection change is reflected in the earlier-mentioned *Sample Text Preview* section of the window. Once again, the developer can traverse through each font by using the up and down arrow keys on the keyboard.

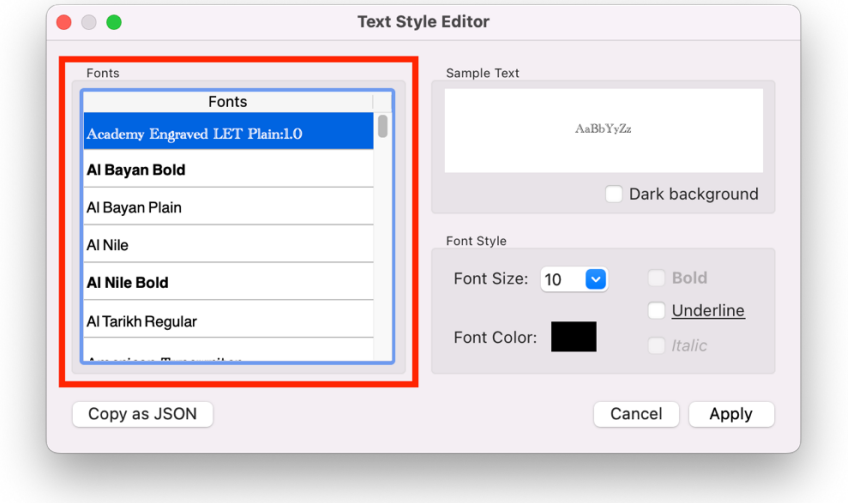

Image 9. The *Fonts* list box of the *Text Style Editor* modal dialog

Font style options

This section serves to generate the rest of the text style by offering font size, color, and style (i.e., bold, underline, italic) options.

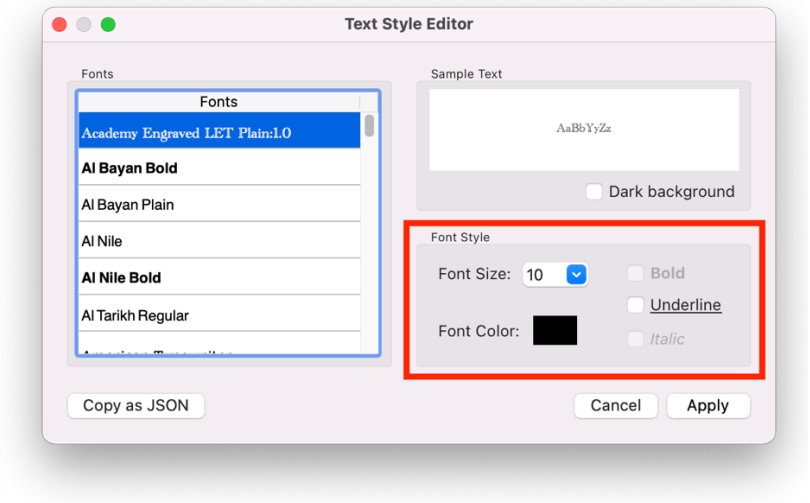

Image 9. The *Font Style Options* section of the *Text Style Editor* modal dialog

Application options

Once a text style is generated, the developer has two options on what they would like to do with the data: copy the data as JSON text or apply the text style directly to the set of selected form objects. These options can be executed their corresponding buttons at the bottom of the modal dialog.

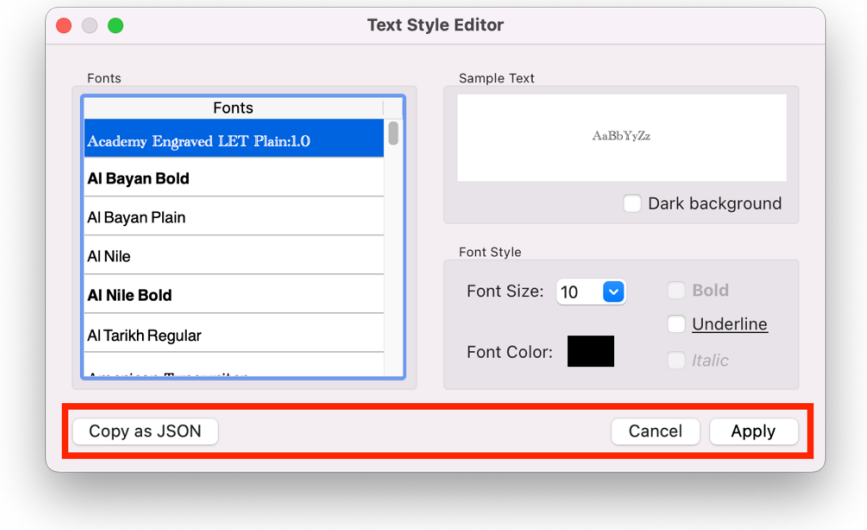

Image 9. The *Application Options* section of the Text *Style Editor* modal dialog

"Apply" button

When the **Apply** button is clicked, the **openTextStyleDialog** class of the form macro will apply the custom text style to the form elements selected by directly manipulating the form definition. Since the main focus of this technical note is on the Text Style Editor form macro itself, you can find more detailed information on the implementation of form editor macros here: https://developer.4d.com/docs/20-R2/FormEditor/macros.

"Copy as JSON" button

When the **Copy as JSON** button has been clicked, the text style is converted into a JSON string of the text style's properties; this string is then copied to the clipboard, where it can be used in the following ways:

- Inserted into the form definition (a.k.a., .4DForm file)
- Inserted into a CSS file
- Used in 4D code by using the Get text from pasteboard command

Below is an example of a custom text style and its JSON string when copied to the clipboard.

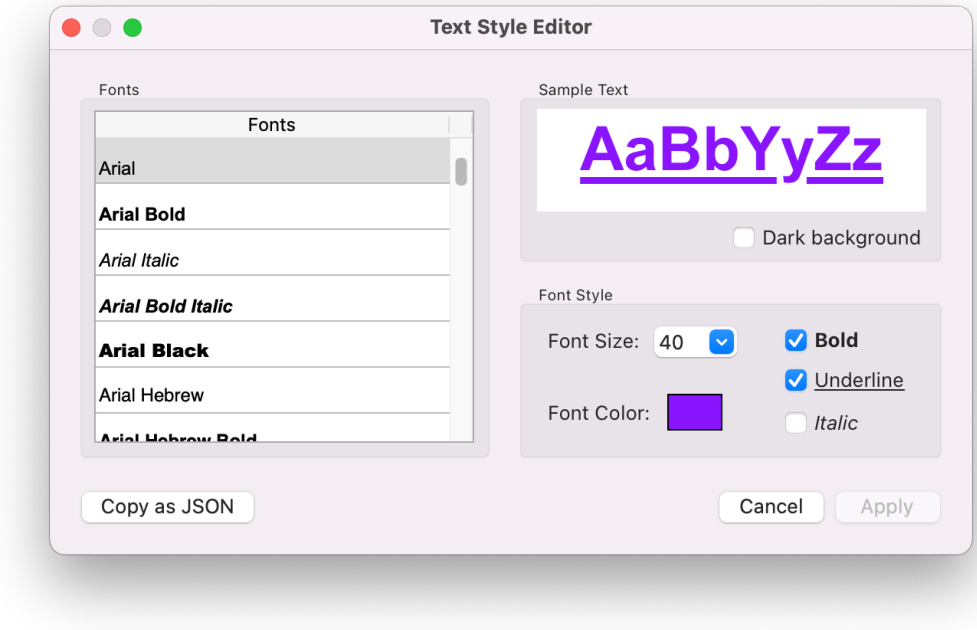

Image 10. A custom text style done in the *Text Style Editor* modal dialog

```
{
    "fontFamily": "Arial",
    "fontSize": 40,
    "stroke": "#8B14FF",
    "fontWeight": "bold",
    "fontStyle": "normal",
    "textDecoration": "underline"
}
```

Code Example 1. The exported JSON string based on the text style in Image 10

Aside: Undo function

Form macros inherently have a feature where any changes made to the form can be reverted by executing the undo function in the form editor. If you are not satisfied with the text style once it has been applied to selected form elements, it is an easy fix to go back and undo the actions as many times as needed.

Customizing the component

This technical note also includes the matrix database used to develop the component; with this, you may customize any part of the form macro to your liking. Throughout the entire form macro's code, comments are inserted at every step of the method or class to make the logic behind the code easier to understand. For more information on how to implement form macros, the following documentation page may be useful: https://doc4d.github.io/docs/20-R2/FormEditor/macros.

From there, all you need to do is compile and build the matrix database into a component and install it onto another database as specified in the **Installing the component on your database** section. Documentation on building 4D components can be found here: https://developer.4d.com/docs/Desktop/building/#build-component.

Conclusion

While the 4D form editor already has an interface to change the font of form elements with text, there are minute inconveniences that prevent the developer from easily previewing each font available on the computer. This technical note has served to explain how these issues can easily be solved and provides a concrete solution to the problem with the *Text Style Editor* component.