Rocketbot Studio : Conflictos de Librerías

El empleo de diversas librerías en Rocketbot puede dar lugar a incompatibilidades si las versiones requeridas por los módulos personalizados no coinciden con las que ofrece la plataforma. Este informe examina las razones detrás de estos conflictos y sugiere soluciones para solucionarlos.

Importante

Es fundamental considerar la versión de Python utilizada al instalar librerías en Rocketbot. La versión 2020 de Rocketbot opera con Python 3.6.8 de 32 bits, mientras que a partir de la versión 2023 se utiliza Python 3.10 de 64 bits.

En caso de ejecutar módulos mediante un proceso externo, la librería correspondiente debe estar instalada en el entorno desde el cual se realiza la ejecución.
Esto permite operar dentro del entorno adecuado, evitar incompatibilidades y asegurar la correcta recepción de la información en Rocketbot.

Causas de los Conflictos

Los conflictos pueden surgir por diversas razones, como la existencia de diferentes versiones de librerías que no coinciden con las de Rocketbot. También pueden presentarse problemas de compatibilidad entre dependencias, ya que múltiples versiones de una misma librería pueden causar errores. Además, la instalación de versiones adicionales puede interferir con las librerías predeterminadas de Rocketbot.

Soluciones Recomendadas

Renombrar el Archivo de la Librería:

En casos donde se vea errores como el siguiente:

```
cannot import name 'deprecated' from 'typing_extensions' (C:\Rocketbot_win_20240528\typing_extensions.pyc)
```

typing_extensions es una librería que viene compilada con Rocketbot y este error suele darse porque viene con una versión más antigua que la instalada para el script, por lo que al ejecutar otra librería que tiene como dependencia typing_extensions, rompe.

En estos casos se recomienda indicar la ruta hacia la lib que instalaron o si el error persiste como workaround se puede renombrar la carpeta de la librería importada o el nombre del archivo.py por uno diferente, por ejemplo en lugar de llamarlo **typing_extensions.py** lo llamaremos

new_typing_extensions.py. En tu código deberás encontrar los imports que hagan tu scripts o las dependencias que importas cambiando lo siguiente:

import typing_extensions por **import new_typing_extensions as typing_extensions** (así se mantiene el nombre original)

Y también (teniendo en cuenta el error del ejemplo):

from typing_extensions import deprecated por **from new_typing_extensions import deprecated**

Con estos cambios, el script ejecutado por Rocketbot entenderá que no debe utilizar typing_extensions que viene nativamente con Rocketbot, sino **new_typing_extensions** instalado manualmente por nosotros.

Este ejemplo se puede aplicar tanto a archivos .py como a carpetas de librerías. Por ejemplo si queremos utilizar una nueva versión de la librería pandas, podemos descargarla localmente, renombrar la carpeta a new_pandas y luego en nuestro script la importamos con ese nombre.

Uso de Procesos Externos:

En los casos donde la solución anterior no funcione, se sugiere ejecutar el proceso mediante popen o a través de un webhook. Esto permite el uso de la versión de Python y librerías del ambiente donde se ejecute en vez de usar los de Rocketbot.

Popen

Para ejecutar su script mediante un Popen realice lo siguiente en un comando Ejecutar Python:

```
from subprocess import Popen, PIPE
process = subprocess.Popen(
        ['python', 'ruta/a/su/script.py', 'arg1', 'arg2', 'argN'],
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE
    )
out, err = process.communicate()
print(out.decode())
```

En su script imprima en consola para obtener lo que necesita en Rocketbot.
En el ejemplo se ejecuta un script donde se le envían argumentos (que bien podrían ser datos de variables de Rocketbot). Luego en el script que está ejecutando toma los argumentos mediante: **sys.argv**
Esto trae cada valor de la lista que se le pasa al popen: **sys.argv[0]** traería python, **sys.argv[1]** ruta_script.py, **sys.argv[2]** arg1 y así según se necesite. Es opcional para el caso en que se necesite enviar datos de Rocketbot al script.

Webhook

Al levantar un puerto local se queda en pausa a la escucha de algún evento

POST o GET.

- Es un módulo que se descarga del market, el manual se encuentra en la carpeta example del módulo, en la ruta Rocketobot/modules/Webhook
- Puedes levantar un endpoint propio donde enviar información y que el robot escuche o usar ngrok para levantar uno (pasos en el manual).
- Al ejecutar el comando de 'Crear webhook', Rocketbot se pausará esperando que se consulte la url configurada en el comando, ya sea con una petición GET o POST.

En el contexto de conflicto de librerías, un workaround utilizando Webhook es crear un .bat de la siguiente manera:

C:/Python312/python.exe "C:\Users\user\Desktop\archivo.py"

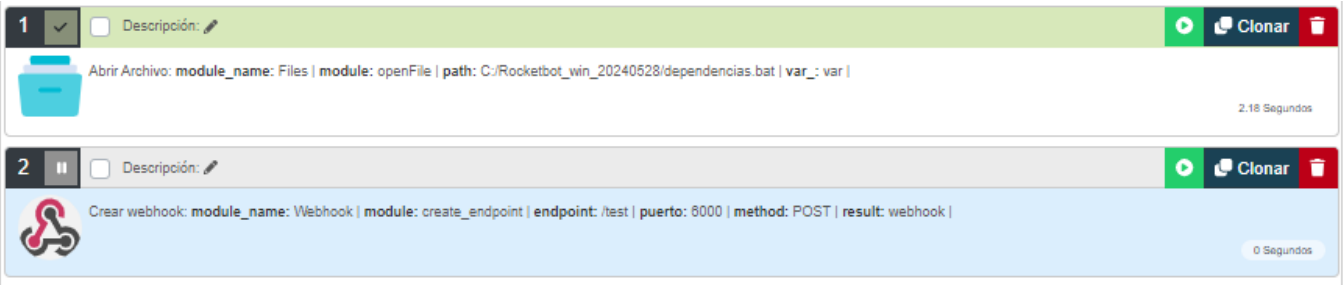
Cambiando el path que está entre comillas por el path hacia el script a ejecutar y el primer path por el path hacia el ejecutable de Python en el ambiente a utilizar.

Dentro del archivo.py que contiene el script a ejecutar se tiene que agregar al principio las importaciones de request y un sleep que dará tiempo a ejecutar el comando de webhook. Al final del script se tiene que agregar la petición post para recibir la información mediante webhook al endpoint levantado:

```
import requests
from time import sleep
sleep(10)
(acá va el código)
requests.post("http://127.0.0.1:6000/test", data={"var1":"data1", "var2":
"data2"})
```

Cambiando 127.0.0.1:6000 por el host y puerto elegido. El parámetro data en este ejemplo es opcional, se puede utilizar si se requiere enviar información a Rocketbot.

En el bot se tienen que crear los siguientes comandos uno seguido del otro:



El comando de webhook se tiene que ejecutar dentro de los segundos declarados en el sleep del archivo.py, en este ejemplo se tiene que ejecutar antes de los 10 segundos.