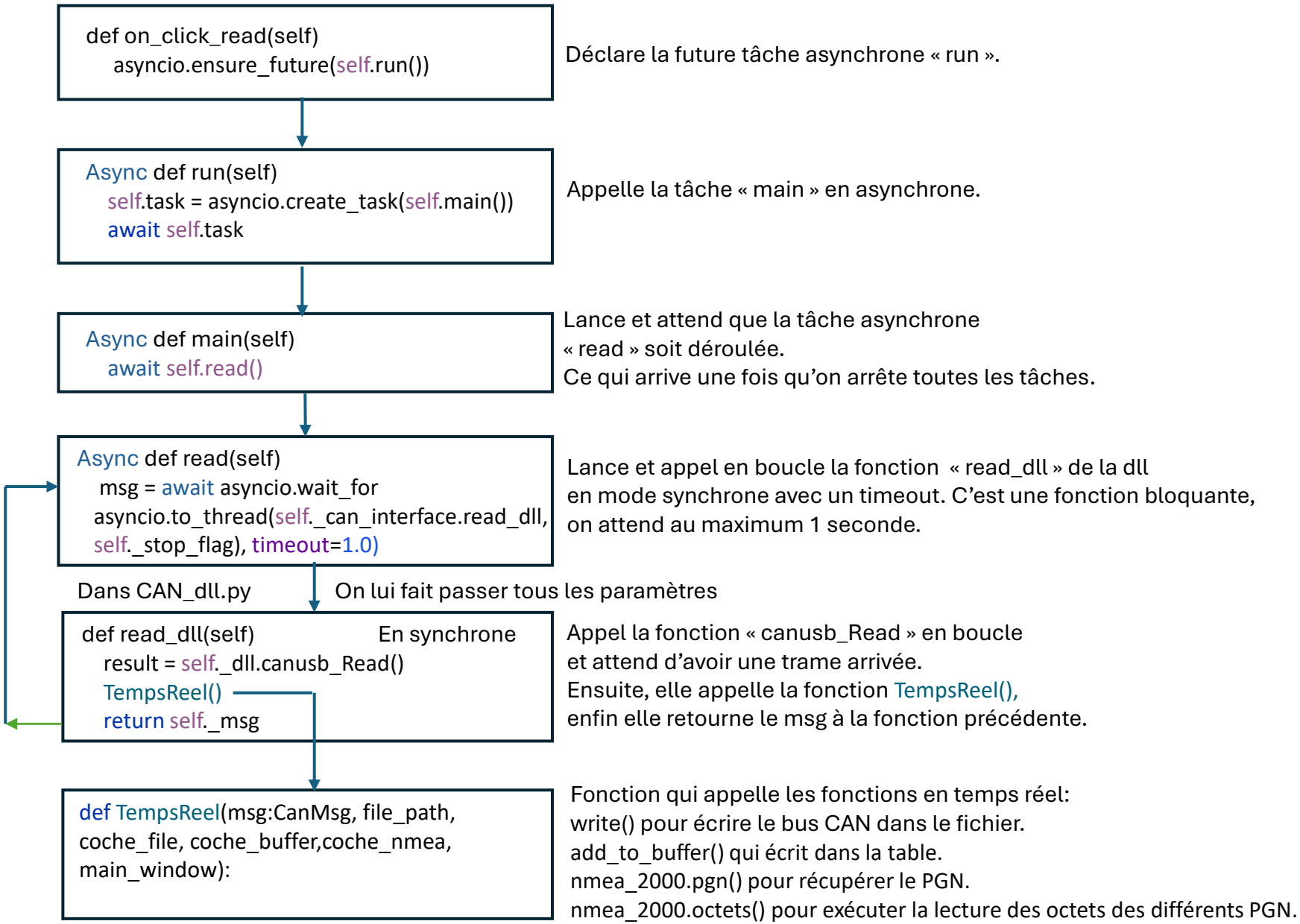
def on_click_read(self)
asyncio.ensure_future(self.run())
Déclare la future tâche asynchrone « run ».
Async def run(self)
self.task = asyncio.create_task(self.main())
await self.task
Appelle la tâche « main » en asynchrone.
Async def main(self)
await self.read()
Lance et attend que la tâche asynchrone « read » soit déroulée.
Ce qui arrive une fois qu'on arrête toutes les tâches.
Async def read(self)
msg = await asyncio.wait_for
asyncio.to_thread(self._can_interface.read_dll,
self._stop_flag), timeout=1.0)
Lance et appel en boucle la fonction « read_dll » de la dll en mode synchrone avec un timeout. C'est une fonction bloquante, on attend au maximum 1 seconde.
Dans CAN_dll.py
On lui fait passer tous les paramètres
def read_dll(self)
En synchrone
result = self._dll.canusb_Read()
TempsReel()
return self._msg
Appel la fonction « canusb_Read » en boucle et attend d'avoir une trame arrivée.
Ensuite, elle appelle la fonction TempsReel(),
enfin elle retourne le msg à la fonction précédente.
def TempsReel(msg:CanMsg, file_path,
coche_file, coche_buffer,coche_nmea,
main_window):
Fonction qui appelle les fonctions en temps réel:
write() pour écrire le bus CAN dans le fichier.
add_to_buffer() qui écrit dans la table.
nmea_2000.pgn() pour récupérer le PGN.
nmea_2000.octets() pour exécuter la lecture des octets des différents PGN.
Ces trois fonctions sont exécutées en fonction des cases à cocher.