`messaging` – Messaging

An EV3 Brick can send information to another EV3 Brick using Bluetooth. This page shows you how to connect multiple bricks and how to write scripts to send messages between them.

Pairing two EV3 Bricks

Before two EV3 bricks can exchange messages, they must be *paired*. You'll need to do this only the first time. First, activate bluetooth on all EV3 bricks as shown in Figure 19.

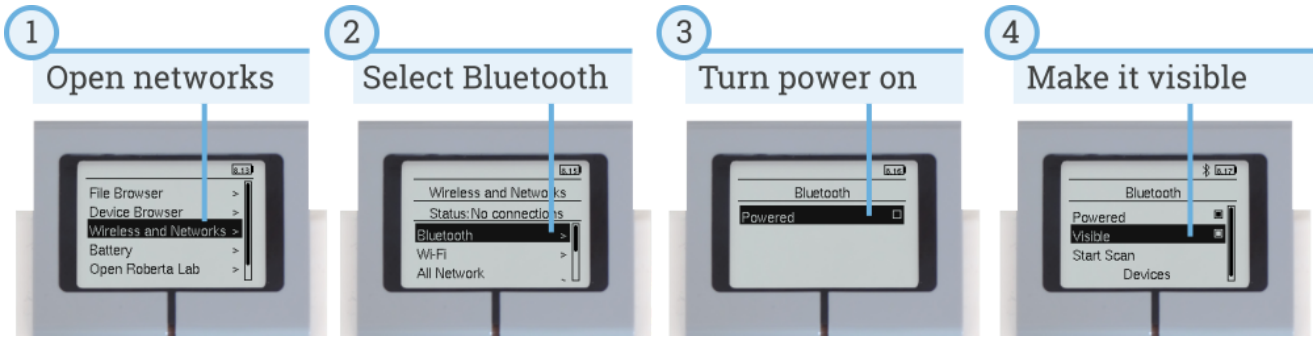

Figure 19 : Turn on Bluetooth and make Bluetooth visible.

Now you can make one EV3 Brick search for the other and pair with it, as shown in Figure 20.

Once they are paired, do *not* click *connect* in the menu that appears. The connection will be made when you run your programs, as described below.

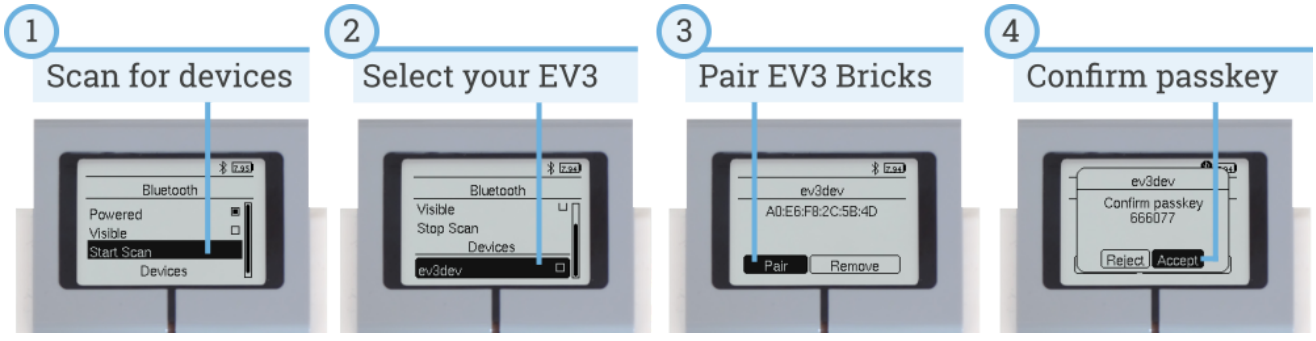

Figure 20 : Pairing one EV3 Brick to another EV3 Brick.

When you scan for Bluetooth devices, you'll see a list of device names. By default, all EV3 Bricks are named *ev3dev*. Click here to learn how to change that name. This makes it easy to tell them apart.

Repeat the steps in Figure 20 if you want to pair more than two EV3 Bricks.

Server and Client

A wireless network consists of EV3 Bricks acting as servers or clients. A example with one server and one client is shown in Figure 21. Messages can be sent in both ways: the server can send a message to the client, and the client can send a message to the server.

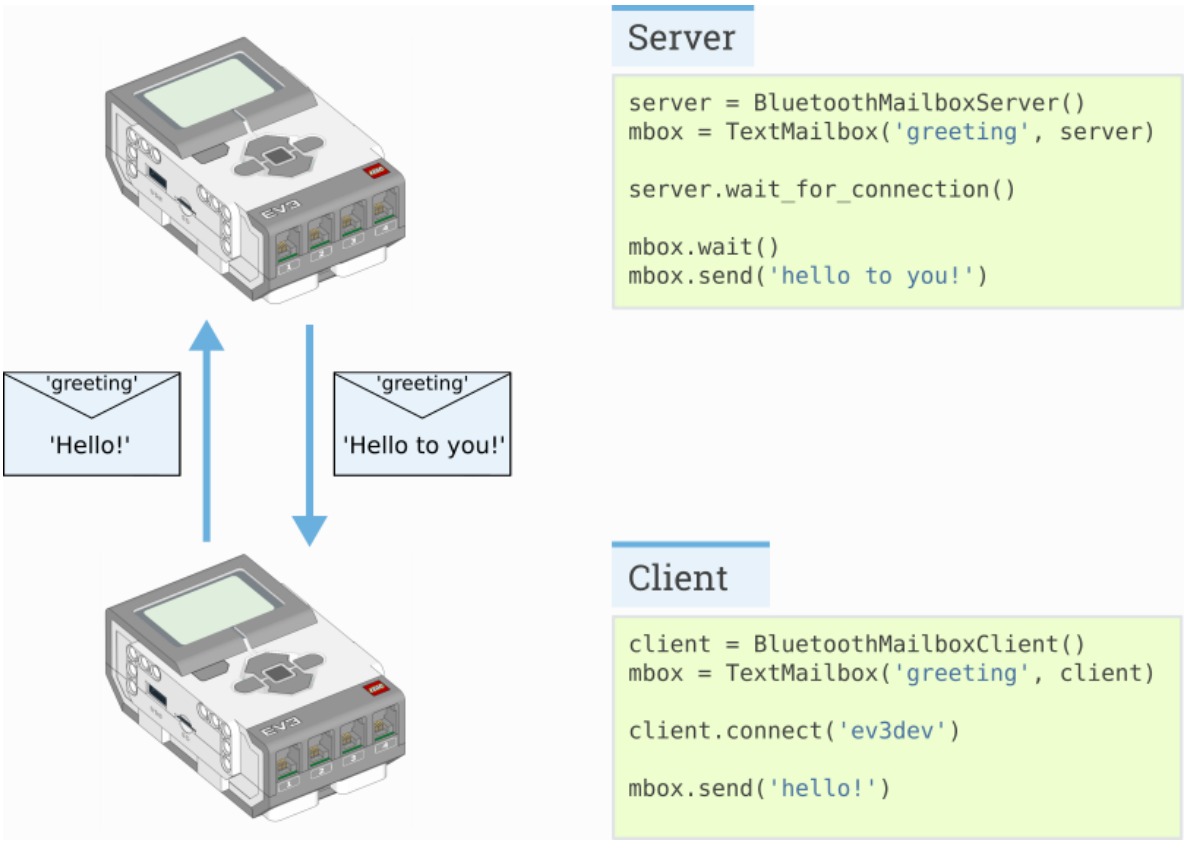

Figure 21 : An example network with one server and one clients.

Show/hide full server example ▲

Example: EV3 Bluetooth Server.

This is the full version of the excerpt shown in Figure 21.

```
#!/usr/bin/env pybricks-micropython

# Before running this program, make sure the client and server EV3 bricks are
# paired using Bluetooth, but do NOT connect them. The program will take care
# of establishing the connection.

# The server must be started before the client!

from pybricks.messaging import BluetoothMailboxServer, TextMailbox

server = BluetoothMailboxServer()
mbox = TextMailbox('greeting', server)

# The server must be started before the client!
print('waiting for connection...')
server.wait_for_connection()
print('connected!')

# In this program, the server waits for the client to send the first message
# and then sends a reply.
mbox.wait()
print(mbox.read())
mbox.send('hello to you!')
```

Show/hide full client example ▲

Example: EV3 Bluetooth Client.

This is the full version of the excerpt shown in Figure 21.

```
#!/usr/bin/env pybricks-micropython

# Before running this program, make sure the client and server EV3 bricks are
# paired using Bluetooth, but do NOT connect them. The program will take care
# of establishing the connection.

# The server must be started before the client!

from pybricks.messaging import BluetoothMailboxClient, TextMailbox

# This is the name of the remote EV3 or PC we are connecting to.
SERVER = 'ev3dev'

client = BluetoothMailboxClient()
mbox = TextMailbox('greeting', client)

print('establishing connection...')
client.connect(SERVER)
print('connected!')

# In this program, the client sends the first message and then waits for the
# server to reply.
mbox.send('hello!')
mbox.wait()
print(mbox.read())
```

The only difference between the client and the server is which one initiates the connection at the beginning of the program:

- The **server** must always be started first. It uses the `BluetoothMailboxServer` class. Then it waits for clients using the `wait_for_connection` method.
- The **client** uses the `BluetoothMailboxClient` class. It connects to the server using the `connect` method.
- After that, sending and receiving messages is done in the same way on both EV3 Bricks.

---

class **BluetoothMailboxServer**

Object that represents a Bluetooth connection from one or more remote EV3s.

The remote EV3s can either be running MicroPython or the standard EV3 firmware.

A "server" waits for a "client" to connect to it.

wait_for_connection(*count=1*)

Waits for a `BluetoothMailboxClient` on a remote device to connect.

Parameters: **count** (*int*) – The number of remote connections to wait for.

Raises: `OSError` – There was a problem establishing the connection.

close()

Closes all connections.

---

class **BluetoothMailboxClient**

Object that represents a Bluetooth connection to one or more remote EV3s.

The remote EV3s can either be running MicroPython or the standard EV3 firmware.

A "client" initiates a connection to a waiting "server".

connect(*brick*)

Connects to an `BluetoothMailboxServer` on another device.

The remote device must be paired and waiting for a connection. See `BluetoothMailboxServer.wait_for_connection()`.

Parameters: **brick** (*str*) – The name or Bluetooth address of the remote EV3 to connect to.

Raises: `OSError` – There was a problem establishing the connection.

close()

Closes all connections.

Mailboxes

Mailboxes are used to send data to and from other EV3 Bricks.

A Mailbox has a `name`, similar to the "subject" of an email. If two EV3 Bricks have a Mailbox with the same name, they can send messages between them. Each EV3 Brick can read its own Mailbox, and send messages to the Mailbox on the other EV3 Brick.

Depending on the type of messages you would like to exchange (bytes, booleans, numbers, or text), you can choose one of the Mailboxes below.

class **Mailbox**(*name*, *connection*, *encode=None*, *decode=None*)

Object that represents a mailbox containing data.

You can read data that is delivered by other EV3 bricks, or send data to other bricks that have the same mailbox.

By default, the mailbox reads and send only bytes. To send other data, you can provide an `encode` function that encodes your Python object into bytes, and a `decode` function to convert bytes back to a Python object.

Parameters:
- **name** (*str*) – The name of this mailbox.
- **connection** – A connection object such as `BluetoothMailboxClient`.
- **encode** (*callable*) – Function that encodes a Python object to bytes.
- **decode** (*callable*) – Function that creates a new Python object from bytes.

read()

Gets the current value of the mailbox.

Returns: The current value or `None` if the mailbox is empty.

send(*value*, *brick=None*)

Sends a value to this mailbox on connected devices.

Parameters:
- **value** – The value that will be delivered to the mailbox.
- **brick** (*str*) – The name or Bluetooth address of the brick or `None` to to broadcast to all connected devices.

Raises: `OSError` – There is a problem with the connection.

wait()

Waits for the mailbox to be updated by remote device.

wait_new()

Waits for a new value to be delivered to the mailbox that is not equal to the current value in the mailbox.

Returns: The new value.

class LogicMailbox(*name*, *connection*)

Object that represents a mailbox containing boolean data.

This works just like a regular `Mailbox`, but values must be `True` or `False`.

This is compatible with the “logic” mailbox type in EV3-G.

Parameters:
- **name** (*str*) – The name of this mailbox.
- **connection** – A connection object such as `BluetoothMailboxClient`.

class NumericMailbox(*name*, *connection*)

Object that represents a mailbox containing numeric data.

This works just like a regular `Mailbox`, but values must be a number, such as `15` or `12.345`

This is compatible with the “numeric” mailbox type in EV3-G.

Parameters:
- **name** (*str*) – The name of this mailbox.
- **connection** – A connection object such as `BluetoothMailboxClient`.

class TextMailbox(*name*, *connection*)

Object that represents a mailbox containing text data.

This works just like a regular `Mailbox`, but data must be a string, such as `'hello!'` or `'My name is EV3'`.

This is compatible with the “text” mailbox type in EV3-G.

Parameters:
- **name** (*str*) – The name of this mailbox.
- **connection** – A connection object such as `BluetoothMailboxClient`.

Making bigger networks

The classes in this module are not limited to just two EV3 Bricks. for example, you can add more clients to your network. An example with pseudo-code is shown in Figure 22.

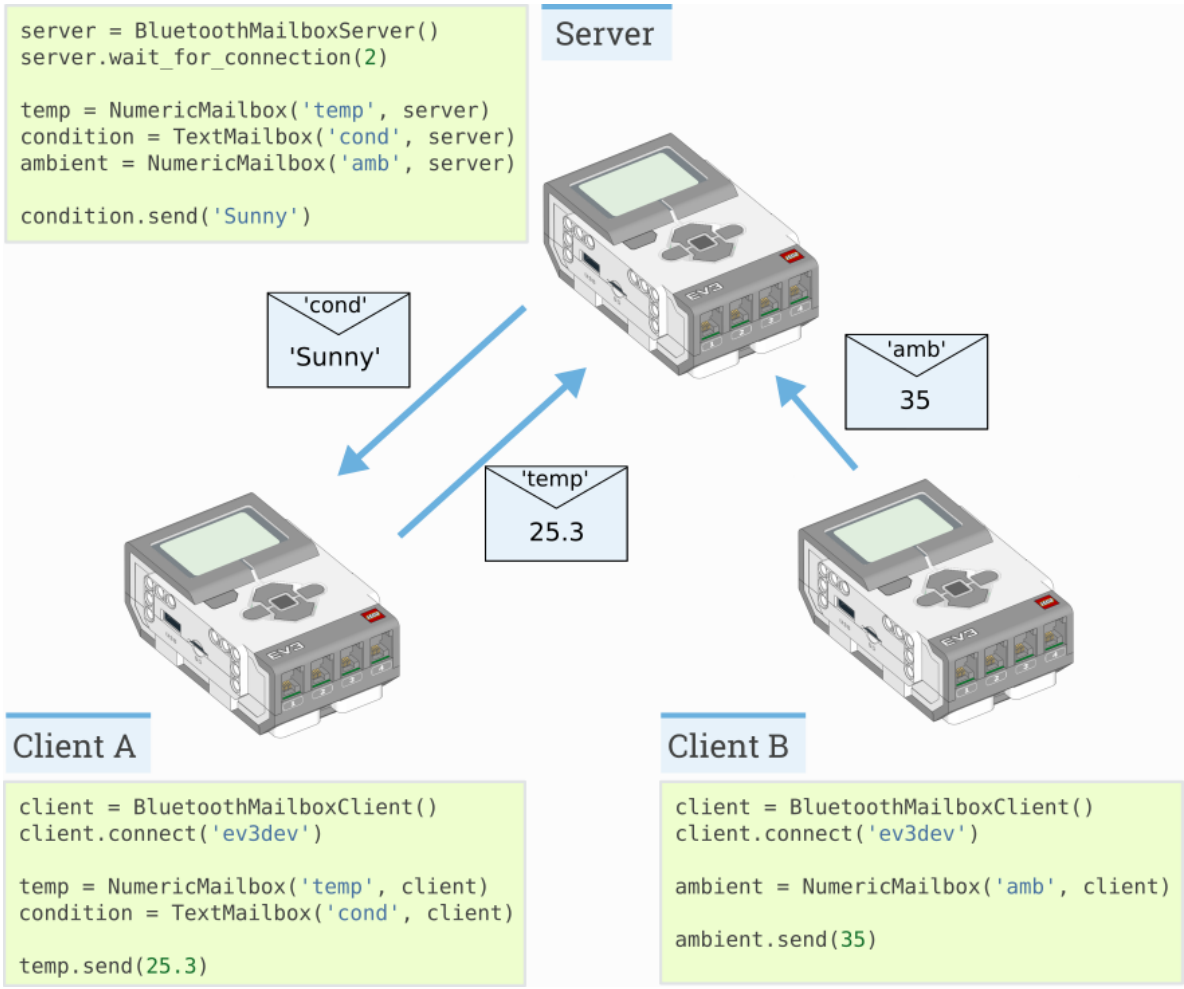

Figure 22 : An example network with one server and two clients.