Proyecto embebidos 20221

Generado por Doxygen 1.9.2

Capítulo 1

Indice jerárquico

1.1. Jerarquía de la clase

Esta lista de herencias esta ordenada aproximadamente por orden alfabético:

Capítulo 2

Índice de clases

2.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

Capítulo 3

Indice de archivos

3.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

Capítulo 4

Documentación de las clases

4.1. Referencia de la Clase core.BD.BD

Métodos públicos

- def __init__ (self)

  Esta clase solo se encarga de conectar la base de datos.
- def ejecutar_consulta (self, query, un_resultado=True)

  Ejecuta la consulta.

Atributos públicos

- conn

4.1.1. Descripción detallada

Definición en la línea 6 del archivo BD.py.

4.1.2. Documentación del constructor y destructor

4.1.2.1. __init__()

```
def core.BD.BD.__init__ (
              self )
```

Esta clase solo se encarga de conectar la base de datos.

implementada en Postgresql.

Definición en la línea 7 del archivo BD.py.

Gráfico de llamadas para esta función:

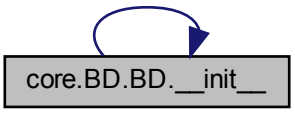
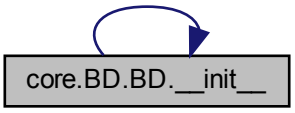


Gráfico de llamadas a esta función:



4.1.3. Documentación de las funciones miembro

4.1.3.1. ejecutar_consulta()

```
def core.BD.BD.ejecutar_consulta (
              self,
              query,
              un_resultado = True )
```

Ejecuta la consulta.

Parámetros

query	Es la consulta que se realizará
un_resultado	Se esfecifica en True si solo se espera una fila de resultados, False si se espera varias filas, por defecto esta en True

Devuelve

Ua o unas tuplas de resultados

Definición en la línea 20 del archivo BD.py.

Gráfico de llamadas para esta función:

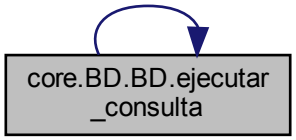
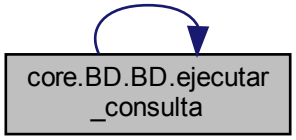


Gráfico de llamadas a esta función:



4.1.4. Documentación de los datos miembro

4.1.4.1. conn

`core.BD.BD.conn`

Definición en la línea 12 del archivo BD.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- core/BD.py

4.2. Referencia de la Clase Bluetooth.Bluetooth

Diagrama de herencias de Bluetooth.Bluetooth

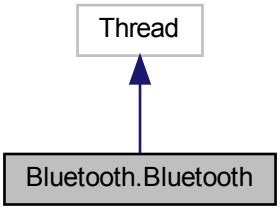
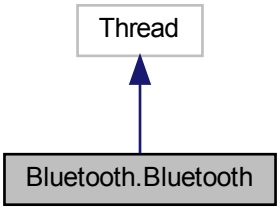


Diagrama de colaboración para Bluetooth.Bluetooth:



Métodos públicos

- def __init__ (self, bluetooth, servo)

  Esta clase se encarga de controlar las acciones de la cerradura por bluetooth.
- def dpad (self, pos)

  Por zona en la cual se presiona el bluedot, se abre o cierra la cerradura: Arrba Abre.
- def bluetooth_main (self)

  Escucha la entrada de BlueDot.
- def run (self)

  Ejecuta la instancia en un hilo de ejecución.

Atributos públicos

- bd
- servo

4.2.1. Descripción detallada

Definición en la línea 6 del archivo Bluetooth.py.

4.2.2. Documentación del constructor y destructor

4.2.2.1. __init__()

```
def Bluetooth.Bluetooth.__init__ (
              self,
              bluetooth,
              servo )
```

Esta clase se encarga de controlar las acciones de la cerradura por bluetooth.

Parámetros

bluetooth	Recibe una instancia de BlueDot
servo	Recibe una instancia de un Servomotor

Definición en la línea 7 del archivo Bluetooth.py.

Gráfico de llamadas para esta función:

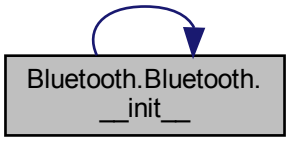
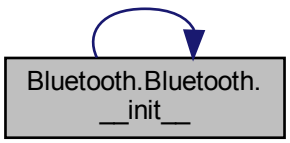


Gráfico de llamadas a esta función:



4.2.3. Documentación de las funciones miembro

4.2.3.1. bluetooth_main()

```
def Bluetooth.Bluetooth.bluetooth_main (
              self )
```

Escucha la entrada de BlueDot.

Definición en la línea 32 del archivo Bluetooth.py.

Gráfico de llamadas para esta función:

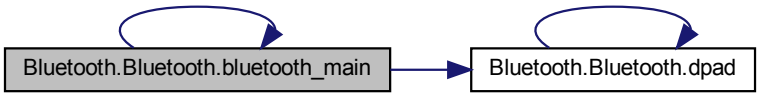
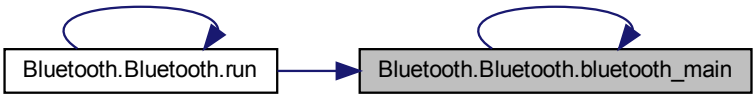


Gráfico de llamadas a esta función:



4.2.3.2. dpad()

```
def Bluetooth.Bluetooth.dpad (
              self,
              pos )
```

Por zona en la cual se presiona el bluedot, se abre o cierra la cerradura: Arrba Abre.

Abajo Cierra

Definición en la línea 17 del archivo Bluetooth.py.

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:

4.2.3.3. run()

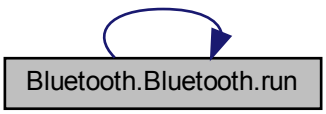
```
def Bluetooth.Bluetooth.run (
              self )
```

Ejecuta la instancia en un hilo de ejecución.

Definición en la línea 40 del archivo Bluetooth.py.

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



4.2.4. Documentación de los datos miembro

4.2.4.1. bd

`Bluetooth.Bluetooth.bd`

Definición en la línea 14 del archivo Bluetooth.py.

4.2.4.2. servo

`Bluetooth.Bluetooth.servo`

Definición en la línea 15 del archivo Bluetooth.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- bluetooth/Bluetooth.py

4.3. Referencia de la Clase cerradura_pwm.Cerradura.Cerradura

4.3.1. Descripción detallada

Definición en la línea 3 del archivo Cerradura.py.

4.3.2. Documentación del constructor y destructor

4.3.2.1. __init__()

`def cerradura_pwm.Cerradura.Cerradura.__init__`

Definición en la línea 4 del archivo Cerradura.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- cerradura_pwm/Cerradura.py

4.4. Referencia de la Clase core.audio_helpers.ConversationStream

Diagrama de herencias de core.audio_helpers.ConversationStream

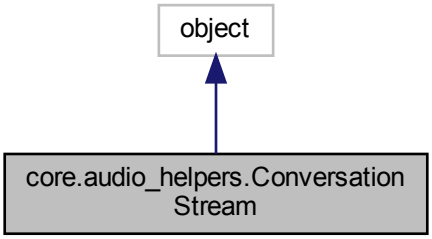
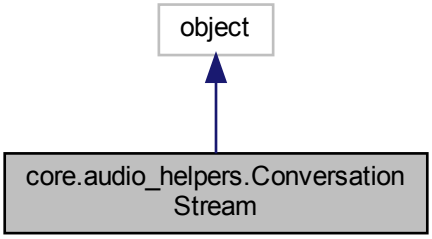


Diagrama de colaboración para core.audio_helpers.ConversationStream:



Métodos públicos

- def __init__ (self, source, sink, iter_size, sample_width)
- def start_recording (self)
- def stop_recording (self)
- def start_playback (self)
- def stop_playback (self)
- def recording (self)
- def playing (self)
- def volume_percentage (self)
- def volume_percentage (self, new_volume_percentage)
- def read (self, size)
- def write (self, buf)
- def close (self)
- def __iter__ (self)
- def sample_rate (self)

4.4.1. Descripción detallada

```
Audio stream that supports half-duplex conversation.

A conversation is the alternance of:
- a recording operation
- a playback operation

Excepted usage:

  For each conversation:
  - start_recording()
  - read() or iter()
  - stop_recording()
  - start_playback()
  - write()
  - stop_playback()

  When conversations are finished:
  - close()

Args:
  source: file-like stream object to read input audio bytes from.
  sink: file-like stream object to write output audio bytes to.
  iter_size: read size in bytes for each iteration.
  sample_width: size of a single sample in bytes.
```

Definición en la línea 238 del archivo audio_helpers.py.

4.4.2. Documentación del constructor y destructor

4.4.2.1. __init__()

```
def core.audio_helpers.ConversationStream.__init__ (
              self,
              source,
              sink,
              iter_size,
              sample_width )
```

Definición en la línea 264 del archivo audio_helpers.py.

4.4.3. Documentación de las funciones miembro

4.4.3.1. __iter__()

```
def core.audio_helpers.ConversationStream.__iter__ (
              self )
```

Returns a generator reading data from the stream.

Definición en la línea 334 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

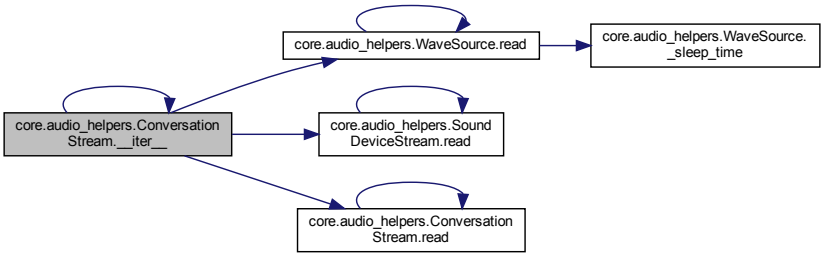
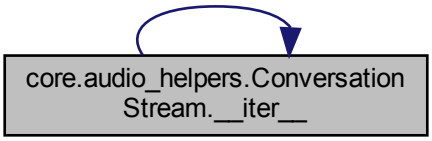


Gráfico de llamadas a esta función:



4.4.3.2. close()

```
def core.audio_helpers.ConversationStream.close (
              self )
```

Close source and sink.

Definición en la línea 329 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

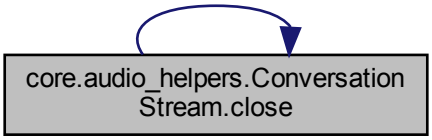
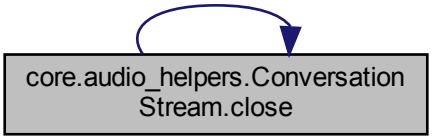


Gráfico de llamadas a esta función:



4.4.3.3. playing()

```
def core.audio_helpers.ConversationStream.playing (
              self )
```

Definición en la línea 304 del archivo audio_helpers.py.

4.4.3.4. read()

```
def core.audio_helpers.ConversationStream.read (
              self,
              size )
```

Read bytes from the source (if currently recording).

Definición en la línea 316 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

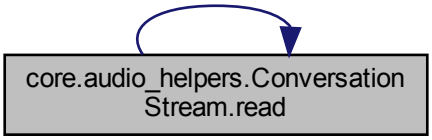
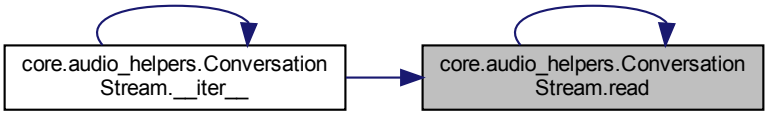


Gráfico de llamadas a esta función:



4.4.3.5. recording()

```
def core.audio_helpers.ConversationStream.recording (
              self )
```

Definición en la línea 300 del archivo audio_helpers.py.

4.4.3.6. sample_rate()

```
def core.audio_helpers.ConversationStream.sample_rate (
              self )
```

Definición en la línea 342 del archivo audio_helpers.py.

4.4.3.7. start_playback()

```
def core.audio_helpers.ConversationStream.start_playback (
              self )
```

Start playback to the audio sink.

Definición en la línea 288 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

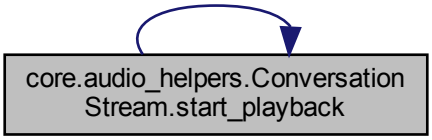
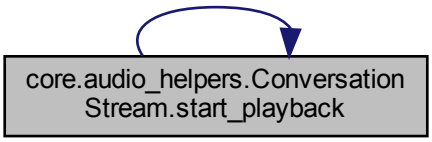


Gráfico de llamadas a esta función:



4.4.3.8. start_recording()

```
def core.audio_helpers.ConversationStream.start_recording (
              self )
```

Start recording from the audio source.

Definición en la línea 275 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

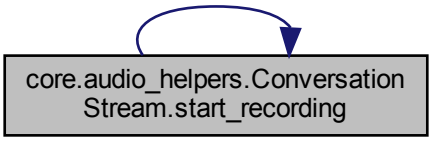
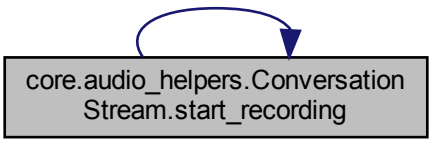


Gráfico de llamadas a esta función:



4.4.3.9. stop_playback()

```
def core.audio_helpers.ConversationStream.stop_playback (
              self )

Stop playback from the audio sink.
```

Definición en la línea 293 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

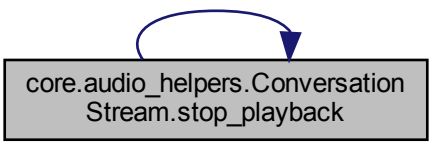
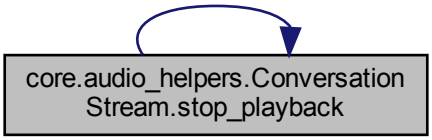


Gráfico de llamadas a esta función:



4.4.3.10. stop_recording()

```
def core.audio_helpers.ConversationStream.stop_recording (
              self )
```

Stop recording from the audio source.

Definición en la línea 281 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

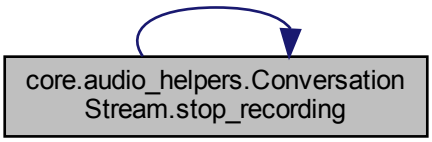
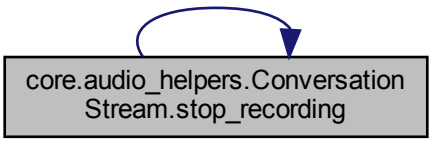


Gráfico de llamadas a esta función:



4.4.3.11. volume_percentage() [1/2]

```
def core.audio_helpers.ConversationStream.volume_percentage (
            self )

The current volume setting as an integer percentage (1-100).
```

Definición en la línea 308 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

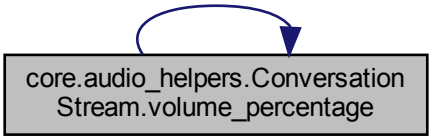
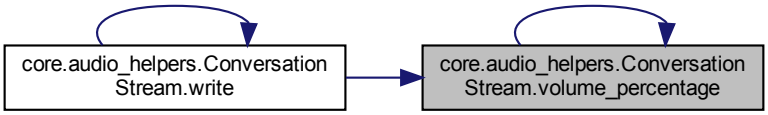


Gráfico de llamadas a esta función:

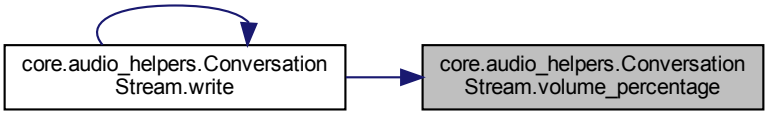


4.4.3.12. volume_percentage() [2/2]

```
def core.audio_helpers.ConversationStream.volume_percentage (
            self,
            new_volume_percentage )
```

Definición en la línea 313 del archivo audio_helpers.py.

Gráfico de llamadas a esta función:



4.4.3.13. write()

```
def core.audio_helpers.ConversationStream.write (
              self,
              buf )
```

Write bytes to the sink (if currently playing).

Definición en la línea 322 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

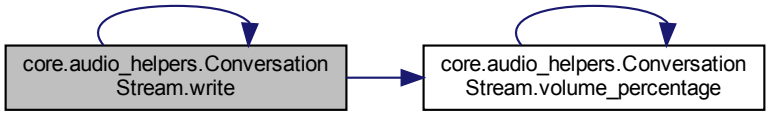
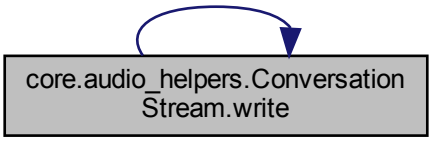


Gráfico de llamadas a esta función:



La documentación para esta clase fue generada a partir del siguiente fichero:

- core/audio_helpers.py

4.5. Referencia de la Clase core.Core.Core

Diagrama de herencias de core.Core.Core

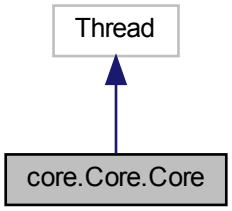
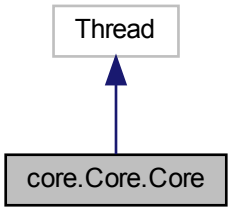


Diagrama de colaboración para core.Core.Core:



Métodos públicos

- def __init__ (self)

  Esta clase sirve como compositor e iniciador de cada sensor, inicia y decara los componestes que se usará para el embebido.
- def telebot_msg_handler (self)

  Este método es el encargado de enviar y recibir mensajes del bot de telegram.
- def run (self)

  Aquí se llaman y se ejecuta en cada hilo los procesos de cada sensor.
- def abrir_cerradura (self)

  Abre la cerradura.
- def cerrar_cerradura (self)

  Cierra la cerradura.
- def cambiar_edo (self)

  Cierra o abre la serradura segun su estado actual.
- def cerradura (self)

  Escucha del interruptor para cambiar el estado de la cerradura.

Atributos públicos

- shutdown_flag
- bd
- status_led_ok
- status_led_error
- led_gen_status
- servo_motor
- gas_sensor
- interruptor_metal
- interruptor_plastico
- distancia_sensor
- bluetooth
- tele_bot
- lector
- mod_gas

4.5.1. Descripción detallada

Definición en la línea 44 del archivo Core.py.

4.5.2. Documentación del constructor y destructor

4.5.2.1. __init__()

```
def core.Core.Core.__init__ (
              self )
```

Esta clase sirve como compositor e iniciador de cada sensor, inicia y decara los componestes que se usará para el embebido.

Definición en la línea 45 del archivo Core.py.

Gráfico de llamadas para esta función:

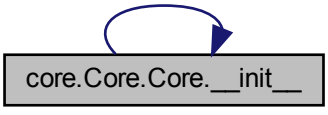
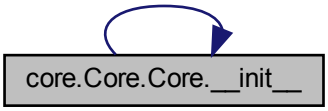


Gráfico de llamadas a esta función:



4.5.3. Documentación de las funciones miembro

4.5.3.1. abrir_cerradura()

```
def core.Core.Core.abrir_cerradura (
              self )
```

Abre la cerradura.

Definición en la línea 121 del archivo Core.py.

Gráfico de llamadas para esta función:

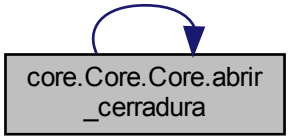
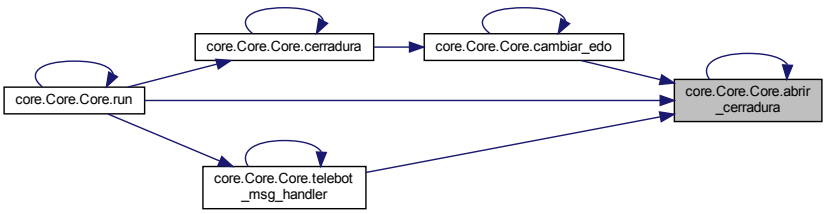


Gráfico de llamadas a esta función:



4.5.3.2. cambiar_edo()

```
def core.Core.Core.cambiar_edo (
              self )
```

Cierra o abre la serradura segun su estado actual.

Definición en la línea 133 del archivo Core.py.

Gráfico de llamadas para esta función:

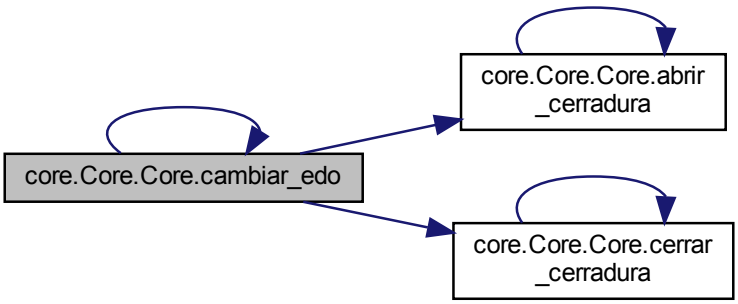
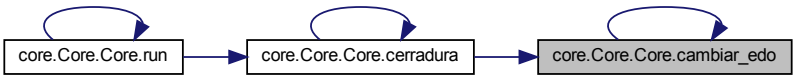


Gráfico de llamadas a esta función:



4.5.3.3. cerradura()

```
def core.Core.Core.cerradura (
              self )
```

Escucha del interruptor para cambiar el estado de la cerradura.

Definición en la línea 142 del archivo Core.py.

Gráfico de llamadas para esta función:

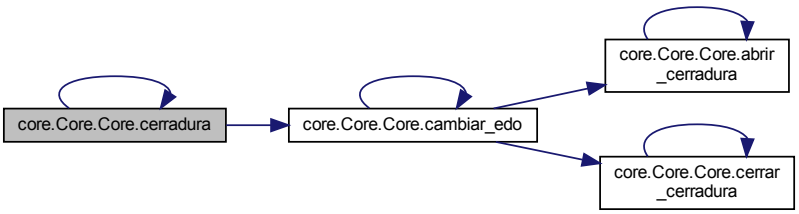
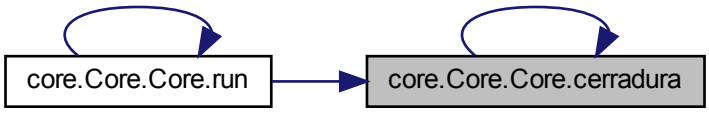


Gráfico de llamadas a esta función:



4.5.3.4. cerrar_cerradura()

```
def core.Core.Core.cerrar_cerradura (
              self )
```

Cierra la cerradura.

Definición en la línea 127 del archivo Core.py.

Gráfico de llamadas para esta función:

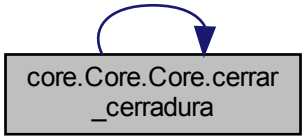
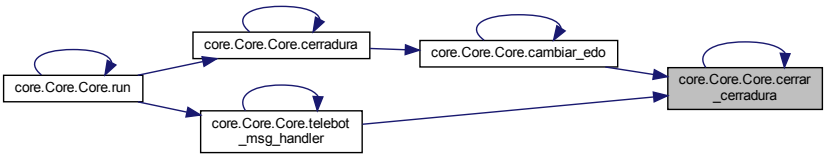


Gráfico de llamadas a esta función:



4.5.3.5. run()

```
def core.Core.Core.run (
              self )
```

Aquí se llaman y se ejecuta en cada hilo los procesos de cada sensor.

Definición en la línea 102 del archivo Core.py.

Gráfico de llamadas para esta función:

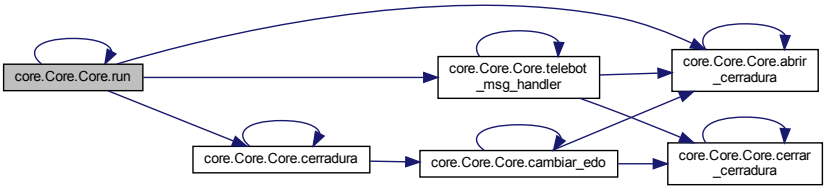
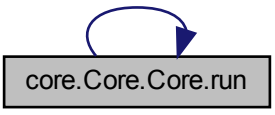


Gráfico de llamadas a esta función:



4.5.3.6. telebot_msg_handler()

```
def core.Core.Core.telebot_msg_handler (
              self )
```

Este método es el encargado de enviar y recibir mensajes del bot de telegram.

Definición en la línea 73 del archivo Core.py.

Gráfico de llamadas para esta función:

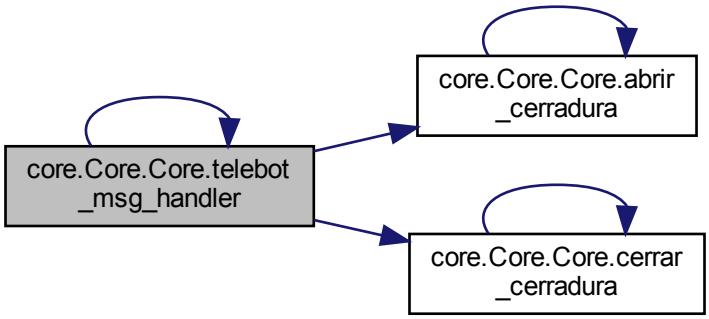
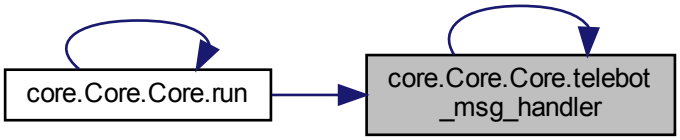


Gráfico de llamadas a esta función:



4.5.4. Documentación de los datos miembro

4.5.4.1. bd

```
core.Core.Core.bd
```

Definición en la línea 52 del archivo Core.py.

4.5.4.2. bluetooth

`core.Core.Core.bluetooth`

Definición en la línea 64 del archivo Core.py.

4.5.4.3. distancia_sensor

`core.Core.Core.distancia_sensor`

Definición en la línea 61 del archivo Core.py.

4.5.4.4. gas_sensor

`core.Core.Core.gas_sensor`

Definición en la línea 58 del archivo Core.py.

4.5.4.5. interruptor_metal

`core.Core.Core.interruptor_metal`

Definición en la línea 59 del archivo Core.py.

4.5.4.6. interruptor_plastico

`core.Core.Core.interruptor_plastico`

Definición en la línea 60 del archivo Core.py.

4.5.4.7. lector

`core.Core.Core.lector`

Definición en la línea 67 del archivo Core.py.

4.5.4.8. led_gen_status

`core.Core.Core.led_gen_status`

Definición en la línea 55 del archivo Core.py.

4.5.4.9. mod_gas

`core.Core.Core.mod_gas`

Definición en la línea 69 del archivo Core.py.

4.5.4.10. servo_motor

`core.Core.Core.servo_motor`

Definición en la línea 56 del archivo Core.py.

4.5.4.11. shutdown_flag

`core.Core.Core.shutdown_flag`

Definición en la línea 51 del archivo Core.py.

4.5.4.12. status_led_error

`core.Core.Core.status_led_error`

Definición en la línea 54 del archivo Core.py.

4.5.4.13. status_led_ok

`core.Core.Core.status_led_ok`

Definición en la línea 53 del archivo Core.py.

4.5.4.14. tele_bot

`core.Core.Core.tele_bot`

Definición en la línea 65 del archivo Core.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- core/Core.py

4.6. Referencia de la Clase core.device_helpers.DeviceRequestHandler

Diagrama de herencias de core.device_helpers.DeviceRequestHandler

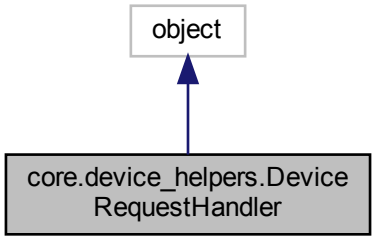
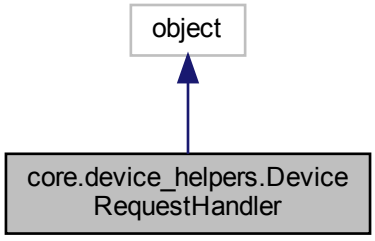


Diagrama de colaboración para core.device_helpers.DeviceRequestHandler:



Métodos públicos

- def __init__ (self, device_id)
- def __call__ (self, device_request)
- def command (self, intent)
- def submit_commands (self, devices, execution)
- def dispatch_command (self, command, params=None)

Atributos públicos

- executor
- device_id
- handlers

4.6.1. Descripción detallada

```
Asynchronous dispatcher for Device actions commands.

Dispatch commands to the given device handlers.

Args:
  device_id: device id to match command against

Example:
  # Use as as decorator to register handler.
  device_handler = DeviceRequestHandler('my-device')
  @device_handler.command('INTENT_NAME')
  def handler(param):
      pass
```

Definición en la línea 29 del archivo device_helpers.py.

4.6.2. Documentación del constructor y destructor

4.6.2.1. __init__()

```
def core.device_helpers.DeviceRequestHandler.__init__ (
              self,
              device_id )
```

Definición en la línea 45 del archivo device_helpers.py.

4.6.3. Documentación de las funciones miembro

4.6.3.1. __call__()

```
def core.device_helpers.DeviceRequestHandler.__call__ (
              self,
              device_request )
```

```
Handle incoming device request.

Returns: List of concurrent.futures for each command execution.
```

Definición en la línea 50 del archivo device_helpers.py.

Gráfico de llamadas para esta función:

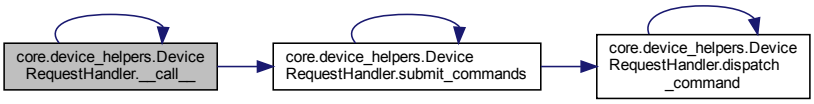


Gráfico de llamadas a esta función:

4.6.3.2. command()

```
def core.device_helpers.DeviceRequestHandler.command (
              self,
              intent )
```

```
Register a device action handlers.
```

Definición en la línea 63 del archivo device_helpers.py.

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:

4.6.3.3. dispatch_command()

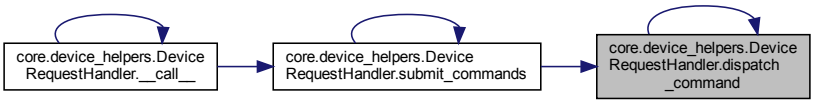
```
def core.device_helpers.DeviceRequestHandler.dispatch_command (
              self,
              command,
              params = None )
```

Dispatch device commands to the appropriate handler.

Definición en la línea 90 del archivo device_helpers.py.

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



4.6.3.4. submit_commands()

```
def core.device_helpers.DeviceRequestHandler.submit_commands (
              self,
              devices,
              execution )
```

```
Submit device command executions.

Returns: a list of concurrent.futures for scheduled executions.
```

Definición en la línea 69 del archivo device_helpers.py.

Gráfico de llamadas para esta función:

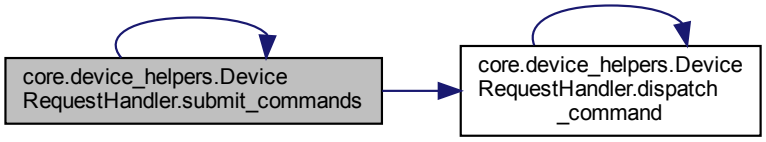
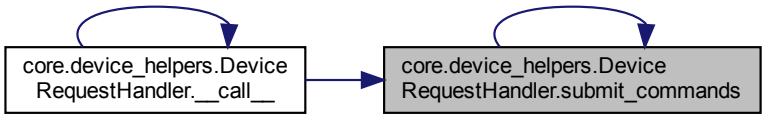


Gráfico de llamadas a esta función:



4.6.4. Documentación de los datos miembro

4.6.4.1. device_id

`core.device_helpers.DeviceRequestHandler.device_id`

Definición en la línea 47 del archivo device_helpers.py.

4.6.4.2. executor

`core.device_helpers.DeviceRequestHandler.executor`

Definición en la línea 46 del archivo device_helpers.py.

4.6.4.3. handlers

`core.device_helpers.DeviceRequestHandler.handlers`

Definición en la línea 48 del archivo device_helpers.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- core/device_helpers.py

4.7. Referencia de la Clase Distancia.Distancia

def **init**(self, dist_sensor, led_status, tele_bot, database, shutdown_flag): Esta Clase Se encarga de instanciar y controlar el sensor de distancia, avisa al usuario si alguien esta cerca de la cerradura.

Diagrama de herencias de Distancia.Distancia

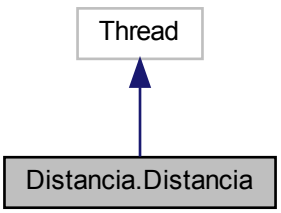
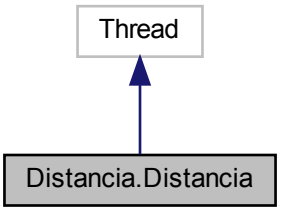


Diagrama de colaboración para Distancia.Distancia:



Métodos públicos

- def sensor_listener (self)

  Escucha del sensor de distancia que envia mensaje por telegram si alguien esta cerca de la puerta.
- def run (self)

  Ejecuta el escucha del sensor de distancia en un hilo de ejecución.

Atributos públicos estáticos

- shutdown_flag
- dist_sensor
- led_status
- tele_bot

4.7.1. Descripción detallada

def **init**(self, dist_sensor, led_status, tele_bot, database, shutdown_flag): Esta Clase Se encarga de instanciar y controlar el sensor de distancia, avisa al usuario si alguien esta cerca de la cerradura.

Parámetros

dist_sensor	Instancia del sensor de distancica
led_status	Instancua del led de estado
tele_bot	Instancia del bot de telegram
database	Instancia del controlador de la BD
shutdown_flag	Bandera de señales

Definición en la línea 8 del archivo Distancia.py.

4.7.2. Documentación de las funciones miembro

4.7.2.1. run()

```
def Distancia.Distancia.run (
              self )
```

Ejecuta el escucha del sensor de distancia en un hilo de ejecución.

Definición en la línea 44 del archivo Distancia.py.

Gráfico de llamadas para esta función:

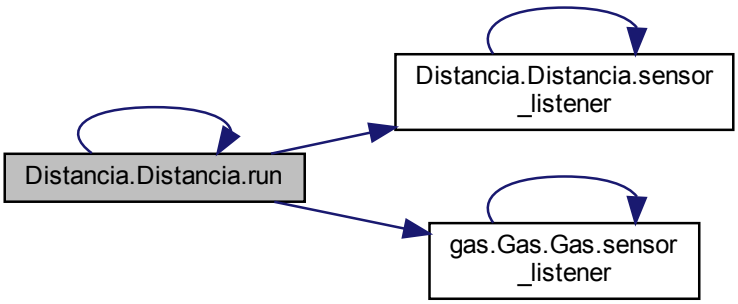


Gráfico de llamadas a esta función:

4.7.2.2. sensor_listener()

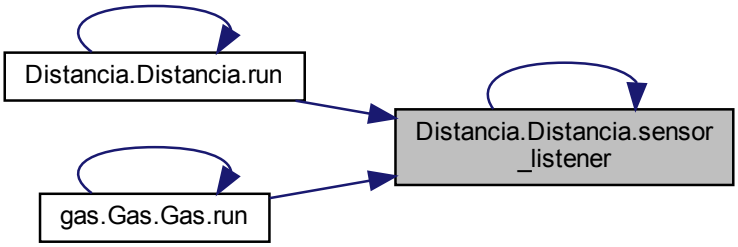
```
def Distancia.Distancia.sensor_listener (
              self )
```

Escucha del sensor de distancia que envia mensaje por telegram si alguien esta cerca de la puerta.

Definición en la línea 27 del archivo Distancia.py.

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



4.7.3. Documentación de los datos miembro

4.7.3.1. dist_sensor

`Distancia.Distancia.dist_sensor [static]`

Definición en la línea 22 del archivo Distancia.py.

4.7.3.2. led_status

`Distancia.Distancia.led_status [static]`

Definición en la línea 23 del archivo Distancia.py.

4.7.3.3. shutdown_flag

```
Distancia.Distancia.shutdown_flag  [static]
```

Definición en la línea 20 del archivo Distancia.py.

4.7.3.4. tele_bot

```
Distancia.Distancia.tele_bot  [static]
```

Definición en la línea 24 del archivo Distancia.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- distancia/Distancia.py

4.8. Referencia de la Clase gas.Gas.Gas

Diagrama de herencias de gas.Gas.Gas

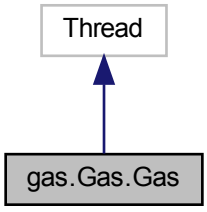
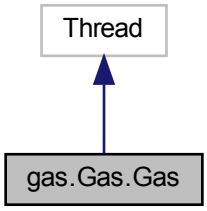


Diagrama de colaboración para gas.Gas.Gas:



Métodos públicos

- def __init__ (self, gas_sensor, led_status, tele_bot, database, shutdown_flag)

  Esta clase se encarga de enviar y de controlar el sensor de gas.
- def sensor_listener (self)

  Rutina en bucle que esta a la escucha del sensor, la envia un mensaje a los usuarios dados de alta en la BD mediante telegram.
- def run (self)

  Ejecuta en un hilo el sensor de gas.

Atributos públicos

- shutdown_flag
- gas_sensor
- led_status
- tele_bot

4.8.1. Descripción detallada

Definición en la línea 8 del archivo Gas.py.

4.8.2. Documentación del constructor y destructor

4.8.2.1. __init__()

```
def gas.Gas.Gas.__init__ (
              self,
              gas_sensor,
              led_status,
              tele_bot,
              database,
              shutdown_flag )
```

Esta clase se encarga de enviar y de controlar el sensor de gas.

Parámetros

gas_sensor	El sensor de gas
led_status	El led a usar
tele_bot	El bot de telegram
database	El manejador de la base de datos
shutdown_flag	La bandera de sañales

Devuelve

Una instancia del sensor de gas

Definición en la línea 9 del archivo Gas.py.

Gráfico de llamadas para esta función:

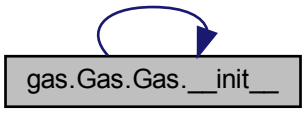
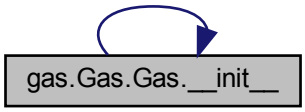


Gráfico de llamadas a esta función:



4.8.3. Documentación de las funciones miembro

4.8.3.1. run()

```
def gas.Gas.Gas.run (
              self )
```

Ejecuta en un hilo el sensor de gas.

Definición en la línea 57 del archivo Gas.py.

Gráfico de llamadas para esta función:

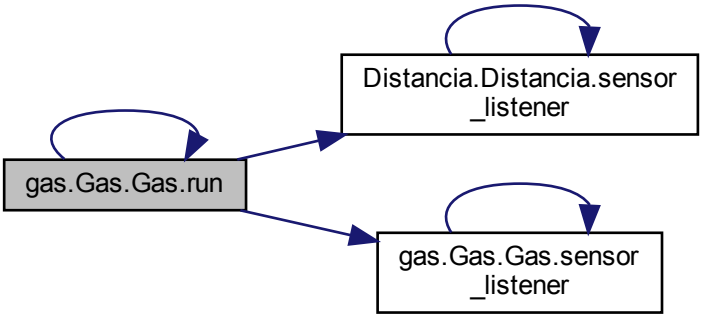
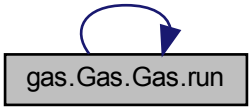


Gráfico de llamadas a esta función:



4.8.3.2. sensor_listener()

```
def gas.Gas.Gas.sensor_listener (
              self )
```

Rutina en bucle que esta a la escucha del sensor, la envia un mensaje a los usuarios dados de alta en la BD mediante telegram.

También enciende y apaga un led si hay gas, humo u gasolina cerca.

Definición en la línea 30 del archivo Gas.py.

Gráfico de llamadas para esta función:

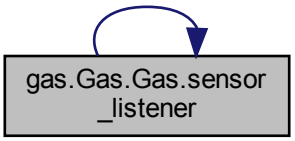
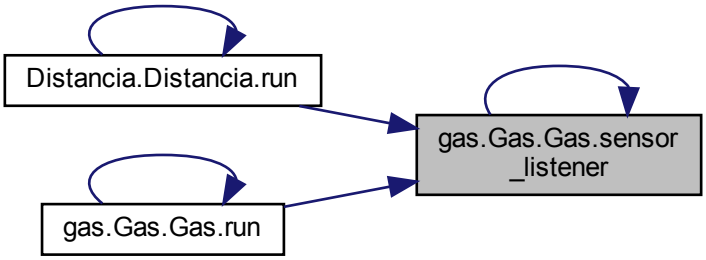


Gráfico de llamadas a esta función:



4.8.4. Documentación de los datos miembro

4.8.4.1. gas_sensor

`gas.Gas.Gas.gas_sensor`

Definición en la línea 24 del archivo Gas.py.

4.8.4.2. led_status

`gas.Gas.Gas.led_status`

Definición en la línea 25 del archivo Gas.py.

4.8.4.3. shutdown_flag

`gas.Gas.Gas.shutdown_flag`

Definición en la línea 22 del archivo Gas.py.

4.8.4.4. tele_bot

`gas.Gas.Gas.tele_bot`

Definición en la línea 26 del archivo Gas.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- gas/Gas.py

4.9. Referencia de la Clase Lector.Lector

Diagrama de herencias de Lector.Lector

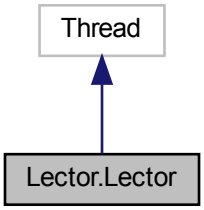
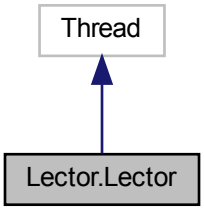


Diagrama de colaboración para Lector.Lector:



Métodos públicos

- def __init__ (self, led_status_r, led_status_g, tele_bot, servo, tty=None)

  Esta clase se encarga de realizar las acciones del lector de huella digital como:
- def guardar_huella (self)

  metodo para salvar huella en la memoria del lector
- def verificar_huella (self)

  Metodo que verifica la existencia de la huella.
- def controlar_puerta (self)

  Metodo que controla la puerta si se coincide la huella con alguna de la BD del lector.
- def run (self)

  Ejecuta el escucha del lector en un hilo de ejecucion.

Atributos públicos

- tty
- finger
- led_status_r
- led_status_g
- tele_bot
- servo

4.9.1. Descripción detallada

Definición en la línea 7 del archivo Lector.py.

4.9.2. Documentación del constructor y destructor

4.9.2.1. __init__()

```
def Lector.Lector.__init__ (
              self,
              led_status_r,
              led_status_g,
              tele_bot,
              servo,
              tty = None )
```

Esta clase se encarga de realizar las acciones del lector de huella digital como:

- Dar de alta la huella
- Borrar Huella
- Verificar existencia de la huella
- Abrir cerradura si la huella coincide con la base de datos

Parámetros

led_status_r	Instancia de led de estado roja
led_status_g	Instancia de led de estado verde
tele_bot	Bot de telegram
servo	Servomotor a controlar
tty	Puerto serial USB de lector de huella, por defecto es /dev/ttyUSBX

Definición en la línea 8 del archivo Lector.py.

Gráfico de llamadas para esta función:

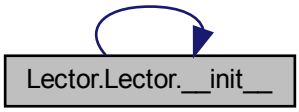
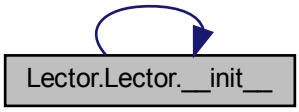


Gráfico de llamadas a esta función:



4.9.3. Documentación de las funciones miembro

4.9.3.1. controlar_puerta()

```
def Lector.Lector.controlar_puerta (
            self )
```

Metodo que controla la puerta si se coincide la huella con alguna de la BD del lector.

Definición en la línea 94 del archivo Lector.py.

Gráfico de llamadas para esta función:

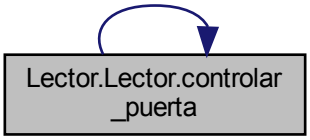
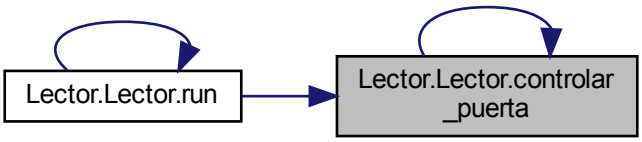


Gráfico de llamadas a esta función:



4.9.3.2. guardar_huella()

```
def Lector.Lector.guardar_huella (
              self )
```

metodo para salvar huella en la memoria del lector

Definición en la línea 36 del archivo Lector.py.

Gráfico de llamadas para esta función:

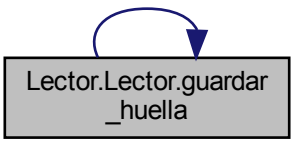
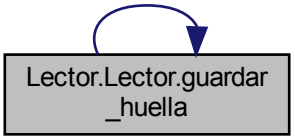


Gráfico de llamadas a esta función:



4.9.3.3. run()

```
def Lector.Lector.run (
              self )
```

Ejecuta el escucha del lector en un hilo de ejecucion.

Definición en la línea 123 del archivo Lector.py.

Gráfico de llamadas para esta función:

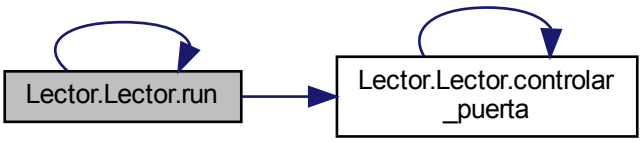
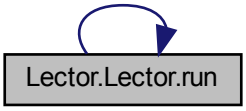


Gráfico de llamadas a esta función:



4.9.3.4. verificar_huella()

```
def Lector.Lector.verificar_huella (
              self )
```

Metodo que verifica la existencia de la huella.

Definición en la línea 73 del archivo Lector.py.

Gráfico de llamadas para esta función:

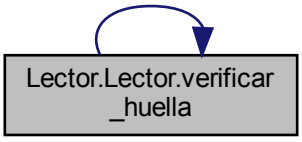
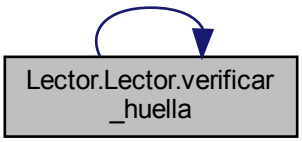


Gráfico de llamadas a esta función:



4.9.4. Documentación de los datos miembro

4.9.4.1. finger

```
Lector.Lector.finger
```

Definición en la línea 29 del archivo Lector.py.

4.9.4.2. led_status_g

`Lector.Lector.led_status_g`

Definición en la línea 31 del archivo Lector.py.

4.9.4.3. led_status_r

`Lector.Lector.led_status_r`

Definición en la línea 30 del archivo Lector.py.

4.9.4.4. servo

`Lector.Lector.servo`

Definición en la línea 33 del archivo Lector.py.

4.9.4.5. tele_bot

`Lector.Lector.tele_bot`

Definición en la línea 32 del archivo Lector.py.

4.9.4.6. tty

`Lector.Lector.tty`

Definición en la línea 26 del archivo Lector.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- lector_huella/Lector.py

4.10. Referencia de la Clase core.Assistant.SampleAssistant

Diagrama de herencias de core.Assistant.SampleAssistant

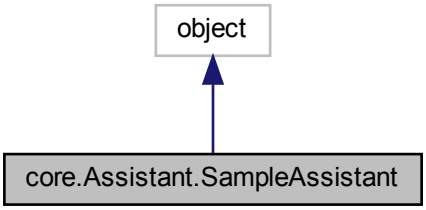
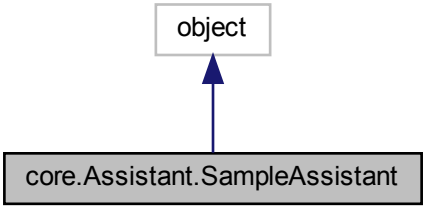


Diagrama de colaboración para core.Assistant.SampleAssistant:



Métodos públicos

- def __init__ (self, language_code, device_model_id, device_id, conversation_stream, display, channel, deadline_sec, device_handler)
- def __enter__ (self)
- def __exit__ (self, etype, e, traceback)
- def is_grpc_error_unavailable (e)
- def assist (self)
- def gen_assist_requests (self)

Atributos públicos

- language_code
- device_model_id
- device_id
- conversation_stream
- display
- conversation_state
- is_new_conversation
- assistant
- deadline
- device_handler

Atributos públicos estáticos

- retry = retry_if_exception(is_grpc_error_unavailable))

4.10.1. Descripción detallada

```
Sample Assistant that supports conversations and device actions.

Args:
  device_model_id: identifier of the device model.
  device_id: identifier of the registered device instance.
  conversation_stream(ConversationStream): audio stream
    for recording query and playing back assistant answer.
  channel: authorized gRPC channel for connection to the
    Google Assistant API.
  deadline_sec: gRPC deadline in seconds for Google Assistant API call.
  device_handler: callback for device actions.
```

Definición en la línea 45 del archivo Assistant.py.

4.10.2. Documentación del constructor y destructor

4.10.2.1. __init__()

```
def core.Assistant.SampleAssistant.__init__ (
              self,
              language_code,
              device_model_id,
              device_id,
              conversation_stream,
              display,
              channel,
              deadline_sec,
              device_handler )
```

Definición en la línea 59 del archivo Assistant.py.

4.10.3. Documentación de las funciones miembro

4.10.3.1. __enter__()

```
def core.Assistant.SampleAssistant.__enter__ (
              self )
```

Definición en la línea 86 del archivo Assistant.py.

4.10.3.2. __exit__()

```
def core.Assistant.SampleAssistant.__exit__ (
              self,
              etype,
              e,
              traceback )
```

Definición en la línea 89 del archivo Assistant.py.

4.10.3.3. assist()

```
def core.Assistant.SampleAssistant.assist (
              self )
```

```
Send a voice request to the Assistant and playback the response.

Returns: True if conversation should continue.
```

Definición en la línea 103 del archivo Assistant.py.

Gráfico de llamadas para esta función:

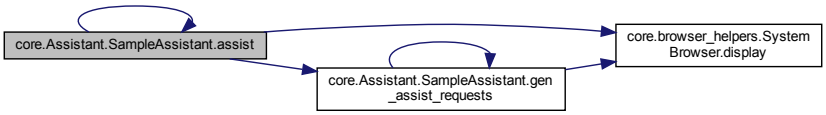
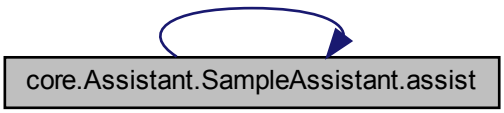


Gráfico de llamadas a esta función:



4.10.3.4. gen_assist_requests()

```
def core.Assistant.SampleAssistant.gen_assist_requests (
              self )

Yields: AssistRequest messages to send to the API.
```

Definición en la línea 171 del archivo Assistant.py.

Gráfico de llamadas para esta función:

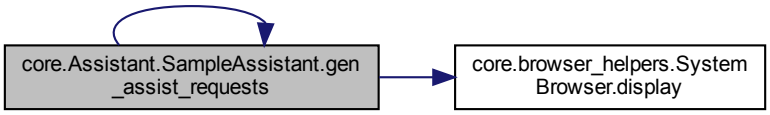
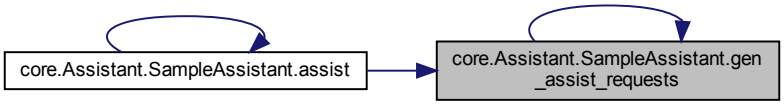


Gráfico de llamadas a esta función:



4.10.3.5. is_grpc_error_unavailable()

```
def core.Assistant.SampleAssistant.is_grpc_error_unavailable (
              e )
```

Definición en la línea 94 del archivo Assistant.py.

4.10.4. Documentación de los datos miembro

4.10.4.1. assistant

```
core.Assistant.SampleAssistant.assistant
```

Definición en la línea 79 del archivo Assistant.py.

4.10.4.2. conversation_state

`core.Assistant.SampleAssistant.conversation_state`

Definición en la línea 74 del archivo Assistant.py.

4.10.4.3. conversation_stream

`core.Assistant.SampleAssistant.conversation_stream`

Definición en la línea 65 del archivo Assistant.py.

4.10.4.4. deadline

`core.Assistant.SampleAssistant.deadline`

Definición en la línea 82 del archivo Assistant.py.

4.10.4.5. device_handler

`core.Assistant.SampleAssistant.device_handler`

Definición en la línea 84 del archivo Assistant.py.

4.10.4.6. device_id

`core.Assistant.SampleAssistant.device_id`

Definición en la línea 64 del archivo Assistant.py.

4.10.4.7. device_model_id

`core.Assistant.SampleAssistant.device_model_id`

Definición en la línea 63 del archivo Assistant.py.

4.10.4.8. display

`core.Assistant.SampleAssistant.display`

Definición en la línea 66 del archivo Assistant.py.

4.10.4.9. is_new_conversation

`core.Assistant.SampleAssistant.is_new_conversation`

Definición en la línea 76 del archivo Assistant.py.

4.10.4.10. language_code

`core.Assistant.SampleAssistant.language_code`

Definición en la línea 62 del archivo Assistant.py.

4.10.4.11. retry

`core.Assistant.SampleAssistant.retry = retry_if_exception(is_grpc_error_unavailable))  [static]`

Definición en la línea 102 del archivo Assistant.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- core/Assistant.py

4.11. Referencia de la Clase core.audio_helpers.SoundDeviceStream

Diagrama de herencias de core.audio_helpers.SoundDeviceStream

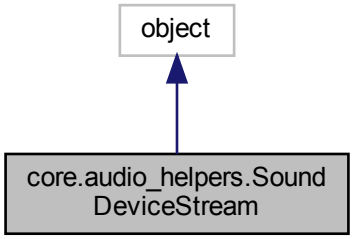
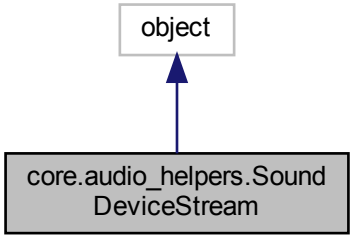


Diagrama de colaboración para core.audio_helpers.SoundDeviceStream:



Métodos públicos

- def __init__ (self, sample_rate, sample_width, block_size, flush_size)
- def read (self, size)
- def write (self, buf)
- def flush (self)
- def start (self)
- def stop (self)
- def close (self)
- def sample_rate (self)

4.11.1. Descripción detallada

```
Audio stream based on an underlying sound device.

It can be used as an audio source (read) and a audio sink (write).

Args:
  sample_rate: sample rate in hertz.
  sample_width: size of a single sample in bytes.
  block_size: size in bytes of each read and write operation.
  flush_size: size in bytes of silence data written during flush operation.
```

Definición en la línea 172 del archivo audio_helpers.py.

4.11.2. Documentación del constructor y destructor

4.11.2.1. __init__()

```
def core.audio_helpers.SoundDeviceStream.__init__ (
              self,
              sample_rate,
              sample_width,
              block_size,
              flush_size )
```

Definición en la línea 183 del archivo audio_helpers.py.

4.11.3. Documentación de las funciones miembro

4.11.3.1. close()

```
def core.audio_helpers.SoundDeviceStream.close (
              self )


Close the underlying stream and audio interface.
```

Definición en la línea 226 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

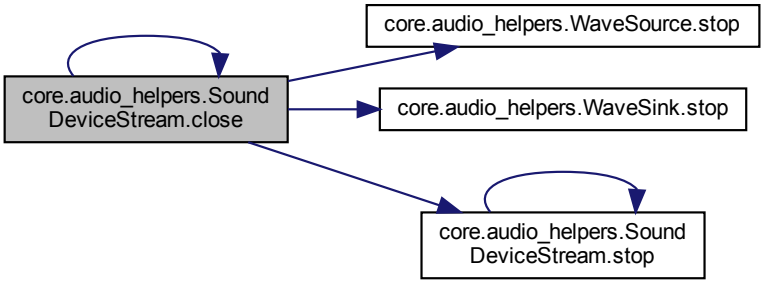


Gráfico de llamadas a esta función:

4.11.3.2. flush()

```
def core.audio_helpers.SoundDeviceStream.flush (
              self )
```

Definición en la línea 212 del archivo audio_helpers.py.

4.11.3.3. read()

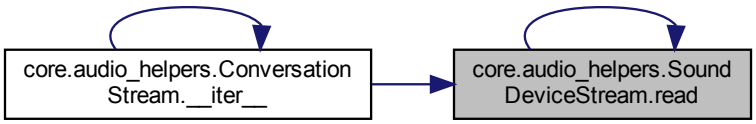
```
def core.audio_helpers.SoundDeviceStream.read (
              self,
              size )
```

Read bytes from the stream.

Definición en la línea 196 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



4.11.3.4. sample_rate()

```
def core.audio_helpers.SoundDeviceStream.sample_rate (
              self )
```

Definición en la línea 234 del archivo audio_helpers.py.

4.11.3.5. start()

```
def core.audio_helpers.SoundDeviceStream.start (
              self )
```

Start the underlying stream.

Definición en la línea 216 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:

4.11.3.6. stop()

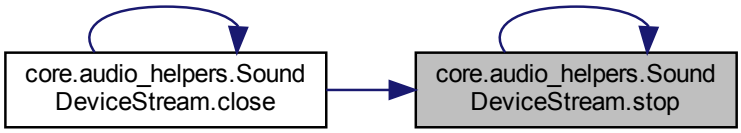
```
def core.audio_helpers.SoundDeviceStream.stop (
              self )

Stop the underlying stream.
```

Definición en la línea 221 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:



4.11.3.7. write()

```
def core.audio_helpers.SoundDeviceStream.write (
              self,
              buf )
```

Write bytes to the stream.

Definición en la línea 204 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

Gráfico de llamadas a esta función:

La documentación para esta clase fue generada a partir del siguiente fichero:

- core/audio_helpers.py

4.12. Referencia de la Clase core.browser_helpers.SystemBrowser

Diagrama de herencias de core.browser_helpers.SystemBrowser

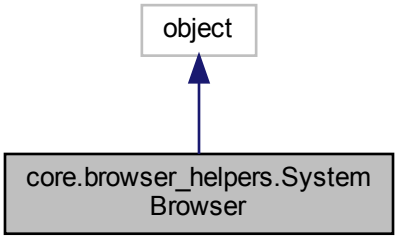
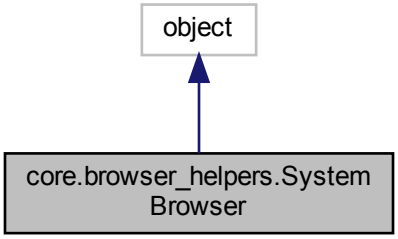


Diagrama de colaboración para core.browser_helpers.SystemBrowser:



Métodos públicos

- def __init__ (self)
- def display (self, html)

Atributos públicos

- tempdir
- filename

4.12.1. Descripción detallada

Definición en la línea 22 del archivo browser_helpers.py.

4.12.2. Documentación del constructor y destructor

4.12.2.1. __init__()

```
def core.browser_helpers.SystemBrowser.__init__ (
              self )
```

Definición en la línea 23 del archivo browser_helpers.py.

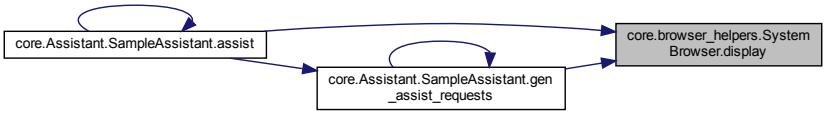
4.12.3. Documentación de las funciones miembro

4.12.3.1. display()

```
def core.browser_helpers.SystemBrowser.display (
              self,
              html )
```

Definición en la línea 27 del archivo browser_helpers.py.

Gráfico de llamadas a esta función:



4.12.4. Documentación de los datos miembro

4.12.4.1. filename

```
core.browser_helpers.SystemBrowser.filename
```

Definición en la línea 25 del archivo browser_helpers.py.

4.12.4.2. tempdir

`core.browser_helpers.SystemBrowser.tempdir`

Definición en la línea 24 del archivo browser_helpers.py.

La documentación para esta clase fue generada a partir del siguiente fichero:

- core/browser_helpers.py

4.13. Referencia de la Clase core.audio_helpers.WaveSink

Diagrama de herencias de core.audio_helpers.WaveSink

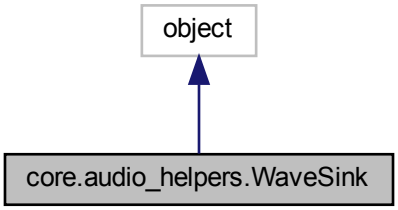
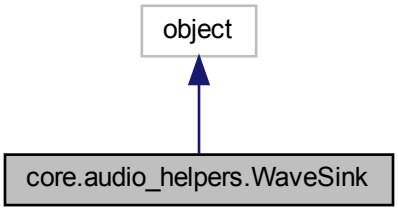


Diagrama de colaboración para core.audio_helpers.WaveSink:



Métodos públicos

- def __init__ (self, fp, sample_rate, sample_width)
- def write (self, data)
- def close (self)
- def start (self)
- def stop (self)
- def flush (self)

4.13.1. Descripción detallada

```
Audio sink that writes audio data to a WAV file.

Args:
  fp: file-like stream object to write data to.
  sample_rate: sample rate in hertz.
  sample_width: size of a single sample in bytes.
```

Definición en la línea 134 del archivo audio_helpers.py.

4.13.2. Documentación del constructor y destructor

4.13.2.1. __init__()

```
def core.audio_helpers.WaveSink.__init__ (
              self,
              fp,
              sample_rate,
              sample_width )
```

Definición en la línea 142 del archivo audio_helpers.py.

4.13.3. Documentación de las funciones miembro

4.13.3.1. close()

```
def core.audio_helpers.WaveSink.close (
              self )

Close the underlying stream.
```

Definición en la línea 157 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

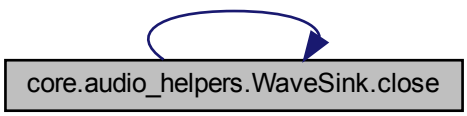
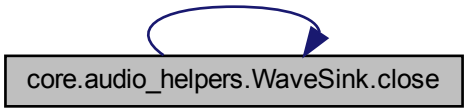


Gráfico de llamadas a esta función:



4.13.3.2. flush()

```
def core.audio_helpers.WaveSink.flush (
              self )
```

Definición en la línea 168 del archivo audio_helpers.py.

4.13.3.3. start()

```
def core.audio_helpers.WaveSink.start (
              self )
```

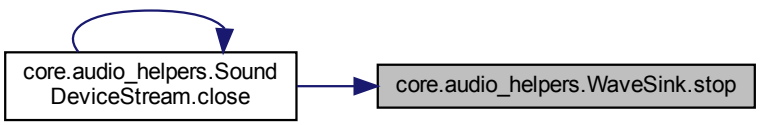
Definición en la línea 162 del archivo audio_helpers.py.

4.13.3.4. stop()

```
def core.audio_helpers.WaveSink.stop (
              self )
```

Definición en la línea 165 del archivo audio_helpers.py.

Gráfico de llamadas a esta función:



4.13.3.5. write()

```
def core.audio_helpers.WaveSink.write (
              self,
              data )


Write bytes to the stream.

Args:
  data: frame data to write.
```

Definición en la línea 149 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

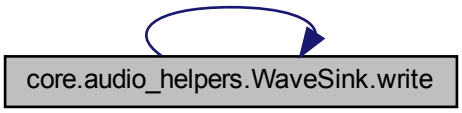
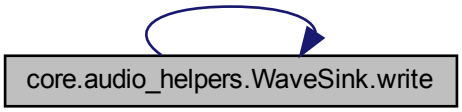


Gráfico de llamadas a esta función:



La documentación para esta clase fue generada a partir del siguiente fichero:

- core/audio_helpers.py

4.14. Referencia de la Clase core.audio_helpers.WaveSource

Diagrama de herencias de core.audio_helpers.WaveSource

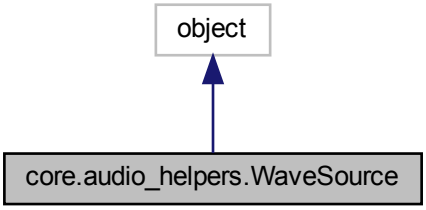
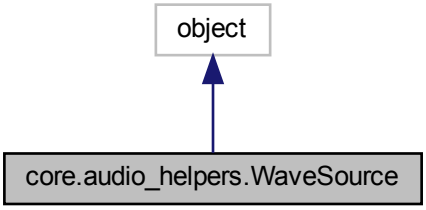


Diagrama de colaboración para core.audio_helpers.WaveSource:



Métodos públicos

- def __init__ (self, fp, sample_rate, sample_width)
- def read (self, size)
- def close (self)
- def start (self)
- def stop (self)
- def sample_rate (self)

4.14.1. Descripción detallada

```
Audio source that reads audio data from a WAV file.

Reads are throttled to emulate the given sample rate and silence
is returned when the end of the file is reached.

Args:
  fp: file-like stream object to read from.
  sample_rate: sample rate in hertz.
  sample_width: size of a single sample in bytes.
```

Definición en la línea 69 del archivo audio_helpers.py.

4.14.2. Documentación del constructor y destructor

4.14.2.1. __init__()

```
def core.audio_helpers.WaveSource.__init__ (
              self,
              fp,
              sample_rate,
              sample_width )
```

Definición en la línea 80 del archivo audio_helpers.py.

4.14.3. Documentación de las funciones miembro

4.14.3.1. close()

```
def core.audio_helpers.WaveSource.close (
              self )

Close the underlying stream.
```

Definición en la línea 112 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

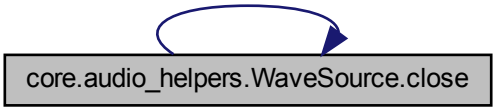
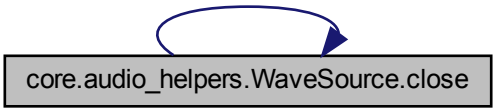


Gráfico de llamadas a esta función:



4.14.3.2. read()

```
def core.audio_helpers.WaveSource.read (
              self,
              size )

Read bytes from the stream and block until sample rate is achieved.

Args:
  size: number of bytes to read from the stream.
```

Definición en la línea 93 del archivo audio_helpers.py.

Gráfico de llamadas para esta función:

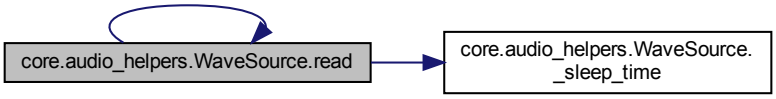
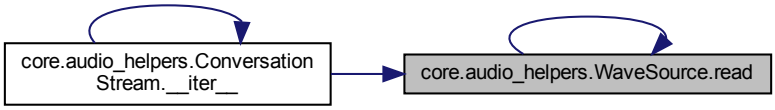


Gráfico de llamadas a esta función:



4.14.3.3. sample_rate()

```
def core.audio_helpers.WaveSource.sample_rate (
              self )
```

Definición en la línea 130 del archivo audio_helpers.py.

4.14.3.4. start()

```
def core.audio_helpers.WaveSource.start (
              self )
```

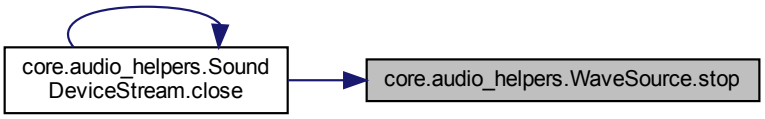
Definición en la línea 123 del archivo audio_helpers.py.

4.14.3.5. stop()

```
def core.audio_helpers.WaveSource.stop (
              self )
```

Definición en la línea 126 del archivo audio_helpers.py.

Gráfico de llamadas a esta función:



La documentación para esta clase fue generada a partir del siguiente fichero:

- core/audio_helpers.py

Capítulo 5

Documentación de archivos

5.1. Bluetooth.py

```
00001 #!/usr/bin/env python3
00002
00003 from signal import pause
00004 from threading import Thread
00005
00006 class Bluetooth(Thread):
00007     def __init__(self, bluetooth, servo):
00008         """!
00009         Esta clase se encarga de controlar las acciones de la cerradura por bluetooth
00010         @param bluetooth Recibe una instancia de BlueDot
00011         @param servo Recibe una instancia de un Servomotor
00012         """
00013         Thread.__init__(self)
00014         self.bdbd = bluetooth
00015         self.servoservo = servo
00016
00017     def dpad(self, pos):
00018         """!
00019         Por zona en la cual se presiona el bluedot, se abre o cierra la cerradura:
00020         Arrba Abre.
00021         Abajo Cierra
00022         """
00023         if pos.top:
00024             self.servoservo.max()
00025             print("Abriendo con BT")
00026         elif pos.bottom:
00027             self.servoservo.min()
00028             print("Cerrando con BT")
00029         else:
00030             pass
00031
00032     def bluetooth_main(self):
00033         """!
00034         Escucha la entrada de BlueDot
00035         """
00036         self.bdbd.when_pressed = self.dpaddpad
00037
00038         pause()
00039
00040     def run(self):
00041         """!
00042         Ejecuta la instancia en un hilo de ejecución
00043         """
00044         self.bluetooth_mainbluetooth_main()
```

5.2. Cerradura.py

```
00001 #!/usr/bin/env python3
00002
00003 class Cerradura:
00004     def __init__:
00005         pass
```

5.3. Assistant.py

```
00001 import concurrent.futures
00002 import json
00003 import logging
00004 import os
00005 import os.path
00006 import pathlib2 as pathlib
00007 import sys
00008 import time
00009 import uuid
00010
00011 import click
00012 import grpc
00013 import google.auth.transport.grpc
00014 import google.auth.transport.requests
00015 import google.oauth2.credentials
00016
00017 from google.assistant.embedded.v1alpha2 import (
00018     embedded_assistant_pb2,
00019     embedded_assistant_pb2_grpc
00020 )
00021 from tenacity import retry, stop_after_attempt, retry_if_exception
00022
00023 try:
00024     from . import (
00025         assistant_helpers,
00026         audio_helpers,
00027         browser_helpers,
00028         device_helpers
00029     )
00030 except (SystemError, ImportError):
00031     import assistant_helpers
00032     import audio_helpers
00033     import browser_helpers
00034     import device_helpers
00035
00036
00037 ASSISTANT_API_ENDPOINT = 'embeddedassistant.googleapis.com'
00038 END_OF_UTTERANCE = embedded_assistant_pb2.AssistResponse.END_OF_UTTERANCE
00039 DIALOG_FOLLOW_ON = embedded_assistant_pb2.DialogStateOut.DIALOG_FOLLOW_ON
00040 CLOSE_MICROPHONE = embedded_assistant_pb2.DialogStateOut.CLOSE_MICROPHONE
00041 PLAYING = embedded_assistant_pb2.ScreenOutConfig.PLAYING
00042 DEFAULT_GRPC_DEADLINE = 60 * 3 + 5
00043
00044
00045 class SampleAssistant(object):
00046     """Sample Assistant that supports conversations and device actions.
00047
00048     Args:
00049       device_model_id: identifier of the device model.
00050       device_id: identifier of the registered device instance.
00051       conversation_stream(ConversationStream): audio stream
00052         for recording query and playing back assistant answer.
00053       channel: authorized gRPC channel for connection to the
00054         Google Assistant API.
00055       deadline_sec: gRPC deadline in seconds for Google Assistant API call.
00056       device_handler: callback for device actions.
00057     """
00058
00059     def __init__(self, language_code, device_model_id, device_id,
00060                  conversation_stream, display,
00061                  channel, deadline_sec, device_handler):
00062         self.language_codelanguage_code = language_code
00063         self.device_model_iddevice_model_id = device_model_id
00064         self.device_iddevice_id = device_id
00065         self.conversation_streamconversation_stream = conversation_stream
00066         self.displaydisplay = display
00067
00068         # Opaque blob provided in AssistResponse that,
00069         # when provided in a follow-up AssistRequest,
00070         # gives the Assistant a context marker within the current state
00071         # of the multi-Assist()-RPC "conversation".
00072         # This value, along with MicrophoneMode, supports a more natural
00073         # "conversation" with the Assistant.
00074         self.conversation_stateconversation_state = None
00075         # Force reset of first conversation.
00076         self.is_new_conversationis_new_conversation = True
00077
00078         # Create Google Assistant API gRPC client.
00079         self.assistantassistant = embedded_assistant_pb2_grpc.EmbeddedAssistantStub(
00080             channel
00081         )
00082         self.deadlinedeadline = deadline_sec
00083
00084         self.device_handlerdevice_handler = device_handler
00085
```

```
00086     def __enter__(self):
00087         return self
00088
00089     def __exit__(self, etype, e, traceback):
00090         if e:
00091             return False
00092         self.conversation_streamconversation_stream.close()
00093
00094     def is_grpc_error_unavailable(e):
00095         is_grpc_error = isinstance(e, grpc.RpcError)
00096         if is_grpc_error and (e.code() == grpc.StatusCode.UNAVAILABLE):
00097             logging.error('grpc unavailable error: %s', e)
00098             return True
00099         return False
00100
00101     @retry(reraise=True, stop=stop_after_attempt(3),
00102            retry=retry_if_exception(is_grpc_error_unavailable))
00103     def assist(self):
00104         """Send a voice request to the Assistant and playback the response.
00105
00106         Returns: True if conversation should continue.
00107         """
00108         continue_conversation = False
00109         device_actions_futures = []
00110
00111         self.conversation_streamconversation_stream.start_recording()
00112         logging.info('Recording audio request.')
00113
00114         def iter_log_assist_requests():
00115             for c in self.gen_assist_requestsgen_assist_requests():
00116                 assistant_helpers.log_assist_request_without_audio(c)
00117                 yield c
00118             logging.debug('Reached end of AssistRequest iteration.')
00119
00120         # This generator yields AssistResponse proto messages
00121         # received from the gRPC Google Assistant API.
00122         for resp in self.assistantassistant.Assist(iter_log_assist_requests(),
00123                                                    self.deadlinedeadline):
00124             assistant_helpers.log_assist_response_without_audio(resp)
00125             if resp.event_type == END_OF_UTTERANCE:
00126                 logging.info('End of audio request detected.')
00127                 logging.info('Stopping recording.')
00128                 self.conversation_streamconversation_stream.stop_recording()
00129             if resp.speech_results:
00130                 logging.info('Transcript of user request: "%s".',
00131                              ' '.join(r.transcript
00132                                       for r in resp.speech_results))
00133             if len(resp.audio_out.audio_data) > 0:
00134                 if not self.conversation_streamconversation_stream.playing:
00135                     self.conversation_streamconversation_stream.stop_recording()
00136                     self.conversation_streamconversation_stream.start_playback()
00137                     logging.info('Playing assistant response.')
00138                 self.conversation_streamconversation_stream.write(resp.audio_out.audio_data)
00139             if resp.dialog_state_out.conversation_state:
00140                 conversation_state = resp.dialog_state_out.conversation_state
00141                 logging.debug('Updating conversation state.')
00142                 self.conversation_stateconversation_state = conversation_state
00143             if resp.dialog_state_out.volume_percentage != 0:
00144                 volume_percentage = resp.dialog_state_out.volume_percentage
00145                 logging.info('Setting volume to %s%%', volume_percentage)
00146                 self.conversation_streamconversation_stream.volume_percentage = volume_percentage
00147             if resp.dialog_state_out.microphone_mode == DIALOG_FOLLOW_ON:
00148                 continue_conversation = True
00149                 logging.info('Expecting follow-on query from user.')
00150             elif resp.dialog_state_out.microphone_mode == CLOSE_MICROPHONE:
00151                 continue_conversation = False
00152             if resp.device_action.device_request_json:
00153                 device_request = json.loads(
00154                     resp.device_action.device_request_json
00155                 )
00156                 fs = self.device_handlerdevice_handler(device_request)
00157                 if fs:
00158                     device_actions_futures.extend(fs)
00159             if self.displaydisplay and resp.screen_out.data:
00160                 system_browser = browser_helpers.system_browser
00161                 system_browser.display(resp.screen_out.data)
00162
00163         if len(device_actions_futures):
00164             logging.info('Waiting for device executions to complete.')
00165             concurrent.futures.wait(device_actions_futures)
00166
00167         logging.info('Finished playing assistant response.')
00168         self.conversation_streamconversation_stream.stop_playback()
00169         return continue_conversation
00170
00171     def gen_assist_requests(self):
00172         """Yields: AssistRequest messages to send to the API."""
```

```
00173
00174         config = embedded_assistant_pb2.AssistConfig(
00175             audio_in_config=embedded_assistant_pb2.AudioInConfig(
00176                 encoding='LINEAR16',
00177                 sample_rate_hertz=self.conversation_streamconversation_stream.sample_rate,
00178             ),
00179             audio_out_config=embedded_assistant_pb2.AudioOutConfig(
00180                 encoding='LINEAR16',
00181                 sample_rate_hertz=self.conversation_streamconversation_stream.sample_rate,
00182                 volume_percentage=self.conversation_streamconversation_stream.volume_percentage,
00183             ),
00184             dialog_state_in=embedded_assistant_pb2.DialogStateIn(
00185                 language_code=self.language_codelanguage_code,
00186                 conversation_state=self.conversation_stateconversation_state,
00187                 is_new_conversation=self.is_new_conversationis_new_conversation,
00188             ),
00189             device_config=embedded_assistant_pb2.DeviceConfig(
00190                 device_id=self.device_iddevice_id,
00191                 device_model_id=self.device_model_iddevice_model_id,
00192             )
00193         )
00194         if self.displaydisplay:
00195             config.screen_out_config.screen_mode = PLAYING
00196         # Continue current conversation with later requests.
00197         self.is_new_conversationis_new_conversation = False
00198         # The first AssistRequest must contain the AssistConfig
00199         # and no audio data.
00200         yield embedded_assistant_pb2.AssistRequest(config=config)
00201         for data in self.conversation_streamconversation_stream:
00202             # Subsequent requests need audio data, but not config.
00203             yield embedded_assistant_pb2.AssistRequest(audio_in=data)
00204
00205
00206
00207 def main(button):
00208     api_endpoint="embeddedassistant.googleapis.com"
00209     credentials="/home/pi/.config/google-oauthlib-tool/credentials.json"
00210     project_id=None
00211     device_model_id=None
00212     device_id=None
00213     device_config="/home/pi/.config/googlesamples-assistant/device_config.json"
00214     lang="es-MX"
00215     display=False
00216     verbose=False
00217     input_audio_file=None
00218     output_audio_file=None
00219     audio_sample_rate=16000
00220     audio_sample_width=2
00221     audio_iter_size=3200
00222     audio_block_size=6400
00223     audio_flush_size=25600
00224     grpc_deadline=185
00225     once=False
00226
00227
00228     logging.basicConfig(level=logging.DEBUG if verbose else logging.INFO)
00229
00230     # Load OAuth 2.0 credentials.
00231     try:
00232         with open(credentials, 'r') as f:
00233             credentials = google.oauth2.credentials.Credentials(token=None,
00234                                                                 **json.load(f))
00235             http_request = google.auth.transport.requests.Request()
00236             credentials.refresh(http_request)
00237     except Exception as e:
00238         logging.error('Error loading credentials: %s', e)
00239         logging.error('Run google-oauthlib-tool to initialize '
00240                       'new OAuth 2.0 credentials.')
00241         sys.exit(-1)
00242
00243     # Create an authorized gRPC channel.
00244     grpc_channel = google.auth.transport.grpc.secure_authorized_channel(
00245         credentials, http_request, api_endpoint)
00246     logging.info('Connecting to %s', api_endpoint)
00247
00248     # Configure audio source and sink.
00249     audio_device = None
00250     if input_audio_file:
00251         audio_source = audio_helpers.WaveSource(
00252             open(input_audio_file, 'rb'),
00253             sample_rate=audio_sample_rate,
00254             sample_width=audio_sample_width
00255         )
00256     else:
00257         audio_source = audio_device = (
00258             audio_device or audio_helpers.SoundDeviceStream(
00259                 sample_rate=audio_sample_rate,
```

```
00260                 sample_width=audio_sample_width,
00261                 block_size=audio_block_size,
00262                 flush_size=audio_flush_size
00263             )
00264         )
00265     if output_audio_file:
00266         audio_sink = audio_helpers.WaveSink(
00267             open(output_audio_file, 'wb'),
00268             sample_rate=audio_sample_rate,
00269             sample_width=audio_sample_width
00270         )
00271     else:
00272         audio_sink = audio_device = (
00273             audio_device or audio_helpers.SoundDeviceStream(
00274                 sample_rate=audio_sample_rate,
00275                 sample_width=audio_sample_width,
00276                 block_size=audio_block_size,
00277                 flush_size=audio_flush_size
00278             )
00279         )
00280     # Create conversation stream with the given audio source and sink.
00281     conversation_stream = audio_helpers.ConversationStream(
00282         source=audio_source,
00283         sink=audio_sink,
00284         iter_size=audio_iter_size,
00285         sample_width=audio_sample_width,
00286     )
00287
00288     if not device_id or not device_model_id:
00289         try:
00290             with open(device_config) as f:
00291                 device = json.load(f)
00292                 device_id = device['id']
00293                 device_model_id = device['model_id']
00294                 logging.info("Using device model %s and device id %s",
00295                              device_model_id,
00296                              device_id)
00297         except Exception as e:
00298             logging.warning('Device config not found: %s' % e)
00299             logging.info('Registering device')
00300             if not device_model_id:
00301                 logging.error('Option --device-model-id required '
00302                               'when registering a device instance.')
00303                 sys.exit(-1)
00304             if not project_id:
00305                 logging.error('Option --project-id required '
00306                               'when registering a device instance.')
00307                 sys.exit(-1)
00308             device_base_url = (
00309                 'https://%s/v1alpha2/projects/%s/devices' % (api_endpoint,
00310                                                              project_id)
00311             )
00312             device_id = str(uuid.uuid1())
00313             payload = {
00314                 'id': device_id,
00315                 'model_id': device_model_id,
00316                 'client_type': 'SDK_SERVICE'
00317             }
00318             session = google.auth.transport.requests.AuthorizedSession(
00319                 credentials
00320             )
00321             r = session.post(device_base_url, data=json.dumps(payload))
00322             if r.status_code != 200:
00323                 logging.error('Failed to register device: %s', r.text)
00324                 sys.exit(-1)
00325             logging.info('Device registered: %s', device_id)
00326             pathlib.Path(os.path.dirname(device_config)).mkdir(exist_ok=True)
00327             with open(device_config, 'w') as f:
00328                 json.dump(payload, f)
00329
00330     device_handler = device_helpers.DeviceRequestHandler(device_id)
00331
00332     @device_handler.command('action.devices.commands.OnOff')
00333     def onoff(on):
00334         if on:
00335             logging.info('Turning device on')
00336         else:
00337             logging.info('Turning device off')
00338
00339     @device_handler.command('com.example.commands.BlinkLight')
00340     def blink(speed, number):
00341         logging.info('Blinking device %s times.' % number)
00342         delay = 1
00343         if speed == "SLOWLY":
00344             delay = 2
00345         elif speed == "QUICKLY":
00346             delay = 0.5
```

```
00347         for i in range(int(number)):
00348             logging.info('Device is blinking.')
00349             time.sleep(delay)
00350 
00351     with SampleAssistant(lang, device_model_id, device_id,
00352                          conversation_stream, display,
00353                          grpc_channel, grpc_deadline,
00354                          device_handler) as assistant:
00355         # If file arguments are supplied:
00356         # exit after the first turn of the conversation.
00357         if input_audio_file or output_audio_file:
00358             assistant.assist()
00359             return
00360 
00361         # If no file arguments supplied:
00362         # keep recording voice requests using the microphone
00363         # and playing back assistant response using the speaker.
00364         # When the once flag is set, don't wait for a trigger. Otherwise, wait.
00365         wait_for_user_trigger = not once
00366         while True:
00367             if wait_for_user_trigger:
00368                 print('Press Enter to send a new request...')
00369                 button.wait_for_press()
00370             continue_conversation = assistant.assist()
00371             # wait for user trigger if there is no follow-up turn in
00372             # the conversation.
00373             wait_for_user_trigger = not continue_conversation
00374 
00375             # If we only want one conversation, break.
00376             if once and (not continue_conversation):
00377                 break
```

5.4. assistant_helpers.py

```
00001 # Copyright (C) 2017 Google Inc.
00002 #
00003 # Licensed under the Apache License, Version 2.0 (the "License");
00004 # you may not use this file except in compliance with the License.
00005 # You may obtain a copy of the License at
00006 #
00007 #     http://www.apache.org/licenses/LICENSE-2.0
00008 #
00009 # Unless required by applicable law or agreed to in writing, software
00010 # distributed under the License is distributed on an "AS IS" BASIS,
00011 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 # See the License for the specific language governing permissions and
00013 # limitations under the License.
00014 
00015 """Helper functions for the Google Assistant API."""
00016 
00017 import logging
00018 
00019 from google.assistant.embedded.v1alpha2 import embedded_assistant_pb2
00020 
00021 
00022 def log_assist_request_without_audio(assist_request):
00023     """Log AssistRequest fields without audio data."""
00024     if logging.getLogger().isEnabledFor(logging.DEBUG):
00025         resp_copy = embedded_assistant_pb2.AssistRequest()
00026         resp_copy.CopyFrom(assist_request)
00027         if len(resp_copy.audio_in) > 0:
00028             size = len(resp_copy.audio_in)
00029             resp_copy.ClearField('audio_in')
00030             logging.debug('AssistRequest: audio_in (%d bytes)',
00031                           size)
00032             return
00033         logging.debug('AssistRequest: %s', resp_copy)
00034 
00035 
00036 def log_assist_response_without_audio(assist_response):
00037     """Log AssistResponse fields without audio data."""
00038     if logging.getLogger().isEnabledFor(logging.DEBUG):
00039         resp_copy = embedded_assistant_pb2.AssistResponse()
00040         resp_copy.CopyFrom(assist_response)
00041         has_audio_data = (resp_copy.HasField('audio_out') and
00042                           len(resp_copy.audio_out.audio_data) > 0)
00043         if has_audio_data:
00044             size = len(resp_copy.audio_out.audio_data)
00045             resp_copy.audio_out.ClearField('audio_data')
00046             if resp_copy.audio_out.ListFields():
00047                 logging.debug('AssistResponse: %s audio_data (%d bytes)',
00048                               resp_copy,
00049                               size)
```

```
00050             else:
00051                 logging.debug('AssistResponse: audio_data (%d bytes)',
00052                               size)
00053             return
00054         logging.debug('AssistResponse: %s', resp_copy)
```

5.5. audio_helpers.py

```
00001 # Copyright (C) 2017 Google Inc.
00002 #
00003 # Licensed under the Apache License, Version 2.0 (the "License");
00004 # you may not use this file except in compliance with the License.
00005 # You may obtain a copy of the License at
00006 #
00007 #     http://www.apache.org/licenses/LICENSE-2.0
00008 #
00009 # Unless required by applicable law or agreed to in writing, software
00010 # distributed under the License is distributed on an "AS IS" BASIS,
00011 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 # See the License for the specific language governing permissions and
00013 # limitations under the License.
00014
00015 """Helper functions for audio streams."""
00016
00017 import array
00018 import logging
00019 import math
00020 import time
00021 import threading
00022 import wave
00023
00024 import click
00025 import sounddevice as sd
00026
00027
00028 DEFAULT_AUDIO_SAMPLE_RATE = 16000
00029 DEFAULT_AUDIO_SAMPLE_WIDTH = 2
00030 DEFAULT_AUDIO_ITER_SIZE = 3200
00031 DEFAULT_AUDIO_DEVICE_BLOCK_SIZE = 6400
00032 DEFAULT_AUDIO_DEVICE_FLUSH_SIZE = 25600
00033
00034
00035 def normalize_audio_buffer(buf, volume_percentage, sample_width=2):
00036     """Adjusts the loudness of the audio data in the given buffer.
00037
00038     Volume normalization is done by scaling the amplitude of the audio
00039     in the buffer by a scale factor of 2^(volume_percentage/100)-1.
00040     For example, 50 % volume scales the amplitude by a factor of 0.414,
00041     and 75 % volume scales the amplitude by a factor of 0.681.
00042     For now we only sample_width 2.
00043
00044     Args:
00045       buf: byte string containing audio data to normalize.
00046       volume_percentage: volume setting as an integer percentage (1-100).
00047       sample_width: size of a single sample in bytes.
00048     """
00049     if sample_width != 2:
00050         raise Exception('unsupported sample width:', sample_width)
00051     scale = math.pow(2, 1.0*volume_percentage/100)-1
00052     # Construct array from bytes based on sample_width, multiply by scale
00053     # and convert it back to bytes
00054     arr = array.array('h', buf)
00055     for idx in range(0, len(arr)):
00056         arr[idx] = int(arr[idx]*scale)
00057     buf = arr.tostring()
00058     return buf
00059
00060
00061 def align_buf(buf, sample_width):
00062     """In case of buffer size not aligned to sample_width pad it with 0s"""
00063     remainder = len(buf) % sample_width
00064     if remainder != 0:
00065         buf += b'\0' * (sample_width - remainder)
00066     return buf
00067
00068
00069 class WaveSource(object):
00070     """Audio source that reads audio data from a WAV file.
00071
00072     Reads are throttled to emulate the given sample rate and silence
00073     is returned when the end of the file is reached.
00074
00075     Args:
```

```
00076       fp: file-like stream object to read from.
00077       sample_rate: sample rate in hertz.
00078       sample_width: size of a single sample in bytes.
00079     """
00080     def __init__(self, fp, sample_rate, sample_width):
00081         self._fp_fp = fp
00082         try:
00083             self._wavep_wavep = wave.open(self._fp_fp, 'r')
00084         except wave.Error as e:
00085             logging.warning('error opening WAV file: %s, '
00086                             'falling back to RAW format', e)
00087             self._fp_fp.seek(0)
00088             self._wavep_wavep = None
00089         self._sample_rate_sample_rate = sample_rate
00090         self._sample_width_sample_width = sample_width
00091         self._sleep_until_sleep_until = 0
00092
00093     def read(self, size):
00094         """Read bytes from the stream and block until sample rate is achieved.
00095
00096         Args:
00097           size: number of bytes to read from the stream.
00098         """
00099         now = time.time()
00100         missing_dt = self._sleep_until_sleep_until - now
00101         if missing_dt > 0:
00102             time.sleep(missing_dt)
00103         self._sleep_until_sleep_until = time.time() + self._sleep_time_sleep_time(size)
00104         data = (self._wavep_wavep.readframes(size)
00105                 if self._wavep_wavep
00106                 else self._fp_fp.read(size))
00107         #  When reach end of audio stream, pad remainder with silence (zeros).
00108         if not data:
00109             return b'\x00' * size
00110         return data
00111
00112     def close(self):
00113         """Close the underlying stream."""
00114         if self._wavep_wavep:
00115             self._wavep_wavep.close()
00116         self._fp_fp.close()
00117
00118     def _sleep_time(self, size):
00119         sample_count = size / float(self._sample_width_sample_width)
00120         sample_rate_dt = sample_count / float(self._sample_rate_sample_rate)
00121         return sample_rate_dt
00122
00123     def start(self):
00124         pass
00125
00126     def stop(self):
00127         pass
00128
00129     @property
00130     def sample_rate(self):
00131         return self._sample_rate
00132
00133
00134 class WaveSink(object):
00135     """Audio sink that writes audio data to a WAV file.
00136
00137     Args:
00138       fp: file-like stream object to write data to.
00139       sample_rate: sample rate in hertz.
00140       sample_width: size of a single sample in bytes.
00141     """
00142     def __init__(self, fp, sample_rate, sample_width):
00143         self._fp_fp = fp
00144         self._wavep_wavep = wave.open(self._fp_fp, 'wb')
00145         self._wavep_wavep.setsampwidth(sample_width)
00146         self._wavep_wavep.setnchannels(1)
00147         self._wavep_wavep.setframerate(sample_rate)
00148
00149     def write(self, data):
00150         """Write bytes to the stream.
00151
00152         Args:
00153           data: frame data to write.
00154         """
00155         self._wavep_wavep.writeframes(data)
00156
00157     def close(self):
00158         """Close the underlying stream."""
00159         self._wavep_wavep.close()
00160         self._fp_fp.close()
00161
00162     def start(self):
```

```
00163         pass
00164
00165     def stop(self):
00166         pass
00167
00168     def flush(self):
00169         pass
00170
00171
00172 class SoundDeviceStream(object):
00173     """Audio stream based on an underlying sound device.
00174
00175     It can be used as an audio source (read) and a audio sink (write).
00176
00177     Args:
00178       sample_rate: sample rate in hertz.
00179       sample_width: size of a single sample in bytes.
00180       block_size: size in bytes of each read and write operation.
00181       flush_size: size in bytes of silence data written during flush operation.
00182     """
00183     def __init__(self, sample_rate, sample_width, block_size, flush_size):
00184         if sample_width == 2:
00185             audio_format = 'int16'
00186         else:
00187             raise Exception('unsupported sample width:', sample_width)
00188         self._audio_stream = sd.RawStream(
00189             samplerate=sample_rate, dtype=audio_format, channels=1,
00190             blocksize=int(block_size/2),  # blocksize is in number of frames.
00191         )
00192         self._block_size = block_size
00193         self._flush_size = flush_size
00194         self._sample_rate = sample_rate
00195
00196     def read(self, size):
00197         """Read bytes from the stream."""
00198         buf, overflow = self._audio_stream_audio_stream.read(size)
00199         if overflow:
00200             logging.warning('SoundDeviceStream read overflow (%d, %d)',
00201                             size, len(buf))
00202         return bytes(buf)
00203
00204     def write(self, buf):
00205         """Write bytes to the stream."""
00206         underflow = self._audio_stream_audio_stream.write(buf)
00207         if underflow:
00208             logging.warning('SoundDeviceStream write underflow (size: %d)',
00209                             len(buf))
00210         return len(buf)
00211
00212     def flush(self):
00213         if self._audio_stream_audio_stream.active and self._flush_size_flush_size > 0:
00214             self._audio_stream_audio_stream.write(b'\x00' * self._flush_size_flush_size)
00215
00216     def start(self):
00217         """Start the underlying stream."""
00218         if not self._audio_stream_audio_stream.active:
00219             self._audio_stream_audio_stream.start()
00220
00221     def stop(self):
00222         """Stop the underlying stream."""
00223         if self._audio_stream_audio_stream.active:
00224             self._audio_stream_audio_stream.stop()
00225
00226     def close(self):
00227         """Close the underlying stream and audio interface."""
00228         if self._audio_stream_audio_stream:
00229             self.stopstop()
00230             self._audio_stream_audio_stream.close()
00231             self._audio_stream_audio_stream = None
00232
00233     @property
00234     def sample_rate(self):
00235         return self._sample_rate_sample_rate
00236
00237
00238 class ConversationStream(object):
00239     """Audio stream that supports half-duplex conversation.
00240
00241     A conversation is the alternance of:
00242     - a recording operation
00243     - a playback operation
00244
00245     Excepted usage:
00246
00247       For each conversation:
00248       - start_recording()
00249       - read() or iter()
```

```
00250       - stop_recording()
00251       - start_playback()
00252       - write()
00253       - stop_playback()
00254
00255       When conversations are finished:
00256       - close()
00257
00258     Args:
00259       source: file-like stream object to read input audio bytes from.
00260       sink: file-like stream object to write output audio bytes to.
00261       iter_size: read size in bytes for each iteration.
00262       sample_width: size of a single sample in bytes.
00263     """
00264     def __init__(self, source, sink, iter_size, sample_width):
00265         self._source_source = source
00266         self._sink_sink = sink
00267         self._iter_size_iter_size = iter_size
00268         self._sample_width_sample_width = sample_width
00269         self._volume_percentage_volume_percentage = 50
00270         self._stop_recording_stop_recording = threading.Event()
00271         self._source_lock_source_lock = threading.RLock()
00272         self._recording_recording = False
00273         self._playing_playing = False
00274
00275     def start_recording(self):
00276         """Start recording from the audio source."""
00277         self._recording_recording = True
00278         self._stop_recording_stop_recording.clear()
00279         self._source_source.start()
00280
00281     def stop_recording(self):
00282         """Stop recording from the audio source."""
00283         self._stop_recording_stop_recording.set()
00284         with self._source_lock_source_lock:
00285             self._source_source.stop()
00286         self._recording_recording = False
00287
00288     def start_playback(self):
00289         """Start playback to the audio sink."""
00290         self._playing_playing = True
00291         self._sink_sink.start()
00292
00293     def stop_playback(self):
00294         """Stop playback from the audio sink."""
00295         self._sink_sink.flush()
00296         self._sink_sink.stop()
00297         self._playing_playing = False
00298
00299     @property
00300     def recording(self):
00301         return self._recording_recording
00302
00303     @property
00304     def playing(self):
00305         return self._playing_playing
00306
00307     @property
00308     def volume_percentage(self):
00309         """The current volume setting as an integer percentage (1-100)."""
00310         return self._volume_percentage_volume_percentage
00311
00312     @volume_percentage.setter
00313     def volume_percentage(self, new_volume_percentage):
00314         self._volume_percentage_volume_percentage = new_volume_percentage
00315
00316     def read(self, size):
00317         """Read bytes from the source (if currently recording).
00318         """
00319         with self._source_lock_source_lock:
00320             return self._source_source.read(size)
00321
00322     def write(self, buf):
00323         """Write bytes to the sink (if currently playing).
00324         """
00325         buf = align_buf(buf, self._sample_width_sample_width)
00326         buf = normalize_audio_buffer(buf, self.volume_percentagevolume_percentagevolume_percentage)
00327         return self._sink_sink.write(buf)
00328
00329     def close(self):
00330         """Close source and sink."""
00331         self._source_source.close()
00332         self._sink_sink.close()
00333
00334     def __iter__(self):
00335         """Returns a generator reading data from the stream."""
00336         while True:
```

```
00337             if self._stop_recording_stop_recording.is_set():
00338                 return
00339             yield self.readread(self._iter_size_iter_size)
00340 
00341     @property
00342     def sample_rate(self):
00343         return self._source_source._sample_rate
00344 
00345 
00346 @click.command()
00347 @click.option('--record-time', default=5,
00348               metavar='<record time>', show_default=True,
00349               help='Record time in secs')
00350 @click.option('--audio-sample-rate',
00351               default=DEFAULT_AUDIO_SAMPLE_RATE,
00352               metavar='<audio sample rate>', show_default=True,
00353               help='Audio sample rate in hertz.')
00354 @click.option('--audio-sample-width',
00355               default=DEFAULT_AUDIO_SAMPLE_WIDTH,
00356               metavar='<audio sample width>', show_default=True,
00357               help='Audio sample width in bytes.')
00358 @click.option('--audio-iter-size',
00359               default=DEFAULT_AUDIO_ITER_SIZE,
00360               metavar='<audio iter size>', show_default=True,
00361               help='Size of each read during audio stream iteration in bytes.')
00362 @click.option('--audio-block-size',
00363               default=DEFAULT_AUDIO_DEVICE_BLOCK_SIZE,
00364               metavar='<audio block size>', show_default=True,
00365               help=('Block size in bytes for each audio device '
00366                     'read and write operation..'))
00367 @click.option('--audio-flush-size',
00368               default=DEFAULT_AUDIO_DEVICE_FLUSH_SIZE,
00369               metavar='<audio flush size>', show_default=True,
00370               help=('Size of silence data in bytes written '
00371                     'during flush operation'))
00372 def main(record_time, audio_sample_rate, audio_sample_width,
00373          audio_iter_size, audio_block_size, audio_flush_size):
00374     """Helper command to test audio stream processing.
00375 
00376     - Record 5 seconds of 16-bit samples at 16khz.
00377     - Playback the recorded samples.
00378     """
00379     end_time = time.time() + record_time
00380     audio_device = SoundDeviceStream(sample_rate=audio_sample_rate,
00381                                      sample_width=audio_sample_width,
00382                                      block_size=audio_block_size,
00383                                      flush_size=audio_flush_size)
00384     stream = ConversationStream(source=audio_device,
00385                                 sink=audio_device,
00386                                 iter_size=audio_iter_size,
00387                                 sample_width=audio_sample_width)
00388     samples = []
00389     logging.basicConfig(level=logging.INFO)
00390     logging.info('Starting audio test.')
00391 
00392     stream.start_recording()
00393     logging.info('Recording samples.')
00394     while time.time() < end_time:
00395         samples.append(stream.read(audio_block_size))
00396     logging.info('Finished recording.')
00397     stream.stop_recording()
00398 
00399     stream.start_playback()
00400     logging.info('Playing back samples.')
00401     while len(samples):
00402         stream.write(samples.pop(0))
00403     logging.info('Finished playback.')
00404     stream.stop_playback()
00405 
00406     logging.info('audio test completed.')
00407     stream.close()
00408 
00409 
00410 if __name__ == '__main__':
00411     main()
```

5.6. BD.py

```
00001 #!/usr/bin/env python3
00002 
00003 import psycopg2 as psy
00004 from os import getenv
00005 
```

```
00006 class BD:
00007     def __init__(self):
00008         """!
00009         Esta clase solo se encarga de conectar la base de datos. implementada en
00010         Postgresql.
00011         """
00012         self.connconn = psy.connect(
00013             user=getenv("PGUSER"),
00014             password=getenv("PGPASSWORD"),
00015             host=getenv("PGHOST"),
00016             port="5432",
00017             database=getenv("PGDATABASE")
00018         )
00019 
00020     def ejecutar_consulta(self, query, un_resultado=True):
00021         """!
00022         Ejecuta la consulta
00023         @param query Es la consulta que se realizará
00024         @param un_resultado Se esfecifica en True si solo se espera una fila de
00025         resultados, False si se espera varias filas, por defecto esta en True
00026 
00027         @return Ua o unas tuplas de resultados
00028         """
00029         cur = self.connconn.cursor()
00030         cur.execute(query)
00031         if (un_resultado):
00032             return cur.fetchone()
00033         return cur.fetchall()
```

5.7. browser_helpers.py

```
00001 # Copyright (C) 2018 Google Inc.
00002 #
00003 # Licensed under the Apache License, Version 2.0 (the "License");
00004 # you may not use this file except in compliance with the License.
00005 # You may obtain a copy of the License at
00006 #
00007 #     http://www.apache.org/licenses/LICENSE-2.0
00008 #
00009 # Unless required by applicable law or agreed to in writing, software
00010 # distributed under the License is distributed on an "AS IS" BASIS,
00011 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 # See the License for the specific language governing permissions and
00013 # limitations under the License.
00014 
00015 import os.path
00016 import tempfile
00017 import webbrowser
00018 
00019 ASSISTANT_HTML_FILE = 'google-assistant-sdk-screen-out.html'
00020 
00021 
00022 class SystemBrowser(object):
00023     def __init__(self):
00024         self.tempdirtempdir = tempfile.mkdtemp()
00025         self.filenamefilename = os.path.join(self.tempdirtempdir, ASSISTANT_HTML_FILE)
00026 
00027     def display(self, html):
00028         with open(self.filenamefilename, 'wb') as f:
00029             f.write(html)
00030         webbrowser.open(self.filenamefilename, new=0)
00031 
00032 
00033 system_browser = SystemBrowser()
```

5.8. Core.py

```
00001 #!/usr/bin/env python3
00002 
00003 import os
00004 import time
00005 import click
00006 import telebot
00007 from threading import Thread, Event
00008 from signal import pause
00009 from bluedot import BlueDot
00010 from gpiozero import Servo, LED, DistanceSensor, LightSensor, Button
00011 from pyfingerprint.pyfingerprint import PyFingerprint
00012 from gas.Gas import Gas
```

```
00013 from bluetooth.Bluetooth import Bluetooth
00014 from lector_huella.Lector import Lector
00015 from threading import Thread
00016 from time import sleep
00017 from core.BD import BD
00018 from core.Assistant import main as ass_main
00019 from google.assistant.embedded.v1alpha2 import (
00020     embedded_assistant_pb2,
00021     embedded_assistant_pb2_grpc
00022 )
00023 try:
00024     from . import (
00025         assistant_helpers,
00026         audio_helpers,
00027         browser_helpers,
00028         device_helpers
00029     )
00030 except (SystemError, ImportError):
00031     import assistant_helpers
00032     import audio_helpers
00033     import browser_helpers
00034     import device_helpers
00035
00036
00037 ASSISTANT_API_ENDPOINT = 'embeddedassistant.googleapis.com'
00038 END_OF_UTTERANCE = embedded_assistant_pb2.AssistResponse.END_OF_UTTERANCE
00039 DIALOG_FOLLOW_ON = embedded_assistant_pb2.DialogStateOut.DIALOG_FOLLOW_ON
00040 CLOSE_MICROPHONE = embedded_assistant_pb2.DialogStateOut.CLOSE_MICROPHONE
00041 PLAYING = embedded_assistant_pb2.ScreenOutConfig.PLAYING
00042 DEFAULT_GRPC_DEADLINE = 60 * 3 + 5
00043
00044 class Core(Thread):
00045     def __init__(self):
00046         """!
00047         Esta clase sirve como compositor e iniciador de cada sensor, inicia y
00048         decara los componestes que se usará para el embebido.
00049         """
00050         Thread.__init__(self)
00051         self.shutdown_flagshutdown_flag = Event()
00052         self.bdbd = BlueDot()
00053         self.status_led_okstatus_led_ok = LED(3,active_high=False) # verde
00054         self.status_led_errorstatus_led_error = LED(4,active_high=False) # rojo
00055         self.led_gen_statusled_gen_status = LED(2,active_high=False) # azul
00056         self.servo_motorservo_motor = Servo(17)
00057         self.servo_motorservo_motor.min()
00058         self.gas_sensorgas_sensor = LightSensor(21)
00059         self.interruptor_metalinterruptor_metal = Button(19)
00060         self.interruptor_plasticointerruptor_plastico = Button(13)
00061         self.distancia_sensordistancia_sensor = DistanceSensor(echo=20, trigger=26)
00062         # self.asistente = Asistente(self.interruptor_metal)
00063         self.__key_api_telegram__key_api_telegram = os.getenv("TELEGRAM_API")
00064         self.bluetoothbluetooth = Bluetooth(self.bdbd, self.servo_motorservo_motor)
00065         self.tele_bottele_bot = telebot.TeleBot(self.__key_api_telegram__key_api_telegram)
00066         self.bdbd = BD()
00067         self.lectorlector = Lector(self.status_led_errorstatus_led_error,
      self.status_led_okstatus_led_ok,
00068                                    self.tele_bottele_bot, self.servo_motorservo_motor, '/dev/ttyUSB0')
00069         self.mod_gasmod_gas = Gas(self.gas_sensorgas_sensor, self.led_gen_statusled_gen_status,
      self.tele_bottele_bot,
00070                                   self.bdbd, self.shutdown_flagshutdown_flag)
00071
00072
00073     def telebot_msg_handler(self):
00074         """!
00075         Este método es el encargado de enviar y recibir mensajes del bot de telegram
00076         """
00077         bot = self.tele_bottele_bot
00078
00079         @bot.message_handler(commands=['abrir'])
00080         def abrir(message):
00081             print("Abrir con cliente Telegram")
00082             self.abrir_cerraduraabrir_cerradura()
00083             msg = "Abierto"
00084             bot.reply_to(message, msg)
00085
00086         @bot.message_handler(commands=['cerrar'])
00087         def cerrar(message):
00088             print("Cerrar con cliente Telegram")
00089             self.cerrar_cerraduracerrar_cerradura()
00090             msg = "Cerrado"
00091             bot.reply_to(message, msg)
00092
00093         @bot.message_handler(func=lambda message: True)
00094         def echo_message(message):
00095             print("Enviando datos del chat id...")
00096             bot.reply_to(message, "Tu chat id es:")
00097             sleep(1)
```

```
00098             bot.reply_to(message, message.chat.id)
00099
00100         bot.infinity_polling()
00101
00102     def run(self):
00103         """!
00104         Aquí se llaman y se ejecuta en cada hilo los procesos de cada sensor
00105         """
00106         self.led_gen_statusled_gen_status.blink(on_time=1, off_time=5)
00107         self.mod_gasmod_gas.start()
00108         print("\033[32mIniciando Modulo de Bluetooth\033[0m")
00109         self.bluetoothbluetooth.start()
00110         self.abrir_cerraduraabrir_cerradura()
00111         print("\033[32mIniciando Asistente\033[0m")
00112         # self.asistente.start()
00113         Thread(target=ass_main, args=(self.interruptor_metalinterruptor_metal,)).start()
00114         Thread(target=self.cerraduracerradura).start()
00115         print("\033[32mIniciando Lector de huella\033[0m")
00116         self.lectorlector.start()
00117         print("\033[32mIniciando Modulo de Telegram\033[0m")
00118         self.telebot_msg_handlertelebot_msg_handler()
00119
00120
00121     def abrir_cerradura(self):
00122         """!
00123         Abre la cerradura
00124         """
00125         self.servo_motorservo_motor.max()
00126
00127     def cerrar_cerradura(self):
00128         """!
00129         Cierra la cerradura
00130         """
00131         self.servo_motorservo_motor.min()
00132
00133     def cambiar_edo(self):
00134         """!
00135         Cierra o abre la serradura segun su estado actual
00136         """
00137         if self.servo_motorservo_motor.value == -1.0:
00138             self.abrir_cerraduraabrir_cerradura()
00139         else:
00140             self.cerrar_cerraduracerrar_cerradura()
00141
00142     def cerradura(self):
00143         """!
00144         Escucha del interruptor para cambiar el estado de la cerradura
00145         """
00146         while True:
00147             if self.interruptor_plasticointerruptor_plastico.is_pressed:
00148                 print("Cambia")
00149                 self.cambiar_edocambiar_edo()
00150                 self.interruptor_plasticointerruptor_plastico.wait_for_release()
```

5.9. device_helpers.py

```
00001 # Copyright (C) 2017 Google Inc.
00002 #
00003 # Licensed under the Apache License, Version 2.0 (the "License");
00004 # you may not use this file except in compliance with the License.
00005 # You may obtain a copy of the License at
00006 #
00007 #     http://www.apache.org/licenses/LICENSE-2.0
00008 #
00009 # Unless required by applicable law or agreed to in writing, software
00010 # distributed under the License is distributed on an "AS IS" BASIS,
00011 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00012 # See the License for the specific language governing permissions and
00013 # limitations under the License.
00014
00015 """Helper functions for the Device Actions."""
00016
00017 import concurrent.futures
00018 import logging
00019 import sys
00020
00021
00022 key_inputs_ = 'inputs'
00023 key_intent_ = 'intent'
00024 key_payload_ = 'payload'
00025 key_commands_ = 'commands'
00026 key_id_ = 'id'
00027
```

```
00028
00029 class DeviceRequestHandler(object):
00030     """Asynchronous dispatcher for Device actions commands.
00031
00032     Dispatch commands to the given device handlers.
00033
00034     Args:
00035       device_id: device id to match command against
00036
00037     Example:
00038       # Use as as decorator to register handler.
00039       device_handler = DeviceRequestHandler('my-device')
00040       @device_handler.command('INTENT_NAME')
00041       def handler(param):
00042           pass
00043     """
00044
00045     def __init__(self, device_id):
00046         self.executorexecutor = concurrent.futures.ThreadPoolExecutor(max_workers=1)
00047         self.device_iddevice_id = device_id
00048         self.handlershandlers = {}
00049
00050     def __call__(self, device_request):
00051         """Handle incoming device request.
00052
00053         Returns: List of concurrent.futures for each command execution.
00054         """
00055         fs = []
00056         if key_inputs_ in device_request:
00057             for input in device_request[key_inputs_]:
00058                 if input[key_intent_] == 'action.devices.EXECUTE':
00059                     for command in input[key_payload_][key_commands_]:
00060                         fs.extend(self.submit_commandssubmit_commands(**command))
00061         return fs
00062
00063     def command(self, intent):
00064         """Register a device action handlers."""
00065         def decorator(fn):
00066             self.handlershandlers[intent] = fn
00067         return decorator
00068
00069     def submit_commands(self, devices, execution):
00070         """Submit device command executions.
00071
00072         Returns: a list of concurrent.futures for scheduled executions.
00073         """
00074         fs = []
00075         for device in devices:
00076             if device[key_id_] != self.device_iddevice_id:
00077                 logging.warning('Ignoring command for unknown device: %s'
00078                                 % device[key_id_])
00079                 continue
00080             if not execution:
00081                 logging.warning('Ignoring noop execution')
00082                 continue
00083             for command in execution:
00084                 f = self.executorexecutor.submit(
00085                     self.dispatch_commanddispatch_command, **command
00086                 )
00087                 fs.append(f)
00088         return fs
00089
00090     def dispatch_command(self, command, params=None):
00091         """Dispatch device commands to the appropriate handler."""
00092         try:
00093             if command in self.handlershandlers:
00094                 self.handlershandlers[command](**params)
00095             else:
00096                 logging.warning('Unsupported command: %s: %s',
00097                                 command, params)
00098         except Exception as e:
00099             logging.warning('Error during command execution',
00100                             exc_info=sys.exc_info())
00101             raise e
```

5.10. Distancia.py

```
00001 #!/usr/bin/env python3
00002 from emoji import emojize
00003 from time import sleep
00004 from signal import pause
00005 from threading import Thread
00006
```

```
00007
00008 class Distancia(Thread):
00009         """!
00010     def __init__(self, dist_sensor, led_status, tele_bot, database, shutdown_flag):
00011         Esta Clase Se encarga de instanciar y controlar el sensor de distancia,
00012         avisa al usuario si alguien esta cerca de la cerradura.
00013         @param dist_sensor Instancia del sensor de distancica
00014         @param led_status Instancua del led de estado
00015         @param tele_bot Instancia del bot de telegram
00016         @param database Instancia del controlador de la BD
00017         @param shutdown_flag Bandera de señales
00018         """
00019         Thread.__init__(self)
00020         self.shutdown_flagshutdown_flag = shutdown_flag
00021
00022         self.dist_sensordist_sensor = dist_sensor
00023         self.led_statusled_status = led_status
00024         self.tele_bottele_bot = tele_bot
00025         self.__conn__conn = database
00026
00027     def sensor_listener(self):
00028         """!
00029         Escucha del sensor de distancia que envia mensaje por telegram si
00030         alguien esta cerca de la puerta
00031         """
00032         while not self.shutdown_flagshutdown_flag.is_set():
00033             if (self.dist_sensordist_sensor.distance < 600):
00034                 records = self.__conn__conn.ejecutar_consulta(
00035                     """SELECT chat_id FROM alta_notificaciones where
00036                     proximidad is true""")
00037                 self.led_statusled_status.blink(on_time=60, off_time=1, n=1)
00038                 if records:
00039                     for chat_id in records:
00040                         self.tele_bottele_bot.send_message(chat_id, emojize("""
00041                         Alguien esta cerca de la puerta :puerta:
00042                         """, language='es'))
00043
00044     def run(self):
00045         """!
00046         Ejecuta el escucha del sensor de distancia en un hilo de ejecución
00047         """
00048         self.sensor_listenersensor_listener()
```

5.11. __init__.py

5.12. __init__.py

5.13. __init__.py

5.14. Gas.py

```
00001 #!/usr/bin/env python3
00002
00003 from emoji import emojize
00004 from time import sleep
00005 from signal import pause
00006 from threading import Thread
00007
00008 class Gas(Thread):
00009     def __init__(self, gas_sensor, led_status, tele_bot, database, shutdown_flag):
00010         """!
00011         Esta clase se encarga de enviar y de controlar el sensor de gas
00012
00013         @param gas_sensor El sensor de gas
00014         @param led_status El led a usar
00015         @param tele_bot El bot de telegram
00016         @param database El manejador de la base de datos
00017         @param shutdown_flag La bandera de sañales
00018
00019         @return Una instancia del sensor de gas
00020         """
00021         Thread.__init__(self)
00022         self.shutdown_flagshutdown_flag = shutdown_flag
00023
```

```
00024         self.gas_sensorgas_sensor = gas_sensor
00025         self.led_statusled_status = led_status
00026         self.tele_bottele_bot = tele_bot
00027         self.__conn__conn = database
00028         self.__contador__contador = 10
00029 
00030     def sensor_listener(self):
00031         """!
00032         Rutina en bucle que esta a la escucha del sensor, la envia un mensaje a
00033         los usuarios dados de alta en la BD mediante telegram. También enciende
00034         y apaga un led si hay gas, humo u gasolina cerca.
00035         """
00036         while True:
00037             sleep(5)
00038             print(self.gas_sensorgas_sensor.value)
00039             print(self.__contador__contador)
00040             if (self.gas_sensorgas_sensor.value  == 0 and self.__contador__contador >= 10):
00041                 self.__contador__contador = 0
00042                 records = self.__conn__conn.ejecutar_consulta(
00043                     """SELECT chat_id FROM alta_notificaciones where
00044                     humo is true""")
00045                 self.led_statusled_status.blink(on_time=0.1,off_time=0.1,n=700)
00046                 if records:
00047                     for chat_id in records:
00048                         print("humo")
00049                         self.tele_bottele_bot.send_message(chat_id, emojize("""
00050                         Se detecta humo o gas cerca de tu puerta :fuego:
00051                         """, language='es'))
00052             if self.__contador__contador < 10:
00053                 self.__contador__contador = self.__contador__contador + 1
00054             else:
00055                 pass
00056 
00057     def run(self):
00058         """!
00059         Ejecuta en un hilo el sensor de gas
00060         """
00061         self.sensor_listenersensor_listener()
```

5.15. Lector.py

```
00001 #!/usr/bin/env python3
00002 import time
00003 from os import popen
00004 from threading import Thread
00005 from pyfingerprint.pyfingerprint import PyFingerprint
00006 
00007 class Lector(Thread):
00008     def __init__(self,  led_status_r, led_status_g, tele_bot, servo, tty=None):
00009         """!
00010         Esta clase se encarga de realizar las acciones del lector de huella
00011         digital como:
00012         - Dar de alta la huella
00013         - Borrar Huella
00014         - Verificar existencia de la huella
00015         - Abrir cerradura si la huella coincide con la base de datos
00016 
00017         @param led_status_r Instancia de led de estado roja
00018         @param led_status_g Instancia de led de estado verde
00019         @param tele_bot Bot de telegram
00020         @param servo Servomotor a controlar
00021         @param tty Puerto serial USB de lector de huella, por defecto es /dev/ttyUSBX
00022         """
00023         Thread.__init__(self)
00024         cmd = 'ls /dev/ | grep ttyUSB'
00025         if not tty:
00026             self.ttytty = '/dev/' + popen(cmd).read().replace("\n","")
00027         else:
00028             self.ttytty = tty
00029         self.fingerfinger = PyFingerprint(self.ttytty, 57600, 0xFFFFFFFF, 0x00000000)
00030         self.led_status_rled_status_r = led_status_r
00031         self.led_status_gled_status_g = led_status_g
00032         self.tele_bottele_bot = tele_bot
00033         self.servoservo = servo
00034         # self.message = message
00035 
00036     def guardar_huella(self):
00037         """!
00038         metodo para salvar huella en la memoria del lector
00039         """
00040         count = 10
00041         self.led_status_gled_status_g.on()
00042         while (count <= 0 and not self.fingerfinger.readImage()):
```

```
00043             time.sleep(0.5)
00044             count = count - 1
00045         self.led_status_gled_status_g.off()
00046         self.fingerfinger.convertImage(0x01)
00047 
00048         result = self.fingerfinger.searchTemplate()
00049         positionNumber = result[0]
00050 
00051         if ( positionNumber >= 0 ):
00052             self.led_status_gled_status_g.blink(on_time=0.2, off_time=0.2, n=3)
00053             return 'Ya esta registrada de la huella'
00054 
00055         self.led_status_gled_status_g.off()
00056         time.sleep(2)
00057         self.led_status_gled_status_g.on()
00058         count = 10
00059         self.led_status_gled_status_g.on()
00060         while (count <= 0 and not self.fingerfinger.readImage()):
00061             time.sleep(0.5)
00062             count = count - 1
00063         self.led_status_gled_status_g.off()
00064         self.fingerfinger.convertImage(0x02)
00065 
00066         if ( self.fingerfinger.compareCharacteristics() == 0 ):
00067             return 'No son la misma huella, intentelo de nuevo...'
00068 
00069         self.fingerfinger.createTemplate()
00070         positionNumber = self.fingerfinger..storeTemplate()
00071         return 'Se registro la huella exitosamente'
00072 
00073     def verificar_huella(self):
00074         """!
00075         Metodo que verifica la existencia de la huella
00076         """
00077         count = 10
00078         self.led_status_gled_status_g.on()
00079         while (count <= 0 and not self.fingerfinger.readImage()):
00080             time.sleep(0.5)
00081             count = count - 1
00082         self.led_status_gled_status_g.off()
00083         self.fingerfinger.convertImage(0x01)
00084         result = self.fingerfinger.searchTemplate()
00085 
00086         positionNumber = result[0]
00087         accuracyScore = result[1]
00088 
00089         if ( positionNumber == -1 ):
00090             return "Verificación fallida"
00091         else:
00092             return "Verificación confirmada"
00093 
00094     def controlar_puerta(self):
00095         """!
00096         Metodo que controla la puerta si se coincide la huella con alguna de la
00097         BD del lector
00098         """
00099         while True:
00100             print('Waiting for finger...')
00101 
00102 
00103             while ( self.fingerfinger.readImage() == False ):
00104                 pass
00105 
00106 
00107             self.fingerfinger.convertImage(0x01)
00108 
00109 
00110             result = self.fingerfinger.searchTemplate()
00111             positionNumber = result[0]
00112 
00113             if ( positionNumber >= 0 ):
00114                 self.led_status_gled_status_g.blink(on_time=0.25, off_time=0.25, n=3)
00115                 if self.servoservo.value == -1.0:
00116                     self.servoservo.max()
00117                 else:
00118                     self.servoservo.min()
00119             else:
00120                 self.led_status_rled_status_r.blink(on_time=0.25, off_time=0.25, n=3)
00121             time.sleep(3)
00122 
00123     def run(self):
00124         """!
00125         Ejecuta el escucha del lector en un hilo de ejecucion
00126         """
00127         self.controlar_puertacontrolar_puerta()
```

5.16. main.py

```
00001 #!/usr/bin/env python3
00002
00003 import signal
00004 from core.Core import Core
00005
00006 def service_shutdown(signum, frame):
00007     """
00008     Intento de ontrolar señales
00009     """
00010     print('Deteniendo el proceso (%d)' % signum)
00011     raise Exception()
00012
00013
00014 def main():
00015     """
00016     Función principal que inicia el programa
00017     """
00018     signal.signal(signal.SIGTERM, service_shutdown)
00019     signal.signal(signal.SIGINT, service_shutdown)
00020     try:
00021         while True:
00022             core = Core()
00023             core.start()
00024             print('inicio')
00025
00026     except Exception:
00027         core.shutdown_flag.set()
00028         core.join()
00029         exit(0)
00030
00031
00032 if __name__=='__main__':
00033     main()
```

Índice alfabético