The MicroPython environment exposes certain OS-level functions, primarily via the standard os module
However, not all commands from the MicroPython os library are implemented—especially in older firmware versions—
so the best way to check availability is to try executing the command in the Python REPL on your device.
import os
print(os.listdir())
This would list files in the device's filesystem, if supported by your firmware.
BLE Message Flow (Canvas BLE + UWB) :
The set of available OS functions may vary depending on your Canvas firmware version.
If a command is not implemented, you will receive an error. Always test commands directly on your device

```
Phone         <----->        NX040 Tag
  |                              |
  | Connect BLE                  |
  |----------------------------->|
  |                              | ble_conn_cb()
  |                              | -> set security level
  | Initialize (0x0A msg)        |
  |----------------------------->| qpp_rx_cb()
  |                              | -> initialize_android() or initialize_ios()
  | Receive TX (0x01)            |
  |<-----------------------------| gattserver.notify(..., "QPP TX", ...)
  | Configure (0x0B)             |
  |----------------------------->| -> configure_android()
  | Start UWB session            |
```

BLE Stack — Layer Breakdown :

```
+---------------------+      ← Your Python app code (e.g., GATT server)
|      Application    |
+---------------------+      ← Canvas API (canvas_ble module)
| Generic Attribute   | GATT — Read/write characteristics
+---------------------+
| Attribute Protocol  | ATT — Data exchange protocol
+---------------------+
| Generic Access      | GAP — Advertising, connection
+---------------------+
| Security Manager    | Pairing, bonding, encryption
+---------------------+
| Logical Link Control| L2CAP — Packet framing
+---------------------+
| Link Layer          | LL — BLE radio protocol
+---------------------+
| Physical Layer (PHY)| Actual radio signals on 2.4 GHz
```

Advertiser:

Method	Description
add_canvas_data()	Adds Canvas-specific BLE advertisement data (e.g., UWB support, provisioning info).
add_data()	Adds raw bytes to the advertisement payload.
add_ltv()	Adds standard BLE [Length-Type-Value] formatted data (used for most BLE fields like name, service UUIDs).
add_smp_uuid()	Adds the BLE SMP (Security Manager Protocol) UUID to indicate pairing support.
add_tag_string()	Adds a human-readable name string to the advertisement (like "UWB" or "TAG").
clear_buffer()	Clears either the advertising or scan response buffer. Takes a boolean: True for advertising buffer, False for scan response.
set_channel_mask()	Sets the BLE advertising channel mask (rarely used; controls which BLE channels 37/38/39 to use).
set_directed()	Makes the advertisement "directed" to a specific central device

(rare in typical Canvas use).
set_interval() Sets the advertisement interval range, in milliseconds. E.g. adv.set_interval(100, 200)
set_phys() Sets the PHY (1M, 2M, Coded) used during advertising.
set_properties() Sets advertisement properties like connectable, scannable, etc. Example: set_properties(True, True, False).
start() Begins BLE advertising with the current buffer.
stop() Stops BLE advertising.
update() Updates the active advertisement payload without restarting.
validate_data() Validates that advertisement buffer contents are valid BLE fields.

Connection:

Method	Description
change_security()	Requests a specific security level (e.g., encrypted or authenticated pairing). You typically call this after connecting.
delete_pairing()	Removes the stored pairing information for this connection. Use it when pairing fails or before retrying. Takes a bonded argument (True or False).
disconnect()	Disconnects the BLE connection with the central device (e.g., phone).
get_addr()	Returns the Bluetooth address of the connected device as a tuple or string.
get_rssi()	Gets the current signal strength (RSSI) of the connection, in dBm. Useful for proximity/quality checks.
set_security_cb()	Registers a callback to be called when security level changes (e.g., after pairing completes). Used to detect success/failure of pairing.

GattClient – Methods && Flags

Method	Description
discover()	Initiates service and characteristic discovery on the connected BLE peripheral. You must call this before reading or writing.
get_dict()	Returns the discovered services/characteristics in a Python dictionary format for easy access.
read(name)	Reads the value of a characteristic by its name (defined in the peripheral's GATT table).
write(name, data, flags)	Writes data to a characteristic by name. Flags determine how the write is done (acknowledged or not).
set_callback(callback)	Sets a callback function to be called when the client receives a notification or indication from the peripheral.
set_name(name)	Sets a name to identify this GATT client (used in logs/ debugging, optional).
subscribe(name, flags)	Subscribes to a characteristic to receive notifications or indications (based on flag).
FLAG_INDICATE	Subscribe with indications (reliable; requires acknowledgment from client).
FLAG_NOTIFY	Subscribe with notifications (fast, no ACK).
FLAG_READ	Used internally for read access.
FLAG_WRITE_ACK	Write with response (ACK from peripheral).
FLAG_WRITE_NO_ACK	Write without response (faster but no confirmation).

GattServer :

Method	Description
build_from_dict()	Initializes the GATT server using a Python dictionary (the

GATT table).
This defines services, characteristics, permissions, and callbacks.
indicate(name, data) Sends an indication to the connected central for the specified characteristic.
Indications require acknowledgment.
notify(connection, name, data) Sends a notification (no acknowledgment required) to the connected central for a given characteristic.
read(name) Returns the current value of a characteristic (by name).
write(name, data) Updates the local value of a characteristic on the server. Used internally or to reflect app state.
start() Starts the GATT server, making it ready to accept connections.
stop() Stops the GATT server, disconnecting any connected centrals.

EVENT_ATTR_VALUE Data was written to a characteristic by the central (write event).
EVENT_CCCD_BOTH Client enabled both notifications and indications.
EVENT_CCCD_INDICATE Client enabled only indications.
EVENT_CCCD_NOTIFY Client enabled only notifications.
EVENT_CCCD_NONE Client disabled all updates (notify + indicate).
EVENT_INDICATION_OK Indication was acknowledged by the client.
EVENT_INDICATION_TIMEOUT Indication timed out waiting for a client acknowledgment.

FLAG_READ Characteristic is readable.
FLAG_WRITE_ACK Writable with response (default behavior).
FLAG_WRITE_NO_ACK Writable without response (faster).
FLAG_NOTIFY Can send notifications to client.
FLAG_INDICATE Can send indications to client.

Scanner:

Method	Description
filter_add(type, value)	Adds a scan filter of a given type (e.g., name, UUID, manufacturer data). Only matching devices are reported.
filter_reset()	Clears all previously added scan filters.
set_phys()	Sets the PHY to use for scanning (1M, Coded PHY, etc.).
set_timing(window_ms, interval_ms)	Sets the scanning timing parameters — window is the active scan time per interval.
start(callback)	Begins scanning. The callback is called when a matching advertisement is found.
stop()	Stops BLE scanning.
FILTER_ADDR	Filter by advertiser address
FILTER_DATA	Raw data match
FILTER_MANUF_DATA	Filter by manufacturer-specific data
FILTER_NAME	Filter by advertisement name string
FILTER_UUID	Filter by UUID (usually a service UUID)
TYPE_CONNECTABLE	Connectable advertisement
TYPE_DIRECTED	Directed advertisement to a known peer
TYPE_EXTENDED	Extended advertisement
TYPE_LEGACY	Legacy advertisement (standard 31-byte payload)
TYPE_SCANNABLE	Scannable advertisement
TYPE_SCAN_RESPONSE	Scan response packet