



b.note

v3.0.0

**Manuel du développeur
(2024-06-25)**



SOMMAIRE

Description	4
Matériel nécessaire	4
Accès aux sources de b.note	4
Connexion à b.note	5
Avec filezilla si l'adresse ip de b.note est 192.168.1.10	5
En ssh	5
Activation de l'option de développement	6
Installation d'un dossier de développement	6
Développement sur b.note sans PycharmPE	7
Développement Sur PC avec PyCharmPE (Professional Edition)	7
Installation	7
Activation de la licence d'essai	8
Configuration du 'Python Interpreter'	8
Modification des packages de bnote	14
Gestion des packages python	14
Gestion des librairies linux	14
Réalisation et exécution d'un update de bnote	14
Exécution d'un update de bnote	15
Structure de l'application	16
L'application	16
Les événements clavier	16
Clavier de commande	16
Clavier braille	16
Clavier curseur routine	17
Autres évènements	17
Fonction	17
Timer	18
Le rafraîchissement de la plage braille	18
Objets standards	18
Les menus	18
Les boîtes de dialogue	19
Eléments	19
Boîtes de dialogue prédéfinies	20
Gestion du braille	20
Les fonctions clés	20
Synthèse vocale	20

Description

Ce manuel décrit comment développer une nouvelle application dans b.note.

Les applications internes de b.note sont entièrement écrites en python 3 sur une RaspberryPi 3 modèle A+.

L'interface utilisateur de ces applications a été écrite de façon à permettre un développement rapide de nouvelles applications.

Elle donne accès à :

- Une barre de menu,
- Des boîtes de dialogue,
- Une zone document ayant accès aux événements clavier et à l'affichage en braille,
- Une synthèse vocale.

Matériel nécessaire

- Un ordinateur personnel pouvant supporter "Pycharm Professional Edition",
- Un b.note,
- Une connexion wifi.

Note :

L'utilisation de PycharmPE permet la pose de points d'arrêt, la visualisation des variables et le pas à pas sur un programme s'exécutant sur b.note (remote debugging). C'est de loin la méthode de développement la plus confortable.

Il est néanmoins possible de modifier le source de l'application directement sur b.note sans outils de développement.

Accès aux sources de b.note

Les sources de b.note sont la propriété d'Eurobraille, un développeur ou une société voulant développer une nouvelle application devra donc en faire la demande à Eurobraille qui lui enverra l'intégralité des sources.

C'est avec plaisir que nous vous remercions de l'intérêt que vous portez à notre produit et si vous réalisez une application qui peut profiter à l'ensemble des utilisateurs de b.note nous sommes prêts à l'intégrer dans la version officiel de b.note.

Connexion à b.note

Dans les préférences de b.note, il est possible de définir une connexion wifi, il faudra pour cela connaître le ssid et le mot de passe wifi de l'endroit où l'on se trouve et se rapporter au manuel utilisateur de b.note.

b.note est paramétré pour demander une adresse ip au serveur dhcp du réseau local. Cette adresse est visible dans les préférences de b.note (rubrique wifi).

Une fois ces étapes accomplies, b.note est accessible en console ssh ou avec filezilla par exemple.

login : pi

mot de passe : euroberry

Avec filezilla si l'adresse ip de b.note est 192.168.1.10

Général	Avancé	Paramètres de transfert	Jeu de caractères
Protocole :	SFTP - SSH File Transfer Protocol ▼		
Hôte :	192.168.1.10	Port :	
Type d'authentification :	Normale ▼		
Identifiant :	pi		
Mot de passe :		

En ssh

```
$ssh pi@192.168.1.10
```

```
pi@192.168.1.10's password:
```

```
Linux raspberrypi 5.10.63-v7+ #1488 SMP Thu Nov 18 16:14:44 GMT 2021 armv7l
```

The programs included with the Debian GNU/Linux system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

Last login: Wed Jan 12 09:54:12 2022 from 192.168.1.60

pi@raspberrypi:~ \$

Activation de l'option de développement

Dans le menu préférence dans la section interface utilisateur basculer l'option mode développeur sur oui.

ou

Une option de développement permet de passer b.note en mode développement. Pour cela modifier le fichier /home/pi/.b.note/settings.txt en remplaçant la ligne :

```
"system": {
  "braille_type": "dot-8",
  "games": false,
  "developer": false
},
en :
"system": {
  "braille_type": "dot-8",
  "games": false,
  "developer": true
},
```

Une fois ce paramètre modifié et après redémarrage de b.note :

- Une application exemple 'skeleton' se trouve dans la barre du menu application. Cette application permet de voir comment une application peut gérer ses menus, ses boîtes de dialogue, l'afficheur braille et les événements clavier. Les premiers essais pourront être réalisés en modifiant cette application.

Installation d'un dossier de développement

Pour créer un environnement de développement indépendant de l'application bnote installée sur l'appareil un dossier develop/ est déjà créé sur la sd. Il contient un fichier .tar.gz contenant les sources de l'application bnote.

```
pi@raspberrypi:~/develop $tar -xvf bnote-3.0.0.tar.gz
```

```
pi@raspberrypi:~/develop $mv bnote-3.0.0 bnote
```

```
pi@raspberrypi:~/develop $cd bnote
```

```
pi@raspberrypi:~/develop/bnote $ sh ./setup.sh
```

Développement sur b.note sans PycharmPE

Une fois l'installation des sources et de l'environnement virtuel dans le dossier develop/bnote, il est possible de modifier les fichiers sources (.py) de b.note soit localement en ssh avec nano par exemple puis de lancer l'application manuellement.

Arrêter le service de lancement de l'application :

```
pi@raspberrypi:~ $sudo systemctl stop bnote.service
```

Lancer manuellement l'application :

```
pi@raspberrypi:~/develop/bnote $ source venv/bin/activate
```

```
(venv)pi@raspberrypi:~/develop/bnote $python3 __main__.py
```

ou

```
(venv)pi@raspberrypi:~/develop/bnote $python3 __main__debug__.py
```

Cette seconde méthode a l'avantage de rendre en ssh les traces de l'application visibles sur la console.

Pour deactiver environnement virtuel python :

```
(venv)pi@raspberrypi:~/develop/bnote $ deactivate
```

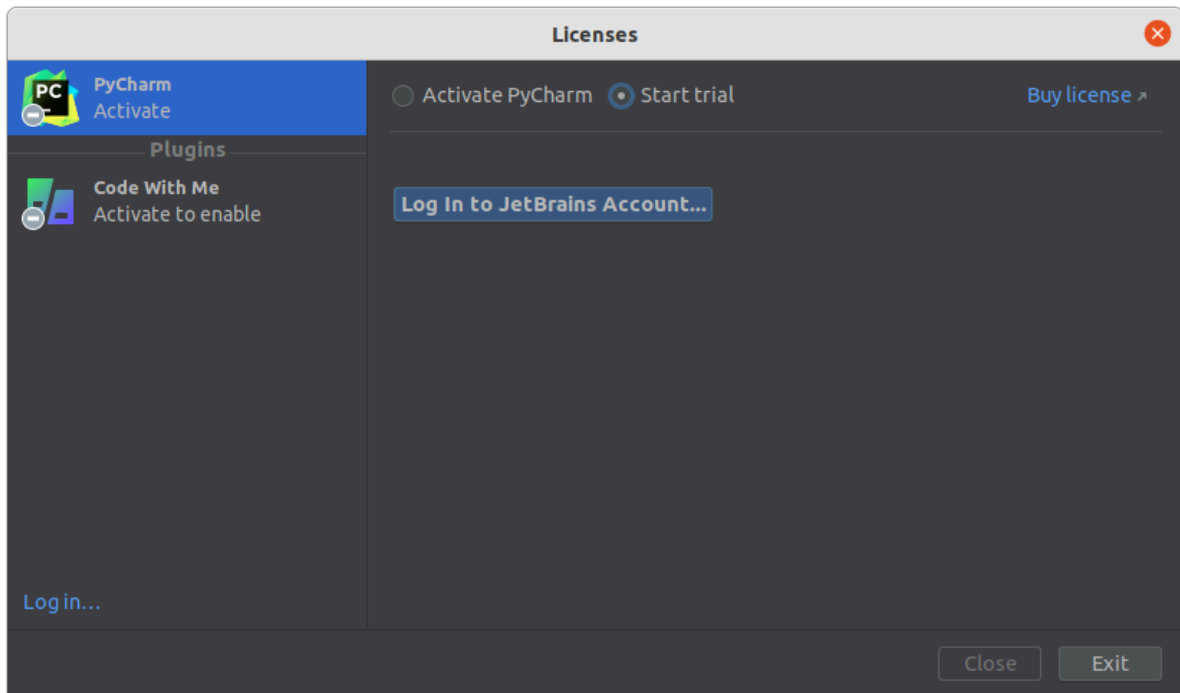
Développement Sur PC avec PyCharmPE (Professional Edition)

PyCharmPE contrairement à la version CE (Community Edition) permet le "remote debugging". Nous l'avons testé sous Ubuntu et Windows et elle existe aussi pour Mac. Cette version de PyCharm est sous licence (Il existe une version d'essai limitée à 30 jours), l'éditeur JetBrains propose différents abonnements dont un mensuel.

Installation

- Télécharger une version sur <https://www.jetbrains.com/pycharm/>
- Sous linux : Extraire le fichier tar.gz et suivre les instructions du fichier
- Sous Windows : Lancer l'installateur (.exe)

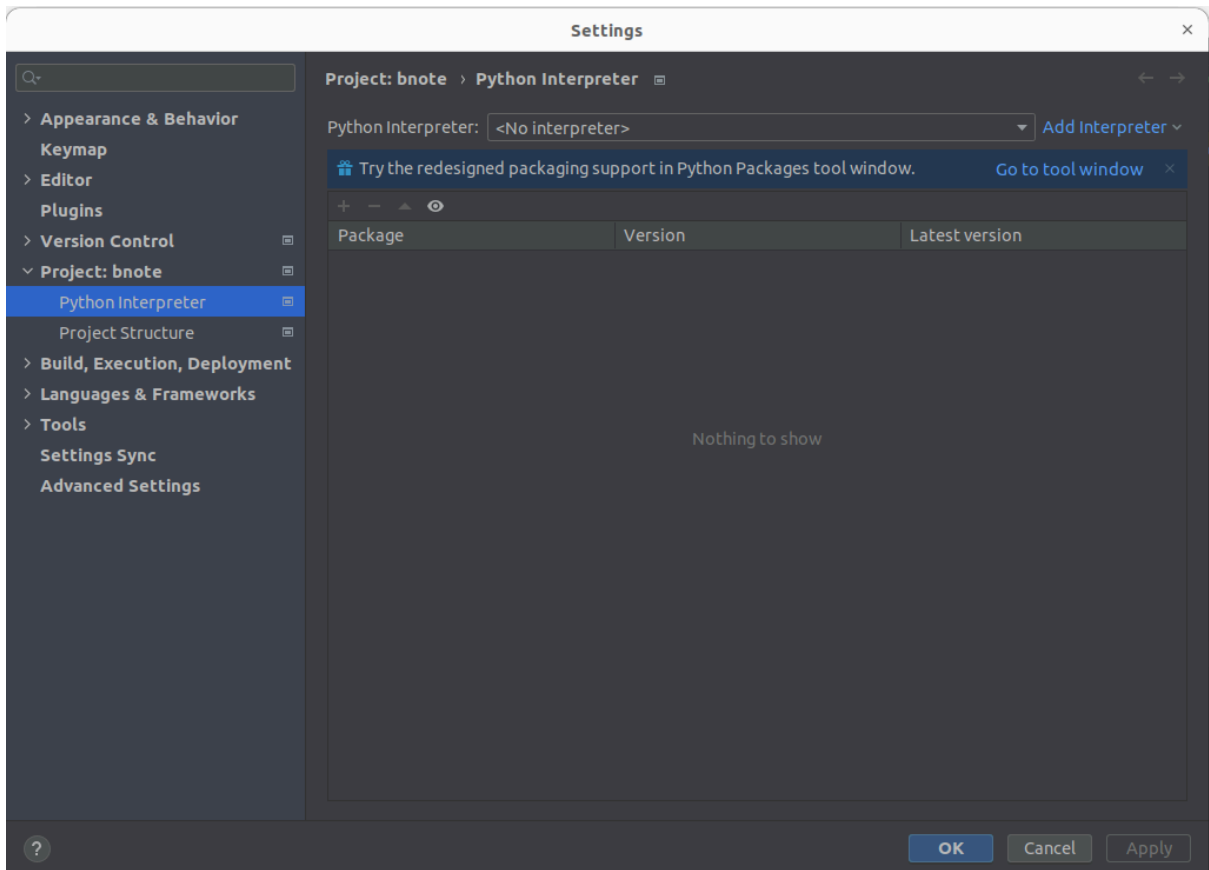
Activation de la licence d'essai



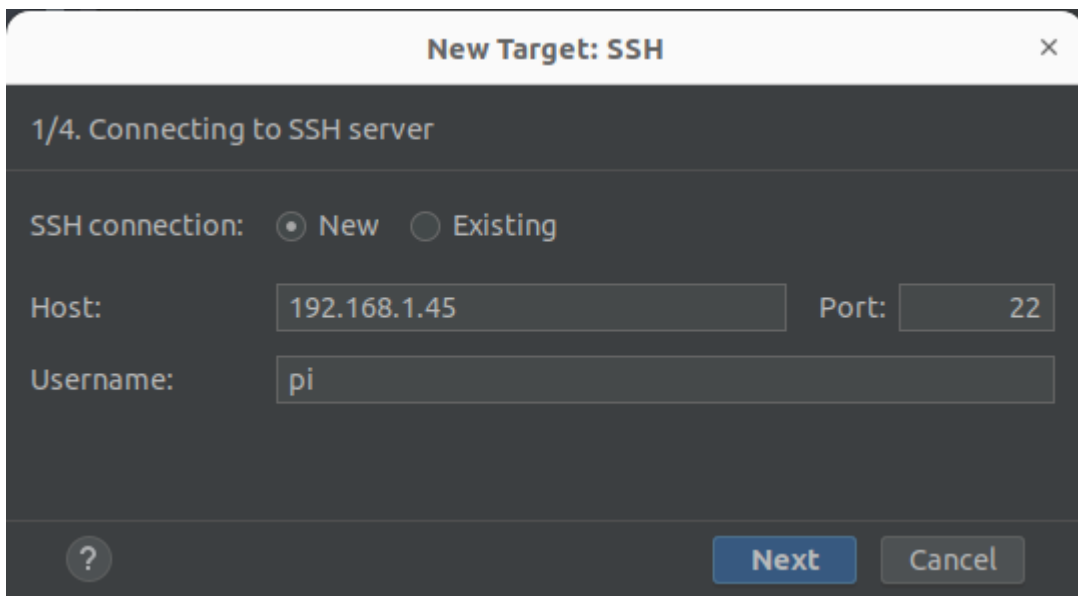
Cliquez sur Log in to JetBrains Account...

Configuration du 'Python Interpreter'

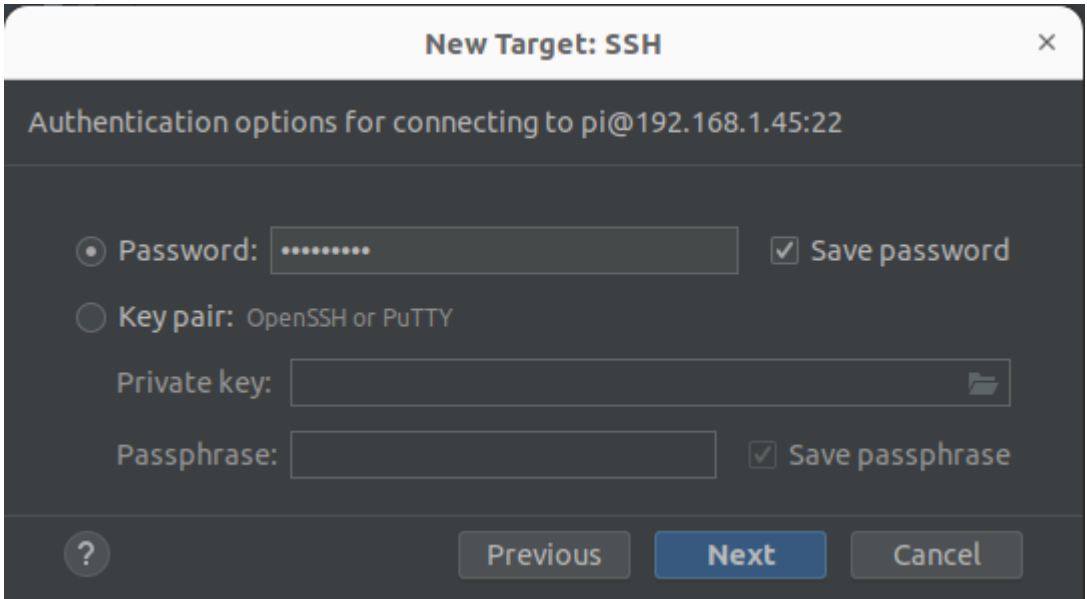
Cliquez sur le menu File>Settings, puis ouvrir Project:bnote>Python interpreter



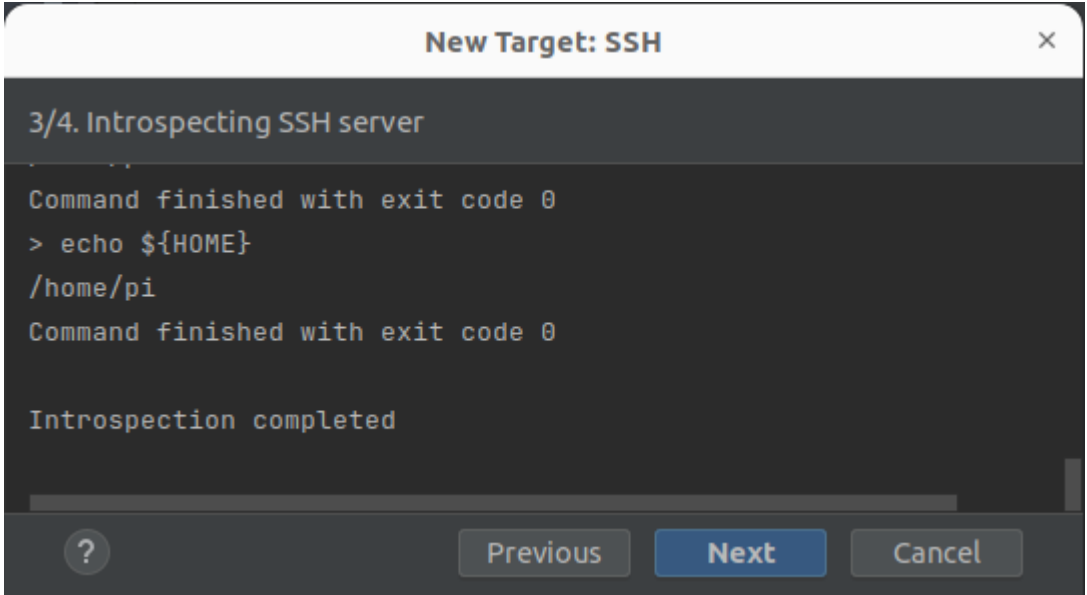
puis cliquer sur Add interpreter>on SSH
renseigner l'adresse ip et le nom d'utilisateur (pi)



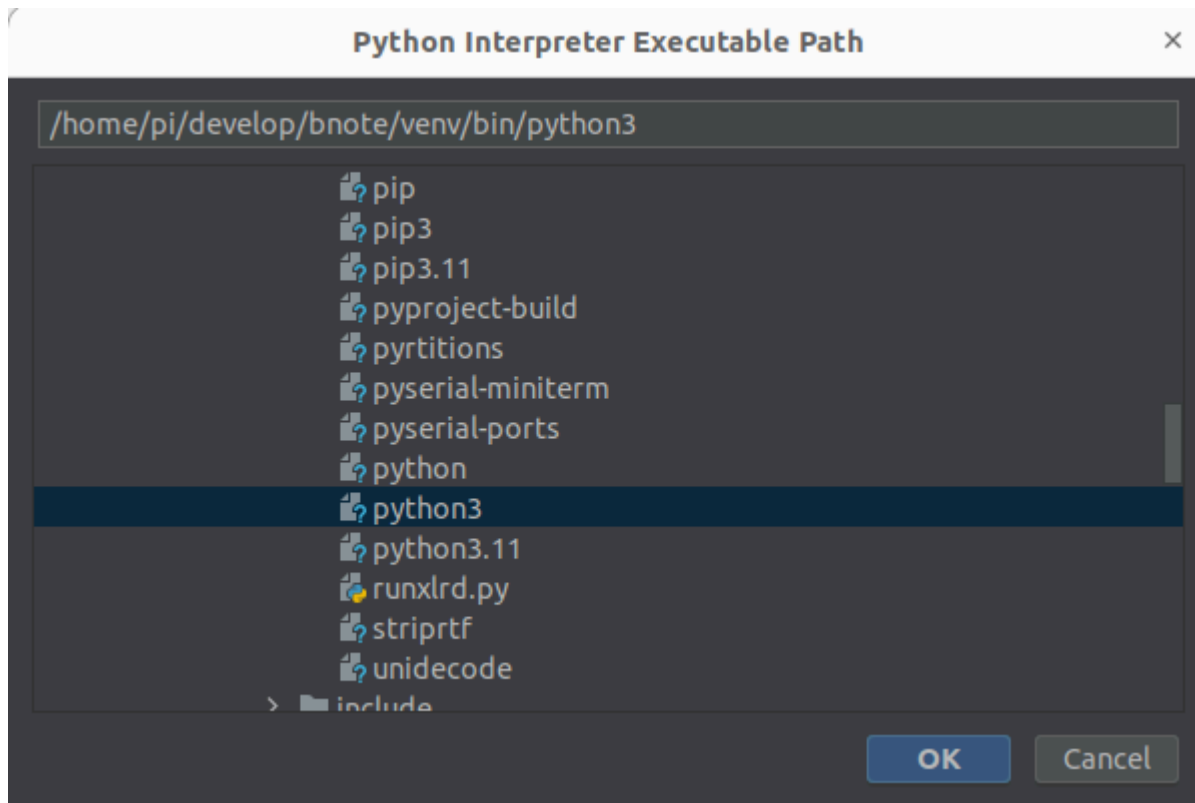
Cliquer sur Next puis renseigner le mot de passe (euroberry)



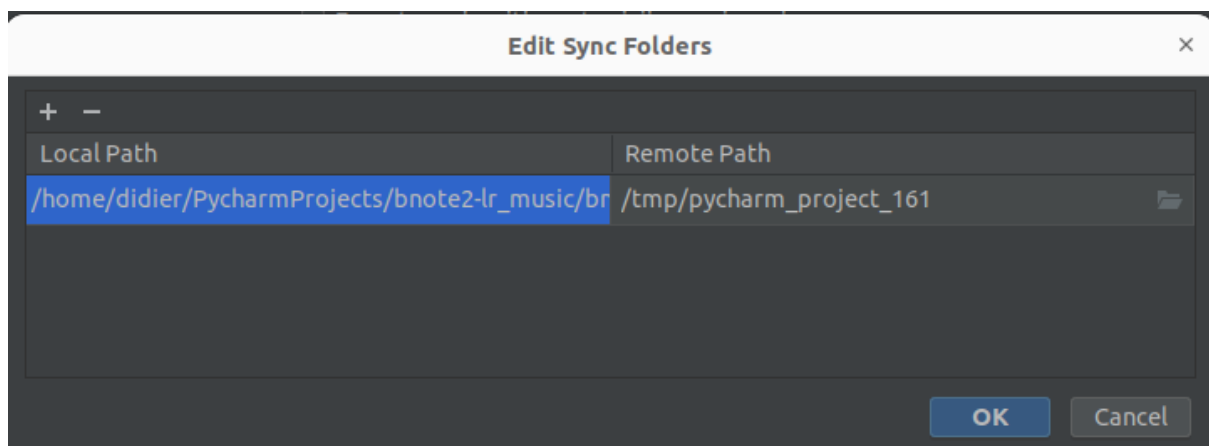
cliquer sur next, pycharm execute alors un test de connexion

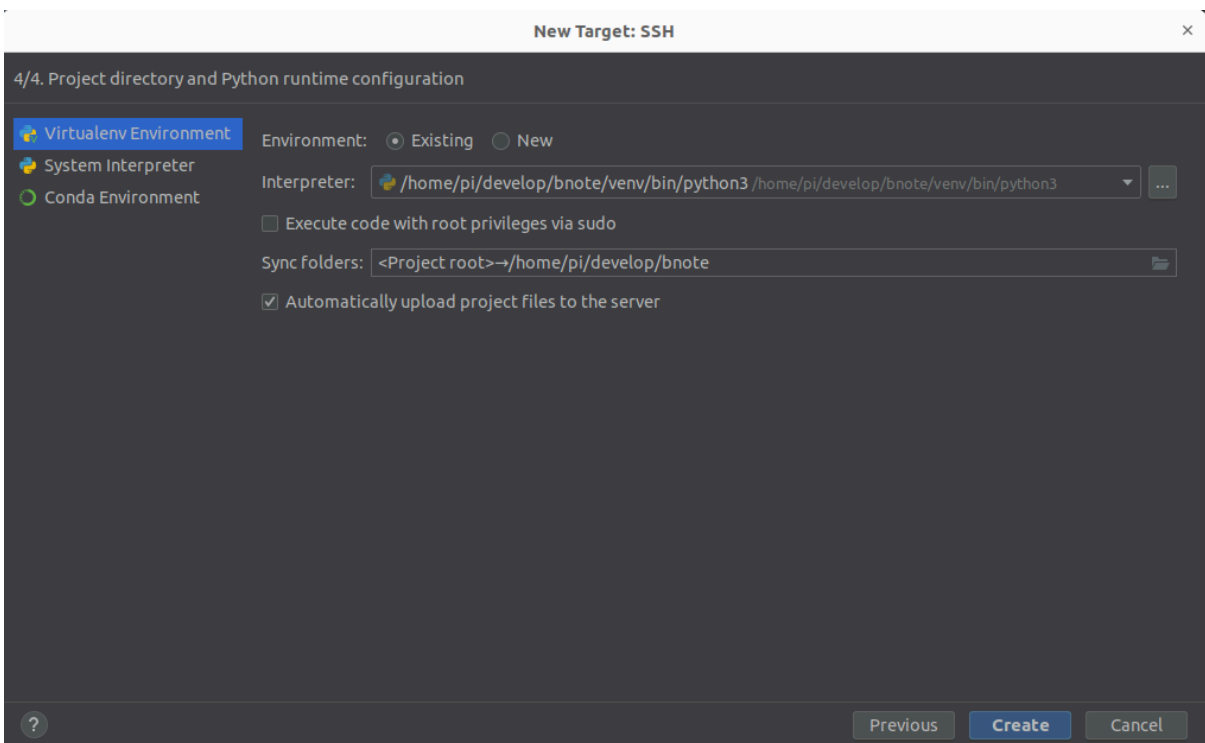
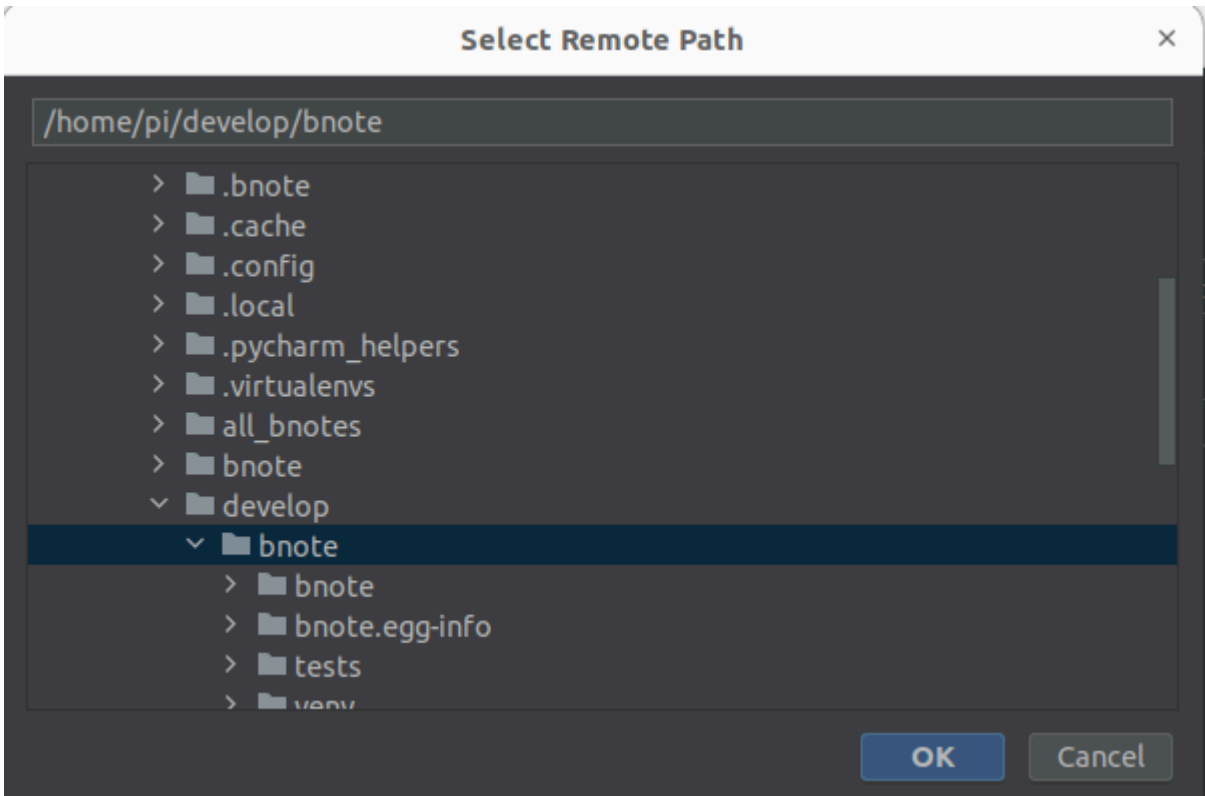


Cliquer sur next, puis cocher le bouton existing et définir le VirtualEnv Environnement sur /home/pi/develop/bnote/venv/bin/python3

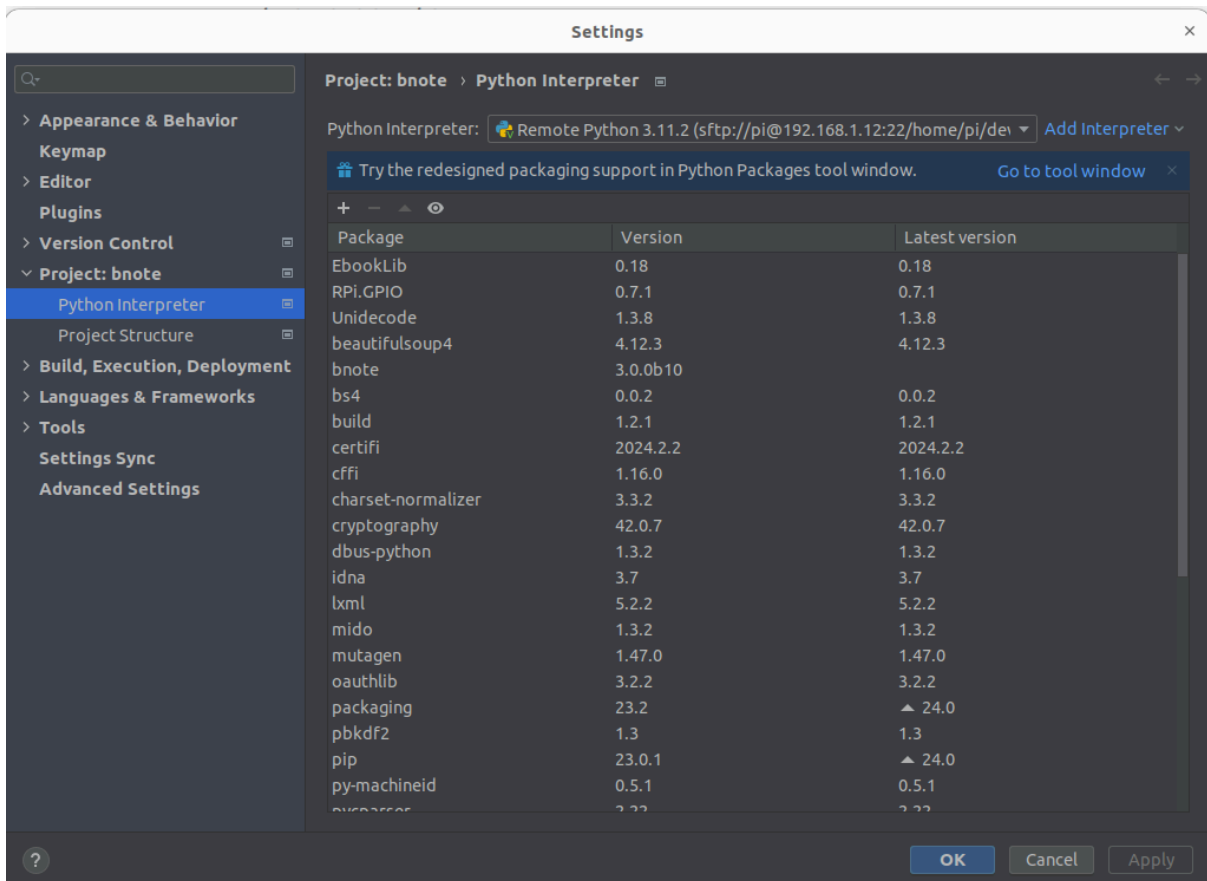


Cliquer sur le dossier de Sync folders pour définir l'emplacement où les sources seront synchronisées avec Pycharm





Cliquer sur ok puis sur create



Une fois ces opérations réalisées, vous pouvez faire clic droit sur le dossier bnote racine du projet puis choisir `Deployment > Upload to ...` pour synchroniser les sources de votre dossier de développement de bnote avec celles de pycharm.

Vous pouvez ensuite lancer `bnote_start.py` depuis pycharm.

Pour plus d'information consulter le lien suivant :

<https://www.jetbrains.com/help/pycharm/remote-debugging-with-product.html>

Modification des packages de bnote

Gestion des packages python

En ssh sur le bnote, depuis votre dossier de développement, vous pourrez taper:

```
pi@raspberrypi:~/develop/bnote $ source venv/bin/activate  
(venv)pi@raspberrypi:~/develop/bnote $pip install mon_package
```

Pour récupérer le package et ses dépendances vous pouvez utiliser les commandes suivantes:

```
pi@raspberrypi:~/develop/bnote $mkdir -p /tmp/packages  
pi@raspberrypi:~/develop/bnote $pip download --dest /tmp/packages requests  
Vous pourrez alors le télécharger et l'inclure dans le dossier whl/ de votre update et ajouter son nom et sa version dans pyproject.toml de votre projet bnote.
```

pour retirer un package python.

```
pi@raspberrypi:~/develop/bnote $pip uninstall mon_package
```

Gestion des librairies linux

En ssh sur le bnote, depuis votre dossier de développement, vous pourrez taper:

```
pi@raspberrypi:~/develop/bnote $sudo apt install ma_librairie  
pour ajouter une librairie.  
pi@raspberrypi:~/develop/bnote $sudo apt remove ma_librairie  
pour retirer une librairie.
```

Lorsque vous réaliserez l'update, il vous faudra tenir compte des changements de librairie en mettant à jour le fichier libraries.txt.

Chacune des commandes de ce fichier sera exécutée lors de l'installation de la nouvelle version de l'application. Une erreur durant ces exécution se soldera par un échec de l'installation de la version.

Ce fichier ne cessera de grandir au fil des mises à jour.

Réalisation et exécution d'un update de bnote

En ssh sur le develop/bnote, depuis votre dossier de développement taper:

```
pi@raspberrypi:~/develop/bnote $ sh ./generate.sh  
Cela générera un fichier bnote-....whl.zip au nom de la version de bnote définie dans pyproject.toml.
```

Exécution d'un update de bnote

C'est le fichier bnote...whl.zip qui permettra l'installation de l'application. Il suffira de le copier sur le bnote cible et de l'exécuter depuis l'explorateur de fichier de bnote.

La nouvelle version sera installée dans le dossier all_bnotes avec son propre environnement virtuel.

Il est donc possible de faire cohabiter plusieurs versions de bnote et de choisir celle qui sera lancée au démarrage.

Structure de l'application

Le code source se trouve le dossier /home/pi/b.note

Le dossier documents visibles depuis l'application se trouve dans /home/pi/.b.note

L'application

Toutes les applications de b.note dérivent d'une classe b.noteApp située dans le fichier b.note_app.py.

C'est grâce à cette classe de base que fonctionne les menus, les boîtes de dialogue et le mécanisme de rafraîchissement braille de l'application.

Les événements clavier

Les événements clavier ont été catégorisés en 4 types qui correspondent chacun à une fonction python de l'application, elle viennent surcharger celles de la classe de base, il est donc important qu'elles appellent les fonctions de la classe de base tel que cela est fait dans skeleton.py pour assurer le bon fonctionnement des boîtes de dialogue et des menus.

Clavier de commande

L'appui sur une ou plusieurs touches des 2 pavés de 4 touches déclenchent l'événement ci-dessous :

```
def input_command(self, data, modifier, key_id) -> bool:
    """
    Does what is expected for this command key.
    :param data: ?
    :param modifier: bits field (see Keyboard.BrailleModifier)
    :param key_id: (see Keyboard.KeyId)
    :return: True if command treated, otherwise False
    """
```

Note : Un événement avec un `key_id = Keyboard.KeyId.KEY_NONE` est généré lorsque toutes les touches sont relâchées. Il est à ignorer pour les applications de b.note.

Clavier braille

Le combinaisons de touche du clavier braille sont réparties en 2 catégories :

- Les combinaisons conduisant à un caractère alphanumérique (`input_character()`)
- Les combinaisons conduisant à une fonction (`input_bramigraph()`)

```
def input_character(self, modifier, character, data) -> bool:
    """
    Do what needs to be done for this braille modifier and character.
    :param modifier: bits field (see Keyboard.BrailleModifier)
```



```

        :param character: unicode char
        :param data: brut braille comb. for advanced treatment
        :return: True if command treated, otherwise False
        """

def input_bramigraph(self, modifier, bramigraph) -> bool:
    """
    Do what needs to be done for this modifier and bramigraph.
    :param modifier: bits field (see Keyboard.BrailleModifier)
    :param bramigraph: braille function (see
Keyboard.BrailleFunction)
    :return: True if command treated, otherwise False
    """

```

Clavier curseur routine

L'appui sur une touche curseur routine appelée touche interactive dans b.note déclenche la fonction ci-dessous :

```

def input_interactive(self, modifier, position, key_type) -> bool:
    """
    Do what needs to be done for this modifier and cursor routine
event.
    :param modifier: bits field (see Keyboard.BrailleModifier)
    :param position: index of key (based 1)
    :param key_type: see Keyboard.InteractiveKeyType
    :return: True if command treated, otherwise False
    """

```

Autres évènements

Fonction

Si l'application exploite le multi-tâche, elle peut avoir besoin de déclencher des fonctions qui seront prises en compte par le processus principal, ces évènement déclenchent la fonction ci-dessous :

```

def input_function(self, *args, **kwargs) -> bool:
    """
    Call when function is not treated by base class of this class.
    :param args[0]: The function id
    :param kwargs:
    :return: True if function treated.
    """

```

Pour déclencher une fonction, il suffit d'écrire :

```
self._put_in_function_queue(FunctionId.FUNCTION_...)
```

Timer

Un événement timer tombe toutes les secondes

```
def on_timer(self):  
    """  
    Event each seconds  
    :return: None  
    """
```

Le rafraîchissement de la plage braille

Une seule fonction permet d'afficher une ligne de texte sur l'afficheur braille

```
def set_data_line(self):  
    """  
    Construct the braille display line from document  
    :return: None (self._braille_display.set_data_line is done)  
    """
```

En général, cette fonction permet d'envoyer sur l'afficheur braille la ligne du document de l'application.

Le codage de cette fonction consiste à construire 3 buffers de texte

- Un buffer de texte alphanumérique (pour esyviewer notamment)
- Un buffer de braille unicode décrivant les points fixes
- Un buffer de braille unicode décrivant les points clignotants

puis d'appeler `self._braille_display.set_data_line` avec ces 3 buffers comme paramètre.

Objets standards

Les menus

Les menus sont gérés par 2 classes du dossier ui :

- `UiMenuBar` La barre de menu ou un sous menu
- `UiMenuItem` Un élément de menu terminal

Un `UiMenuBar` est caractérisé par :

```
'name': le nom du sous menu,  
'action': (None par défaut) Une fonction de l'application,  
'ui_objects': Une liste décrivant son contenu, elle sera composée  
de UiMenuBar et de UiMenuItem,  
'is_root': (False par défaut) ce booléen permet de marquer  
l'élément racine de la description de menu,  
'focused_object': 0
```

Un `UiMenuItem` est caractérisé par :

```
'name': le nom de l'élément de menu,
```

'shortcut_modifieur': les modifieurs du raccourci clavier associé à cet élément,

'shortcut_key': la description de la touche de raccourci,

'action': Chaque élément de menu terminal est associé à une fonction de l'application nommé par convention

```
def _exec_menu_nom_de_la_fonction(self),
```

'action_param': (None par défaut) Ce paramètre permet de définir des paramètres à passer lors du lancement de l'action,

'is_hide': (False par défaut) Masquage de l'élément de menu,

Exemple de création d'un menu :

```
def __create_menu(self):
    # Instantiate menu (A menu bar with 1 sub menu of 2 menu items and one menu
    item).
    return UiMenuBar(
        name=_("skeleton"),
        # Call on ESC bramigraph key
        is_root=True,
        menu_item_list=[
            UiMenuBar(
                name=_("&group"),
                menu_item_list=[
                    UiMenuItem(name=_("&menu_1"), action=self._exec_menu_1),
                    UiMenuItem(name=_("&menu_2"), action=self._exec_menu_2),
                ],
            ),
            UiMenuItem(name=_("&say hello"), action=self._exec_say_hello),
            UiMenuItem(name=_("&about"), action=self._exec_about,
                shortcut_modifieur=Keyboard.BrailleModifier.BRAILLE_FLAG_NONE,
                shortcut_key = Keyboard.BrailleFunction.BRAMIGRAPH_F1),
        ],
    )
```

Les boîtes de dialogue

Eléments

Les boîtes de dialogue sont gérées par les classes du dossier ui suivantes :

- UiDialogBox La boîte de dialogue
- UiCheckBox Une case à cocher
- UiListBox Une liste
- UiEditBox Une boîte éditable
- UiLabel Un texte non modifiable
- UiButton Un bouton

Boîtes de dialogue prédéfinies

Des boîtes de dialogue standard sont déjà définies dans le but de globaliser les boîtes à usage général et d'alléger le code des applications. Elles correspondent aux classes suivantes :

- `UiMessageDialogBox` Une boîte de dialogue composé d'un nom, d'un message et d'une liste de bouton,
- `UiInfoDialogBox` Une boîte de dialogue d'information composée d'un message et d'un bouton Ok,

Gestion du braille

Le braille est géré par une classe globale `b.noteApp.lou` qui correspond à une instance de `liblouis` dans la langue en usage pour l'appareil.

Les fonctions clés

```
def to_dots_8(self, txt):  
    """  
    Convert a string into an unicode braille string (8 dots) (x28nn chars).  
    : param txt : (str) alphanumeric string of text  
    : return : (str) unicode braille string (\u28xx...)  
    """
```

Cette fonction permet de convertir un texte en une chaîne de caractères braille. C'est la seule fonction utile pour une application qui reste en braille informatique 8 points, elle permettra d'envoyer en ligne convertie à l'afficheur braille.

Synthèse vocale

`b.note` dispose d'un point d'entrée global pour activer la synthèse vocale

```
def speak(text, lang_id=None, volume=None, speed=None,  
purge_before_speak=True):
```

Pour une utilisation plus avancée de la synthèse vocale il faudra directement s'interfacer à la classe `SpeechManager()`.