Programmer une extension Nautilus en Python est une excellente façon d'ajouter des fonctionnalités personnalisées (item du menu clic droit) au gestionnaire de fichiers Nautilus sous GNOME. Voici un guide de base pour vous aider à démarrer :

Prérequis

Environnement : Assurez-vous d'avoir un environnement de développement Python installé.

Nautilus-Python : Vous aurez besoin du paquet `nautilus-python`, qui permet d'écrire des extensions Nautilus en Python. Vous pouvez l'installer via la ligne de commande :

sudo apt-get install python3-nautilus

Sources avec exemples d'extensions simples : GitHub - GNOME/nautilus-python

Étapes pour créer une extension Nautilus

Créer un fichier d'extension : Créez un fichier Python, par exemple `mon-extension.py`, et placez-le dans le répertoire `~/.local/share/nautilus-python/extensions/`.

Écrire le code de l'extension, par exemple « hello wolrd » :

```
# Importation de la bibliothèque gi qui permet d'utiliser des bibliothèques GObject Introspection
import gi

# Spécification de la version de Nautilus à utiliser, ici la version 4.0
gi.require_version('Nautilus', '4.0')

# Importation des modules Nautilus et GObject depuis la bibliothèque gi.repository
from gi.repository import Nautilus, GObject

# Définition de la classe HelloWorldExtension qui hérite de GObject.GObject et Nautilus.MenuProvider
class HelloWorldExtension(GObject.GObject, Nautilus.MenuProvider):
    # Constructeur de la classe, appelé lors de la création d'une instance de la classe
    def __init__(self):
        # Appel du constructeur de la classe parente (GObject.GObject)
        super().__init__()

    # Méthode de rappel (callback) appelée lorsque l'élément de menu est activé
    def menu_activate_cb(self, menu, file):
        # Affichage du message "Hello, World!" dans la console
        print("Hello, World!")

    # Méthode appelée par Nautilus pour obtenir les éléments de menu à afficher dans le menu contextuel du fond de la fenêtre
    def get_background_items(self, window):
        # Création d'un nouvel élément de menu avec un nom, une étiquette et une info-bulle
        item = Nautilus.MenuItem(
            name='HelloWorldExtension::Hello_World',  # Nom unique de l'élément de menu
            label='Hello World',  # Texte affiché dans le menu
            tip='Prints Hello, World! to the console'  # Info-bulle affichée lorsque la souris survole l'élément
        )
        # Connexion du signal 'activate' de l'élément de menu à la méthode de rappel menu_activate_cb
        item.connect('activate', self.menu_activate_cb, None)
        # Retourne une liste contenant l'élément de menu créé
        return [item]
```

Rendre le fichier exécutable : Assurez-vous que votre fichier est exécutable :

```
chmod +x ~/.local/share/nautilus-python/extensions/mon-extension.py
```

Redémarrer Nautilus : Pour que les changements prennent effet, vous devez redémarrer Nautilus. Vous pouvez le faire en exécutant la commande suivante :

```
nautilus -q
```

Cela fermera toutes les fenêtres Nautilus ouvertes. Vous pouvez ensuite rouvrir Nautilus pour voir votre extension en action. Rouvrir **Nautilus depuis un terminal (commande nautilus)**, permet de voir les **messages dans la console**, comme les erreurs au chargement de l'extension ou les commandes de debug print('test')

Débogage

- **Journal système filtré sur les erreurs « Nautilus »** : `journalctl -f | grep -i nautilus`

Ressources supplémentaires

- **Documentation** : Overview: nautilus-python Reference Manual
- Exemple ancien : Comment écrire des extensions de nautilus avec nautilus-python

Autres paquets

`python3-nautilus` : C'est le paquet principal qui vous permet d'écrire des extensions Nautilus en Python. Il est indispensable.

1. **`libnautilus-extension-dev`** : Ce paquet est généralement nécessaire pour le développement d'extensions Nautilus, car il fournit les fichiers d'en-tête et les bibliothèques nécessaires pour compiler et développer des extensions.
2. **`python3-gi`** : Ce paquet est nécessaire pour utiliser PyGObject, qui permet d'accéder aux bibliothèques basées sur GObject, comme GTK, depuis Python. Cela est essentiel pour interagir avec les API de Nautilus en Python.
3. **`libgtk-3-dev`** et **`gir1.2-nautilus-3.0`** : Ces paquets sont nécessaires pour le développement avec GTK et pour s'assurer que vous avez les liaisons GI (GObject Introspection) nécessaires pour Nautilus.

Mon extension « Ouvrir_avec_Pinta »

```
import subprocess
from gi.repository import Nautilus, GObject
from typing import List

class TestExtension(GObject.GObject, Nautilus.MenuProvider):
    def __init__(self):
        super().__init__()
        print("Initialized Ouvrir_avec_Pinta extension")

    # Fonction exécutée au clic sur le menu item
    def menu_activate_cb(
        self,
        menu: Nautilus.MenuItem,
        files: List[Nautilus.FileInfo],
    ) -> None:
        print("menu_activate_cb")

        # Collecter tous les chemins de fichiers
        file_paths = [file.get_location().get_path() for file in files]
        print('file_paths =', file_paths)

        # Construire la commande pour Pinta
        command = ["pinta"] + file_paths
        print('command =', command)

        # Envoyer les chemins à Pinta
        subprocess.Popen(command)

    # Fonction liste les éléments sélectionnés par l'utilisateur lors du clic
    def get_file_items(
        self,
        files: List[Nautilus.FileInfo],
    ) -> List[Nautilus.MenuItem]:
        if len(files) < 1:
            return []

        # Vérifier que tous les fichiers sont des images
        for file in files:
            if not file.get_mime_type().startswith('image/'):
                return []

        # Déclaration du menu item
        item = Nautilus.MenuItem(
            name="David_extension:Open_in_Pinta",
            label="Ouvrir avec Pinta",
            tip="Ouvrir avec Pinta",
        )
        item.connect("activate", self.menu_activate_cb, files)

        return [item]

    # Even though we're not using background items, Nautilus will generate
    # a warning if the method isn't present
    def get_background_items(
        self,
        current_folder: Nautilus.FileInfo,
    ) -> List[Nautilus.MenuItem]:
        return []
```