Aabu

Geste de savoir

J'ai créé une extension GNOME Shell

30 juillet 2022

Table des matières

Introduction

Gnome Shell est partie intégrante de l'environnement de bureau par défaut sur Ubuntu. Il gère notamment la zone de notitifcation et on peut le contrôler ou y ajouter des fonctionnalités à l'aide d'extensions. Au cours de la mise à jour agitée de ma version d'Ubuntu, j'ai été confronté à un souci d'extension Gnome Shell qui m'a rendu curieux de leur fonctionnement.

Je me suis renseigné sur le sujet et j'ai mis en pratique sur une extension toute simple pour afficher la température de mon CPU et qui remplacera une extension qui en fait trop pour mon besoin. Ce billet raconte ce que j'ai appris en chemin!

1. Trouver la documentation de base

Les extensions pour GNOME sont écrites en GNOME Javascript (GJS pour faire court), une variante de JavaScript qui propose notamment des API pour manipuler les différents composants de GNOME, et en particulier GNOME Shell.

La documentation est un peu morcelée, et contrairement à ce à quoi je m'attendais, je n'ai pas pu aller sur le site de GNOME et naviguer simplement du genre GNOME Shell > Développeurs > Extensions GNOME Shell. Mon moteur de recherche favori a été d'un grand secours et j'ai fini par trouver ce que je cherchais.

Le plus important est le guide GJS, qu'on retrouve sur un site dédié [1]. En particulier, ce site propose toute une section dédié au développement d'extensions avec des guides introductifs et c'est ce que j'ai utilisé pour me mettre un pied à l'étrier. J'ai trouvé tout cela clairement expliqué, bien amené et bien présenté.

Pour les détails d'API, cette documentation [2] est utile, bien que pas si pratique quand on ne connaît pas exactement ce qu'on cherche.

1. Vous noterez qu'il ne s'agit pas du domaine principal de GNOME ni d'un de ses sous-domaines, même si ces domaines lient probablement d'une manière ou d'une autre vers le guide.

Comme on le verra tout à l'heure, les extensions ont à disposition un espace de nom global qui donne directement accès à des éléments de GNOME Shell, mais qui se trouve être mal documenté (ou alors encore une fois, je ne l'ai pas trouvé facilement). La meilleure documentation semble être le code lui-même , côté GNOME Shell.

2. Composer le squelette

2.1. Template d'extension

Concrètement, une extension est composée au minimum de deux fichiers: `metadata.json` qui décrit l'extension (un identifiant unique, un nom, une description, etc.) et `extension.js` le point d'entrée du code de l'extension.

Je ne m'attarde pas sur les métadonnées qui ont assez peu d'intérêt. Il faut juste remplir cela correctement selon les recommandations.

Intéressons-nous plutôt à `extension.js`. Le contenu le plus basique ou presque qu'on puisse imaginer est le code ci-dessous, inspiré de la documentation officielle.

Parmi les deux variantes au choix, j'ai choisi d'utiliser celle avec un objet `Extension`. On a alors la fonction `init` appelée par le gestionnaire d'extension de GNOME Shell pour… initialiser l'extension, ainsi que les méthodes `enable` (pour charger l'extension) et `disable` (pour faire le ménage derrière) qui lui sont aussi connues. On a là une interface très simple utilisée par le gestionnaire d'extensions pour faire son job.

```
1  const ExtensionUtils = imports.misc.extensionUtils;
2  const Me = ExtensionUtils.getCurrentExtension();
3
4
5  class Extension {
6      constructor() {
7      }
8
9      enable() {
10         log(`enabling ${Me.metadata.name}`);
11     }
12
13     disable() {
14         log(`disabling ${Me.metadata.name}`);
15     }
16 }
17
18
19 function init() {
20     log(`initializing ${Me.metadata.name}`);
21     return new Extension();
```

2. Sur un sous-domaine de gnome.org ce coup-ci !

```
}
```

2.2. Squelette de l'interface

J'ai envie d'indiquer la température dans le panneau des indicateurs, en haut à droite (là où sont l'indicateur de batterie, connectivité réseau, statut de synchronisation de stockage en ligne, etc). Il s'agit donc:

— de causer avec le panneau des indicateurs (l'API le permet, tant mieux !);
— lui ajouter un indicateur ;
— prévoir une zone de texte dans cet indicateur pour afficher le texte qu'on souhaite (quelque chose du genre «54°C»).

Je n'ai besoin de rien de plus: pas de paramètres, pas de menu, pas d'images, rien qu'un petit texte. Cela donne le code suivant.

```
const Clutter = imports.gi.Clutter;
const St = imports.gi.St;
const Gio = imports.gi.Gio;

const ExtensionUtils = imports.misc.extensionUtils;
const Me = ExtensionUtils.getCurrentExtension();
const Main = imports.ui.main;
const PanelMenu = imports.ui.panelMenu;


class Extension {
    constructor() {
        this._indicator = null;
        this._temperatureLabel = null;
    }

    enable() {
        this._temperatureLabel = new St.Label({text : "--- °C",
            y_align: Clutter.ActorAlign.CENTER});

        let indicatorName = `${Me.metadata.name} Indicator`;
        this._indicator = new PanelMenu.Button(0.0, indicatorName,
            false);
        this._indicator.add_child(this._temperatureLabel);

        Main.panel.addToStatusArea(indicatorName, this._indicator);
    }

    disable() {
        this._indicator.destroy();
        this._indicator = null;
```

```
    }
}


function init() {
    return new Extension();
}
```

Le fonctionnement est très similaire à n'importe quelle bibliothèque graphique: on appelle ou crée des composants, on crée la structure en ajoutant des composants comme enfants, on dispose d'options pour le *packing*, etc.

3. Ajouter l'affichage de la température CPU

Maintenant, il ne manque plus que l'essentiel: régulièrement relever la température depuis le capteur adéquat et mettre à jour l'affichage en conséquence.

Le capteur qui m'intéresse est mappé vers le fichier `/sys/class/hwmon/hwmon4/temp1_input` [1] qu'il s'agit alors de lire. Je me suis inspiré du code de mon extension actuelle: je lis le fichier de manière asynchrone puis mets à jour l'indicateur avec son contenu. Ensuite, un callback appelé toutes les secondes s'occupe de mettre à jour régulièrement.

Le code (dont la propreté n'est pas garantie, je connais très peu JavaScript) n'est pas compliqué. La principale difficulté à l'écriture consiste à identifier les fonctions utiles dans les différentes API.

```
const Clutter = imports.gi.Clutter;
const St = imports.gi.St;
const Gio = imports.gi.Gio;

const ExtensionUtils = imports.misc.extensionUtils;
const Me = ExtensionUtils.getCurrentExtension();
const Main = imports.ui.main;
const PanelMenu = imports.ui.panelMenu;
const Mainloop = imports.mainloop;


const delayMs = 1000;
const sensorFilename = "/sys/class/hwmon/hwmon4/temp1_input";
const indicatorName = `${Me.metadata.name} Indicator`;
const defaultText = "--- °C";


class Extension {
```

1. La manière dont le système s'occupe de mapper les capteurs vers le système de fichier est un sujet à part entière que je ne développerai pas dans ce billet.

```
19      constructor() {
20          this._indicator = null;
21          this._temperatureLabel = null;
22          this._callback = null;
23      }
24
25      enable() {
26          this._temperatureLabel = new St.Label({text : defaultText,
                y_align: Clutter.ActorAlign.CENTER});
27
28          this._indicator = new PanelMenu.Button(0.0, indicatorName,
                false);
29          this._indicator.add_child(this._temperatureLabel);
30
31          Main.panel.addToStatusArea(indicatorName, this._indicator);
32
33          this._callback = Mainloop.timeout_add(delayMs,
                this._update.bind(this));
34      }
35
36      disable() {
37          this._indicator.destroy();
38          this._indicator = null;
39          Mainloop.source_remove(this._callback);
40          this._callback = null;
41      }
42
43      _update() {
44          const sensorFile = Gio.File.new_for_path(sensorFilename);
45          sensorFile.load_contents_async(null, (source, result) => {
46              let [success, sensorFileContent, _] =
                    source.load_contents_finish(result);
47              let labelText = success ?
                    this._format(sensorFileContent) : defaultText;
48              this._temperatureLabel.set_text(labelText);
49          });
50          return true;
51      }
52
53      _format(content) {
54          const value = content.toString().slice(0, 2);
55          return `${value} °C`;
56      }
57  }
58
59
60  function init() {
61      return new Extension();
62  }
```

Conclusion

Il n'est pas possible de conclure le billet sans montrer le résultat.

FIGURE 3.1. – Et voilà! Je sais, pas très amusant.

Quoi qu'il en soit, je me retrouve avec une extension qui répond pile-poil à mon besoin, légère et en laquelle j'ai confiance, sans compter les connaissances supplémentaires acquises sur le chemin. Que demander de plus?

Minitiature du tutoriel: logo de Gnome (source).