Bluetooth API - Documentation

Last revision : 01/02/2024

Table of contents

Bluetooth API

Introduction

The Bluetooth API is a programming interface that allows developers to establish a wireless connection between the ***aRdent smart glasses*** and other Bluetooth-enabled devices such as a computer, smartphone, or portable keypad. This API provides a simple and efficient way to transfer data between these devices, allowing computing power and software functionality to be offloaded to an external device.

Using the Bluetooth API, developers can create apps that communicate with the aRdent smart glasses, providing users with relevant information while they work. Data can be transmitted in textual or visual form, depending on the intended application.

This document describes in detail the various functions and methods of the Bluetooth API, along with instructions for using them in any Bluetooth-enabled programming language.

Prerequisites

Before you start using the Bluetooth API for aRdent smart glasses, you must ensure that you have the following prerequisites:

- A Bluetooth-enabled device that supports Bluetooth version 4.0 or later.
- An operating system compatible with Bluetooth 4.0 or later (for example, Windows 8 or later, macOS 10.10 or later, Android 4.3 or later, iOS 7 or later).
- A basic understanding of programming and Bluetooth communication concepts.

If you need help obtaining any of these prerequisites, please consult the documentation for your device or operating system, or contact your device vendor's technical support.

Connection

The aRdent smart glasses only work in ***peripheral mode***, which means they cannot initiate a Bluetooth connection themselves. It is therefore up to your device to detect the glasses and initiate the connection. You don't need to pair them to establish a connection, which means you can connect them even if they've never been connected to your device before.

To establish a connection with the aRdent glasses, you must use the MAC address of the glasses. To enable two-way communication between the two devices, the connection must be established using the ***Generic Attribute Profile (GATT)*** Bluetooth Low Energy (BLE). Using this profile, you can exchange data with aRdent glasses.

The aRdent glasses have a public Bluetooth name which is `GYW aRdent`. However, on some devices they may appear under the generic name `bluenrg!` Bluetooth BLE. This can be useful to know if you are unable to detect glasses with their public name.

Communication

Bluetooth communication between your device and aRdent smart glasses is via the Bluetooth Low Energy (BLE) GATT (Generic Attribute Profile). BLE is a lighter version of classic Bluetooth, designed for applications requiring low power consumption, such as wearable devices.

GATT is a profile specification for BLE communications, which defines a hierarchical data structure for ***services*** and ***characteristics***. Services are collections of related characteristics that can be used to perform a particular function, while characteristics are individual data objects that contain specific information. Figure 2 is a diagram of a profile meeting this specification.

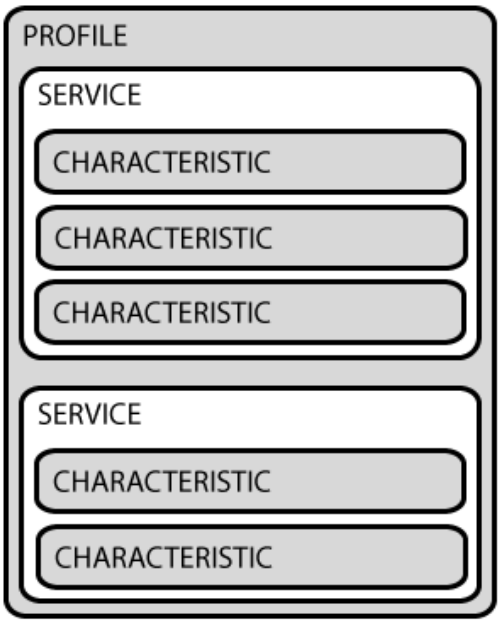


Figure 2 : Profil GATT

To communicate with aRdent smart glasses, your device must establish a BLE GATT connection with the aRdent device. Once connected, your device can query the services and characteristics available on aRdent glasses to retrieve information, update it, or send commands.

Display

Bluetooth communication with aRdent smart glasses for information display is based on a single service containing two distinct characteristics. The first one is dedicated to sending ***data*** to the glasses, while the second is used to ***control*** how this data is used and displayed.

For example, to send text to display on the glasses, you must write the text on the characteristic linked to the data. Then, you must send a control instruction to the characteristic dedicated to the control to indicate that text must be displayed and

describe how it must be displayed. This principle is used for all display features available with aRdent smart glasses.

API Features

Principle

The display features on aRdent smart glasses are managed by a single service called *Display Service*, which contains two characteristics: the *Data Characteristic* and the *Control Characteristic*. Table 1 below summarizes their properties.

	UUID
Display Service	`9f3443f3-5149-4d53-9b92-35def7b82e51`
Control Characteristic	`9f3443f3-5149-4d53-9b92-35def7b82e52`
Data Characteristic	`9f3443f3-5149-4d53-9b92-35def7b82e53`

Table 1 : Service properties and characteristics for display

To perform a display operation on the screen, it is necessary to write data on the *Data Characteristic*, then describe what to do with this data on the *Control Characteristic*. However, some operations, such as turning on or resetting the screen, do not require data and it is then possible to omit sending data on the *Data Characteristic*.

Sending data

Sending data is an essential step to display information on smart glasses. In Bluetooth Low Energy (BLE), the maximum size of a data packet is ***20 bytes***. This is why it is necessary to divide the data you wish to send into blocks of 20 bytes.

In terms of characteristics, the *Control Characteristic* does not accept more than 20 bytes per instruction. On the other hand, the *Data Characteristic* can accept up to 4096 bytes of data. However, it is necessary to transmit them in blocks of 20 bytes or less.

It is important to note that as long as nothing is sent to the *Control Characteristic*, new data transmitted on the *Data Characteristic* is added to the end of that previously sent. An instruction on the *Control Characteristic* always resets this data buffer.

Examples

The examples provided in this section are made in Python with the bleak library. This library is based on a `BleakClient` which controls the connection and communication between BLE devices.

Example 1 shows how to send the value `0x01` (1 in hexadecimal) to the *Control Characteristic*.

Turning on the screen

Before you can display information on the glasses screen, it is necessary to ***turn on this screen manually***. To do this, simply send the control code `0x01` to the *Control Characteristic*. No data is needed for this operation, so the *Data Characteristic* is not used.

It is generally recommended to wait 500 milliseconds after sending the command to allow time for the screen to light up correctly.

Example 1 corresponds to the Python code used to turn on the screen.

```python
async with BleakClient(MAC_ADDRESS) as client:
    # Write [0x01] on the Control characteristic
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x01]),
    )
```

Example 1 : Python code to turn on the screen

Resetting the screen

The smart glasses screen can be reset using the control code `0x05` and by sending an RGBA8888 color on the data characteristic.

This method removes everything and clears the screen with a color. For clearing just a part of the screen, rectangles can be used instead.

Example 2 shows how to fill the screen in white.

```python
async with BleakClient(MAC_ADDRESS) as client:
    # Clear Display and set a white screen
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x05, 0xFF, 0xFF, 0xFF, 0xFF]),
    )
```

Example 2 : Python code for clearing the screen in white

Displaying text

To ***display text*** on the screen, you must first write the text you want to display on the *Data Characteristic* encoded in UTF-8. Once all text is sent, the control instruction can be written to the *Control Characteristic*.

This instruction is composed of the following attributes:

1 byte	2 bytes	2 bytes	5 bytes	1 byte	4 bytes
`0x03`	Horizontal position	Vertical position	Font name	Font size	Color

- Positions are expressed in pixels from the left for horizontal, and from the top for vertical. These positions are *signed* and encoded in *Little Endian*.
- Font name can be one of the following:

	Font name
Roboto Mono Regular	`robmn`
Roboto Mono Bold	`robmb`
Roboto Mono Italic	`robmi`
Roboto Mono Bold Italic	`robme`

- Font size must be a non-null single byte positive number.
- Color is an RGBA8888 value.

Newlines are not supported. The management of lines of text is left to the application. This means that to send text on two lines, you must necessarily perform this display operation twice.

The font used is always `Roboto Mono`. The character size at font size 18 is 10 x 25 px. The character size is always directly proportional to the font size. If you double the font size, you also double the character size in each dimension.

Example 3 shows how to do this in Python.

```
async with BleakClient(MAC_ADDRESS) as client:
    # Write text on Data Characteristic
    text = "Hello World"
    data = bytes(text, 'utf-8')
    # Split data into chunks of 20 bytes
```

```
    i = 0
    while i < len(data):
         await client.write_gatt_char(
             data_characteristic
             data[i:i + 20],
         )
         i += 20

    # Wait for the data to be treated
    time.sleep(0.1)

    # Text parameters
    x = 100
    y = 200
    font_name = "robmn"
    font_size = 30
    color = 0x000000ff

    # Send control
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x03]) +
        x.to_bytes(2, 'little', signed=True) +
        y.to_bytes(2, 'little', signed=True) +
        font_name.encode("utf-8") +
        font_size.to_bytes(1, "little") +
        color.to_bytes(4, "big"),
    )
```

Example 3 : Python code to display "Hello World" on screen in (100, 200)

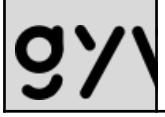
Displaying icons

The smart glasses also have a series of SVG ***icons*** that can be displayed on the screen. To do this, the first step is to write the name of the icon on the *Data Characteristic*. Table 3 lists all the icons available on the device. Predefined icon names are always less than 20 bytes long which allows this data to be sent in one go, without needing to divide it into blocks of 20 bytes or less. Once the file name is sent, it is possible to send a control instruction on the *Control Characteristic* to display the icon.

This instruction is composed of the following attributes:

1 byte	2 bytes	2 bytes	4 bytes	1 bytes
`0x02`	Horizontal position	Vertical position	Color	Scale

- Positions are expressed in the same way as for text.
- Color is an RGBA8888 value.
- Scale is a multiplication factor that will make the icon larger or smaller. A scale of 1 will always display a 48x48 icon. It can range from 0.01 to 13.7, but it has to be converted into a single byte, which Example 4 shows how to do.

	build		nfc		key_0
	camera		person		key_1
	chat		prev		key_2
	check		rename		key_3
	cloud_backup		right		key_4
	cloud_done		settings		key_5
	done		uncheck		key_6
	down		up		key_7
	edit		warning		key_8
	file		wifi_off		key_9
	folder		wifi		key_A
	gyw				key_B

	help				key_C
	info				key_D
	left				key_star
	location				key_#

Table 3 : Available icons

Example 4 shows how to display the icon "**down**".

```
async with BleakClient(MAC_ADDRESS) as client:
    # Write icon filename on Data Characteristic
    icon = "down.svg"
    await client.write_gatt_char(
         data_characteristic,
         bytes(icon, 'utf-8'),
    )

    # Wait for the data to be treated
    time.sleep(0.1)

    # Position of the icon element
    x = 50
    y = 100
    color = 0x000000ff
    scale = 2.5

    def clamp(n, smallest, largest):
        """Clamp a value between two bounds."""
        return max(smallest, min(n, largest))

    def encode_scale(scale):
        """Encode the scale into a single byte."""
        scale = clamp(scale, 0.01, 13.7)
        if scale >= 0:
            byte = round((scale - 1.0) * 10.0)
        else:
            byte = round(-scale * 100.0)

        return byte.to_bytes(1, “little”, signed=True)
```

```
    # Send control
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x02]) +
        x.to_bytes(2, 'little', signed=True) +
        y.to_bytes(2, 'little', signed=True) +
        encode_scale(scale) +
        color.to_bytes(4, “big”)
    )
```

Example 4 : Python code to display a down arrow at (50, 100)

Displaying ***custom images*** is also possible by sending the filename of your image copied on the glasses instead of sending a predefined icon name. Both SVG and PNG formats are supported. The default behavior is to fill the non-transparent parts of the image with the color supplied in the command, but if it's desirable to preserve the original image colors, pass a 32-bit 0 value as color (0x00, 0x00, 0x00, 0x00).

Drawing rectangles

Rectangles can be useful for drawing bullet points, separation lines, or clearing a part of the screen.

Drawing a colored rectangle is achieved by sending the following control instruction on the *Control Characteristic*:

1 byte	2 bytes	2 bytes	2 bytes	2 bytes	4 bytes
`0x0C`	Horizontal position	Vertical position	Width	Height	Color

- Positions are expressed in the same way as for text.
- Width and height are both unsigned positive integers encoded on 2 bytes.
- Color is an RGBA8888 value.

If you set the color to the value 0, the rectangle will have the same color as the background, so you can use it to clear a part of the screen.

Example 5 shows how to draw a rectangle.

```
async with BleakClient(MAC_ADDRESS) as client:
    x = 50
    y = 100
    width = 210
    height = 70
```

```
    color = 0x0000ffff

    # Send control
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x0C]) +
        x.to_bytes(2, 'little', signed=True) +
        y.to_bytes(2, 'little', signed=True) +
        width.to_bytes(2, “little”) +
        height.to_bytes(2, “little”) +
        color.to_bytes(4, “big”),
    )
```

Example 5 : Python code to display a blue rectangle at (50, 100)

Drawing spinners

Spinners are images that rotate creating an animation. They can be useful as progress indicators.

Displaying a spinner is achieved by first sending the name of the image that we want to use on the *Data Characteristic*. Currently, there is only one spinner predefined on the glasses, which is “**`spinner_1.svg`**”, however you can use any SVG image for that matter. Then we send the following control instruction on the *Control Characteristic* to display the spinner:

1 byte	2 bytes	2 bytes	4 bytes	1 byte	1 byte	1 byte
`0x0D`	Horizontal position	Vertical position	Color	Scale	Animation timing function	Spins per second

- Positions and color are expressed in the same way as for text.
- Scale is encoded the same way as for icons.
- Animation timing function is the ID of the animation curve that you want to use. Possible values are:
    - 0 = linear
    - 1 = ease-in
    - 2 = ease-out
- Spins per second:
    - Minimum is 0, encoded as 0
    - Maximum is 25.5, encoded as 255

A few important notes:

- Spinners will always be drawn on top of other drawings regardless if they were created before.
- Placing multiple spinners may break their rotation animation (a known bug).
- They may not display properly (having screen tearing) if they are positioned at the top half of the screen, or if they are too big, so prefer putting them on the bottom half and making them smaller.

Example 6 shows how to draw a spinner:

```
async with BleakClient(MAC_ADDRESS) as client:
    x = 50
    y = 100
    color = 0x0000ffff
    scale = 2.5
    animation_timing_function = 2  # ease-out
    spins_per_second = 3.7

    # Write spinner filename on Data Characteristic
    await client.write_gatt_char(
         data_characteristic,
         bytes(“spinner_1.svg”, 'utf-8'),
    )

    # Send control
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x0D]) +
        x.to_bytes(2, 'little', signed=True) +
        y.to_bytes(2, 'little', signed=True) +
        color.to_bytes(4, “big”) +
        encode_scale(scale) +  # the same function as for icon drawings
        animation_timing_function.to_bytes(1, “little”),
        int(spins_per_second * 10).to_bytes(1, “little”)
    )
```

Example 6 : Python code to display a blue spinner at (50, 100)

Set contrast and brightness

The default contrast and brightness settings should be good enough, but they can also be changed. The control code is `0x06` for contrast, and `0x07` for brightness. Both commands must be given 1 byte, the amount of contrast or brightness. 0 is the minimum, 255 is the maximum.

Example 7 shows how to change the contrast and brightness:

```
async with BleakClient(MAC_ADDRESS) as client:
    # Set contrast to 50%.
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x06, 128]),
    )
    # Set brightness to 100%.
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x07, 255]),
    )
```

Example 7 : Python code to change contrast and brightness.

Lock screen orientation

The screen display is automatically oriented in the correct direction, regardless of whether the user uses the smart glasses with the left or right eye. If we want the screen to keep its orientation, we must send an instruction on the *Control Characteristic*. This instruction is composed of 2 bytes, the control code **`0x02`** and a boolean indicating whether the orientation must be locked or not.

Example 8 shows how to lock and unlock rotation.

```
async with BleakClient(MAC_ADDRESS) as client:
    # Lock the screen rotation.
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x0A, True]),
    )
    # Unlock the screen rotation.
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x0A, False]),
    )
```

Example 8 : Python code to lock screen orientation.

Enable or disable backlight

Disabling the backlight will turn the screen off. The control code for backlight is **`0x0B`** and the command must be given a boolean, whether you want to turn the backlight on or off.

Example 9 shows how to change the backlight:

```
async with BleakClient(MAC_ADDRESS) as client:
    # Turn off the backlight.
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x0B, False]),
    )
    # Re-enable it.
    await client.write_gatt_char(
        control_characteristic,
        bytearray([0x0B, True]),
    )
```

Example 9 : Python code to change the backlight.

Control codes

Table 4 lists the control codes available and which can be used on the *Control Characteristic* to describe an operation.

Control code	Instruction
0x01	START_DISPLAY
0x02	DISPLAY_IMAGE
0x03	DISPLAY_TEXT
0x05	CLEAR_DISPLAY
0x06	SET_CONTRAST
0X07	SET_BRIGHTNESS
0x0A	LOCK_SCREEN_ROTATION
0X0B	ENABLE_BACKLIGHT
0x0C	DISPLAY_RECTANGLE
0x0D	DISPLAY_SPINNER

Table 4 : *Control Characteristic* control codes

Get Your Way

Company Description

Get Your Way (GYW) has developed aRdent, smart glasses designed to improve the comfort, safety and efficiency of employees in businesses. Thanks to a maximum simplification approach, aRdent is much simpler and more comfortable to use and implement than existing smart glasses on the market. The technology used is assisted reality (aR), a subcategory of augmented reality (AR) which makes it possible to assist operators while maintaining their attention on the main activity they are doing.

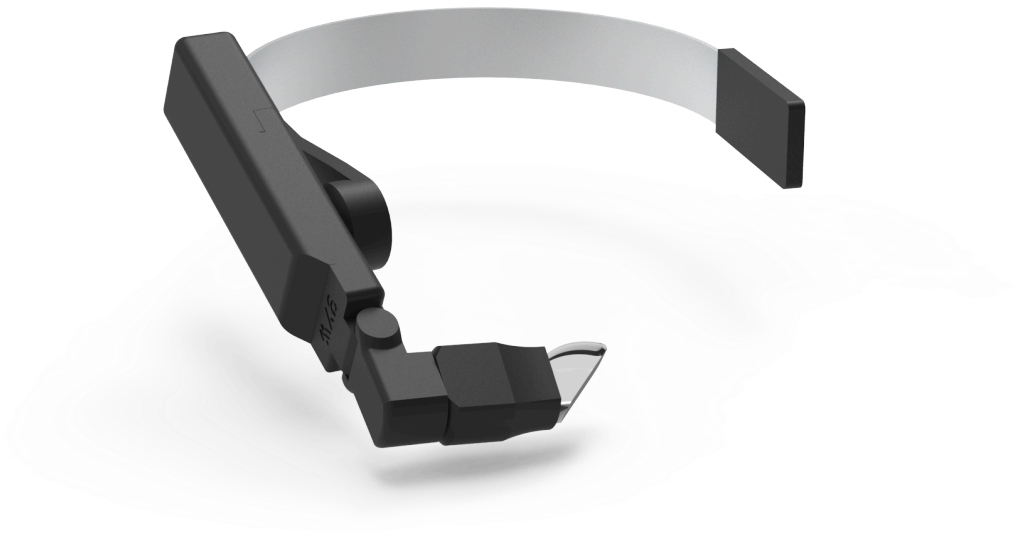


Figure 1 : aRdent smart glasses

The product (shown in Figure 1) takes the form of an optical module to be placed in users' peripheral field of vision to provide them with relevant information while they work. The data displayed can be textual or visual, depending on the intended application. aRdent comes with a Bluetooth API for quick and easy communication with other devices such as a computer, smartphone or portable keypad. The computing power and software functionalities can thus be deported and implemented on an external device.

Contact

For any questions, requests, or suggestions relating to the Bluetooth API of aRdent connected glasses, do not hesitate to contact Get Your Way by email at support@getyourway.be