Settings, Storage and UI

This guide is assuming you're hot off the heels of following Your First Plugin, and so will use that same plugin as a test case for adding user-configurable functionality.

This guide will also cover the general usage of shelter's UI tools, including Solid and shelter UI, and plugin storage.

Specifically, we'll be adding an option to use a nicer looking tag instead of just a boring span.

If you've never used Solid, I STRONGLY suggest that you go through the interactive Solid tutorial. It will get you to speed on how Solid works!

There's also a couple of nice videos.

Storage

First, we need to use the plugin store to keep track of if the user wants this or not. We'll grab the store, and in case this is the first run, set a default for our option:

```js
const {
  // other stuff ...

  plugin: { store }
} = shelter;

store.fancyTag ??= true; // set default
```

This behaves just like an object, and whenever you assign to one of the top level properties, it will save it. All the top level properties of these objects are reactive, in that if you use one in a Solid effect or JSX, it will automatically update when changed.

This makes them work to an almost "magical" (but predictable) degree within your UI code - you'll see later. 😃

You may store anything as long as it is serializable. The list of types allowed can be found here, circular references are not allowed. Notably, functions are not allowed.

Writing the actual feature

So let's go to our `handleElem` function, specifically this part:

js
```js
elem.firstElementChild.textContent += ` (${message.author.username})`;
```

We'll keep this, as the whole point here is a setting, but we'll introduce a nicer looking version too:

jsx
```jsx
if (store.fancyTag)
  elem.firstElementChild.append(<span class="showuname-tag">{message.auth
else
  elem.firstElementChild.textContent += ` (${message.author.username})`;
```

And add some CSS:

css
```css
/* styles.css
   thx to wiz for styles btw */
.showuname-tag {
  font-weight: 600;
  border-radius: 5px;
  padding: 0 3px;
  background: var(--background-surface-highest);
}
```

js

```
import css from "./styles.css";
shelter.plugin.scoped.ui.injectCss(css);
```

TIP

You should go read the Lune docs about CSS Modules, btw 😉

Writing the Settings UI

Now we have a setting stored, and we do something when its on, so we need to expose it to the user. We do this by exposing a Solid component called `settings` , which will be rendered in a modal when the user clicks the settings icon on the plugin card (or when we manually call `shelter.plugin.showSettings()`).

jsx

```
const { SwitchItem } = shelter.ui;

export const settings = () => (
  <SwitchItem value={store.fancyTag} onChange={(v) => {store.fancyTag = v
    Use fancy looking tags
  </SwitchItem>
)
```

Reactivity

That's... kinda it! But let's take a detour through reactivity to see if we can make these tags change out immediately when we flip this switch.

Let's go back to this code:

js

```js
if (store.fancyTag)
  elem.firstElementChild.append(<span class="showuname-tag">{message.auth
else
  elem.firstElementChild.textContent += ` (${message.author.username})`;
```

This selectively injects something based on the value, then it's done. One-shot. However, `store` is reactive, and we can use it with Solid to automatically react when it changes. To do this, we need to make the difference between the two cases in our UI, not just normal code.

We would rewrite it using either a tenary inside JSX, or as a use of the `Show` component, as so:

jsx

```jsx
// when you're putting non-trivial solid onto the page you should *ideall
// this is not necessary, and by all means test your code without them, b
const { SwitchItem, ReactiveRoot } = shelter.ui;
import { Show } from "solid-js";

elem.firstElementChild.append(
  <ReactiveRoot>
    <Show when={store.fancyTag} fallback={` (${message.author.username})`
      <span class="showuname-tag">{message.author.username}</span>
    </Show>
  </ReactiveRoot>
);
```

This is much more like what your code would look like if this was an app you were writing yourself, not just a plugin, and it will indeed change out all the shown usernames to be fancy/boring when the switch is flipped.

Now you know how to use settings, storage, and have seen reactive storage with UI at work!

Previous page

Next page