shelter Injector Integration

shelter, as a client mod, provides a lot of functionality that is useful to writers of custom clients and the like.

Examples of things you may do, which uses (as of writing) shelter to:

- display its settings ui in a stable way
- fix some screenshare issues with your custom discord client
- reimplement RPC, perhaps using arRPC

Whatever you want to do, shelter can provide you with very stable settings UI capability and a platform to inject custom code into discord from, which can be more convenient than somehow injecting your own modifications, especially since you get a robust set of APIs to build your tweaks on top of!

To better facilitate this, shelter has some special integration options for use in custom clients or other injectors, to make the experience less buggy and more seamless for the end user.

Injector Settings Sections

shelter can inject sections into the sidebar of the Discord user settings.

Without custom integration, it is very hard to predict where your settings will show, and making it a separate zoned off section is very difficult to do well, if not impossible.

To fix this issue, shelter provides *injector sections*, which are displayed above shelter's sections, and above all plugin's sections too.

To use them you need to do one of two things: provide via a global, or use Injector Plugins.

Settings with Injector Plugins

From within injector plugins, all settings sections created via the usual APIs go to injector sections. That's it. Easy. This includes via the scoped API.

You also have access to one special new API: `shelter.settings.setInjectorSections`.

This overwrites the injector settings with the same content that you would set on `SHELTER_INJECTOR_SETTINGS`.

Note that this will override anything added with `registerSection`, but is useful if you want to set a lot of sections at once.

Settings with a Global

Note that this method is flawed in a couple of ways, most chiefly that you don't have access to solidjs at this point.

To provide a global, you must ensure that `SHELTER_INJECTOR_SETTINGS` is defined at shelter init and follows the structure:

ts

```ts
// this is the same as the standard settings API
type SettingsSection =
  | ["divider"]
  | ["header", string]
  | ["section", string, string, Component, object?]
  | ["button", string, string, () => void];

let SHELTER_INJECTOR_SETTINGS: SettingsSection[] | solidjs.Accessor<Setti
```

If it is an array, it will be set at shelter init time. If a solidjs accessor (for a signal), it can be changed later and will be reactively updated.

Example:

```js
window.SHELTER_INJECTOR_SETTINGS = [
  ["divider"]
  ["header", "My Client"],
  ["section", "myclient_settings", "Settings",
    () => {
      // solidjs component here
      const [toggle, setToggle] = shelter.solid.createSignal();
      // have fun without jsx i guess - thats the other of the flaws i me
      return document.createElement("div");
    }
  ],
  ["buttton", "myclient_clog", "Changelog", MyClient.ShowChangelogModal]
];

(0,eval)(shelter);
```

You can also keep this off the global:

```js
new Function("SHELTER_INJECTOR_SECTIONS", shelter)([/*...*/]);
```

Injector Plugins

You may wish to use shelter plugins as a base to build your functionality upon. We have special support for injector supplied plugins, that allow you more control over how they behave.

You can do the following things:

- Ensure that your plugins exist at startup and work as you expect
- Hide the plugins from view
- Disable all plugin actions except showing the settings modal (if you wanted to disable that, just don't provide one!)
- Set injector user settings sections

Note that an info icon will be shown on any visible loader plugins' cards to explain to users that they are a part of your loader, as we have had issues with users not understanding *why* these plugins are here / where they came from.

When your injector plugins are loaded, if they are new, they will be created and turned on. If they existed already, they will be overwritten with the following exceptions:

- `shelter.plugin.store` **content** is kept
- if the following conditions are met, the plugin's on/off state will be kept, else it is just forced on:
  - the plugin is visible to the user (not hidden)
  - the user is allowed to toggle the plugin themselves

Note that while functionality can be disabled in the settings UI, the `shelter.plugins` APIs will completely ignore your prohibitions - hidden plugins will be accessible, and disallowed actions can be performed. This is a purposeful choice - injector plugins are not here to break stuff, they're here to stop the 99% of users shooting themselves in the foot, and provide you a nicer integration experience.

Remote injector plugins ALWAYS have auto-update enabled.

Setting Up Injector Plugins

You set up your injector plugins via a global called `SHELTER_INJECTOR_PLUGINS`, that must be defined at shelter load. It must match this type:

ts

```ts
type InjectorIntegrationOptions = {
  // is the plugin visible in shelter's plugin list?
  isVisible: boolean;
  // what actions are visible to the user on the plugin card
  // irrelevant if hidden but MUST be provided nonetheless
  // opt-in so that future new options are hidden for your plugins by def
  allowedActions: { toggle?: true; delete?: true; edit?: true, update?: t
  // if set, will show your injector's name instead of a generic string i
  loaderName?: string;
};
```

```
type RemotePlugin = [string, InjectorIntegrationOptions]; // string is th
type LocalPlugin = {
  js: string;
  manifest: Record<string, string>; // expected: name, author, descriptio
  injectorIntegration: InjectorIntegrationOptions
};

let SHELTER_INJECTOR_PLUGINS: Record<string, RemotePlugin | LocalPlugin>;
```

For example:

```
window.SHELTER_INJECTOR_PLUGINS = {
  // obviously in a client you're unlikely to use both remote and local t
  "myclient-settings": ["myclient://shelter-plugins/settings.js", { isVis
  "myclient-rpc": {
    js: MyClient.ShelterPlugins.Rpc.CodeText,
    manifest: {
      name: "MyClient RPC Fix",
      author: "MyClient Authors",
      description: "Provides Rich Presence support for MyClient"
    },
    injectorIntegration: {
      isVisible: true,
      allowedActions: { toggle: true }
    }
  }
};
(0, eval)(shelter);
```

Alternatively, you need not use window, to keep it all private:

```
new Function("SHELTER_INJECTOR_PLUGINS", shelter)({/*...*/});
```

Background

How Injectors Work