TiefPrompt Documentation

Tiefseetauchner

October 10, 2026

Contents

1. Introduction 5
    1.1. TiefPrompts primary function 5
    1.2. Overview 5

TiefPrompt Usage Guide

1. Setup 8
    1.1. Primary Controls 8
    1.2. Settings 10
    1.3. Language 10
    1.4. Theming 11
    1.5. Prompter Screen 12
    1.6. Display, Text and Scroll Settings 13
    1.7. Saving and Restoring your Settings 13
    1.8. Keybinding Settings 14
2. The Prompter Screen 15
    2.1. Top Bar 15
    2.2. Bottom Bar 15
    2.3. Display and Text Overlays 15
3. Display Settings 17
    3.1. Flipping 17
    3.2. Control Buttons 18
    3.3. Reading Indicators 18
    3.4. Vertical Margins 19
    3.5. Side Margin 21
    3.6. Colors 22
4. Text Settings 25
    4.1. Text Display Settings 25
    4.2. Font Settings 26
    4.3. Custom Fonts 26
    4.4. Markdown 29
    4.5. Current Chapter Display 30
    4.6. RTL Rendering 31
5. Scroll Settings 32
    5.1. Scroll Speed 32
    5.2. Countdown Timer 32
    5.3. Voice Activation 33
6. Keybindings 36
    6.1. Changing Keybindings 36
    6.2. Default Keybindings 37
7. Save and Restore Settings 39
8. App Variants 42
    8.1. Gated features 43

9. Saved Scripts 45

An Incomplete List of Features in TiefPrompt

1. Script Saving 50
    1.1. Saving the Scratch Script 50
    1.2. Saving a Regular Script 50
    1.3. The Unsaved-Changes Indicator 50
    1.4. Importing Scripts 50
    1.5. Leaving with Unsaved Changes 51
2. Saved Settings 52
    2.1. What's in a Profile 52
    2.2. Applying a Profile 52
    2.3. Naming 52
    2.4. Import and Export 52
3. Chapters 53
    3.1. Markdown Is a Hard Requirement 53
    3.2. What Counts as a Chapter 53
    3.3. Tracking the Current Chapter 53
    3.4. Turning It Off 53
4. Custom Fonts 54
    4.1. Families and Variants 54
    4.2. Uploading Fonts 54
    4.3. How Uploads Merge Into Families 54
    4.4. Moving and Deleting Variants 54
    4.5. Silent Font Fallback 55
5. Scroll and Session Behavior 56
    5.1. Scroll Position Persistence 56
    5.2. The Scratch Script 56
6. Theming 57
    6.1. Two Separate Themes 57
    6.2. Light, Dark, and System 57
    6.3. App Accent Color 57
    6.4. Prompter Theme 57
7. Voice Activation 58
8. Markdown 59

TiefPrompt Contribution Guidelines

1. Filing Issues 61
    1.1. Filing a Bug Report 61
    1.2. Filing a Feature Request 61
2. Contribution Guidelines 62
    2.1. CONTRIBUTING.md 62
        2.1.1. 0. Setup 62

2.1.2. 1. Build both flavors before opening a PR 62
2.1.3. 2. Separate concerns 63
2.1.4. 3. Think about what you've done 63
2.1.5. 4. Don't branch on flavor in the UI 63
2.1.6. 5. State lives in providers; navigate with go_router 63
2.1.7. 6. Localize user-facing strings, but stay in your lane 63
2.1.8. 7. Profit 64
2.1.9. When I add a Feature 64
2.1.10. When I add a Route 64
3. Documentation Guidelines 65

1. Introduction

This documentation primarily exists for guiding you, the user, through the application and explaining features. Historically grown apps often have very complicated interfaces if they weren't designed from the beginning to be obviously accessible. So does TiefPrompt. Here, we shall strive to disentangle that.

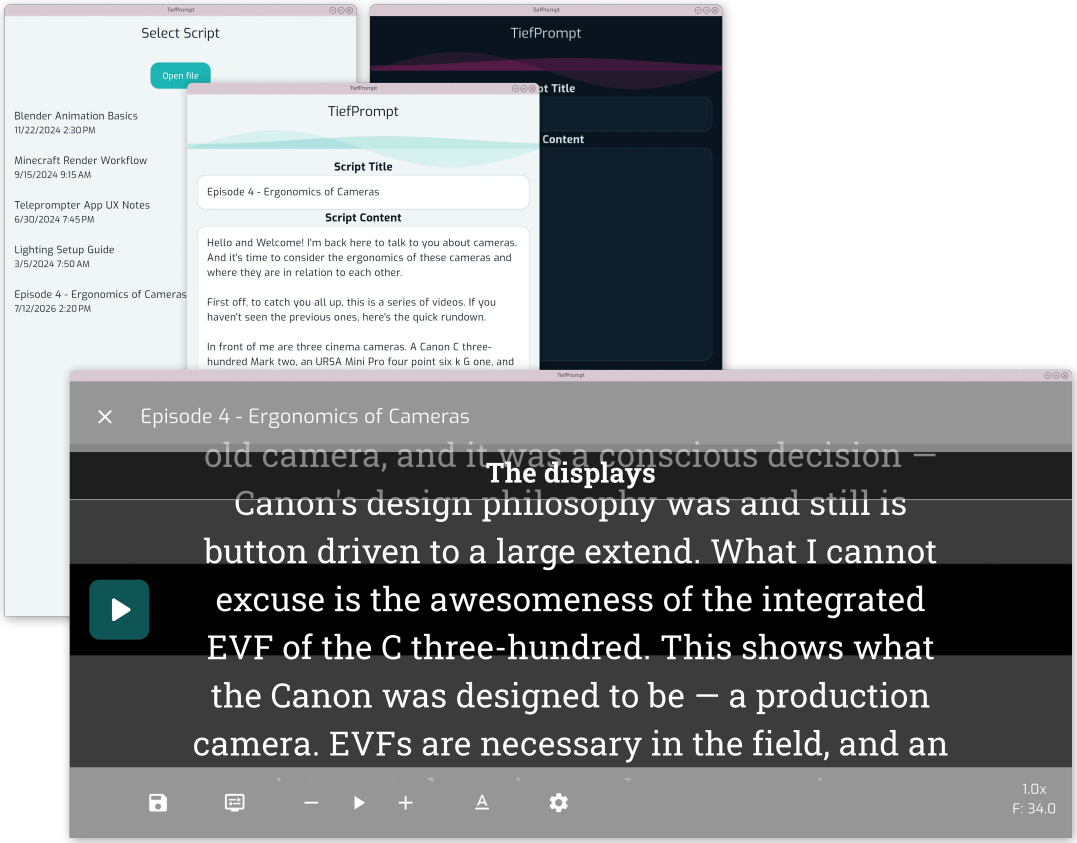


Figure 1: Hero showing an example of TiefPrompts user interface

1.1. TiefPrompts primary function

TiefPrompt is a teleprompter. Its purpose is to scroll text down a screen so you can read it while speaking.

However, the way *how* you want your text to scroll is significant. What should it look like? How should it be formatted?

TiefPrompt gives you as much freedom as possible over that. Customization is a paramount factor for TiefPrompt development and is regarded with pride.

1.2. Overview

This Documentation is split in three parts:

- Usage — starting with Setup the App, you can find out all off basic to advanced usage of TiefPrompt.
- Feature Description — here, you'll find a detailed description of (almost) every feature, giving you full control over the app.

- Contributing — if you're done finding bugs, you can proceed to contributing to the project in various ways — find out more there.

This documentation won't be complete, as documentation goes. But it tries to be somewhat like that.

SECTION 1

TiefPrompt Usage Guide

> ℹ TiefPrompt is an open-source project. However, when downloaded via the App Store or Google Play, it is a freemium app. This means that some features are locked behind an in-app purchase. The FOSS version, which is available on GitHub, provides all features for free. This documentation primarily covers the FOSS version.

TiefPrompt is a relatively simple app designed to be usable without any setup. When first opening the app, reading the changelogs and our plea for support (which is recommended but not required, obviously), you will be greeted with the home screen. From there, you can start using the app immediately, and it will work with the default settings. However, after installing, there are some setup steps you can take to customize the app. If you want to skip the walkthrough of primary app controls, you can jump to the Settings section.

1.1. Primary Controls

The app's home screen has a few primary controls you may want to familiarize yourself with. These are the buttons that allow you to start a new prompter session, open a saved session, or access the settings menu.

First, the Title text box at the top allows you to enter a script title. This is used for saving and loading scripts, and is not required to be filled in. If you do not enter a title, the app will use a default title when saving. See more under Save and Restore Settings.

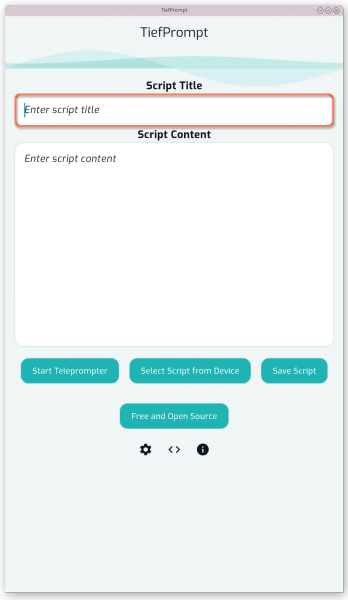


Figure 2: The Title box

Below the title box is the Script content box, which is where you can enter the text you want to scroll in the prompter. This box is effectively to be filled in, and will be empty

when first opening the app. You can leave it empty, however that makes for a boring prompter session.

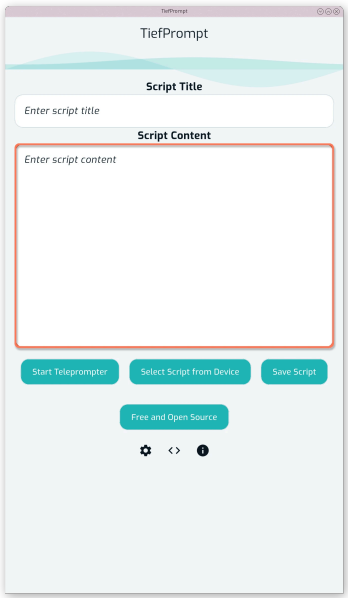


Figure 3: The Script content box

Then there's the primary buttons of the home screen — the "Start Prompter" button (see The Prompter Screen), the "Load Saved Script" and the "Save Script" button (see Saved Scripts).

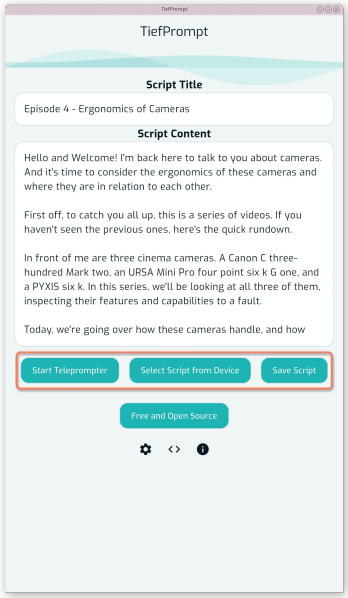


Figure 4: The primary buttons of the home screen

Below that is the button displaying the currently installed app variant. This can be clicked to get a dialog showing information about the app, and it differs between the variants ("FOSS", "Free" and "Pro"). FOSS is the "Free and Open Source Software" version, which provides all features fully for free, while the Freemium versions require an in-app purchase to unlock all features. See App Variants for more information.

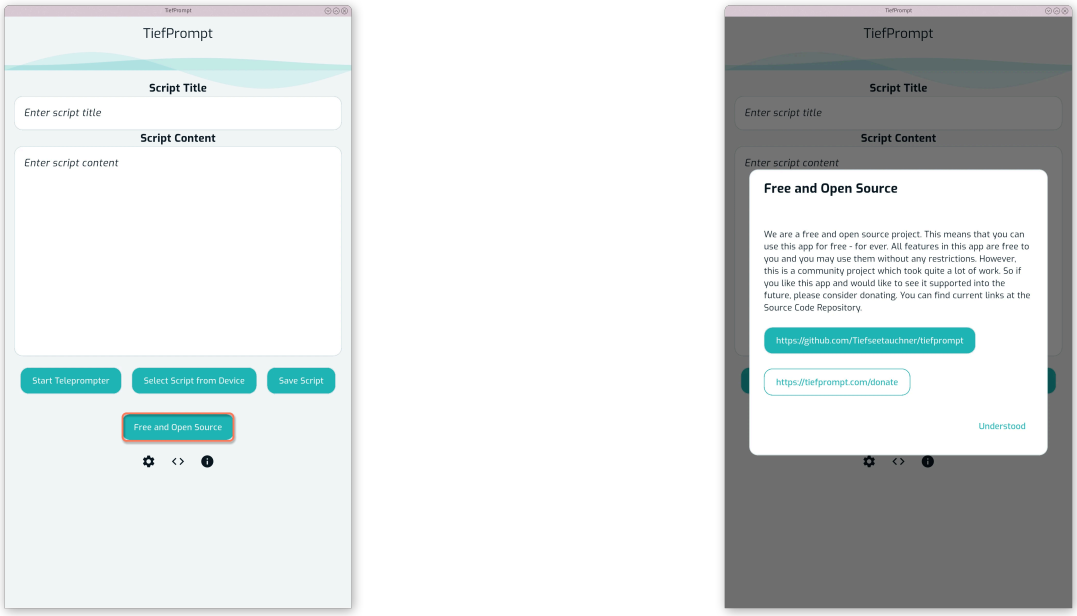


Figure 5: The app variant button, showing currently FOSS being installed, with the FOSS dialog open

1.2. Settings

To access the settings, click the cogwheel below the primary app buttons as shown below.

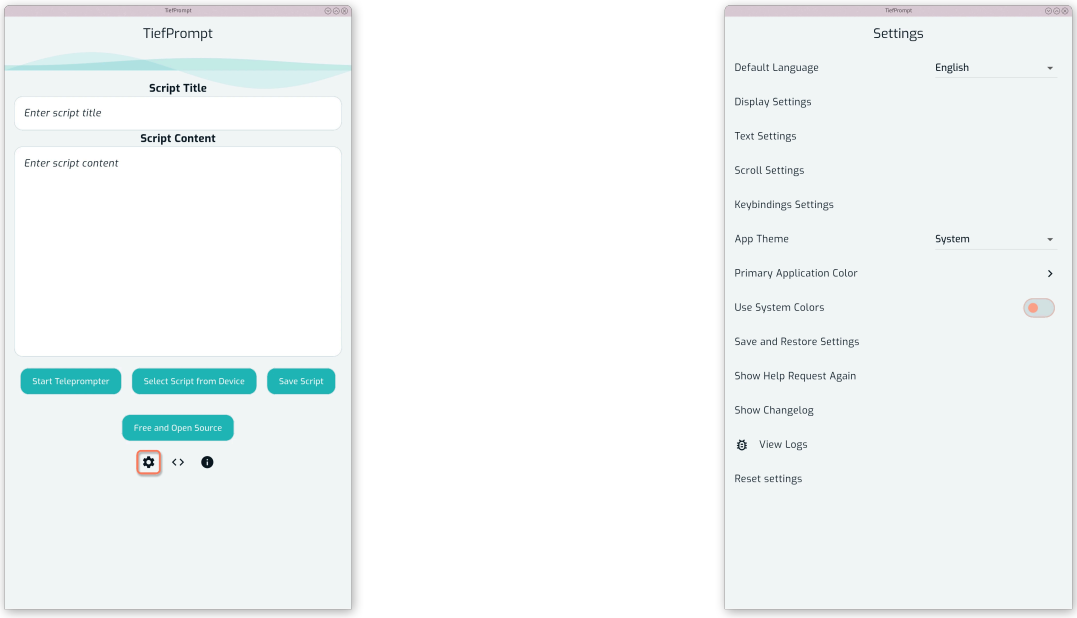


Figure 6: Instructions to open the TiefPrompt Settings

1.3. Language

The primary language TiefPrompt is developed in is English. However, there are other languages available. Among them are:

- Arabic
- German
- Russian

- Simplified Chinese
- Pirate English

After opening the app, you can navigate to the settings menu via the cogwheel below the primary app buttons as shown below, and select the language from the top of the menu.

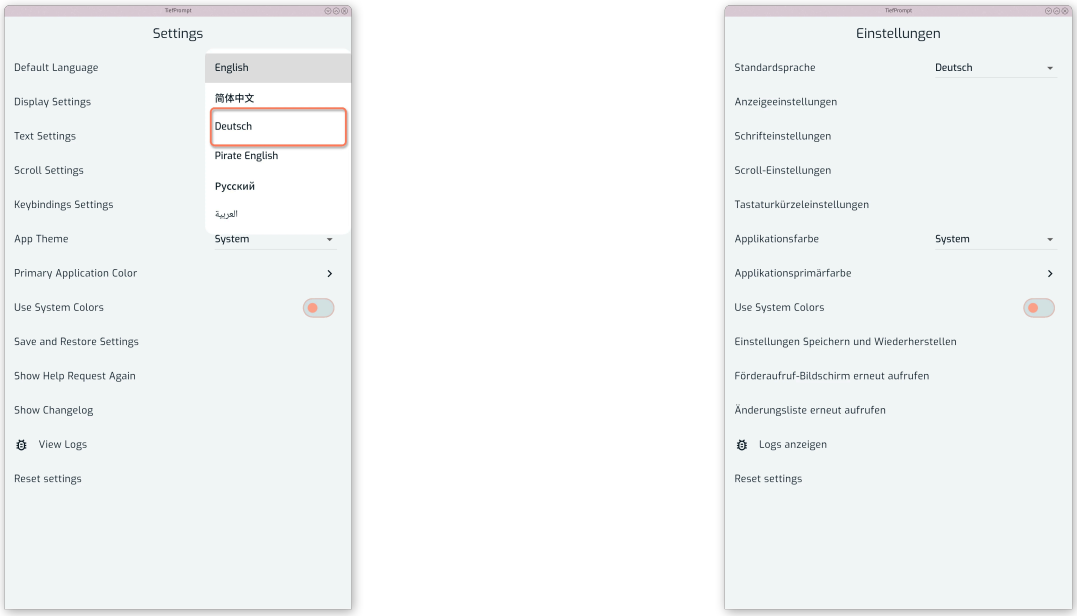


Figure 7: Instructions to change the TiefPrompt Language

1.4. Theming

The apps theme is similarly configurable, this time to either be light, dark or system dependent.

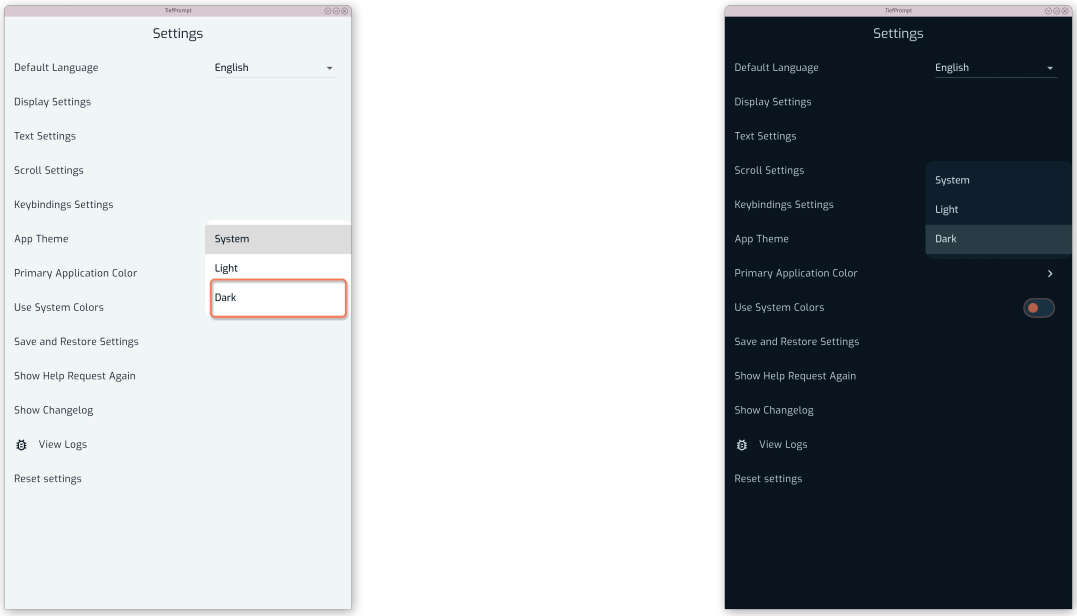


Figure 8: Instructions to change the TiefPrompt Theme

In addition to the theme, there is also a primary color switch that allows you to stylize TiefPrompt's primary color. This affects buttons and the general theme of the app.

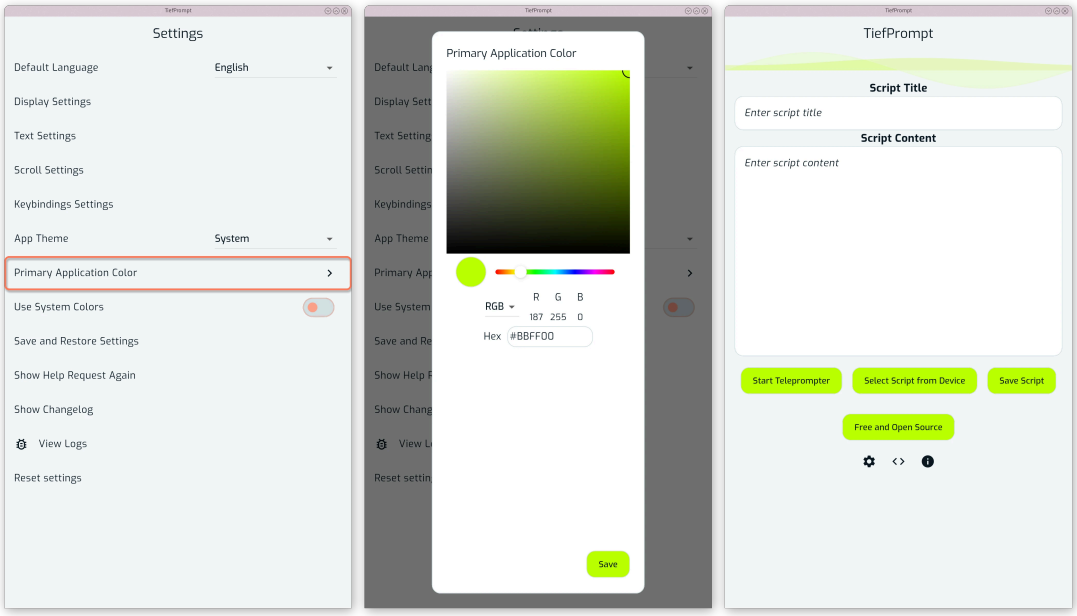


Figure 9: Instructions to change the TiefPrompt Primary Color and example of app afterwards

This primary color change does not however affect the prompter directly; those are separate, different settings.

If running Windows, MacOS, Linux or Android, you can also use the system color scheme — simply enable “Use System Colors”. Note that this will disable the application primary color selection.

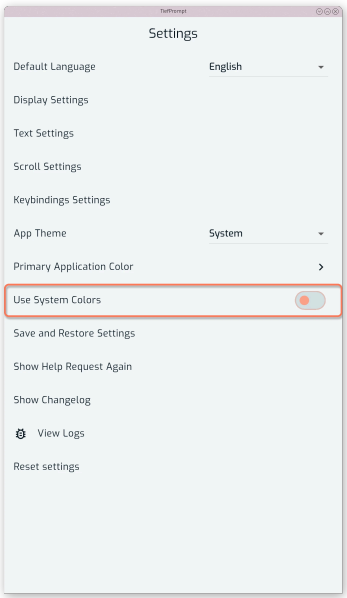


Figure 10: Enable System Color Scheming

1.5. Prompter Screen

Being the most prominent screen of the application, the actual scrolling text screen deserves its own part at The Prompter Screen. But below is a sneak peek of what could be your prompter screen after configuring it.

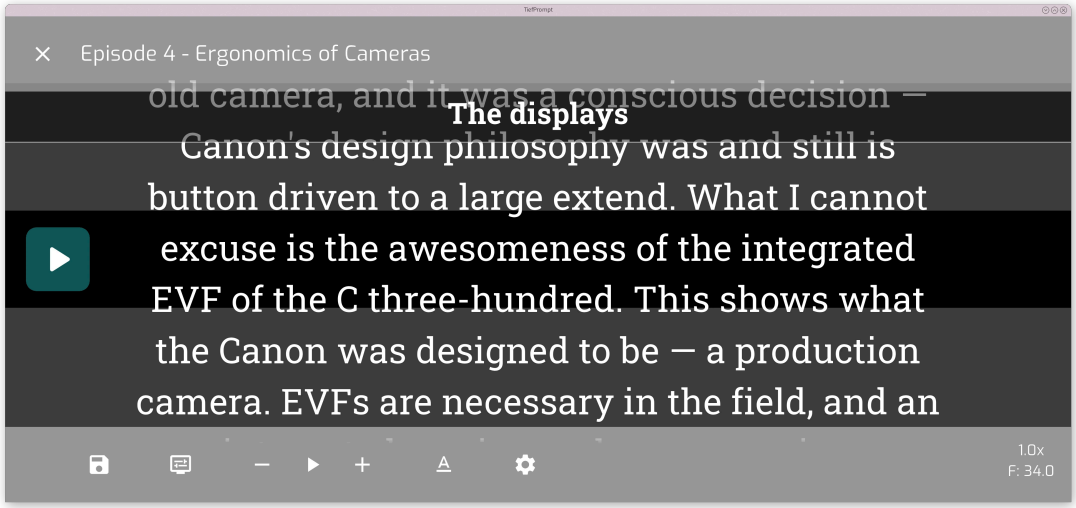


Figure 11: The primary prompter screen, configured with the various tools at hand

1.6. Display, Text and Scroll Settings

The three largest articles of the settings screen are the Display Settings, Text Settings and Scroll Settings screens. All three are too in depth for this page, but are explained further in Display Settings, Text Settings and Scroll Settings respectively.

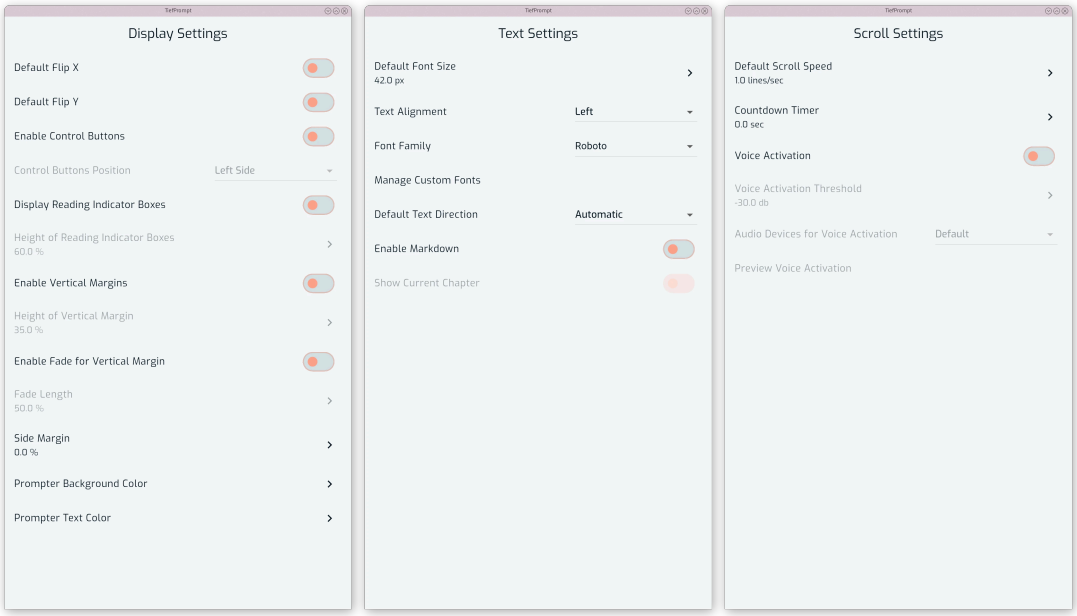


Figure 12: Display Settings, Text Settings and Scroll Settings

1.7. Saving and Restoring your Settings

You can also save and restore settings using the Save Settings Profile feature.

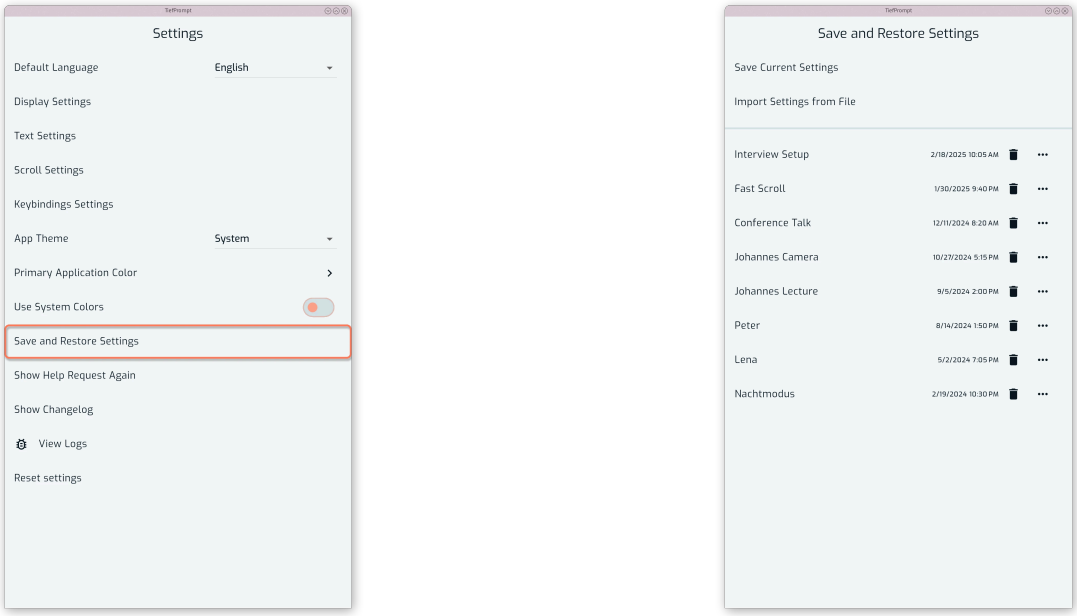


Figure 13: The Saved Settings Screen

See Save and Restore Settings for more information.

1.8. Keybinding Settings

TiefPrompt allows for customization of Keyboard and Controller controls in the app. To adjust the keybindings, access the Keybindings Settings under the settings screen.

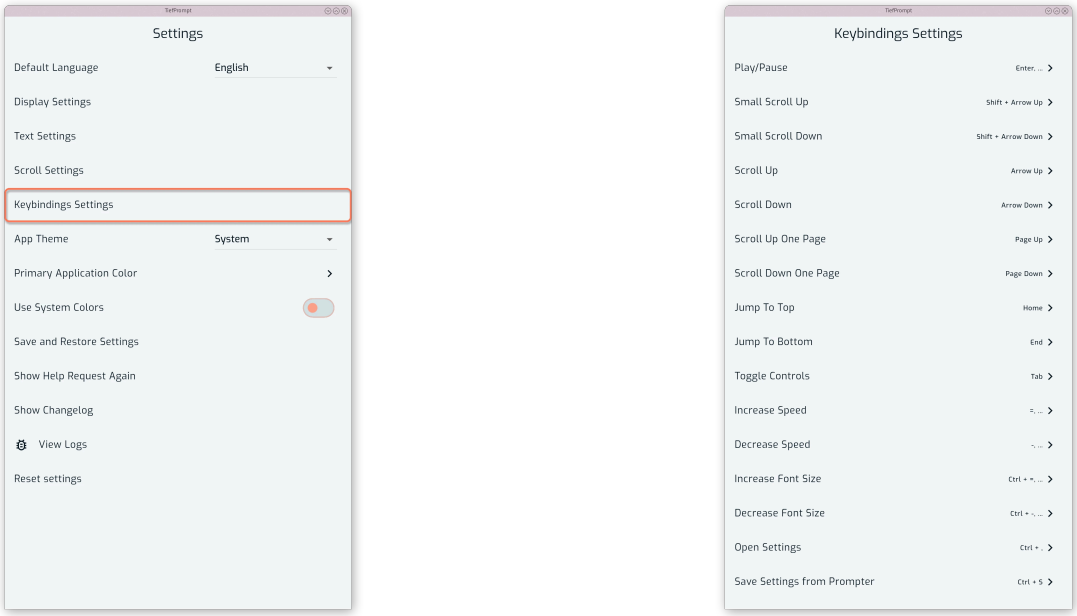


Figure 14: The Keybindings Screen

See Keybindings for more information.

2. The Prompter Screen

The Prompter Screen shows the scrolling text. How that text looks and moves is governed by the settings covered in Display Settings, Text Settings and Scroll Settings, but the top bar, bottom bar and their controls stay the same regardless of configuration.

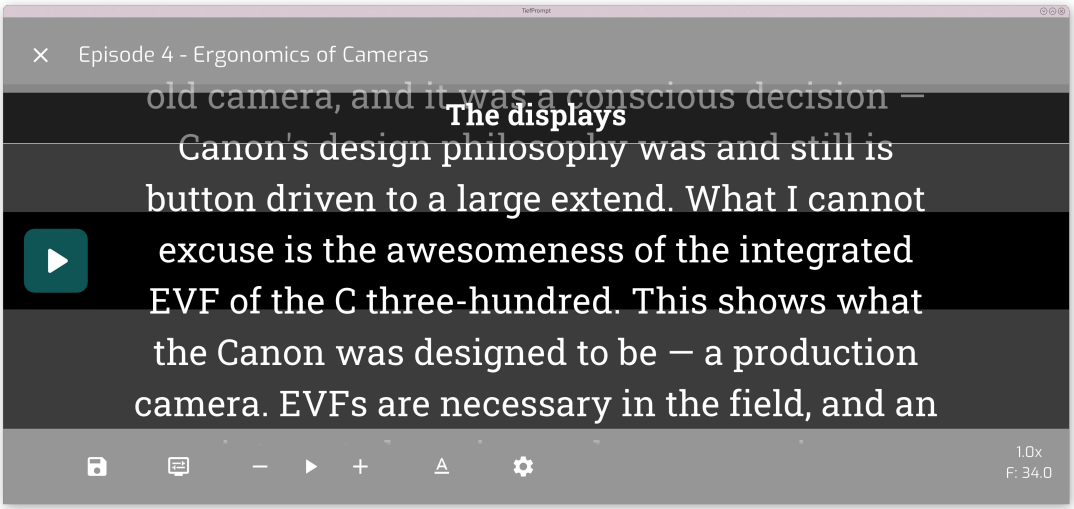


Figure 15: The primary prompter screen, configured with the various tools at hand

2.1. Top Bar

The top bar shows an X button to leave the Prompter Screen and return to script selection, along with the title of the script currently loaded.

2.2. Bottom Bar

The bottom bar holds icons on the left and two readouts on the right.

From left to right, the icons are: Save Prompter Settings, Display Settings overlay toggle, Reduce Speed, Play/Pause, Increase Speed, Text Settings overlay toggle, and Enter Settings.

Save Prompter Settings takes the ephemeral adjustments made through the two overlays below and writes them to the permanent settings. Enter Settings leaves the Prompter Screen for the full settings menu.

The right side shows the current scroll speed in lines per second, and below that, the current font size.

To show/hide the bottom or top bar, tap the screen anywhere outside of the text area.

2.3. Display and Text Overlays

The Display Settings and Text Settings icons each toggle an overlay on top of the scrolling text. Both overlays can be open at the same time, and changes apply immediately. Nothing is written to the permanent settings until Save Prompter Settings is pressed.

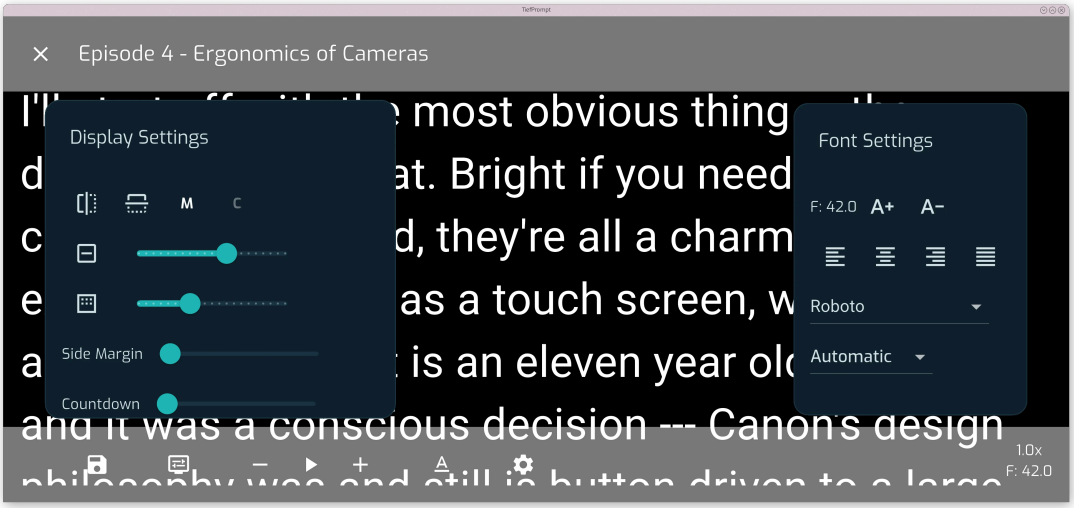


Figure 16: The prompter screen with both the Display Settings and Text Settings overlays open

Each control in these overlays is covered in Display Settings, Text Settings and Scroll Settings.

3. Display Settings

The Display Settings screen, reachable from the main settings menu, holds the defaults applied whenever a script is opened in the Prompter Screen. The overlay described in The Prompter Screen lets you change the same values temporarily during a session; pressing Save Prompter Settings there writes them back to these defaults.

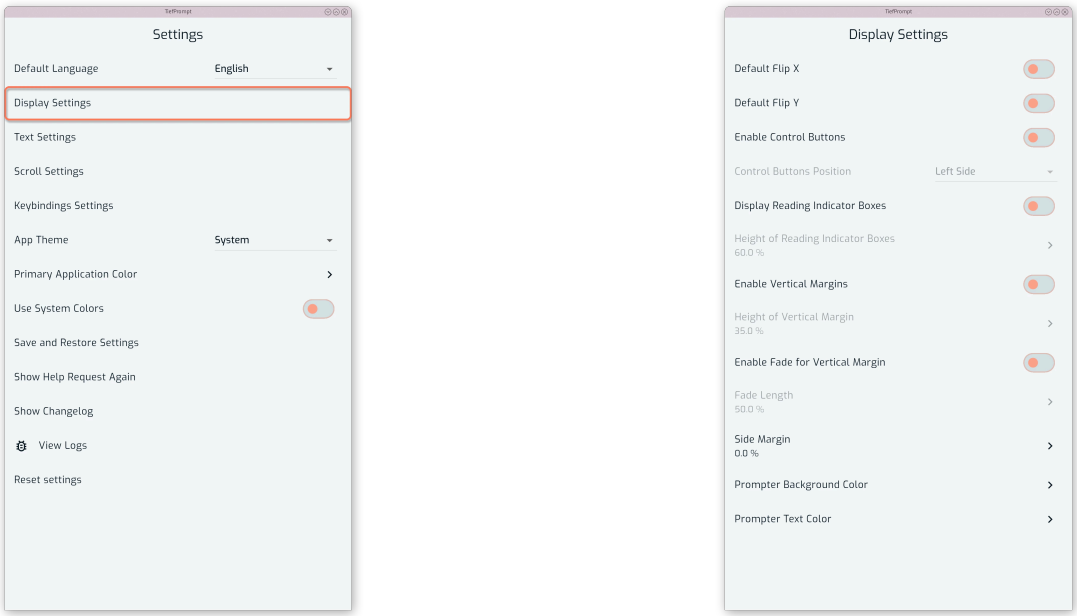


Figure 17: The Display Settings screen

3.1. Flipping

Default Flip X and Default Flip Y mirror the text horizontally or vertically. This is intended for use with a teleprompter rig, where the text is reflected off a piece of glass and needs to be flipped to read correctly.

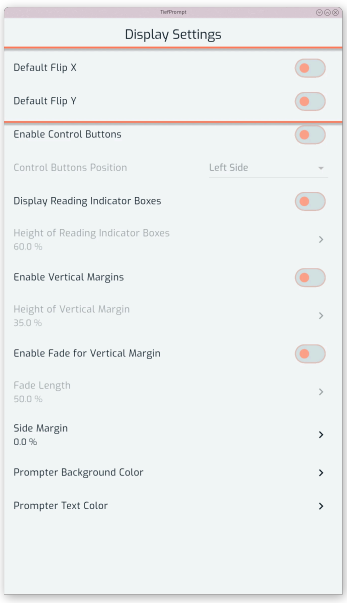


Figure 18: Default Flip X and Default Flip Y toggles

3.2. Control Buttons

Enable Control Buttons shows or hides the on-screen playback controls over the prompter text. These are separate from the controls in the bottom bar, and can be positioned on any side of the screen.

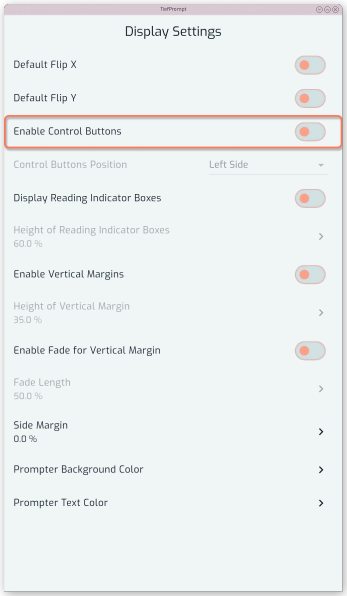


Figure 19: Enable Control Buttons toggle

Control Buttons Position selects which side of the screen they're anchored to: left, right, top or bottom.

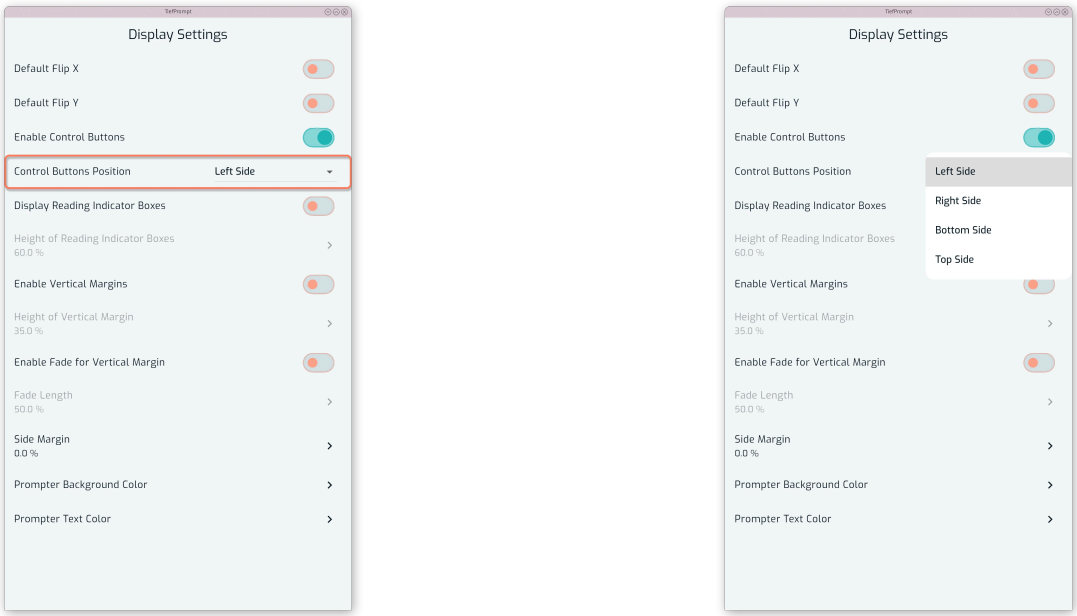


Figure 20: Control Buttons Position dropdown

3.3. Reading Indicators

Display Reading Indicator Boxes draws two boxes across the width of the screen marking the area you should be reading from.

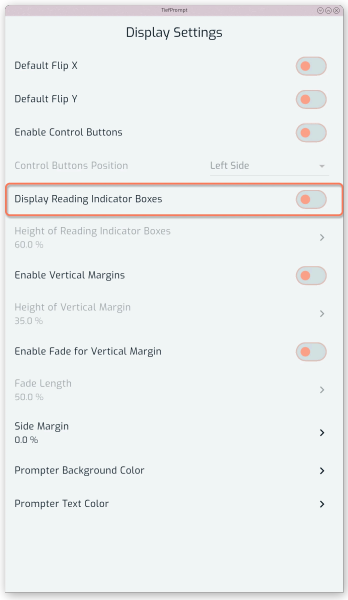


Figure 21: Display Reading Indicator Boxes toggle

Height of Reading Indicator Boxes sets how tall those boxes are, as a percentage of the screen.

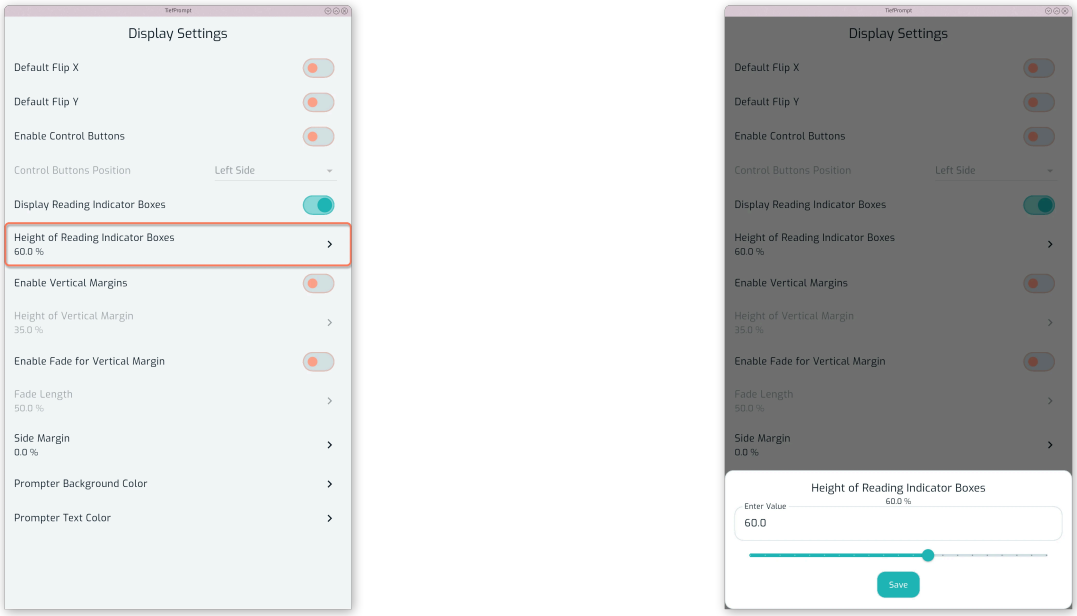


Figure 22: Height of Reading Indicator Boxes slider

3.4. Vertical Margins

Enable Vertical Margins reserves a band of empty space at the top and bottom of the screen so text doesn't scroll all the way to the edges.

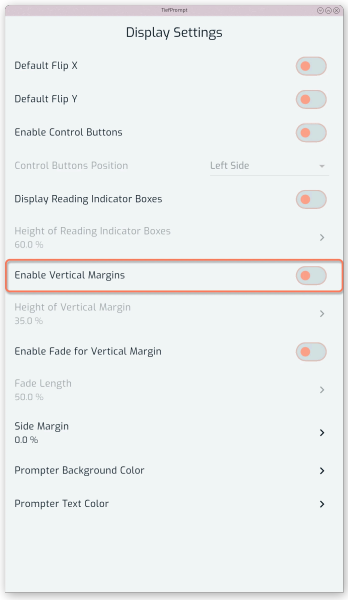


Figure 23: Enable Vertical Margins toggle

Height of Vertical Margins sets how large that band is.

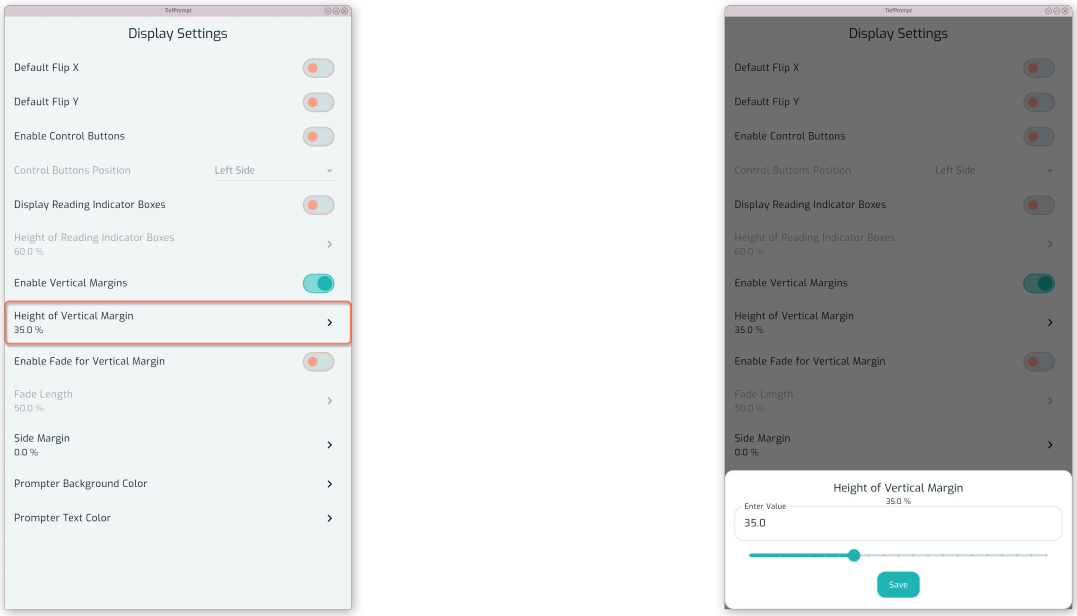


Figure 24: Height of Vertical Margins slider

Enable Fade for Vertical Margin fades the text into the background near the edge of the margin rather than cutting it off sharply.

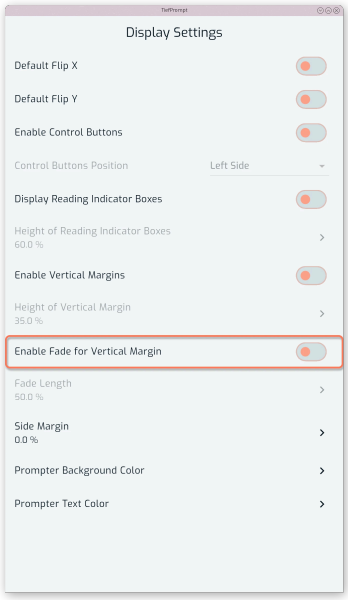


Figure 25: Enable Fade for Vertical Margin toggle

Fade Length controls how far that fade extends.

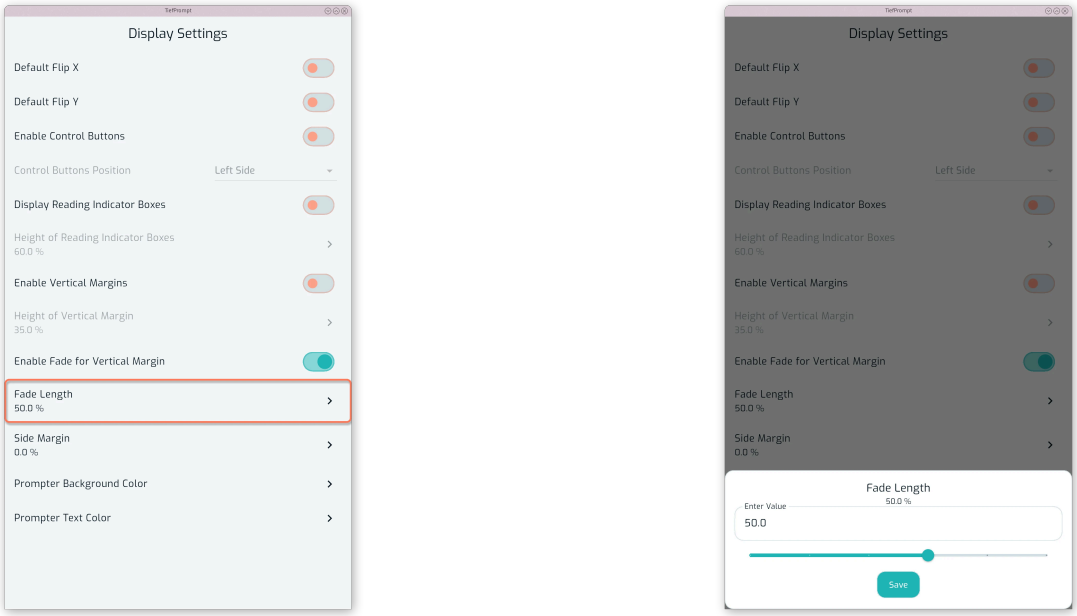


Figure 26: Fade Length slider

3.5. Side Margin

Side Margin adds empty space on the left and right of the text, as a percentage of the screen width.

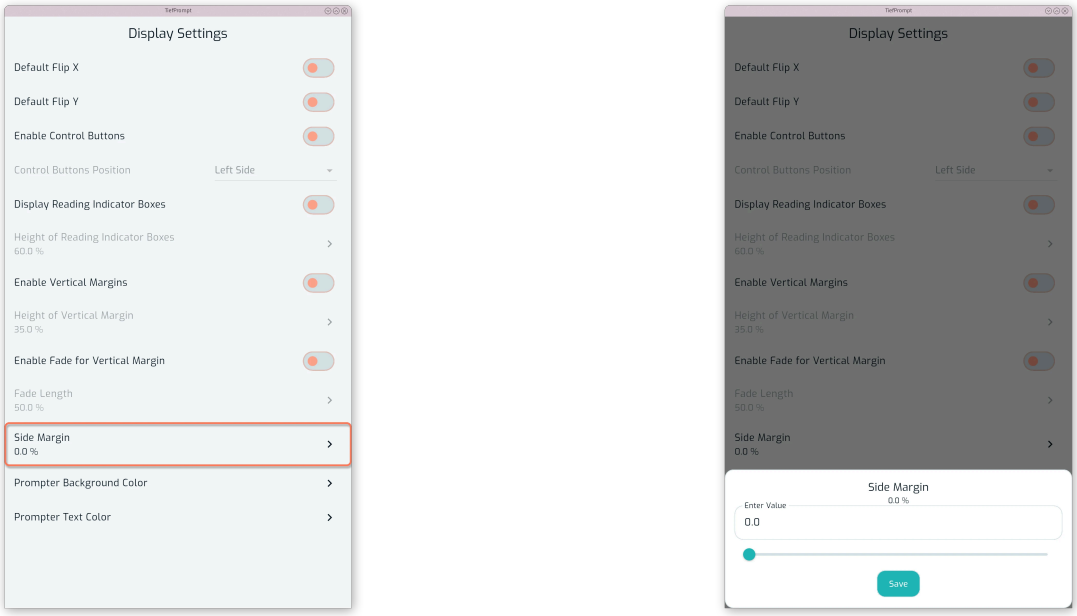


Figure 27: Side Margin slider

3.6. Colors

Prompter Background Color sets the color used behind the scrolling text, independently of the app's own theme and primary color.

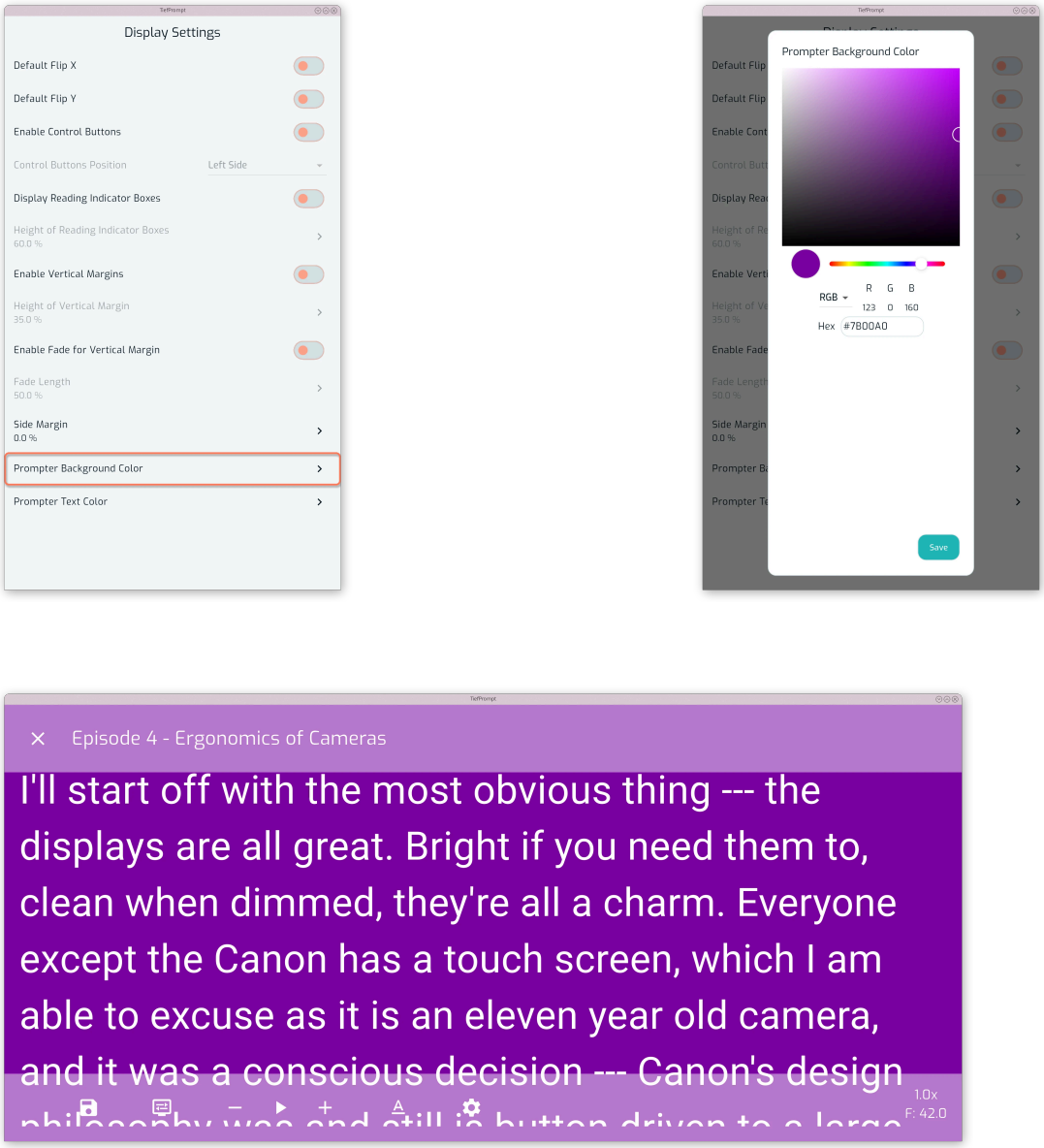


Figure 28: Prompter Background Color picker

Prompter Text Color sets the color of the scrolling text and the affordances of the app themselves.

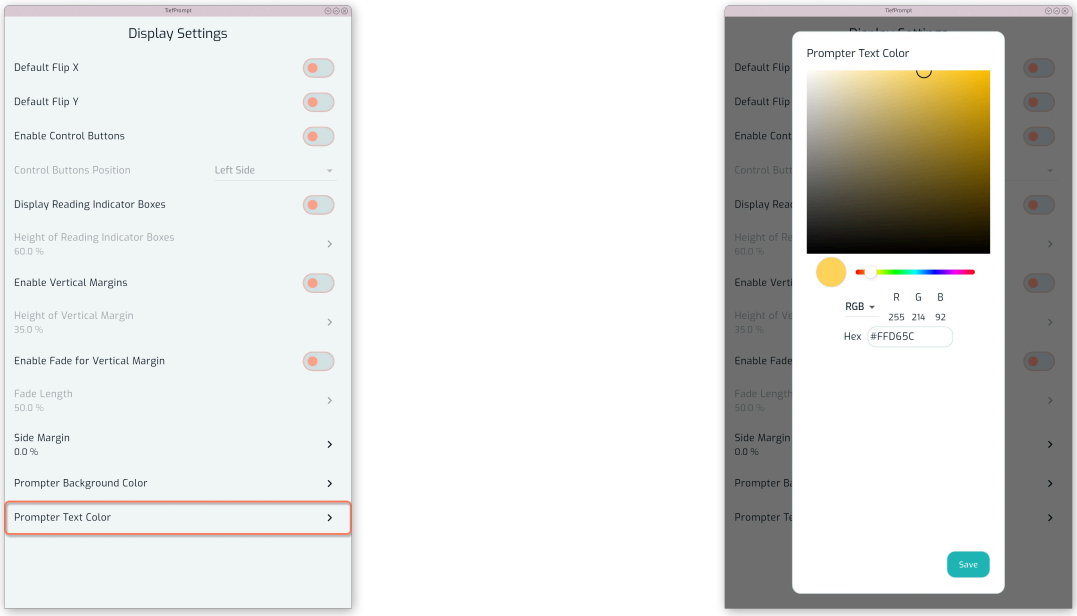


Figure 29: Prompter Text Color picker

4. Text Settings

The Text Settings screen is the primary entrypoint for changing how the text looks in the prompter. The default settings are opinionated — but every setting is configurable.

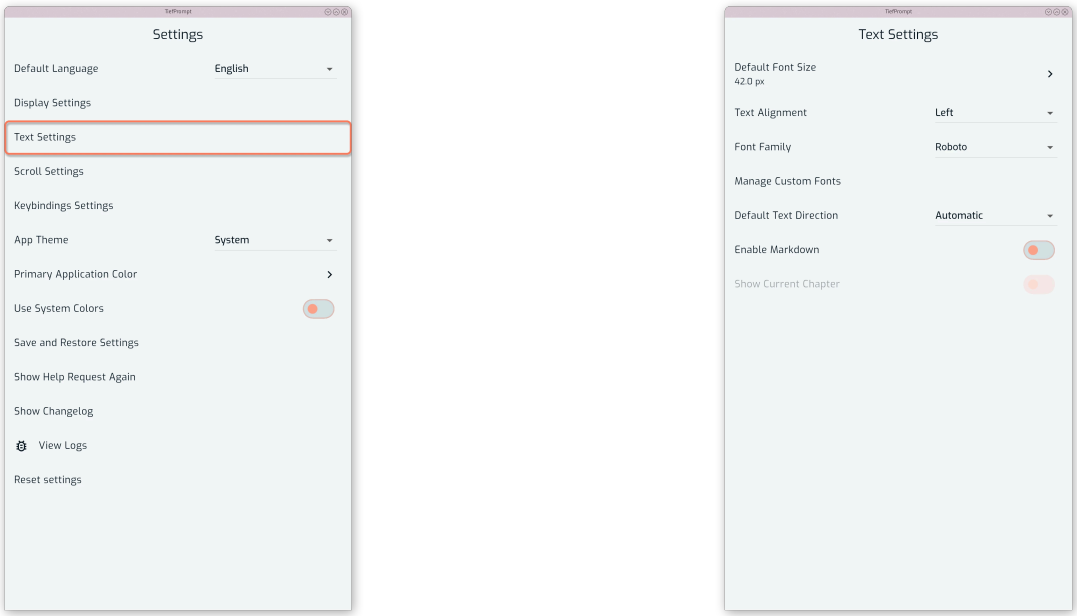


Figure 30: The Text Settings screen

4.1. Text Display Settings

In the Text Settings screen, the text size can be adjusted with a slider by pressing the “Default Font Size” entry:

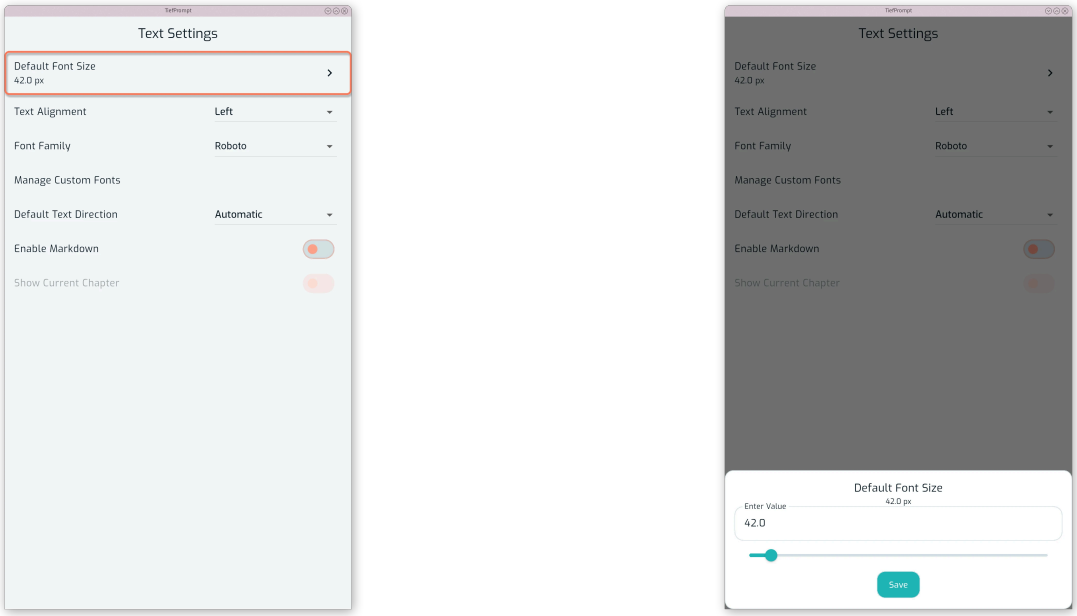


Figure 31: The “Default Font Size” dialog open

The alignment of the text can also be changed via a dropdown. Available settings are left, centered, right, and justified. The default is left.

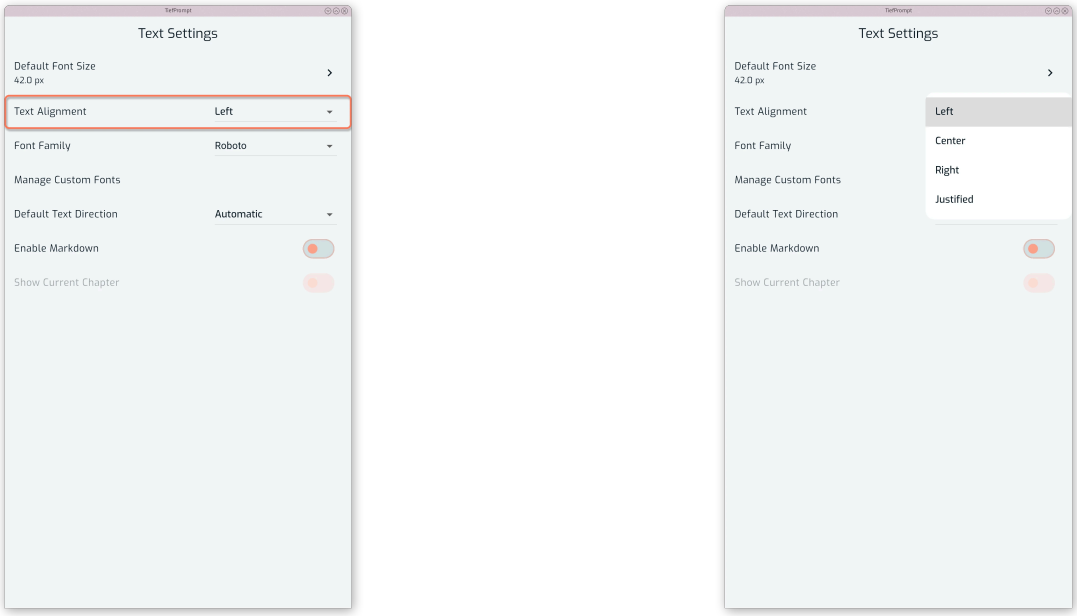


Figure 32: The “Text Alignment” dropdown open

4.2. Font Settings

The font used in the scrolling text and markdown chapter headings (see Text Settings) can be changed via the “Font Family” dropdown. The default is “Roboto”.

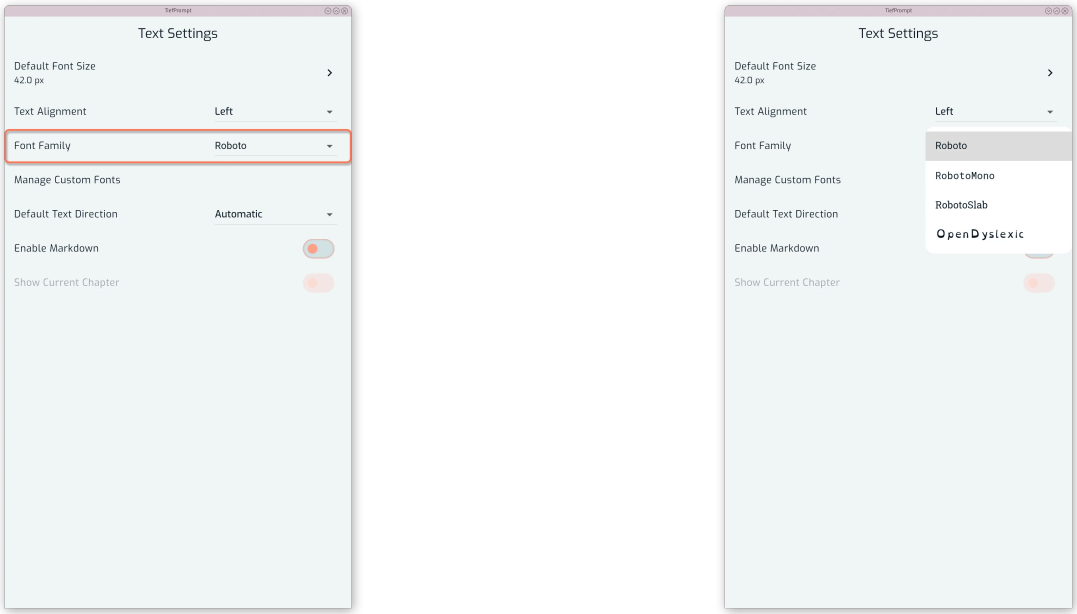


Figure 33: The “Font Family” dropdown open

4.3. Custom Fonts

There is also a posibility to add a custom font to the app. This is done by pressing the “Manage Custom Fonts” entry to get to the settings page, which shows a list of currently installed fonts, as well as the preinstalled fonts, and allows for adding or removing fonts via a file selector. The font file must be in TTF or OTF format, and will be copied to the app’s database. You need not keep the font file around after adding it to the app.

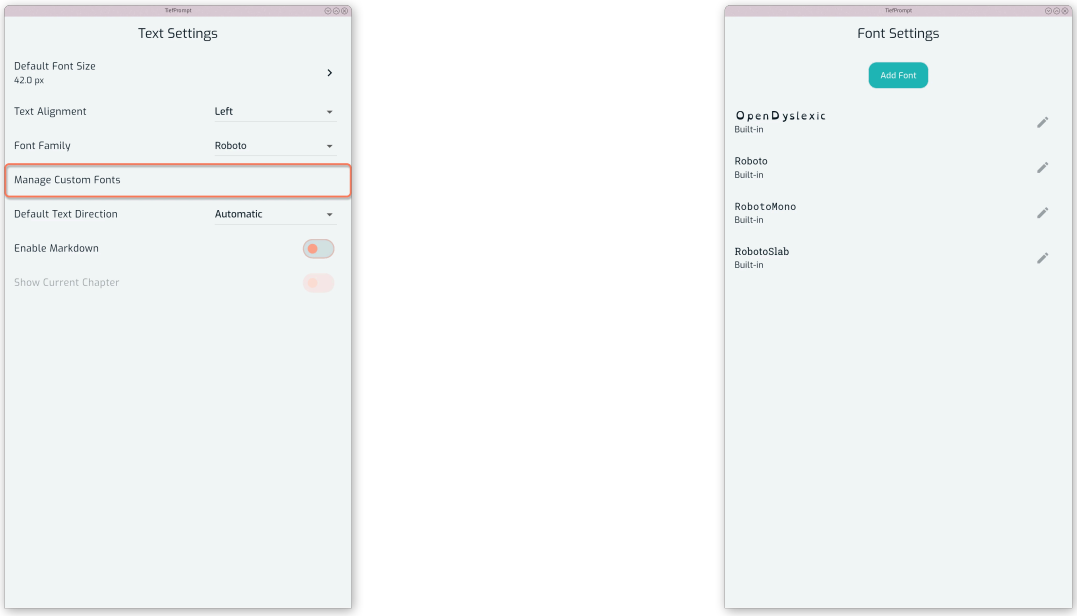


Figure 34: The “Manage Custom Fonts” settings page

Each font can have multiple versions — if you, say, have a font with a regular, a bold, a italics and a bold-italics version as separate files, you can add them all to the app. After adding a font family, you are able to rename it by pressing the edit icon next to it. This will open a dialog where you can enter the new name. This is useful if the app does not detect the font family correctly, or if you want to change the name of a font family.

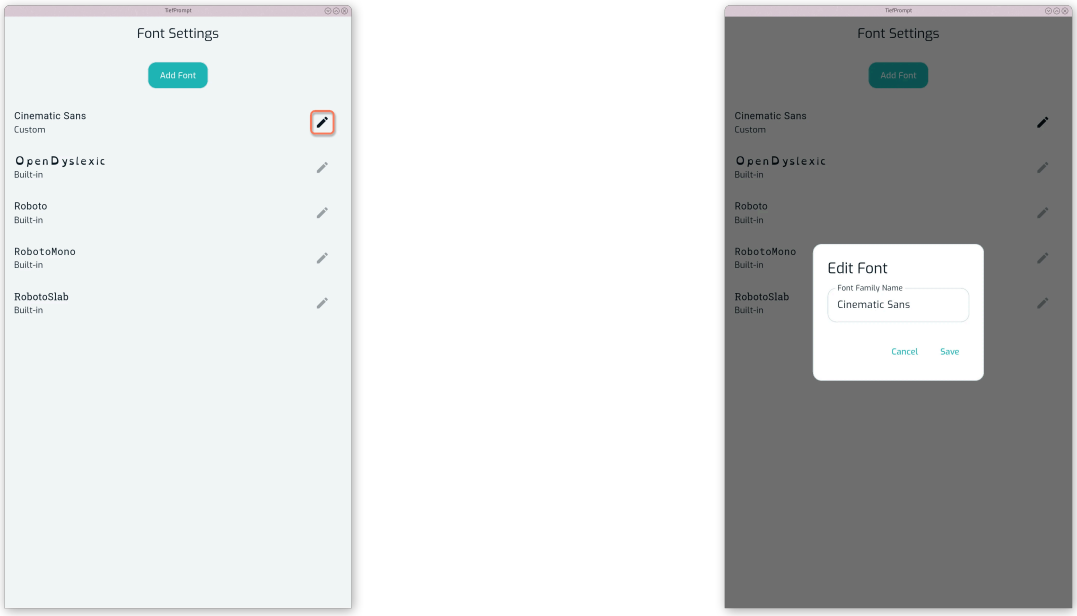


Figure 35: The “Rename Font Family” dialog open

If the font name integrated in the files matches (not the file name!), the app will automatically detect the different versions and use them accordingly. If your fonts do not have matching names, you can manually move the font variant by clicking on a font family and pressing the move icon next to the variant. This will open a dialog where you can select the font family to move the variant to. Integrated fonts cannot be added to or moved.

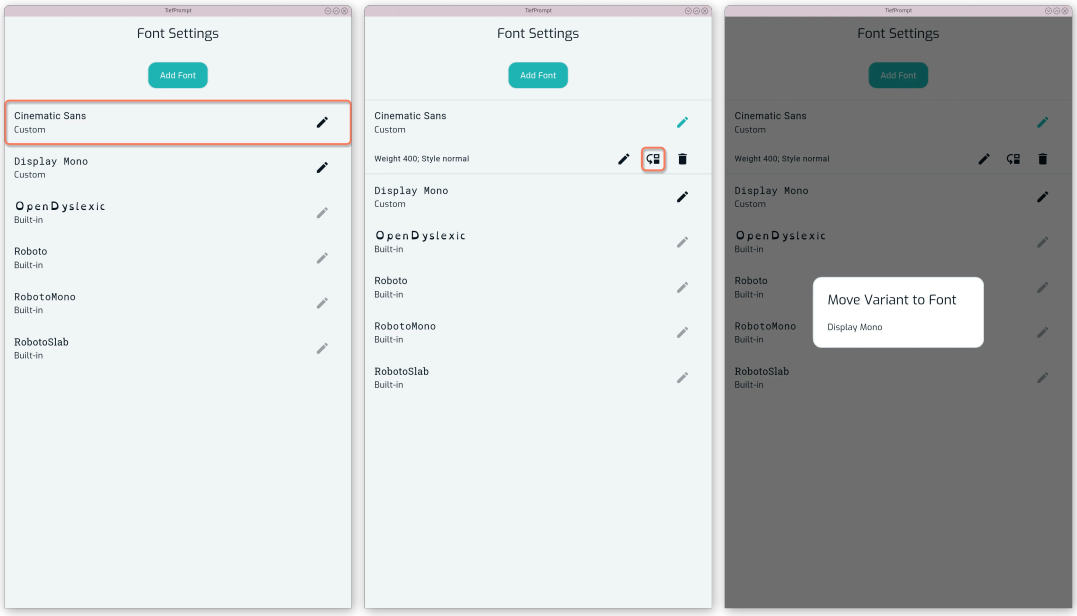


Figure 36: The “Move Font Variant” dialog open

You can also remove a font variant by pressing the trashcan icon next to it. This will remove the font from the app’s database, but will not delete the original file.

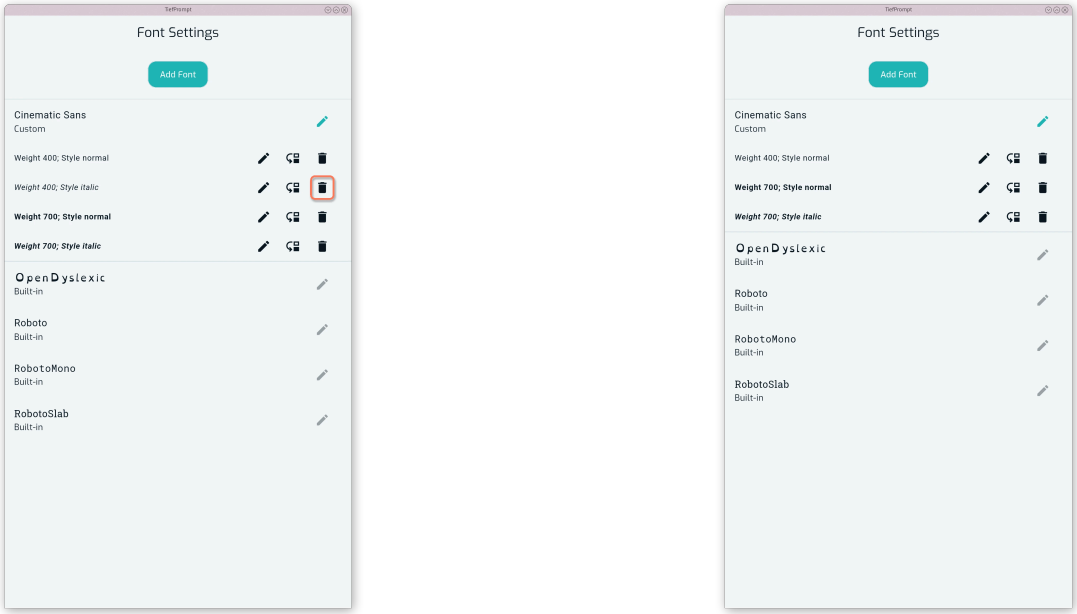


Figure 37: “Delete Variant” highlighted

When deleting the last variant of a font family, the font family will also be removed from the app’s database.

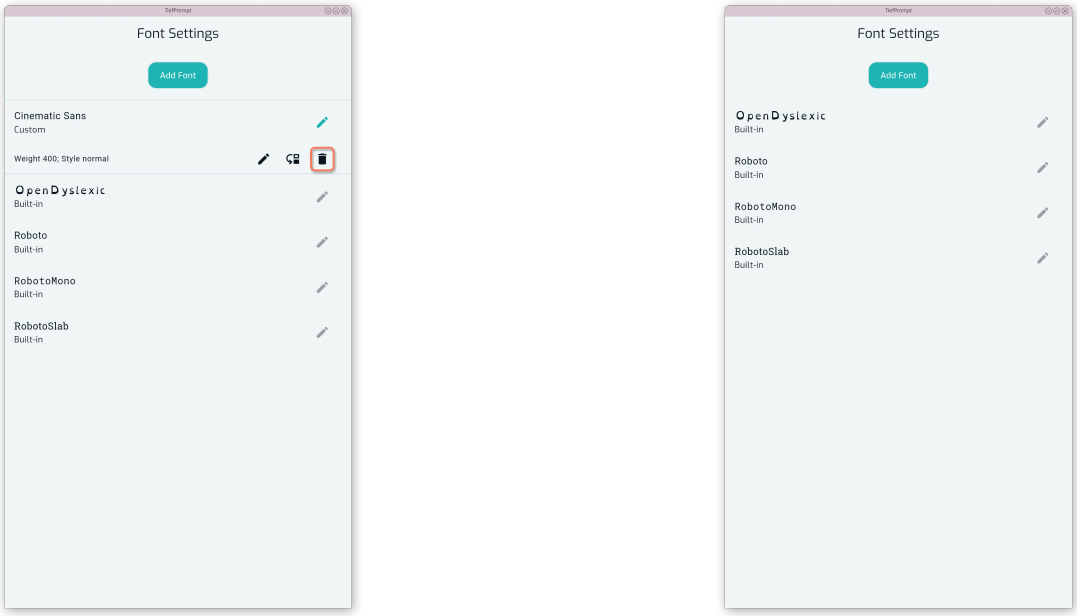


Figure 38: “Delete Variant” highlighted, with last variant of a font family being deleted

Lastly, by pressing the edit icon next to a font variant, you can edit the variant’s weight and whether it is an italic font or not. This is useful if the app does not detect the variant correctly, or if you want to change the weight of a font variant.

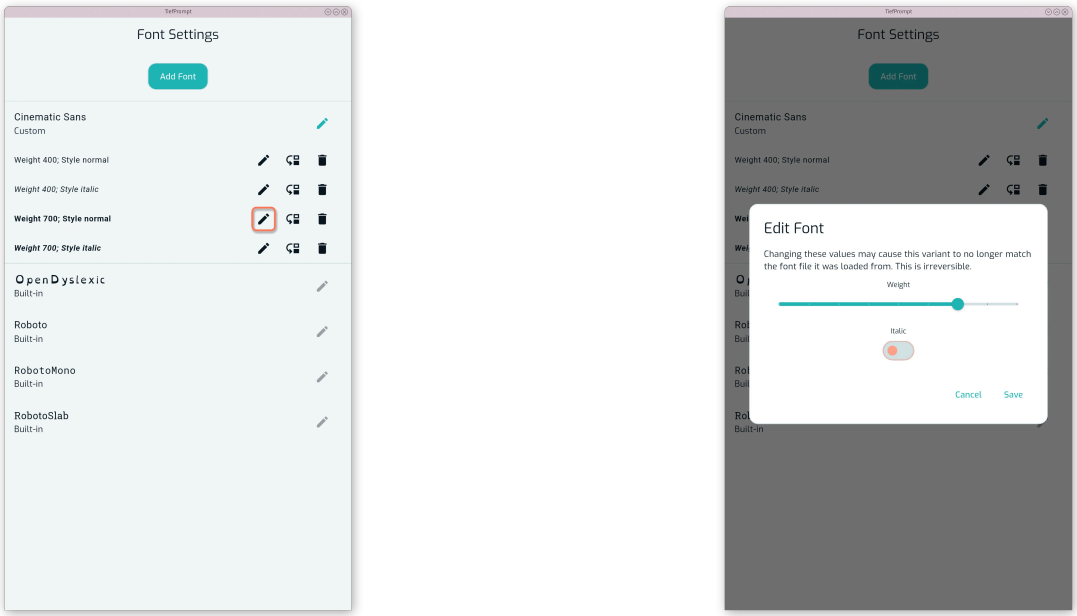


Figure 39: The “Edit Font Variant” dialog open

4.4. Markdown

Markdown rendering can be enabled via the “Enable Markdown” toggle. This will render the text in the prompter as Markdown, allowing for formatting such as bold, italics, headings, and more. The default is off.

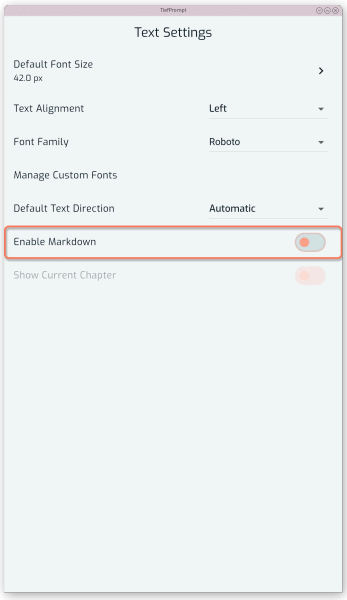


Figure 40: The “Enable Markdown” toggle

This is not a full Markdown renderer, but it does support the most common formatting options. For more information on what is supported, see Markdown Rendering.

4.5. Current Chapter Display

The toggle “Show Current Chapter” controls whether the current chapter is displayed in the prompter. This can only be enabled if Markdown is enabled. The default is off.

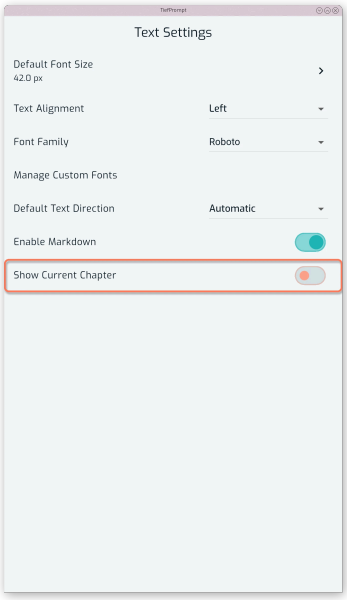


Figure 41: The “Show Current Chapter” toggle

When the current chapter is displayed, it will be shown in a separate line above the scrolling text, and will be updated as the text scrolls. The chapter is determined by the first heading in the Markdown text that has been scrolled by in the prompter.

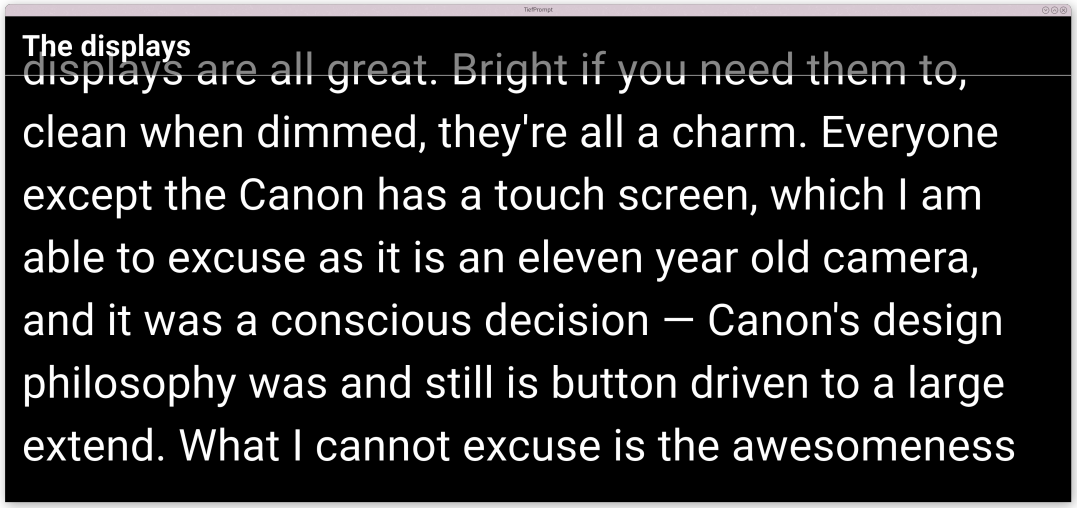


Figure 42: Example of the current chapter being displayed in the prompter

When toggled, the current chapter heading doubles as a clickable element to jump to the previous most chapter. You can thus press the heading and jump to the current chapters start.

Additionally, keybindings are available to jump to the previous/next chapter. See Keybindings

4.6. RTL Rendering

TiefPrompt allows forcing either RTL or LTR text rendering in the prompter. Per default, text is rendered in accordance with standard BIDI algorithms, however, that can be changed in the display settings.

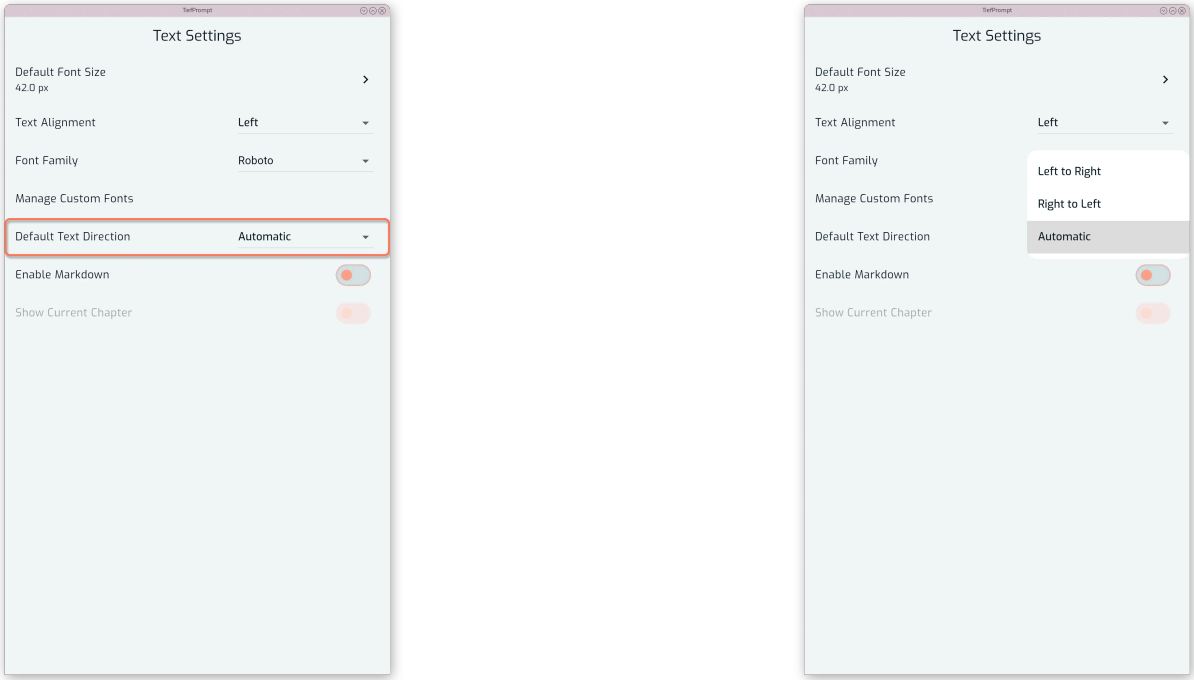


Figure 43: Rendering Direction selection

5. Scroll Settings

The Scroll Settings screen is the configuration section for scrolling and related settings.

5.1. Scroll Speed

Default Scroll Speed sets the speed the text scrolls at when a script is opened, in lines per second.

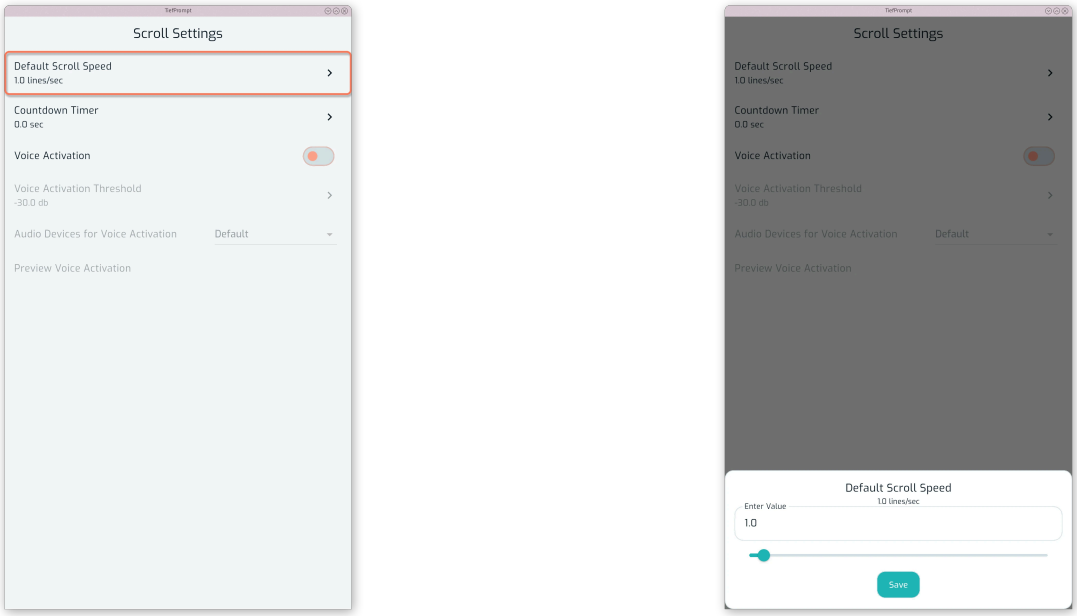


Figure 44: Default Scroll Speed slider

5.2. Countdown Timer

Countdown Timer sets how many seconds pass between pressing play and the text starting to scroll.

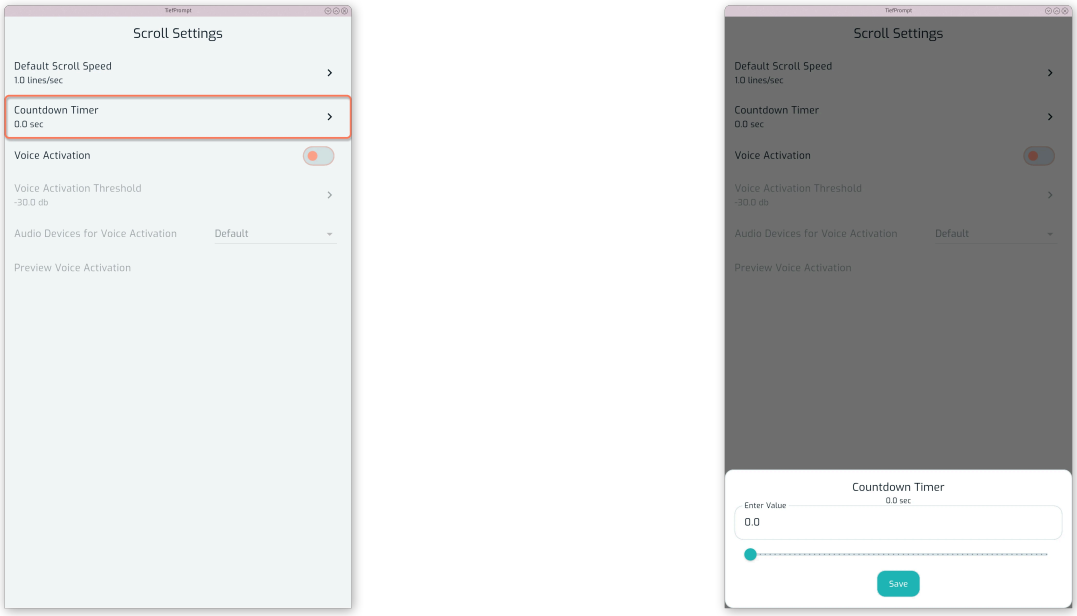


Figure 45: Countdown Timer slider

5.3. Voice Activation

Voice Activation allows users to control the teleprompter with their voice. You can enable voice activation and test it on the Scroll Settings screen.

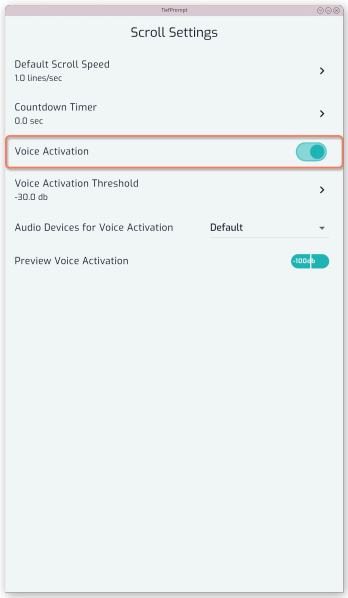


Figure 46: Voice Activation toggle

When voice activation is enabled, the prompter only scrolls when it detects sound above the configured threshold.

You can adjust the sensitivity of the voice activation to ensure it responds appropriately to your voice.

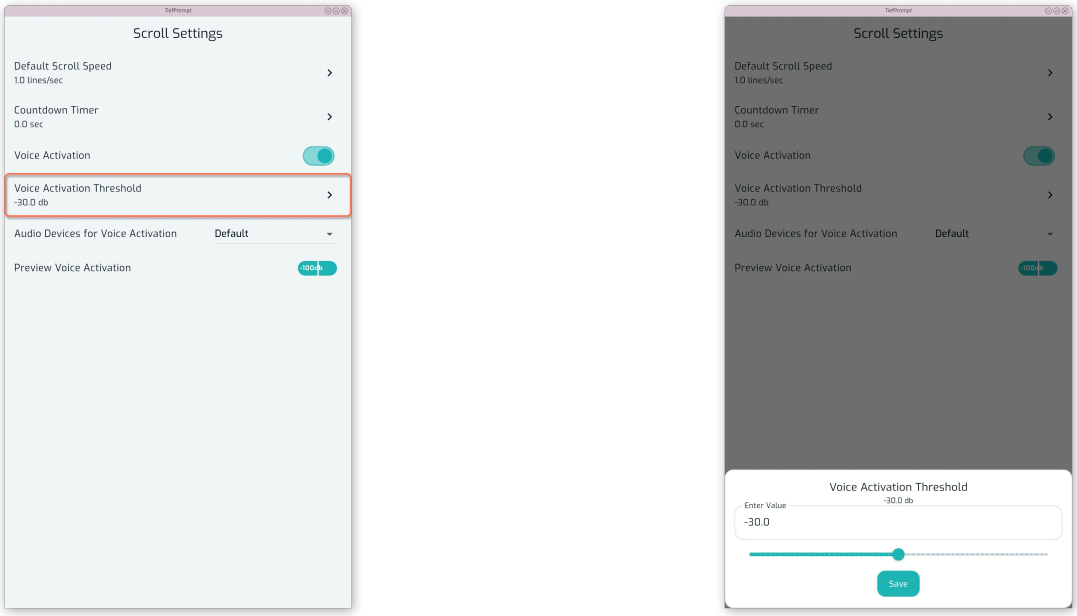


Figure 47: Voice Activation sensitivity slider

The audio device used can be selected from the dropdown menu.

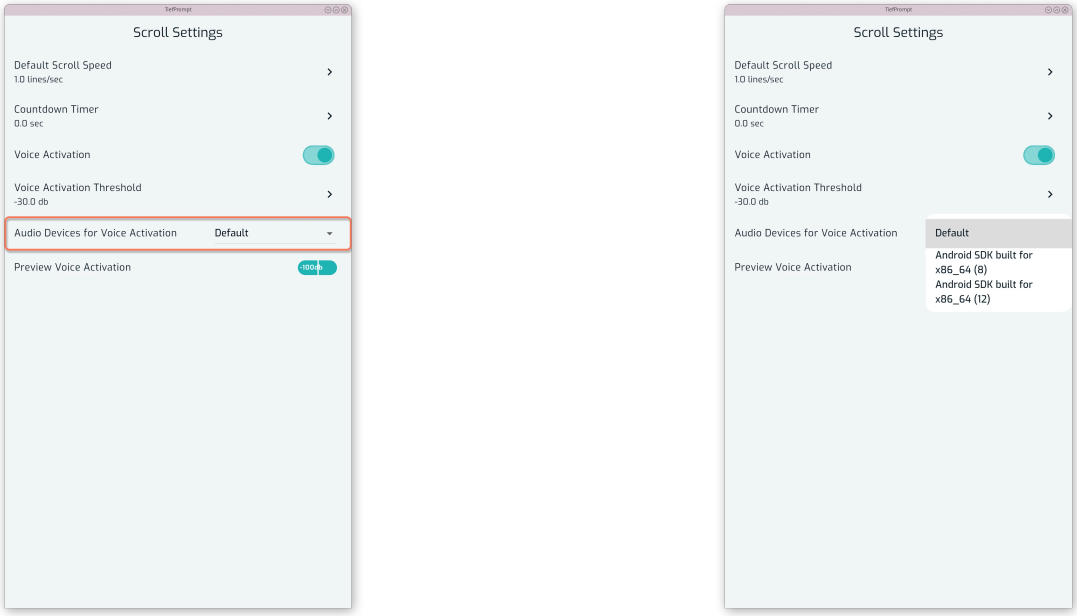


Figure 48: Audio Device dropdown

You may test the voice activation by enabling the Preview Voice Activation option.

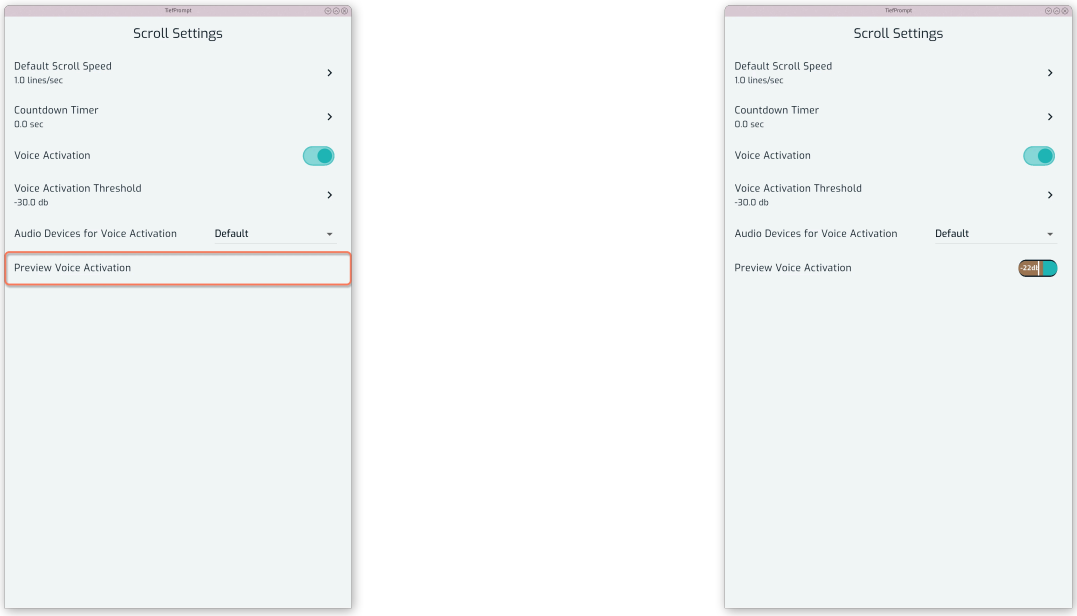


Figure 49: Preview Voice Activation toggle

Learn more at Voice Activation.

6. Keybindings

The keybindings screen is the primary entrypoint for changing how to control the prompter. The default controls are opinionated — but every keybinding is configurable.

Keybinding configuration is primarily for keyboards and controllers. Mouse buttons are not configurable, and the mouse wheel is always used for scrolling. Each binding (an action in the prompter) can have an arbitrary amount of keyboard shortcuts and controller buttons assigned to it, and a shortcut can be assigned to multiple actions.

> ⚠ As per current version v1.1.0+, there is a bug where only the first listed action for a keybinding is actually used. This means that if you assign the same key to multiple actions, only the first one will be triggered. This will be fixed in a future release.

6.1. Changing Keybindings

The Keybindings can be changed in the Keybindings Settings screen, which is accessible from the settings menu. Each action has a list of currently assigned shortcuts, and can be assigned a new one or have an old one removed by pressing the keybinding in the list. This will open a dialog as shown below.

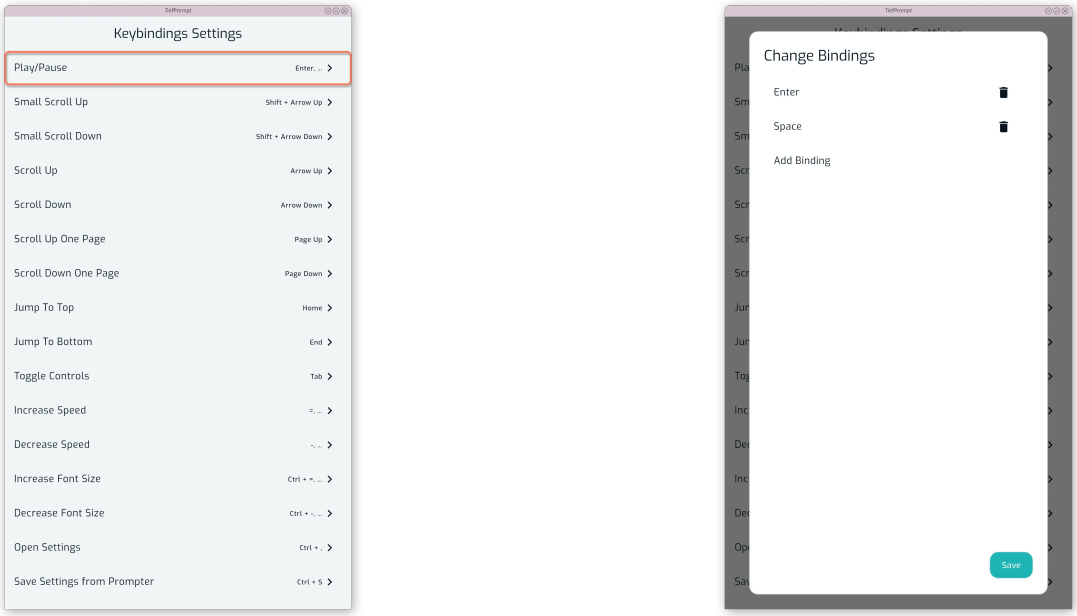


Figure 50: The Keybinding Settings screen, with the “Change Bindings” dialog open

A new keybinding can be assigned by pressing the “Add Binding” button, which opens another dialog where you can press the desired key or controller button.

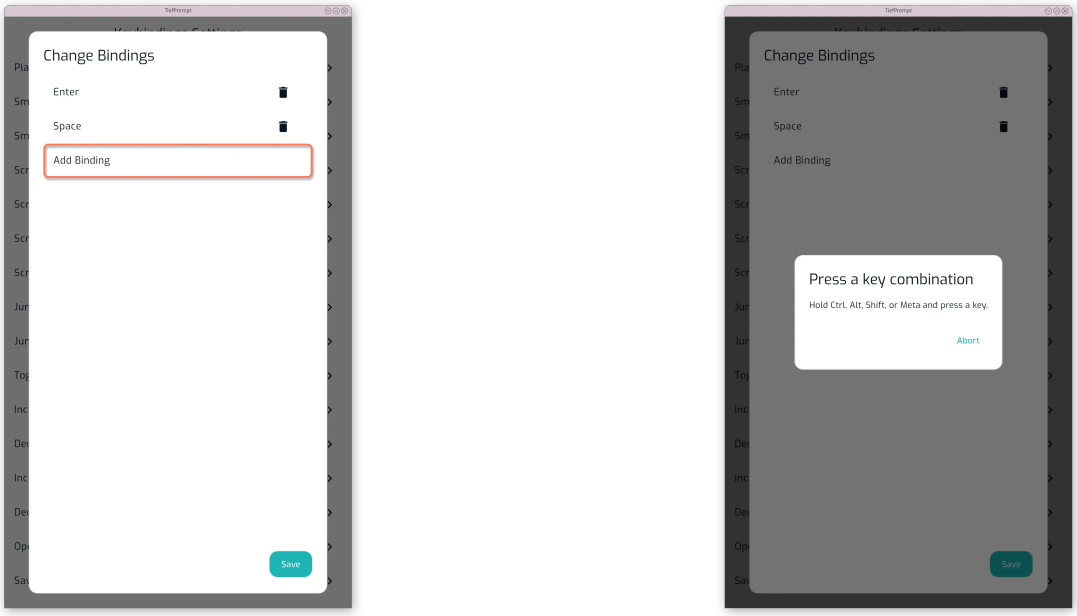


Figure 51: The “Add Binding” dialog, waiting for a key or controller button to be pressed

A keybinding can be removed by pressing the trashcan icon next to it in the list of currently assigned bindings. This will remove the binding immediately, with no confirmation dialog.

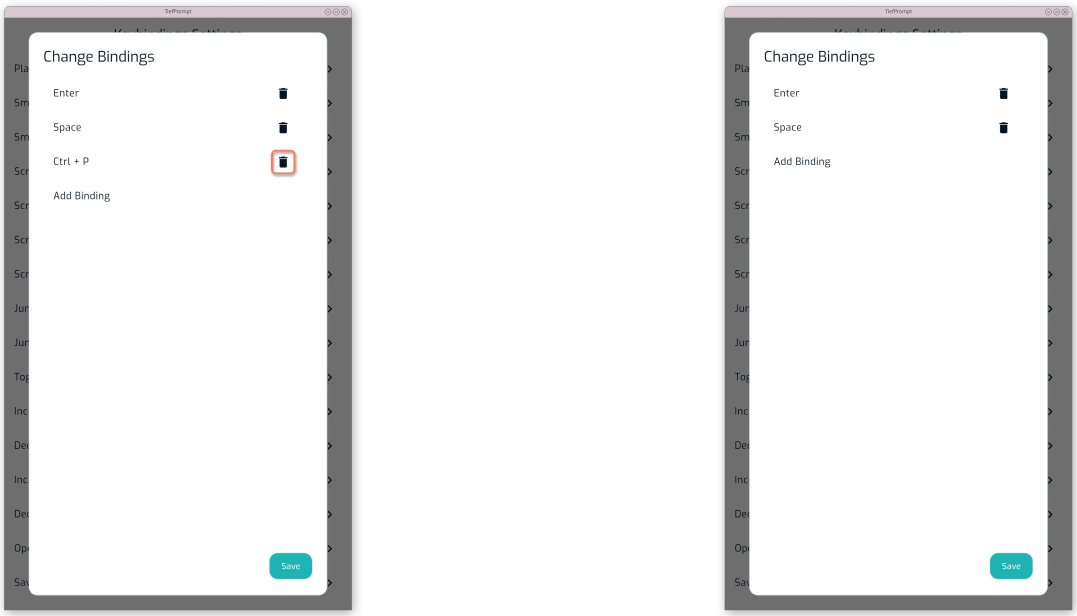


Figure 52: The Keybinding Settings screen, with the “Change Bindings” dialog open and a keybinding being removed

6.2. Default Keybindings

The default keybindings are as follows:

- Play/Pause: `Enter`, `Space`
- Scroll Up Small: `Shift + Up Arrow`
- Scroll Down Small: `Shift + Down Arrow`
- Scroll Up: `Up Arrow`
- Scroll Down: `Down Arrow`

- Page Up: `Page Up`
- Page Down: `Page Down`
- Jump to Start: `Home`
- Jump to End: `End`
- Toggle Controls: `Tab`
- Speed Up: `=`, `Numpad +`
- Speed Down: `-`, `Numpad -`
- Font Size Up: `Ctrl + =`, `Ctrl + Numpad +`
- Font Size Down: `Ctrl + -`, `Ctrl + Numpad -`
- Open Settings: `Ctrl + ,`
- Save Settings from Prompter: `Ctrl + S`
- Jump to Previous Chapter: `,`
- Jump to Next Chapter: `.`

7. Save and Restore Settings

As customization is an integral part of TiefPrompt, and we're aware of multiple-user workflows, there is the option to save and restore settings. That is positioned right below the primary color control, giving access to a list of named, saved settings as well as the option to import or export settings files.

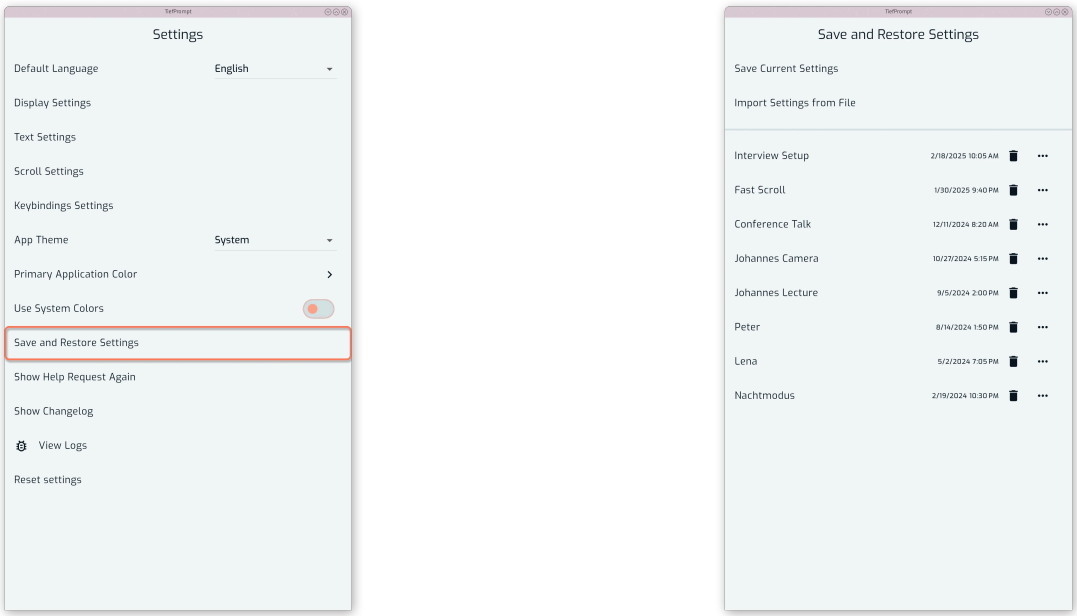


Figure 53: The Saved Settings Screen

To save settings, press the button labeled "Save Current Settings" and enter a name. The new setting appears in the list below, and you can apply it by pressing on it.

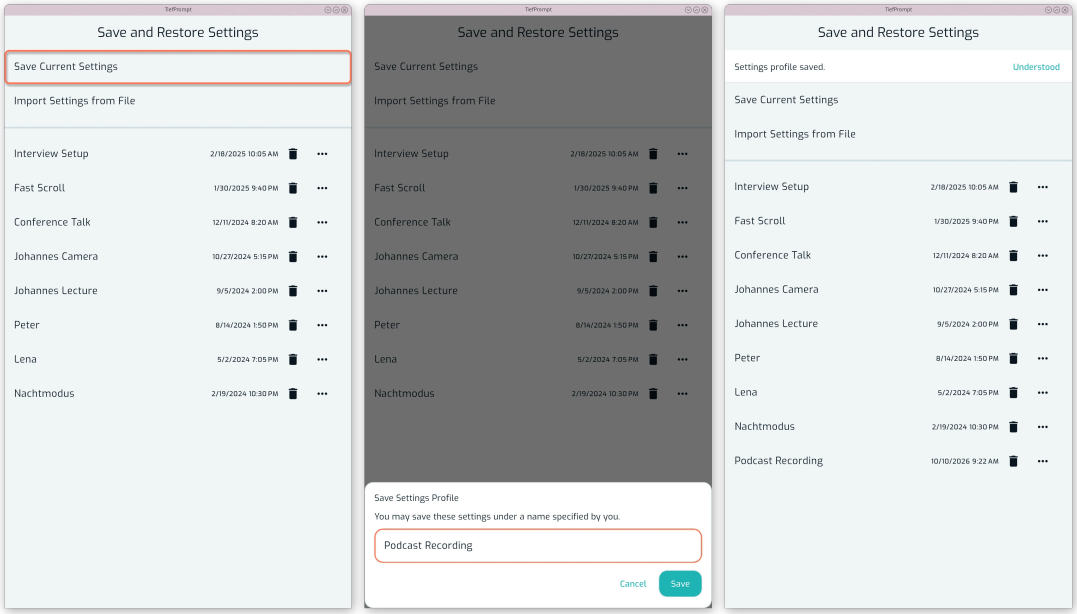


Figure 54: Instructions to Save Settings

When applying settings, the whole app will be updated. Any currently unsaved changes will be lost, so make sure to save your current settings before applying a new one.

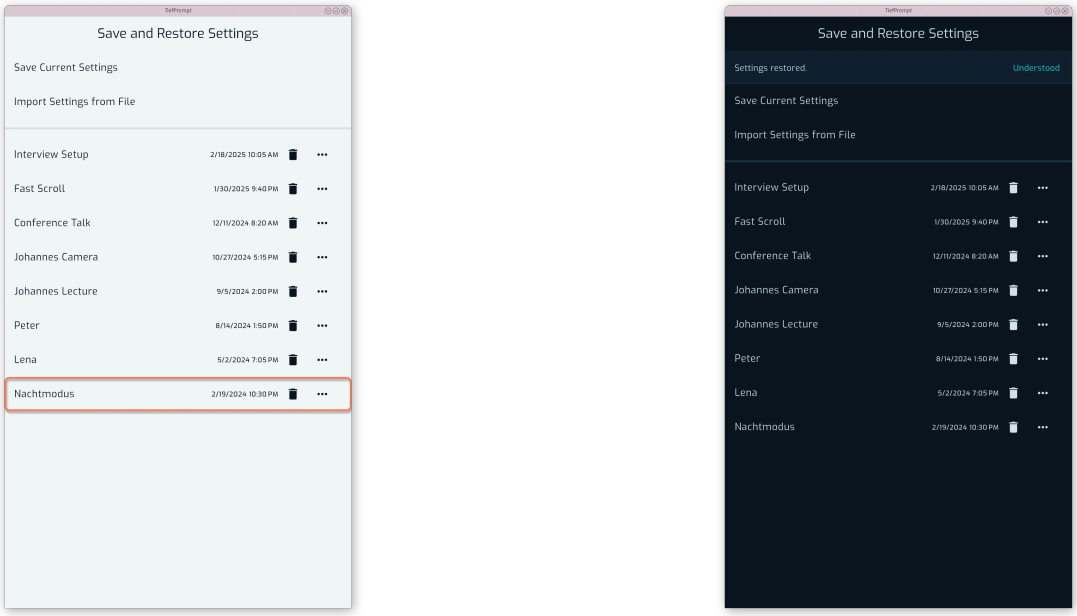


Figure 55: Instructions to Apply Settings

Once saved, you can also export or delete a settings profile by pressing the three buttons to the right of each profile:

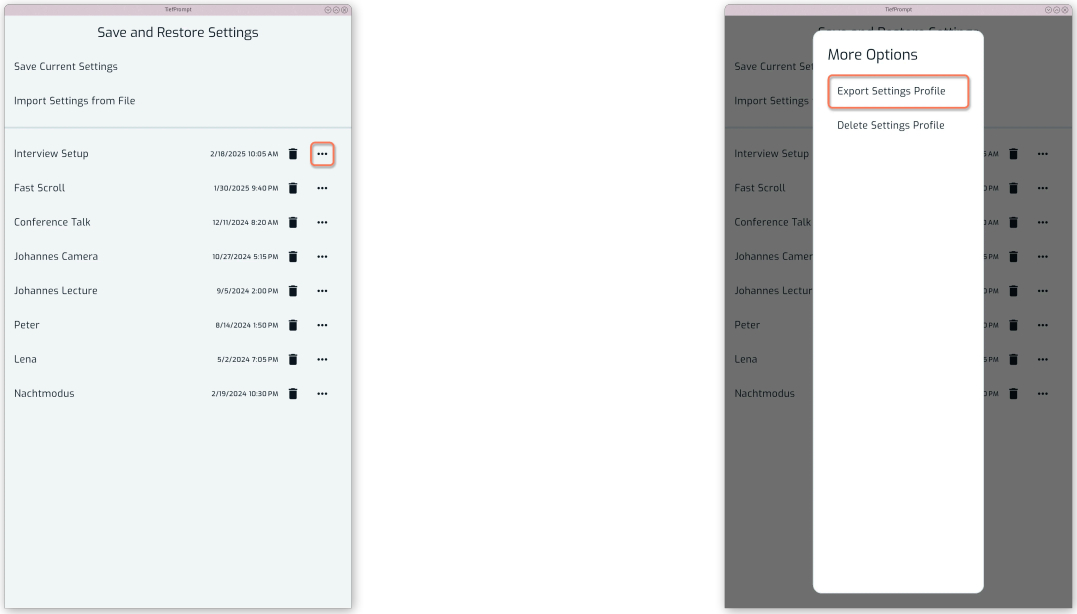


Figure 56: Instructions to Export Settings

After entering the file name and location of your settings profile, you are able to copy it to a different device and import it from the same screen:

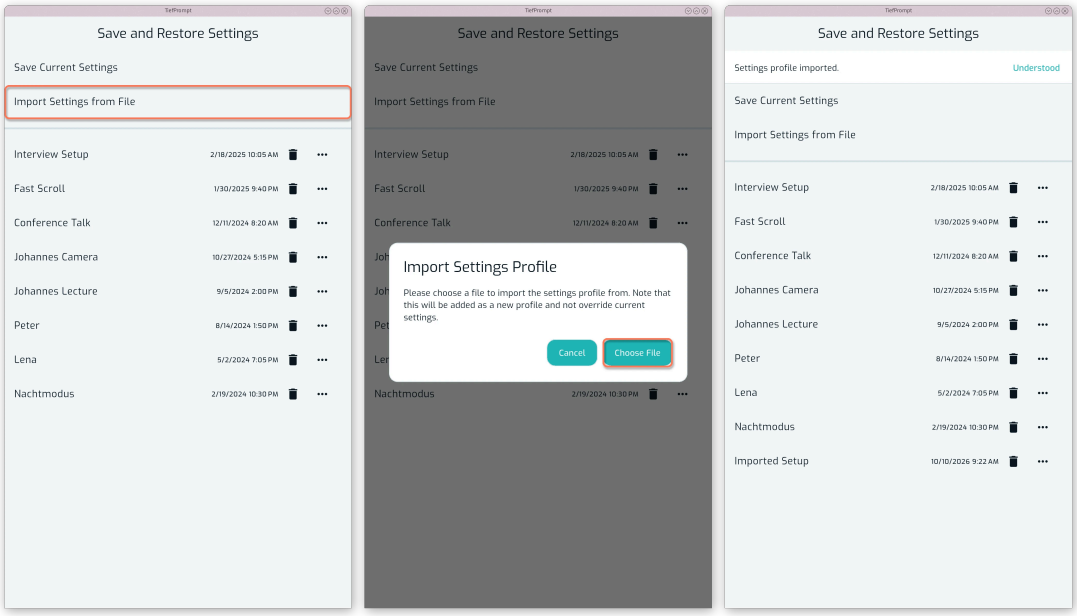


Figure 57: Instructions to Import Settings

To delete a profile, press the trashcan icon next to it and confirm.

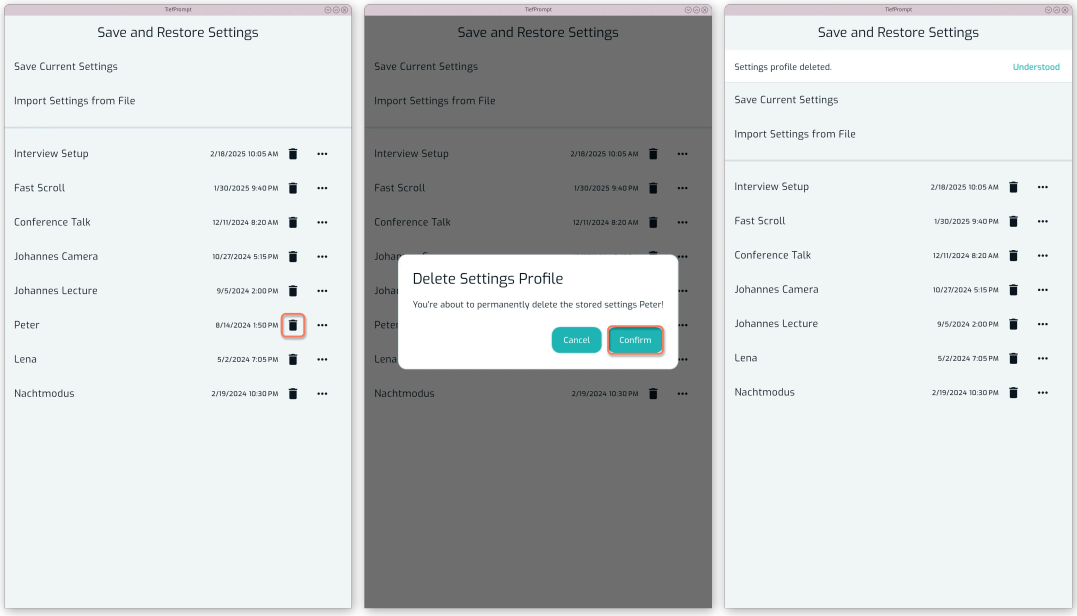


Figure 58: Instructions to Delete Settings

8. App Variants

TiefPrompt is available in multiple variants, each with its own set of features and capabilities. The main variants are:

- FOSS: The “Free and Open Source Software” version, which provides all features fully for free. This version is available on GitHub and F-Droid.
- Free: The freemium version available on the App Store, which has some features locked behind an in-app purchase.
- Pro: The freemium version available on the App Store, which has all features unlocked after an in-app purchase.

The core functionality of TiefPrompt is the same across all variants, but the availability of certain features may differ. The FOSS version is fully open-source and free to use, while the Free and Pro versions may require payment for access to certain features.

No proprietary dependencies are leaked into the FOSS version. The FOSS version is fully separated from in-app purchase logic and does not transmit any data to third-party services. The Free and Pro versions, however, include proprietary dependencies and in-app purchase logic. Read our privacy policy° for more information on data collection and usage.

Currently, the app variant can be viewed in the home screen, below the primary buttons. Clicking on it will show a dialog with information about the installed variant.

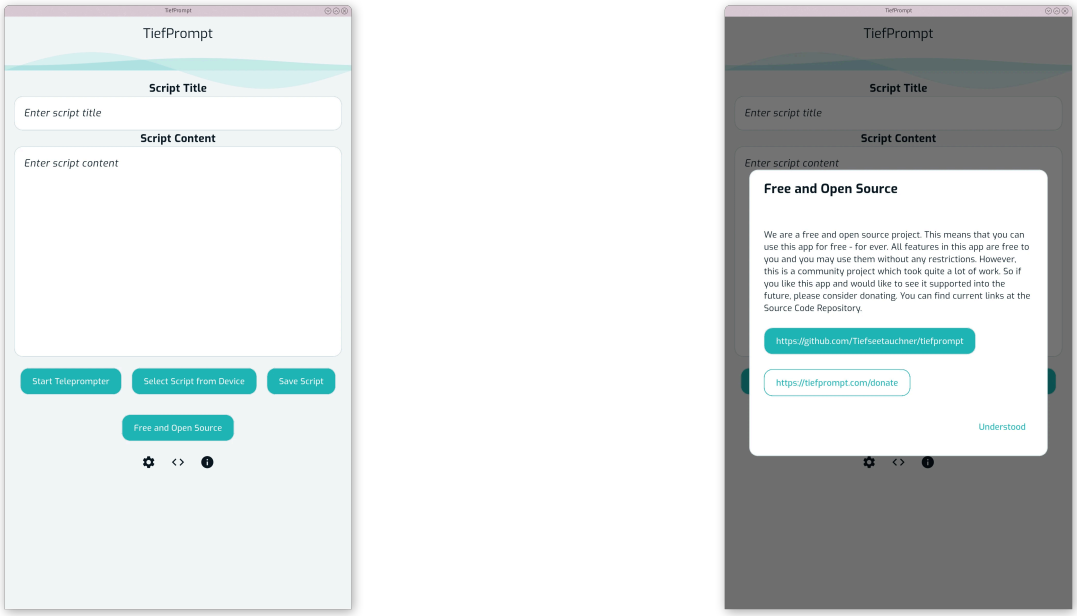


Figure 59: The FOSS variant dialog

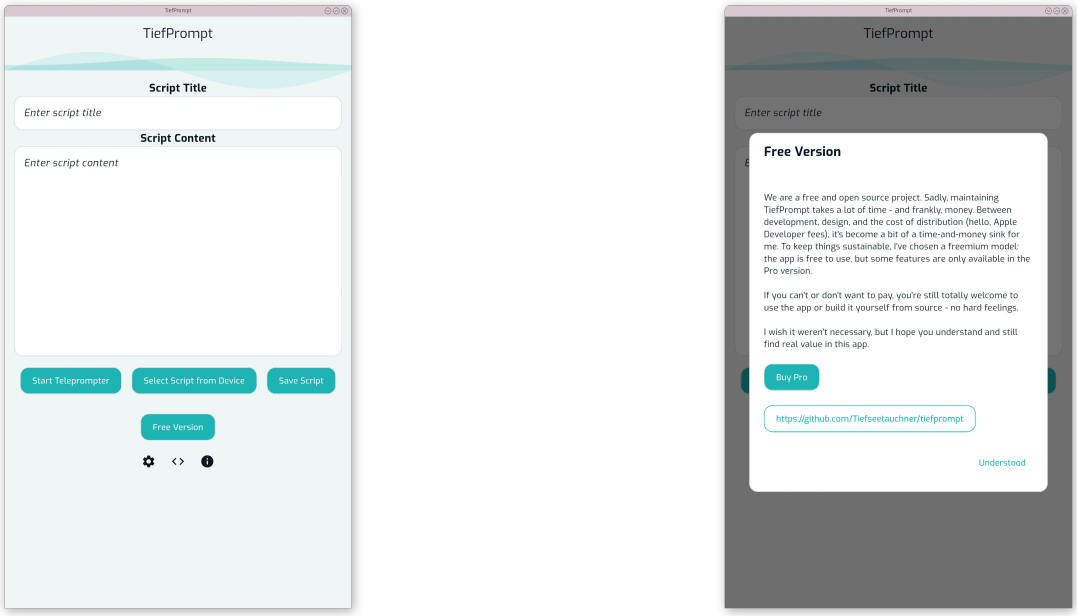


Figure 60: The Free variant dialog

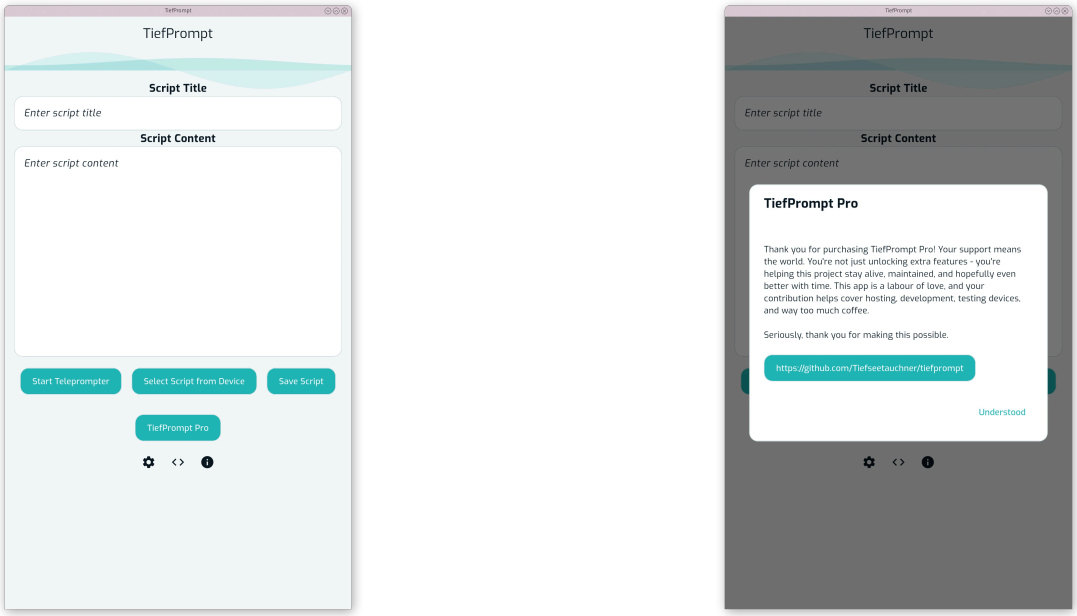


Figure 61: The Pro variant dialog

8.1. Gated features

Currently, the following features are gated behind an in-app purchase in the Freemium variant.

- Reading Indicator Boxes
- Vertical Margins
- Custom Prompter Colors
- Keybindings
- Markdown support and Current Chapter Display
- Saving and Restoring of Settings

When a feature is gated, it will be shown in the menu and prompter as locked. Pressing on it will show a dialog where you can purchase the Pro version to unlock it.

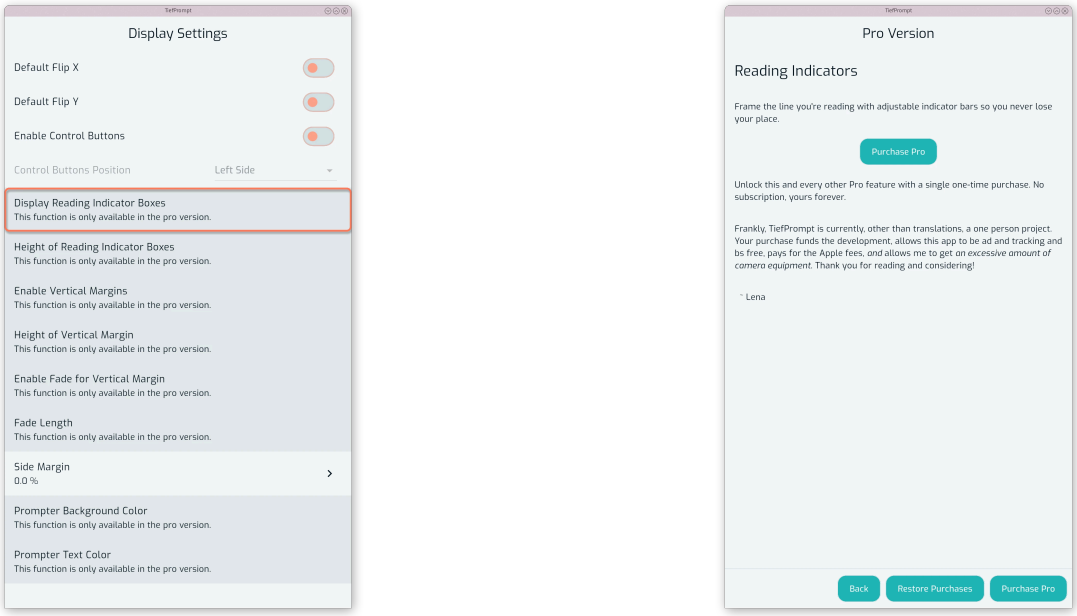


Figure 62: The “Reading Indicator Boxes” feature is locked in the Free variant

9. Saved Scripts

Saving scripts is an obvious feature, until it isn't so obvious anymore.

TiefPrompt stores its scripts not in a file, but in the database of the app. This means that scripts are saved in the app's storage, and can be loaded and deleted from there. The Select Scripts screen is accessible from the home screen, and shows a list of all Select Scripts.

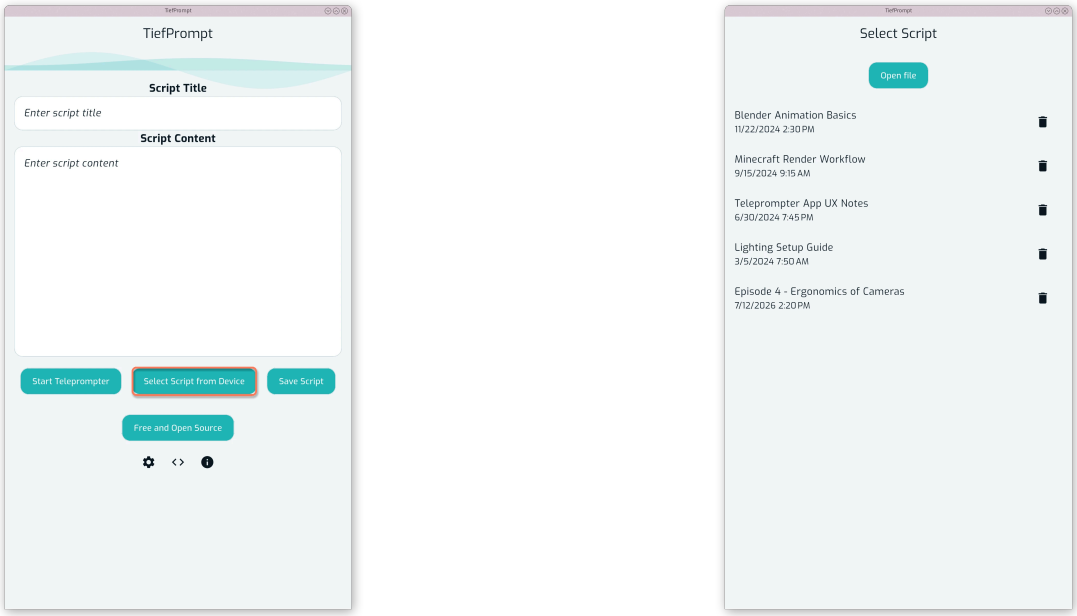


Figure 63: The Select Scripts screen

Saving a script is done by pressing the "Save Script" button on the home screen. It saves the current text in the script content box with the current title in the title box. If you previously loaded a script in the session, you will be prompted to overwrite it, or save as a new script. If you did not load a script (i.e. are on the ephemeral script), it will be saved as a new script.

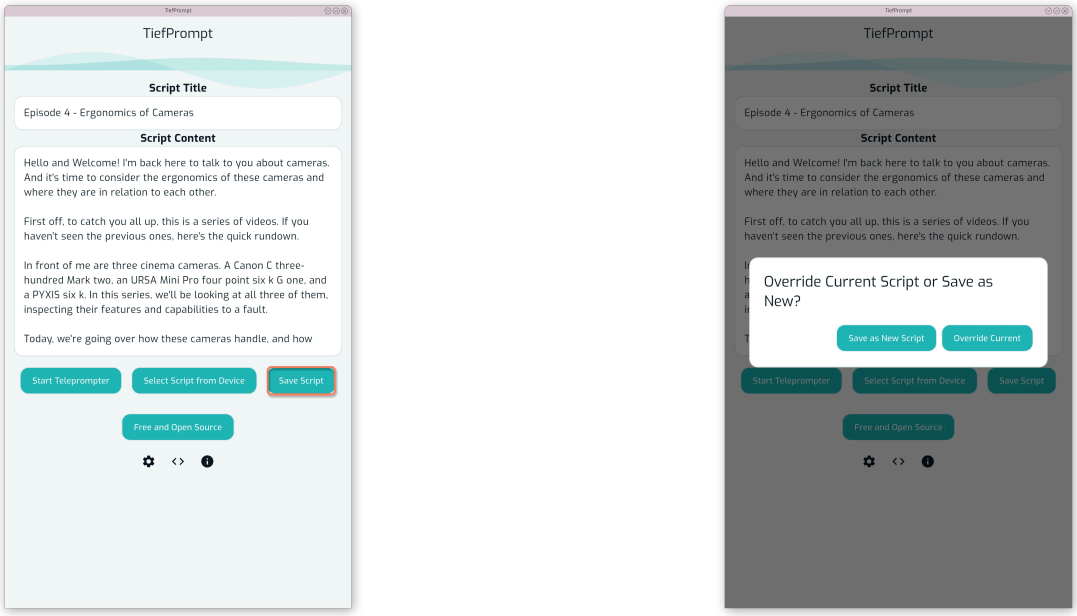


Figure 64: Save Script button and the Override or Save As New dialog

Loading a script is done by pressing on it in the list of Select Scripts. The script content and title will be updated accordingly.

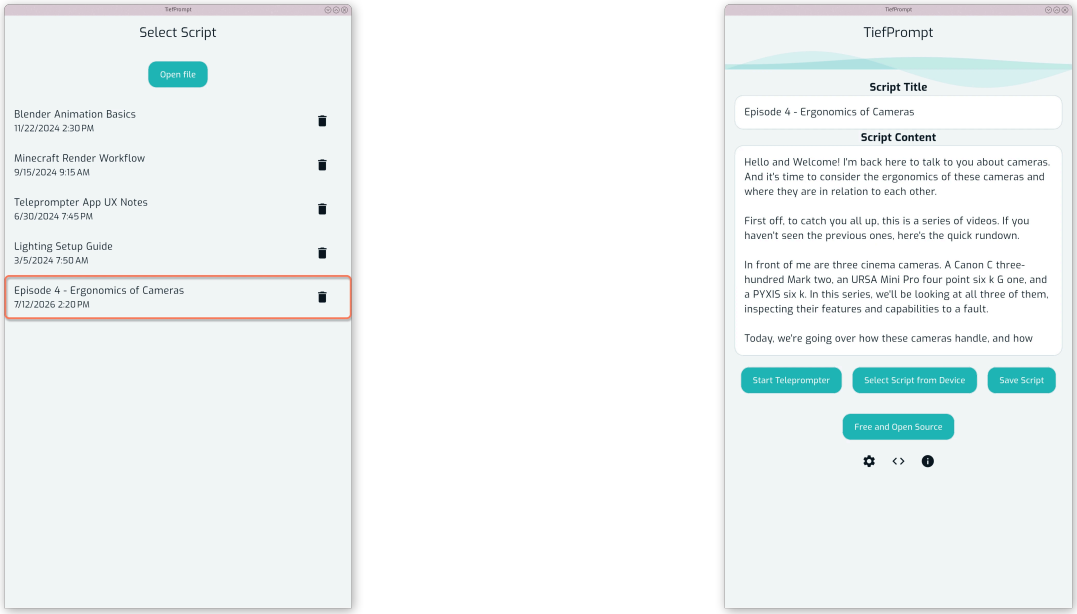


Figure 65: Script being loaded from the Select Scripts screen

When a script is no longer needed, it can be deleted by pressing the trashcan icon next to it in the list of Select Scripts. **There is no confirmation before deletion of a script.**



Figure 66: Script being deleted from the Select Scripts screen

The Select Scripts screen also includes a button up top to import a script from a file. You will be prompted by your operating system to select a file, and a new script entry will be created upon completion of the import, with the name of the file as the title. The creation date is the date of import.

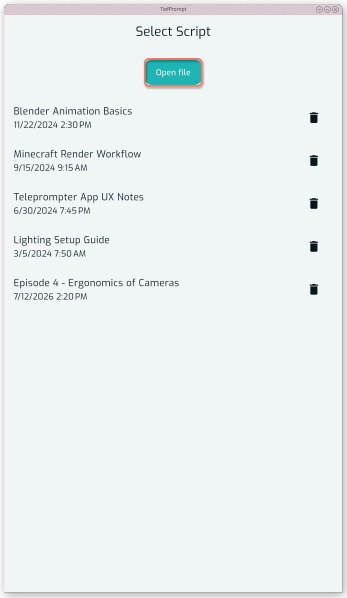


Figure 67: Import Script button

Persisting (saving) a script also persists the last scrolled position of the script. This means that when you load a script, it will be scrolled to the last position it was at when you last closed the prompter. This is useful for continuing a script from where you left off.

ℹ There is currently no export feature for scripts, but it is planned for a future release. To export a script, you may copy the text from the script content box and paste it into a text file, which can then be saved to your device.

SECTION 2

An Incomplete List of Features in TiefPrompt

And the reason they seem counter intuitive

1. Script Saving

This article covers what happens when you press Save, Import, or navigate away from a script – not where the buttons are, which is covered in The Prompter Screen.

1.1. Saving the Scratch Script

Every install starts with one "scratch" script that's always open by default – see Scroll and Session Behavior for how that script is created and maintained. Pressing Save while this scratch script is loaded always does a silent overwrite: the script becomes a permanent, regular script with no dialog or confirmation. A brand new scratch script is created in the background immediately after, but the screen stays on the script you just saved – the new scratch script isn't loaded until the next time the app reconciles its script list (typically the next launch).

1.2. Saving a Regular Script

Once a script is no longer the scratch script, pressing Save always opens a "New Script vs. Override Script" dialog – every time, even if nothing has changed since the last save. There is no silent re-save path once a script is permanent.

New Script inserts a new, separate script and leaves the original untouched. **Override Script** updates the existing script in place.

> ⚠️ Overriding a script resets its creation timestamp to now. The "created" date shown in the Open File list reflects the most recent override, not when the script was first written – if you're expecting "created" to mean "first created," this can read as a bug.

1.3. The Unsaved-Changes Indicator

A plain `*` appears next to the Text field's label (not the title field, and nowhere in any app bar) whenever there are unsaved changes. This is a simple flag flipped on by any keystroke in either the title or text field – there's no comparison against the original content, so typing something and then undoing back to the exact original text still shows as unsaved. The indicator only clears on a successful save or a fresh script load.

1.4. Importing Scripts

Importing a `.txt` or `.md` file always creates a brand-new script, titled after the picked filename including its extension (e.g. `myscript.txt`). Titles aren't required to be unique, and there's no duplicate check – importing the same file twice produces two scripts with identical titles, distinguishable only by their creation time.

1.5. Leaving with Unsaved Changes

The behavior differs depending on how you navigate away. The **Select** button (to open a different script) checks for unsaved changes and shows a discard-confirmation dialog if you're dirty. The **Open File list**, however, deletes scripts immediately when you choose to delete one, with no confirmation dialog.

> ⚠️ Deleting a script from the Open File list is immediate and final – there's no confirmation dialog and no undo.

2. Saved Settings

For instructions on where to click, see Save and Restore Settings. This article instead covers what actually happens when you save, apply, import or export a settings profile ("restore point").

2.1. What's in a Profile

A saved profile is a full snapshot of the app's settings: theme mode, app accent color, prompter background/text colors, and the entire Display Settings / Text Settings / Scroll Settings configuration — plus a private copy of your current Keybindings, cloned into its own record at save time rather than linked to your live bindings. Saving a profile captures all of this at once – there's no way to save, say, only your Text Settings.

Deliberately excluded: anything script-related. No script content, script id, or scroll position is ever part of a profile.

2.2. Applying a Profile

Restoring a profile overwrites your current settings and swaps your active keybindings for the profile's cloned copy. It never touches your open script, scroll position, or anything else about your current session – restoring settings and editing a script are entirely separate concerns.

2.3. Naming

Profiles are identified by the name you give them when saving, but names aren't required to be unique. If you save two profiles with the same name, you'll simply end up with two separate entries that happen to share a label – they're distinguished internally by id and creation time. The only validation on the name field is that it can't be empty.

2.4. Import and Export

Exporting writes a profile – settings, keybindings, and a schema version – to a file you choose, which you can move to another device and import from there.

Importing is a two-step process. The file is first picked and validated, then, once valid, you're asked to name it (pre-filled if the file already had a name) and only then it's saved as a new profile. Importing never silently overwrites an existing profile or auto-applies itself – it always lands as an additional saved profile that you restore explicitly afterward.

⚠ Validation checks for an exact match on schema version, not "version or newer" – a file exported from a newer but otherwise compatible version of TiefPrompt is rejected outright, not just when it's genuinely incompatible.

3. Chapters

3.1. Markdown Is a Hard Requirement

Your script text is parsed into a Markdown structure on every edit regardless of settings, but chapter positions are only ever collected while the Markdown renderer is actually on screen – which only happens when Markdown rendering is enabled in Display Settings. With Markdown rendering off, no chapter positions are ever recorded and the feature is completely inert, not just less polished. This is why the "Show Current Chapter" toggle is greyed out until Markdown rendering is switched on – the UI reflects a real dependency, not an arbitrary restriction.

3.2. What Counts as a Chapter

Every heading in your script – any level, from a top-level heading down to the smallest one – becomes a chapter marker. There's no way to mark only top-level headings as chapters; all heading levels are treated identically.

3.3. Tracking the Current Chapter

As you scroll, TiefPrompt walks through the chapters in the order they appear in your script and keeps track of the last one whose position you've scrolled past. In practice this means the chapter indicator always shows the most recent heading above your current reading position, updating continuously as you scroll through the script.

3.4. Turning It Off

Disabling Markdown rendering or the "Show Current Chapter" toggle doesn't just hide the chapter indicator – it actively clears the tracked chapter state for as long as either is off. Turning either back on starts chapter tracking fresh rather than resuming from a stale position.

4.1. Families and Variants

A font family is a group of one or more variants, where a variant is a specific weight and style combination – for example, a regular weight and a bold italic weight of the same typeface are two different variants of one family. Built-in fonts and the fonts you upload share this exact same structure; the only difference is that built-in fonts can't be edited or removed.

4.2. Uploading Fonts

You can upload one or more `.ttf` / `.otf` files at once. Each file is read independently to extract its weight, italic flag, and embedded family name – if a file has no embedded family name, its filename (without the extension) is used instead. Uploading several files at once processes each one as its own separate addition, not as a single batch, so the merge rules below apply individually to each file.

4.3. How Uploads Merge Into Families

An upload never merges into a **built-in** family, even if the name matches exactly. If no custom family with that name exists yet, the upload becomes a new family; if one does exist and none of its variants share the incoming file's weight and style, the new variant is added to it.

⚠️ Uploading a font named the same as a built-in family does not extend or replace that built-in font – it creates a second, separate custom family with the same name. You'll end up with two same-named entries, not one merged one.

⚠️ If a variant with the same weight and style already exists in a family, the upload is rejected as a conflict – and this currently surfaces as a raw error rather than a clean "this variant already exists" message.

4.4. Moving and Deleting Variants

You can move a variant from one custom family to another. Moving into a built-in family is not allowed, though in practice the family picker doesn't offer built-in families as a destination to begin with. If moving a variant out of a family leaves that family with no variants left, the now-empty family is automatically removed entirely.

4.5. Silent Font Fallback

⚠️If the font currently selected in your settings no longer exists – for example, its last remaining variant was deleted, or a restored settings profile references a font that isn't installed on this device – TiefPrompt automatically switches to whatever font happens to be first in the combined built-in and custom font list, and saves that as your new selection. This isn't guaranteed to be a specific "safe" default font; it's simply whichever one sorts first, and there is no on-screen notice when this happens. If your font ever seems to have switched on its own, this fallback is almost certainly why.

5.1. Scroll Position Persistence

TiefPrompt remembers where you were reading in each script. Your position is saved a second after you stop scrolling, and – separately – at least once every 5 seconds even during a long, continuous scroll, so an unexpected crash or close never loses more than a few seconds of progress. This is only written to the script itself once it's an actual saved script; see Script Saving for when a script becomes permanent.

If a script has never had a position saved, opening it doesn't jump to the very top of the text – it starts about half a screen height down, which lands the first line roughly centered in the reading area instead of pinned to the top edge.

5.2. The Scratch Script

TiefPrompt always keeps exactly one "scratch" script around – an always-available, ready-to-write-in script that doesn't show up in your Open File list. It's created automatically on first install, and every time the app starts, it checks that exactly one exists: if none exist, a new one is created; if one exists, it's loaded as-is.

> ℹ Because saving the scratch script (see Script Saving) creates a fresh replacement scratch script in the background without loading it immediately, it's possible – in entirely normal use – for more than one scratch script to briefly exist at once. If the app ever finds more than one at startup, it treats the extras as real, permanent scripts rather than deleting them, adds them to your visible script list, and creates one fresh scratch script to replace them. You'll see a warning if this happens, though no scripts are lost.

Everything you type into the scratch script is saved automatically about half a second after you stop typing, with no explicit Save action needed – but this never converts it into a permanent script or changes its creation date. Since scratch scripts are hidden from the Open File list, and that's the only place scripts can be deleted from, the scratch script can't be deleted directly – only turned into a permanent script via Save, or replaced the next time the app reconciles a duplicate.

6. Theming

Instructions for changing your theme and colors are covered in Setup the App. This article goes into why the prompter looks and behaves differently from the rest of the app.

6.1. Two Separate Themes

TiefPrompt has two entirely independent themes: the **app theme**, used everywhere except the teleprompter screen, and the **prompter theme**, used only while you're actually reading a script. These aren't the same theme with a few overrides applied to one of them – they're built completely separately, which is why changing your prompter's background and text colors never affects the rest of the app, and vice versa.

6.2. Light, Dark, and System

The light/dark/system choice for the app theme is handled by the standard system-level mechanism for following your device's appearance – TiefPrompt doesn't do any custom detection of its own, it simply hands the choice off and lets the platform resolve it.

6.3. App Accent Color

In the app theme, your chosen accent color is the only color you can customize – everything else (backgrounds, surfaces, borders, body text) is a fixed part of the app's light and dark looks and isn't user-adjustable. The accent color is used for both primary buttons/controls and secondary accents throughout the app.

Text and icons drawn on top of your accent color (for example, the label on a filled button) automatically switch between a dark or light color depending on how light or dark your chosen accent is, so button labels stay readable no matter what color you pick. This automatic contrast handling only applies within the app theme – it has no effect on the prompter.

6.4. Prompter Theme

The prompter's theme is built from a high-contrast dark base, with exactly three things layered on top: your app's accent color, and your explicitly chosen prompter background and text colors from Display Settings. Unlike the app theme, there's no automatic contrast correction here – if you pick a background and text color that are hard to read together, TiefPrompt won't adjust or warn you. The readability of your prompter text and background is entirely in your hands.

7. Voice Activation

Voice Activation allows users to control the teleprompter using their voice. When enabled, the prompter will only scroll when it detects sound above a certain threshold. This feature is particularly useful for hands-free operation, ensuring that the teleprompter responds to the user's voice commands.

Note that **any** noise detected above the threshold will trigger the prompter to scroll, not just the user's voice. Therefore, it is important to set the sensitivity appropriately to avoid unintended scrolling.

You may also use an external microphone to improve voice detection accuracy by selecting the appropriate audio device from the dropdown menu in the Scroll Settings screen.

> ℹ Due to labels being the phone's name on some devices, it may be necessary to manually select the correct audio input to ensure accurate voice detection. The ID of the audio device is displayed in brackets after the label to allow users to identify the correct device.

Note that if an audio device is removed, it will show up as "Unknown Device" in your dropdown and will not be functional until it is reconnected and selected again.

> ⚠ If an audio device is connected while the Scroll Settings screen is active, it may not be immediately recognized. You may need to leave and reenter the screen for detection to work, as the audio devices are not reloaded periodically.

8. Markdown

Markdown support is provided by its own library, `tief_weave`, and is limited to a few features.

Supported are, among others, headings, bold, and italics. To enable markdown, toggle the setting in the prompter or the Text Settings screen.

Markdown rendering using `tief_weave` is currently very limited and does not follow CommonMark. It is a best effort model designed for simple texts.

When markdown rendering is enabled, one may also enable Current Chapter Headings, which will display the most recent markdown heading at the top of the screen.

The rendered overlay can also be used to jump to the previous chapter.

SECTION 3

TiefPrompt Contribution Guidelines

1. Filing Issues

TiefPrompt is an open-source project, and we welcome contributions from the community. If you encounter a bug, have a feature request, or want to contribute in any way, please file an issue on our GitHub repository.

1.1. Filing a Bug Report

When filing a bug report, please provide as much information as possible to help us reproduce and fix the issue. This includes the version of TiefPrompt you are using, the operating system and version where applicable, and a detailed description of the steps to reproduce the bug. Screenshots or screen recordings can also be very helpful. Also please include a log with your report. You can find the log file in the app's settings under "View Logs".

1.2. Filing a Feature Request

When filing a feature request, please provide a clear and concise description of the feature you would like to see implemented. Explain the problem it solves or the improvement it brings to the user experience. While not necessary, if possible, include mockups or examples of similar features in other applications to make our development easier.

This section is verbatim the `CONTRIBUTING.md`.

2.1. CONTRIBUTING.md

This project has more boilerplate than real code, and a couple of hacky, genuinely cursed corners.

A not quite short, opinionated list. Most of these exist because something broke once. The checklists at the end are only important if you need them, if you don't want to see the bodge, don't read them preemptively.

2.1.1. 0. Setup

- Clone with submodules. Either `git clone --recursive <url>`, or if you already cloned: `git submodule update --init --recursive`. Half the project lives in submodules; without them nothing builds. Heads up: `.flutter` *is* the whole Flutter SDK (pinned via submodule, fetched over SSH), so the clone is big and slow and the first `.flutter/bin/flutter` run bootstraps its own Dart. This is normal, go get a coffee. Or use the OS toolchain if you must, but this is at the end of a long sentence on purpose, I don't like the idea of random versions breaking my CI build (been there done that).
- Always use the bundled toolchain: `.flutter/bin/flutter` and `.flutter/bin/dart`, never your system `flutter` / `dart`. The version is pinned for a reason, and mismatched versions cause exactly the kind of mischief you don't want to debug.
- Generate code before you build anything. The Freezed / Riverpod / Drift part files (`*.freezed.dart`, `*.g.dart`, `*.drift.dart`) are not checked in, so a fresh clone is full of "undefined `_$SettingsState`" errors until you run:

```
1 .flutter/bin/dart run build_runner build --delete-conflicting-outputs
```

  (`build_runner watch` if you'd rather it keep up as you go). If you touch the Drift schema you also need `.flutter/bin/dart run drift_dev make-migrations`.

2.1.2. 1. Build both flavors before opening a PR

TiefPrompt ships in two flavors that differ *only* by entrypoint: `lib/main_foss.dart` (F-Droid, direct APK) and `lib/main_freemium.dart` (App Store, Play Store). Freemium pulls in IAP packages; FOSS must build without them.

It is entirely possible to write code that builds fine for freemium but breaks FOSS, because something you imported imported something that imported an IAP-only dependency. This has happened. So build *both* before you push:

```
1 .flutter/bin/flutter build <target> -t lib/main_foss.dart
2 .flutter/bin/flutter build <target> -t lib/main_freemium.dart
```

(or `tools/build.sh -t <targets> -f <freedom>`). A PR that only builds one flavor is not done. (CI should catch it, but darn me if that ever not works…)

2.1.3. 2. Separate concerns

The layers, and what belongs in each:

- `lib/core/`: reusable utilities and shared code
- `lib/models/`: the Drift database and its models
- `lib/providers/`: Riverpod providers that *own* state
- `lib/services/`: Riverpod providers that are stateless and purely functional
- `lib/ui/`: screens and widgets

Hard rules:

- DB access does not live in the UI. Route it through a provider or service.
- Don't reach across layers to dodge writing a provider. Write the provider.

(Yes, `feature_provider` lives in `providers/` but is really a service. It's a known wart, left as-is on purpose, so don't "fix" it as a drive-by, there be dragons.)

2.1.4. 3. Think about what you've done

Before you open the PR, re-read your own diff. Not for typos, for fit. Does it match how the surrounding code already does things, or did you invent a second way? Did you add state to a widget that should have been a provider? Did you hardcode something that has a constant three files over? Most review comments on this repo are things the author would have caught by reading their own diff once.

2.1.5. 4. Don't branch on flavor in the UI

Widgets must never ask "am I FOSS or freemium?". Gating goes through the feature system: wrap things in `_FeatureGate` / `_FeatureGatedIconButton`, or check `featuresProvider.select((s) => s.features.contains(Feature.x))` (see `prompter_bottom_bar.dart` and `app_settings.dart`). Note that `AppSetting` and its derivatives gate themselves automatically off the `Feature` you give them, so for those you just need to pass the right one. The only place flavor actually differs is the entrypoint and its `feature_provider` override. Keep it that way, it's the whole reason the split stays maintainable. (Ignore the BS I did early day with the button on the home screen. If you dare, fix it.)

2.1.6. 5. State lives in providers; navigate with go_router

- Prefer providers over widget-local state so it survives rebuilds. Use `@riverpod` codegen by default; a plain `Notifier` is fine only for trivial single-value toggles (see `controlsVisibleProvider`).
- All navigation goes through `go_router`. Don't call `Navigator.push` directly.
- Routes go in `router_provider.dart`. This may seem confusing, but it has reasons. Reasons I can't remember. But reasons!

2.1.7. 6. Localize user-facing strings, but stay in your lane

Every string a user sees goes through `easy_localization`: `context.tr("Some.Key")`, with the key defined in the translation assets. Hardcoded strings are acceptable only where no `BuildContext` is available, and those must be in English.

Only touch English and your own native language. Do not edit, “improve”, or add other languages, and do not commit AI-generated translations. A wrong translation you can’t read is worse than a missing one, leave those to people who actually speak the language.

2.1.8. 7. Profit

2.1.9. When I add a Feature

A `Feature` is not “a Pro thing”. It’s the unit of gating for the whole app. The point is to be able to move any capability between free and Pro trivially. So: if you implement anything that does something and could *conceivably* be gated, give it a `Feature`. When in doubt, add one.

Always add a *new* `Feature`. Never reuse an existing one because it’s “close enough” (e.g. don’t gate a new line-height control behind `Feature.fontSize`). Reuse looks cheaper today and welds two capabilities together forever, which defeats the entire reason this enum exists. Yes, a new Feature costs you the extra steps below. Pay them.

1. Add the value to the `Feature` enum in `core/constants.dart`.
2. Add it to `kAllFeatures`. Add it to `kFreeFeatures` too if it should be free in the freemium build.
3. Gate it in the UI with `_FeatureGate` / `_FeatureGatedIconButton`, or a `featuresProvider.select(...)` check. Never check the flavor directly. (If the thing is an `AppSetting` or a derivative, it gates itself off the `Feature` you pass it, so just pass the right one.)
4. Add a `BuyProScreen.BuyRequest.<featureName>` translation key. It’s rendered as Markdown on the buy screen when someone taps the locked feature, so give it an h2 header and a little description of what they’re buying.

2.1.10. When I add a Route

1. Add a `GoRoute` in `router_provider.dart` (nest it under `/settings` if it’s a settings subscreen).
2. Create the screen widget under `lib/ui/screens/` (or `screens/settings/`).
3. Navigate to it with `context.push("/your_path")`.
4. If it needs arguments, pass them via `extra:` using a small router-extra class (see `DisabledFeatureScreenRouterExtra`).
5. Re-run `build_runner` (you edited `router_provider.dart`).

3. Documentation Guidelines

Documentation is not written inside the primary TiefPrompt repository, rather living in site-tiefprompt.com°.

Documentation is written in Markdown. Conversion to the website and PDF is done via TiefDownConverter°.

When adapting the documentation, you can run `./build.sh` to build the whole site (slow!) or just `tiefdownconverter convert -m Documentation` to convert only the documentation. Images have to be copied to `Documentation/resources`.

When adding a feature, you can add screenshots to the scrnshot automation in `tiefprompt/integration_test/marketing`. If not, they will be added "magically" (by Tiefseetauchner). If you want, you can also extend the documentation with the feature.

Styling of the website happens in `site-tiefprompt.com/scss`. Javascript goes into `static_resources/js`. If you need to add something significant, better ask, this TiefDown project is a mess.