Build Wireless Products Faster with Zephyr and MicroPython

16 April 2024

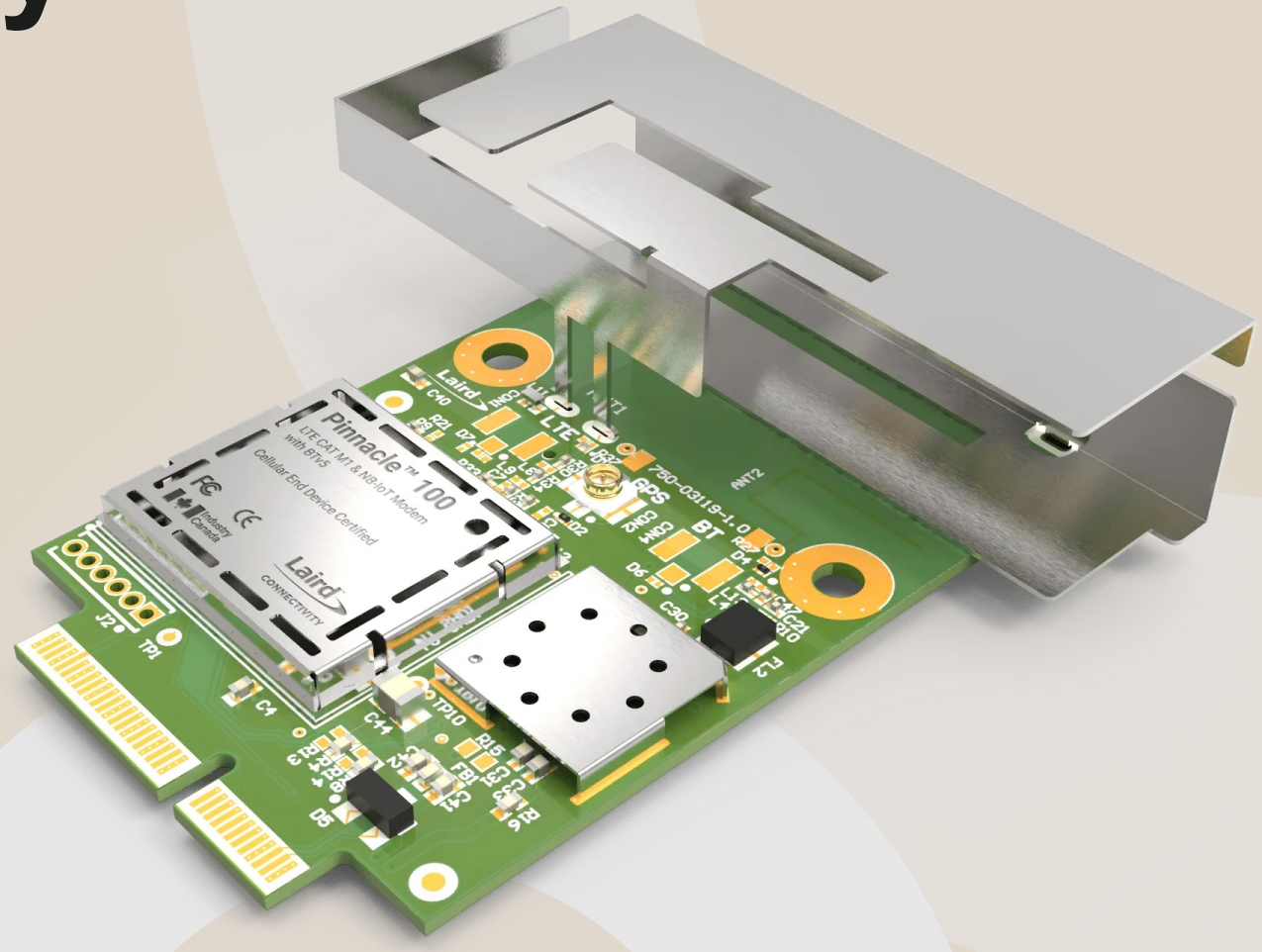
Presented by Ryan Erickson

Intro

Ryan Erickson has been developing wireless products at Ezurio for 15 years. His areas of expertise include Zephyr development and maintenance, Wi-Fi, Bluetooth LE, Bluetooth Classic, 802.15.4, cellular, IoT, manufacturing automation, mobile apps, and more.

Laird Connectivity is now ezurio

Ezurio is Your Connectivity Expert



https://www.ezurio.com/laird-connectivity

Agenda

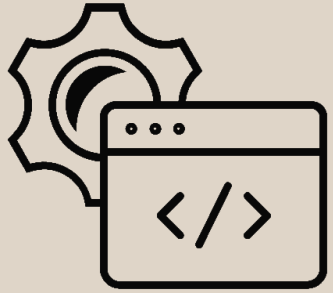
- Canvas Software Suite Overview
- Supported Products
- Canvas Software Suite Firmware Architecture
  - Zephyr Core
  - MicroPython Engine Integration

What is Canvas Software Suite?

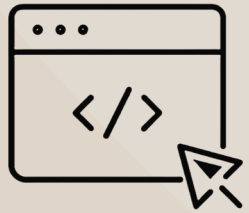
CANVAS SOFTWARE SUITE

SIMPLIFY AND ACCELERATE MODULE INTEGRATION

FIRMWARE



TOOLS



SAMPLES

DOCS

A Software Platform enabling rapid, portable development across our microcontroller-based wireless product portfolio.

CANVAS SOFTWARE SUITE - FIRMWARE

https://www.ezurio.com/canvas/software-suite/development-workflow#firmware

Install Canvas Firmware

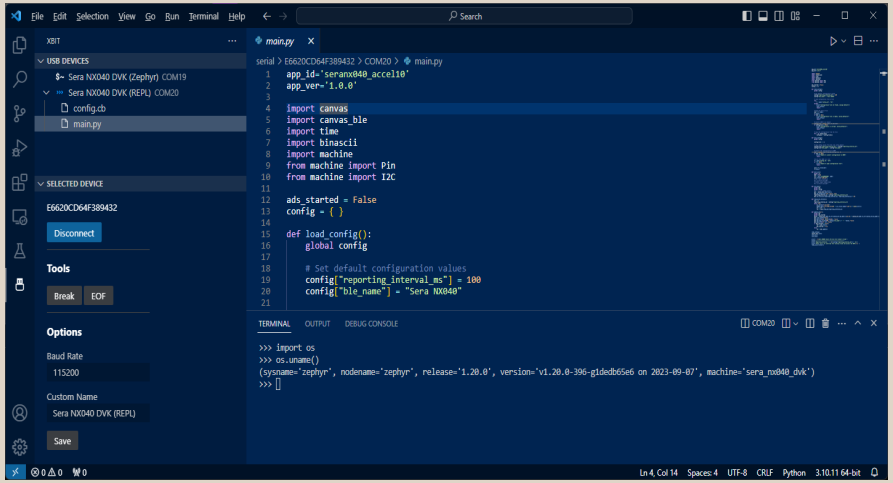
Our Canvas Firmware contains the bootloader, RTOS and middleware required to develop your application. This firmware must be loaded in order to support on-module Python scripts, AT command set operation and interoperability with desktop and mobile tools. Firmware can be programmed using the XBIT Desktop tool.

Product	Description	Download
Sera NX040 DVK	Canvas Firmware - Zephyr + Python	Download
Lyra 24P/S DVK	Canvas Firmware - FreeRTOS + Python	Download
MG100	Canvas Firmware - Zephyr + Python	Download
Pinnacle 100	Canvas Firmware - Zephyr + Python	Download
BL5340DVK	Canvas Firmware - Zephyr + Python	Download
BL654 USB Adapter	Canvas Firmware - Zephyr + Python	Download
Sentrius BT510 Sensor BETA	Canvas Firmware - Zephyr + Python	Download
BL654 DVK BETA	Canvas Firmware - Zephyr + Python	Download
Sentrius BT610 Sensor BETA	Canvas Firmware - Zephyr + Python	Download

CANVAS SOFTWARE SUITE - TOOLS

Xbit VS Code Extension

Create, Load and Interact with On-Device Python Code

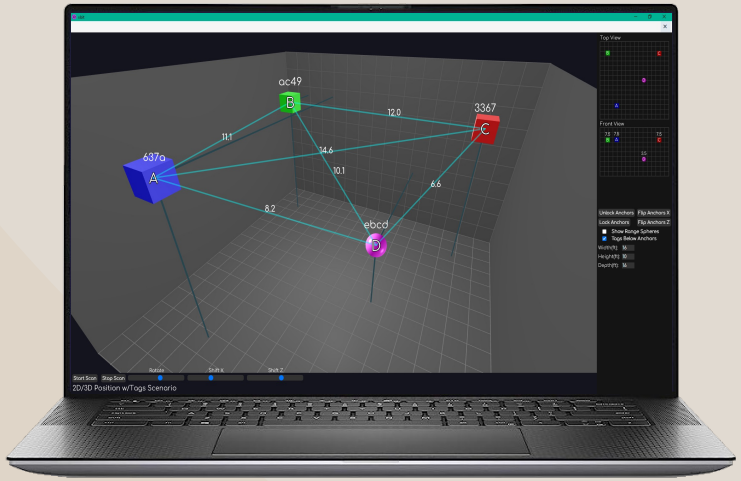


Develop Python scripts and load them onto devices directly from the industry's most popular source code editor.

Access the module's Python REPL to quickly test out APIs and prove out your application code.

Xbit Desktop App

Visualize Data and Interact with Canvas-enabled Devices

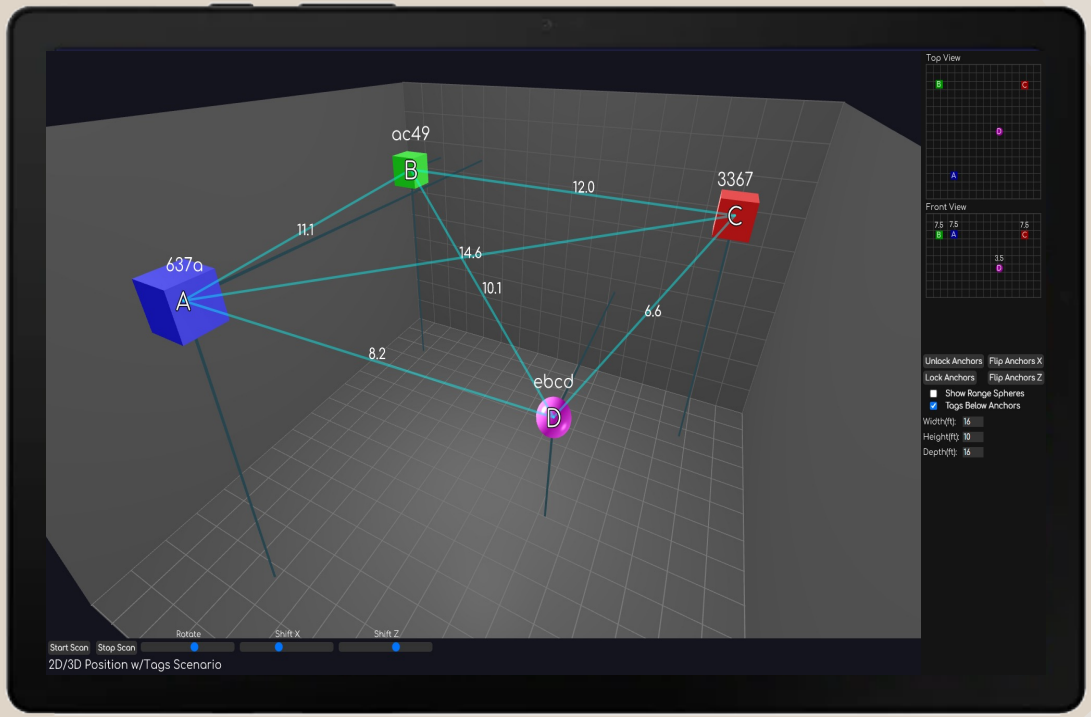


Our In-house designed Xbit Desktop application provides a cross-platform framework for advanced visualization of your device's data over Bluetooth Low Energy. Use one of our provided Applets or create your own with intuitive JavaScript APIs!

CANVAS SOFTWARE SUITE - TOOLS

Xbit Mobile App

Visualize Data and Interact with Canvas-enabled Devices

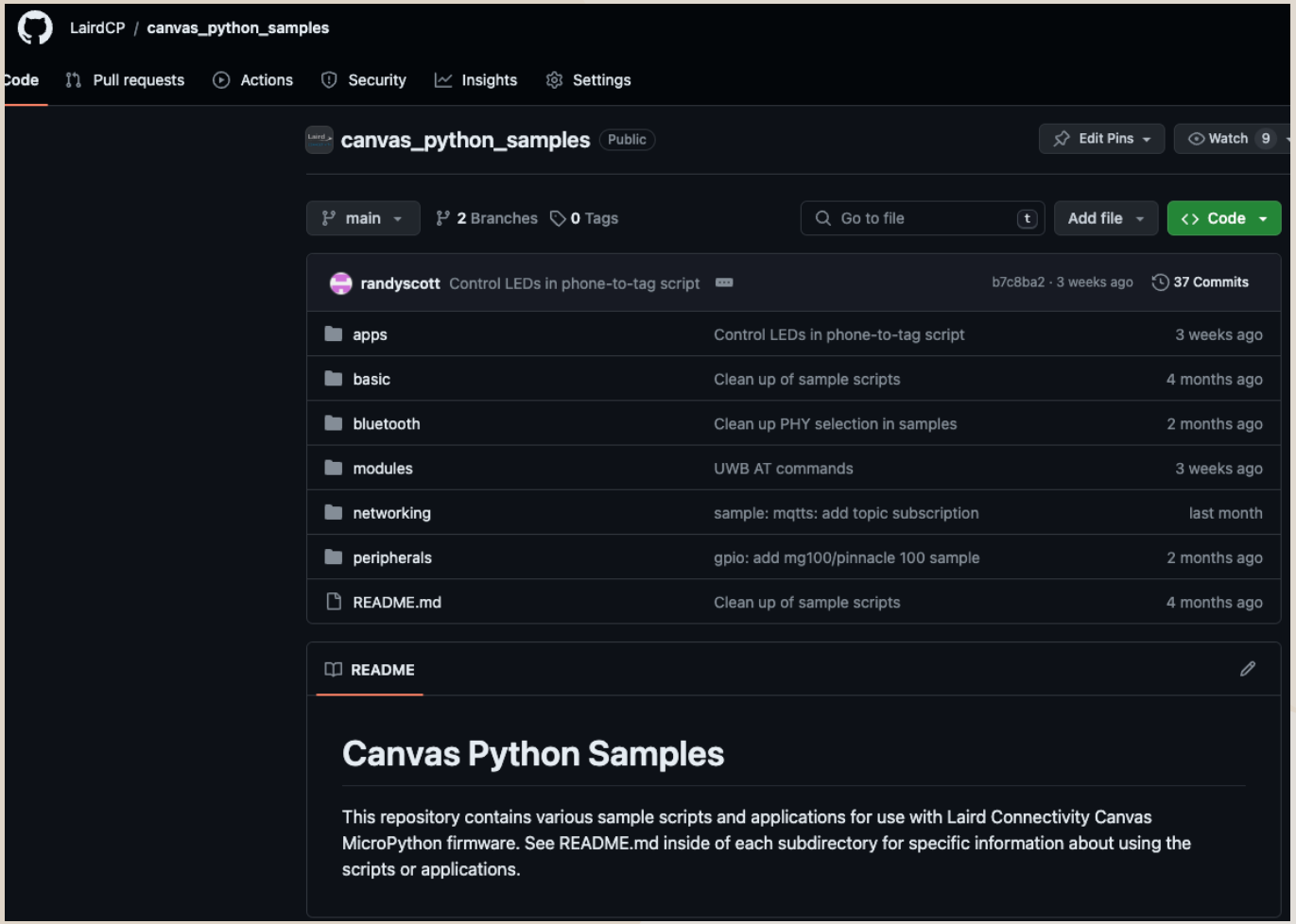


Xbit Mobile provides the same advanced visualization technology as Xbit Desktop. Applet code can be re-used for compatibility with both Desktop and Mobile versions of Xbit.

Target a tablet for a larger view or use a phone for simple workflows like OTA firmware update or device configuration.

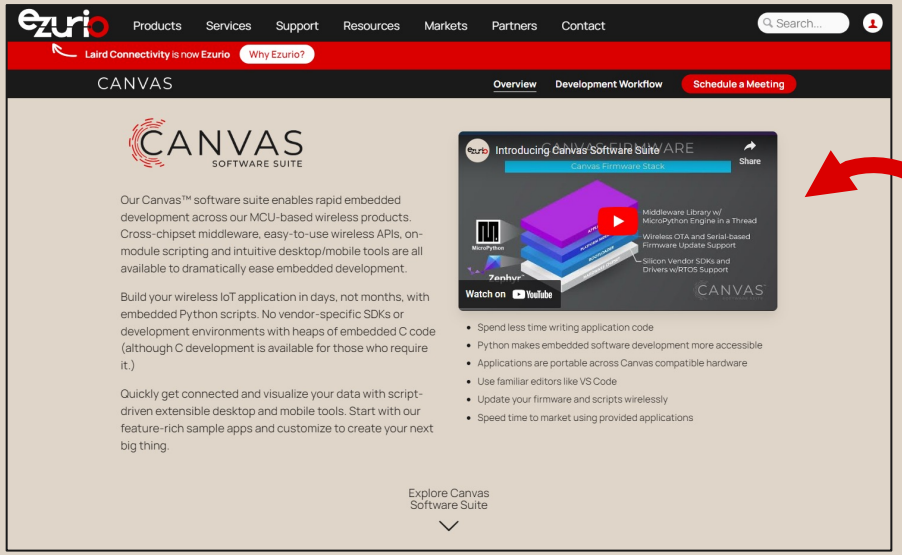
CANVAS SOFTWARE SUITE - SAMPLES

https://github.com/LairdCP/canvas_python_samples



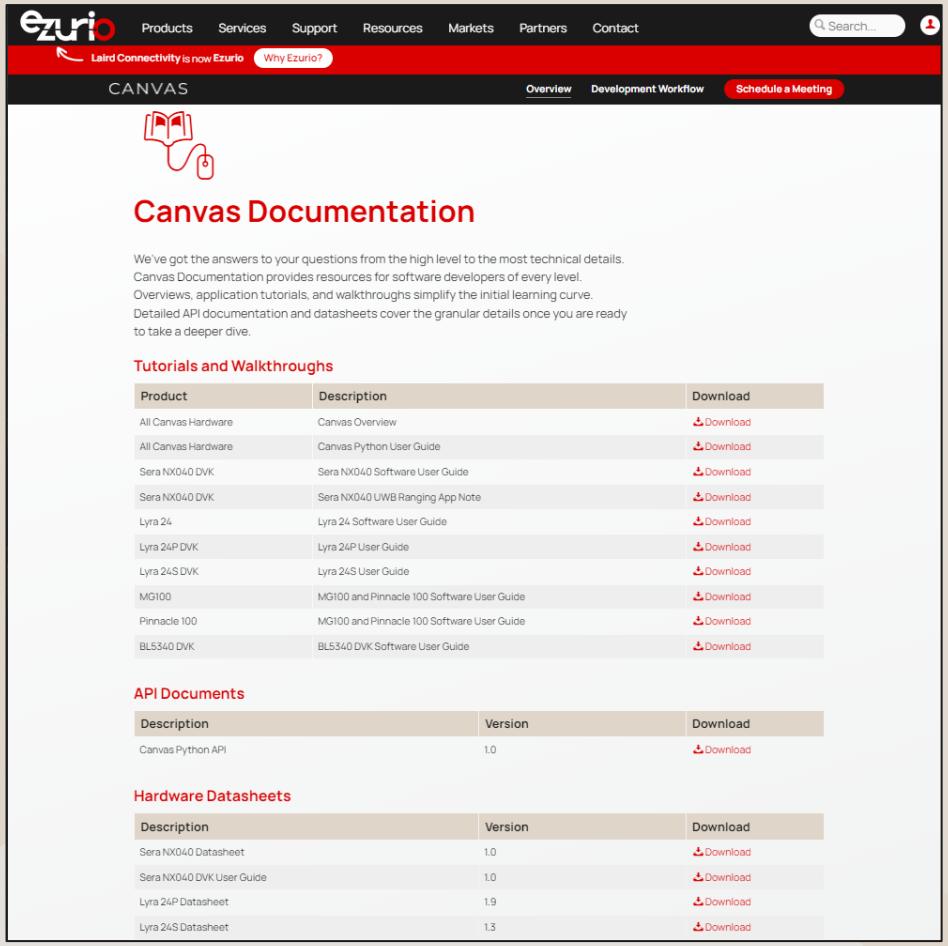
CANVAS SOFTWARE SUITE - DOCUMENTATION

CANVAS SOFTWARE SUITE WEBSITE > ezurio.com/canvas/software-suite



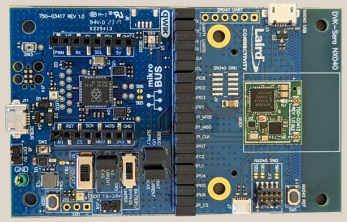
While you're here, watch this video!

Scroll Down Here…

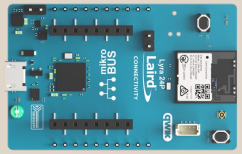


What Products are Supported?

CANVAS-ENABLED PRODUCTS

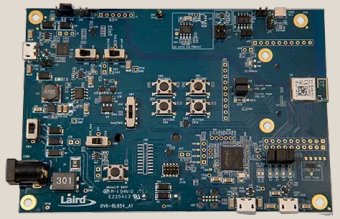


Sera™ NX040 (BLE + UWB)
PN: 453-00174-K1, 453-00175-K1



Lyra™ 24 Series (BLE)
PN: 453-00142-K1, 453-00145-K1, 453-00148-K1, 453-00170-K1

BL654 USB Adapter (BLE)
PN: 451-00004



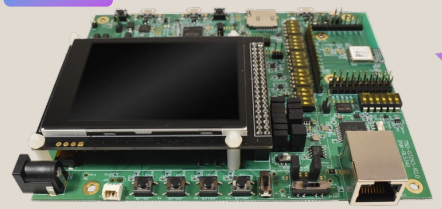
BL654 (BLE)
PN: 455-00001, 455-00002

Sentrius™ BT610 Sensor
PN: 450-00121-K1

Sentrius™ BT510 Sensor
PN: 455-00083, 450-00048B

BL653 and BL653μ (BLE)
PN: 453-00039-K1, 453-00041-K1

NEW!



BL5340 (BLE)
PN: 453-00052-K1, 453-00053-K1

NEW!

Sentrius™ MG100 Gateway
PN: 450-00011-K1, 450-00038-K1, 450-00039-K1, 450-00054-K1

NEW!

Pinnacle™ 100 Cellular LTE-M/NB-IoT/BLE Modem
PN: 453-00010-K1, 453-00011-K1

PLANNED

Sentrius™ BT710 Multi-Sensor
PN: 450-00122-K1, 450-00122B

Support Planned for Future MCU-based Wireless Modules and IoT Products

CANVAS SOFTWARE SUITE – SAMPLE APPS

Sera NX040 UWB

- PYTHON — **UWB Ranging Demo**
- XBIT — **Tag-to-Tag 3D Visualization Applet**

Lyra 24/BL65x BLE

- PYTHON — **Eddystone and iBeacon Samples**
- PYTHON — **BLE Advertiser and Scanner**
- PYTHON — **GATT Client**

Sentrius BT510

- PYTHON — **Tilt Sensor Demo**
- XBIT — **Tilt Sensor 3D Visualization Applet**

Sentrius MG100

- PYTHON — **BLE to MQTT Gateway**
- PYTHON — **LwM2M Device Management**
- PYTHON — **HTTP Client Sample**

BL654 USB Adapter

- PYTHON — **Xbit USB Adapter Script**
- XBIT — **Canvas Hub**
- XBIT — **(All BLE Applets)**

BL654 DVK

- PYTHON — **Button and LED Demo**
- PYTHON — **(All BLE Samples)**

BL5340 DVK

- PYTHON — **BLE to MQTT Gateway**
- PYTHON — **LwM2M Device Management**
- PYTHON — **HTTP Client Sample**

Click Board Samples
(All DVKs w/mikroBUS)

- PYTHON — **Temp and Hum Click Sample**
- PYTHON — **Weather Click**
- PYTHON — **Accel 10 Click**

Basic Samples
(All Devices)

- PYTHON — **Hello World Sample**
- PYTHON — **Device Sleep**
- PYTHON — **Timer Callbacks**

Peripheral I/O
(All Devices)

- PYTHON — **GPIO**
- PYTHON — **I2C**
- PYTHON — **SPI**

Find these and more at github.com/LairdCP/canvas_python_samples

Canvas Firmware Architecture

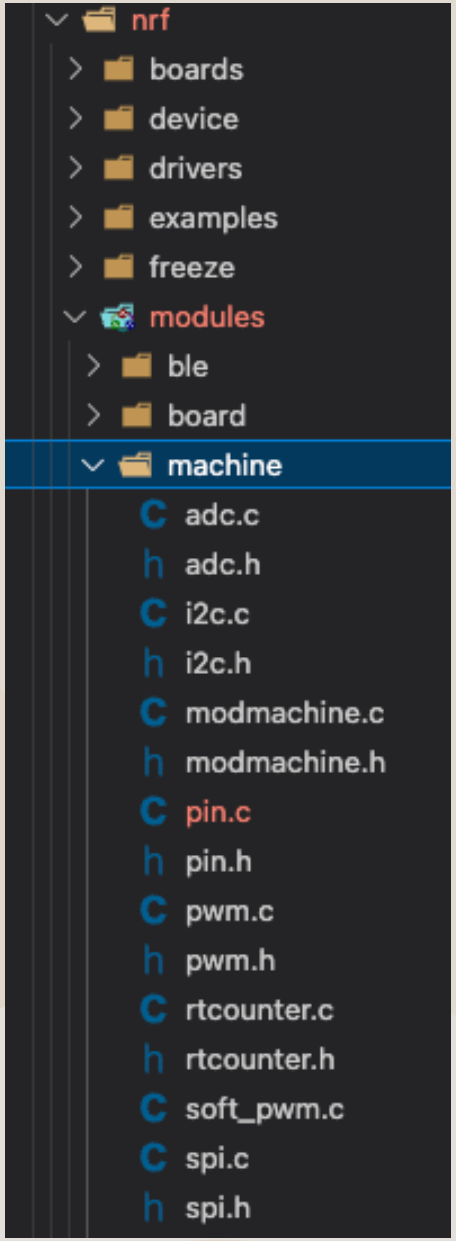
MicroPython

https://github.com/micropython/micropython

- An “implementation of Python 3.x on microcontrollers and small embedded systems.”
- Ports for different embedded systems
  - A port can be RTOS based or bare metal
- Provides source for core subsystems
  - File system, Networking, USB, etc.
  - Subsystems are based on other open-source projects
- Observations
  - Ports vary on “completeness” and maintenance
  - No firmware updates

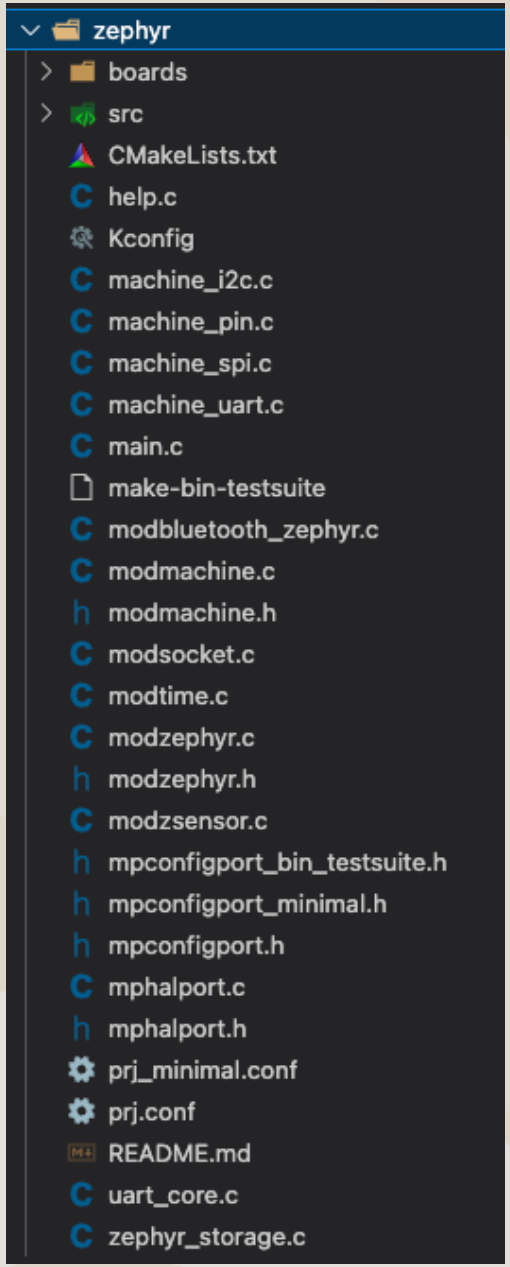
MicroPython nRF port

- Outdated
  - Bluetooth LE based on nRF5 SDK softdevice
- Not supported by vendor
  - No Bluetooth qualifications
- No RTOS benefits
- Manually add features yourself



MicroPython Zephyr Port

- Not up to date with Zephyr
- Does not leverage all Zephyr subsystems
  - Uses its own VFS
  - Limited Bluetooth LE features (lack of python glue)
    - No extended advertising/scanning
    - All PHYs not supported
- No bootloader/firmware update

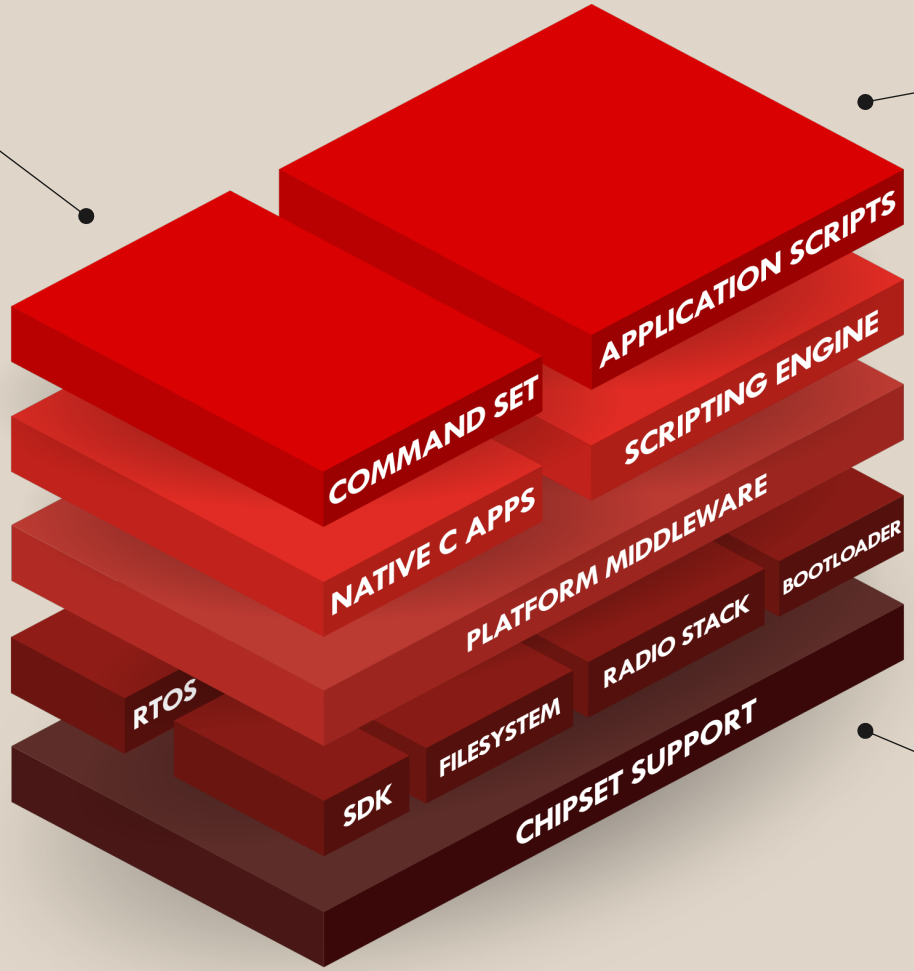


CANVAS PLATFORM FIRMWARE STACK

COMMAND SETS
AT and/or binary command sets make modules easy to integrate with a host

C-BASED APPLICATIONS
Applications requiring “native C” development can build upon middleware API to speed development cycles

STANDARD RTOS
Builds upon standard RTOS platforms (e.g., Zephyr RTOS)



APPLICATION-SPECIFIC SCRIPTS
Quickly enable application-specific functionality with scripting to orchestrate underlying firmware operations

SCRIPTING ENGINE
Language integration layer for common scripting engines

PLATFORM MIDDLEWARE
Ezurio designed abstraction layers enable application portability across MCU-based products, RTOS, scripting language and peripherals

VENDOR SDK & LIBRARIES
Builds upon Hardware-specific, SDKs, Drivers, Bootloaders, Filesystem and other low-level firmware supported by chipset vendors

Canvas MicroPython

Zephyr is the core!

- Zephyr is the core with MicroPython added as a module
- Benefit from all Zephyr features and the community behind it
- MicroPython in a thread

```
- name: micropython
  path: modules/lib/laird_connect/micropython
  revision: v1.21.0
  remote: micropython
- name: micropython_embed
  path: modules/lib/laird_connect/micropython_embed
- name: micropython_embed_tools
  path: modules/lib/laird_connect/micropython_embed_tools
```

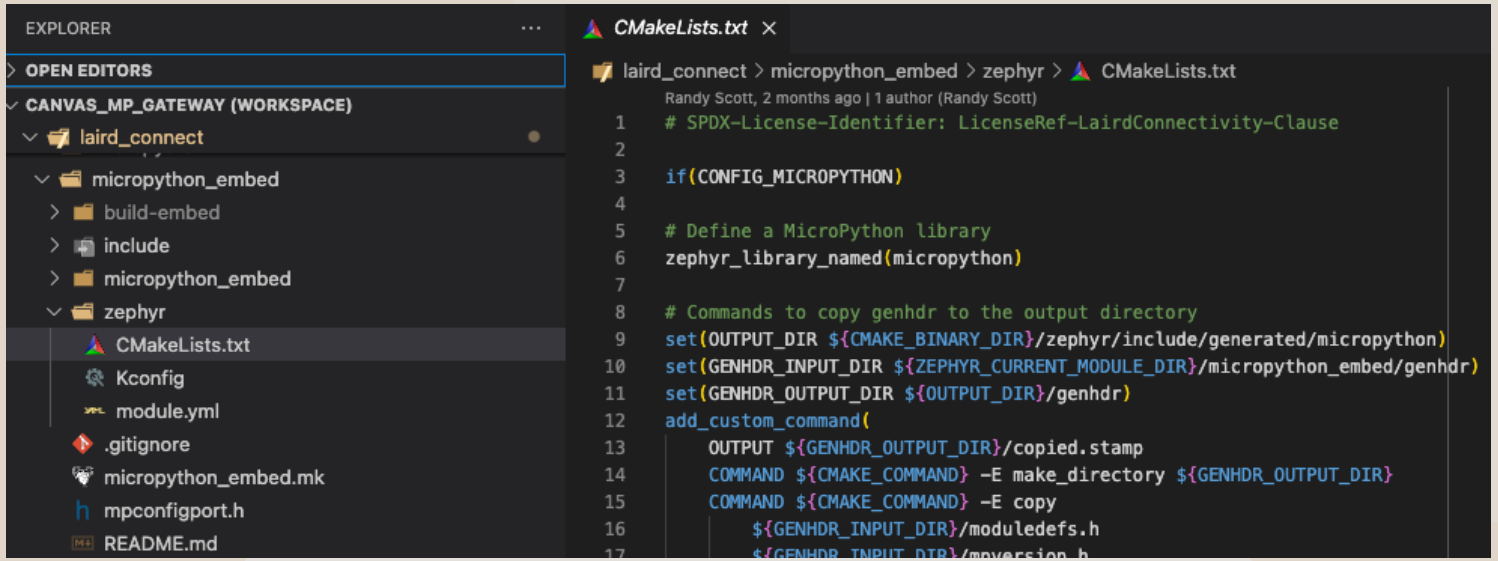
```
SYS_INIT(zephyr_micropython_init, APPLICATION, CONFIG_ZEPHYR_MICROPYTHON_INIT_PRIORITY);

static int zephyr_micropython_init(void)
{
    /* Store and reset the reset cause */
    hwinfo_get_reset_cause(&mp_reset_cause);
    hwinfo_clear_reset_cause();

    /* Initialize the UART */
    (void)mp_platform_uart_init();

    /* Start a thread */
    micropython_main_thread_id =
        k_thread_create(&micropython_thread, micropython_thread_stack,
                K_THREAD_STACK_SIZEOF(micropython_thread_stack),
                micropython_thread_fn, NULL, NULL, NULL,
                CONFIG_ZEPHYR_MICROPYTHON_THREAD_PRIORITY, 0, K_NO_WAIT);
    k_thread_name_set(micropython_main_thread_id, MICROPYTHON_THREAD_NAME);

    return 0;
}
```

Zephyr with MicroPython Embed Port

https://github.com/micropython/micropython/tree/master/ports/embed

The MicroPython embed port is designed to embed MicroPython into an existing C application.

- The embed port was turned into a Zephyr module
  - Can also be used in Non-Zephyr code bases
- MicroPython engine can be started in a thread



Zephyr File System and Device Tree

A file system is a core requirement for a MicroPython device

With Zephyr kconfig and device tree no C code needs to be written to enable a filesystem on a device.

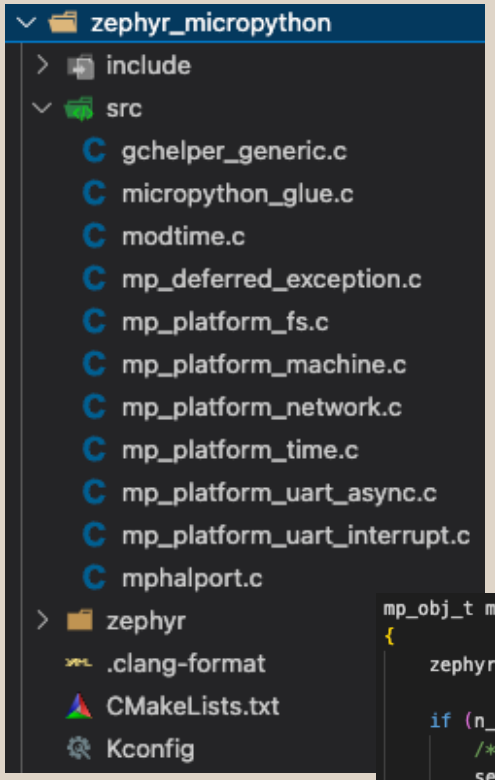
Enable the correct subsystems with kconfig.

```
# Enable the LittleFS file system.
CONFIG_FILE_SYSTEM=y
CONFIG_FILE_SYSTEM_LITTLEFS=y
```

Setup the filesystem partition and automount it on boot with device tree.

```
/ {
    fstab {
        compatible = "zephyr,fstab";
        lfs1: lfs1 {
            compatible = "zephyr,fstab,littlefs";
            mount-point = "/lfs1";
            partition = <&littlefs_storage>;
            automount;
            read-size = <16>;
            prog-size = <16>;
            cache-size = <64>;
            lookahead-size = <32>;
            block-cycles = <512>;
        };
    };
};

&mx25r64 {
    partitions {
        compatible = "fixed-partitions";
        #address-cells = <1>;
        #size-cells = <1>;

        littlefs_storage: partition@200000 {
            label = "littlefs_storage";
            reg = <0x00200000 0x00600000>;
        };
    };
};
```

Connect the Filesystem to MicroPython



Implement a new module to “glue” MP to Zephyr.

Implement file stream operations.

```c
STATIC const mp_map_elem_t zephyr_file_locals_dict_table[] = {
    /* Stream operations */
    { MP_ROM_QSTR(MP_QSTR_read), (mp_obj_t)&mp_stream_read_obj },
    { MP_ROM_QSTR(MP_QSTR_readinto), (mp_obj_t)&mp_stream_readinto_obj },
    { MP_ROM_QSTR(MP_QSTR_readline), (mp_obj_t)&mp_stream_unbuffered_readline_obj },
    { MP_ROM_QSTR(MP_QSTR_readlines), (mp_obj_t)&mp_stream_unbuffered_readlines_obj },
    { MP_ROM_QSTR(MP_QSTR_write), (mp_obj_t)&mp_stream_write_obj },
    { MP_ROM_QSTR(MP_QSTR_flush), (mp_obj_t)&mp_stream_flush_obj },
    { MP_ROM_QSTR(MP_QSTR_close), (mp_obj_t)&mp_stream_close_obj },
    { MP_ROM_QSTR(MP_QSTR_seek), (mp_obj_t)&mp_stream_seek_obj },
    { MP_ROM_QSTR(MP_QSTR_tell), (mp_obj_t)&mp_stream_tell_obj },
    { MP_ROM_QSTR(MP_QSTR___del__), (mp_obj_t)&mp_stream_close_obj },
    { MP_ROM_QSTR(MP_QSTR___enter__), (mp_obj_t)&mp_identity_obj },
    { MP_ROM_QSTR(MP_QSTR___exit__), (mp_obj_t)&mp_stream___exit___obj },
};
STATIC MP_DEFINE_CONST_DICT(zephyr_file_locals_dict, zephyr_file_locals_dict_table);

/* clang-format off */
STATIC const mp_stream_p_t zephyr_text_stream = {
    .read = zephyr_file_read,
    .write = zephyr_file_write,
    .ioctl = zephyr_file_ioctl,
    .is_text = true,
};
MP_DEFINE_CONST_OBJ_TYPE(
    zephyr_text_stream_type,
    MP_QSTR_TextFileIO,
    MP_TYPE_FLAG_NONE,
    protocol, &zephyr_text_stream,
    locals_dict, &zephyr_file_locals_dict);

STATIC const mp_stream_p_t zephyr_binary_stream = {
    .read = zephyr_file_read,
    .write = zephyr_file_write,
    .ioctl = zephyr_file_ioctl
};
MP_DEFINE_CONST_OBJ_TYPE(
    zephyr_binary_stream_type,
    MP_QSTR_BinaryFileIO,
    MP_TYPE_FLAG_NONE,
    protocol, &zephyr_binary_stream,
    locals_dict, &zephyr_file_locals_dict);
```

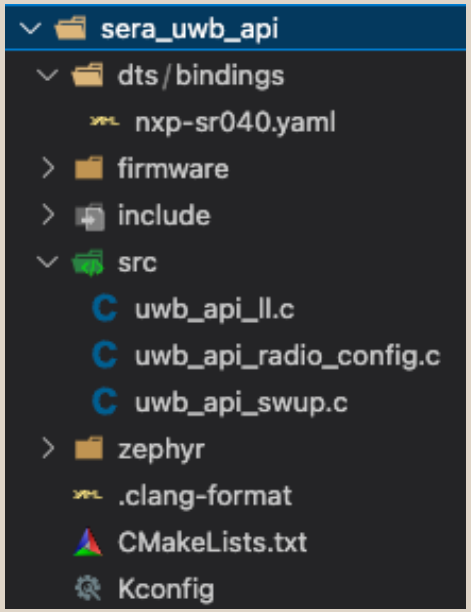
Implement open to get a file object.

```c
mp_obj_t mp_builtin_open(size_t n_args, const mp_obj_t *args, mp_map_t *kwargs)
{
    zephyr_mp_file_descriptor_obj_t *self = NULL;

    if (n_args == 1) {
        /* Default mode to "r" if not provided */
        self = zephyr_file_open(mp_obj_str_get_str(args[0]), "r");
    } else if (n_args >= 2) {
        self = zephyr_file_open(mp_obj_str_get_str(args[0]), mp_obj_str_get_str(args[1]));
    }

    if (self == NULL) {
        mp_raise_OSError(MP_ENOENT);
    }

    return MP_OBJ_FROM_PTR(self);
}
MP_DEFINE_CONST_FUN_OBJ_KW(mp_builtin_open_obj, 1, mp_builtin_open);
```

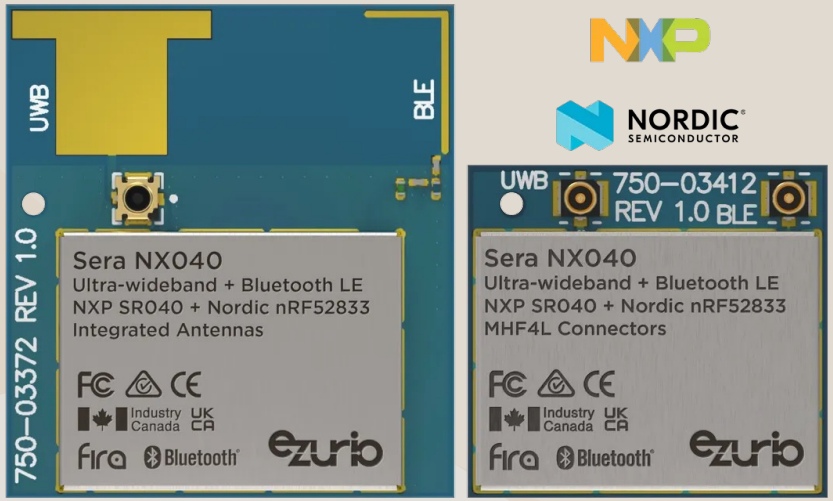
Zephyr Driver Architecture

To add new hardware, re-use Zephyr's driver architecture.

We implemented an out-of-tree driver for the NXP SR040 UWB radio.




```
/* SPI bus to UWB radio */
&spi1 {
    compatible = "nordic,nrf-spim";
    status = "okay";
    cs-gpios = <&gpio0 14 GPIO_ACTIVE_LOW>;
    pinctrl-0 = <&spi1_default>;
    pinctrl-1 = <&spi1_sleep>;
    pinctrl-names = "default", "sleep";
    uwb_radio: sr040@0 {
        compatible = "nxp,sr040";
        reg = <0>;
        spi-max-frequency = <8000000>;
        int-gpios = <&gpio0 15 GPIO_ACTIVE_LOW>;
        ready-gpios = <&gpio0 12 GPIO_ACTIVE_LOW>;
        reset-gpios = <&gpio1 7 GPIO_ACTIVE_LOW>;
    };
};
```



Expose C APIs to MicroPython

Python API is defined with a pyi file.

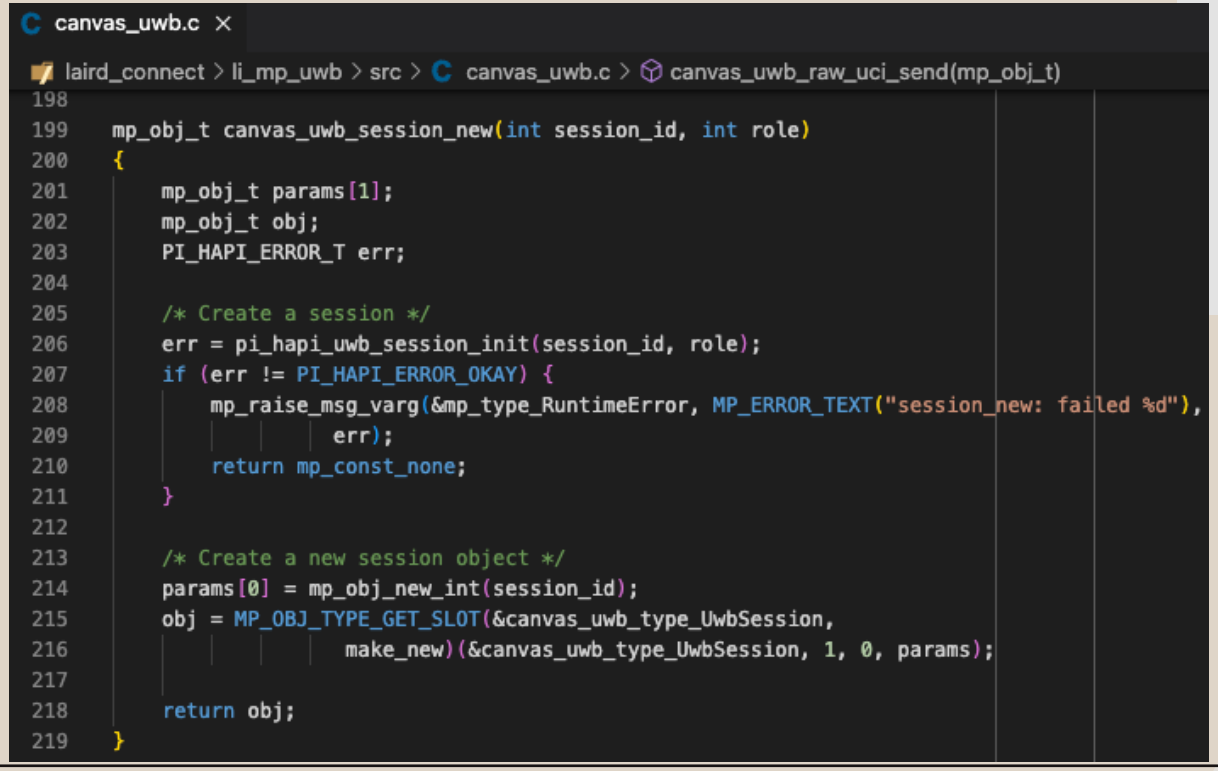
canvas_uwb.pyi

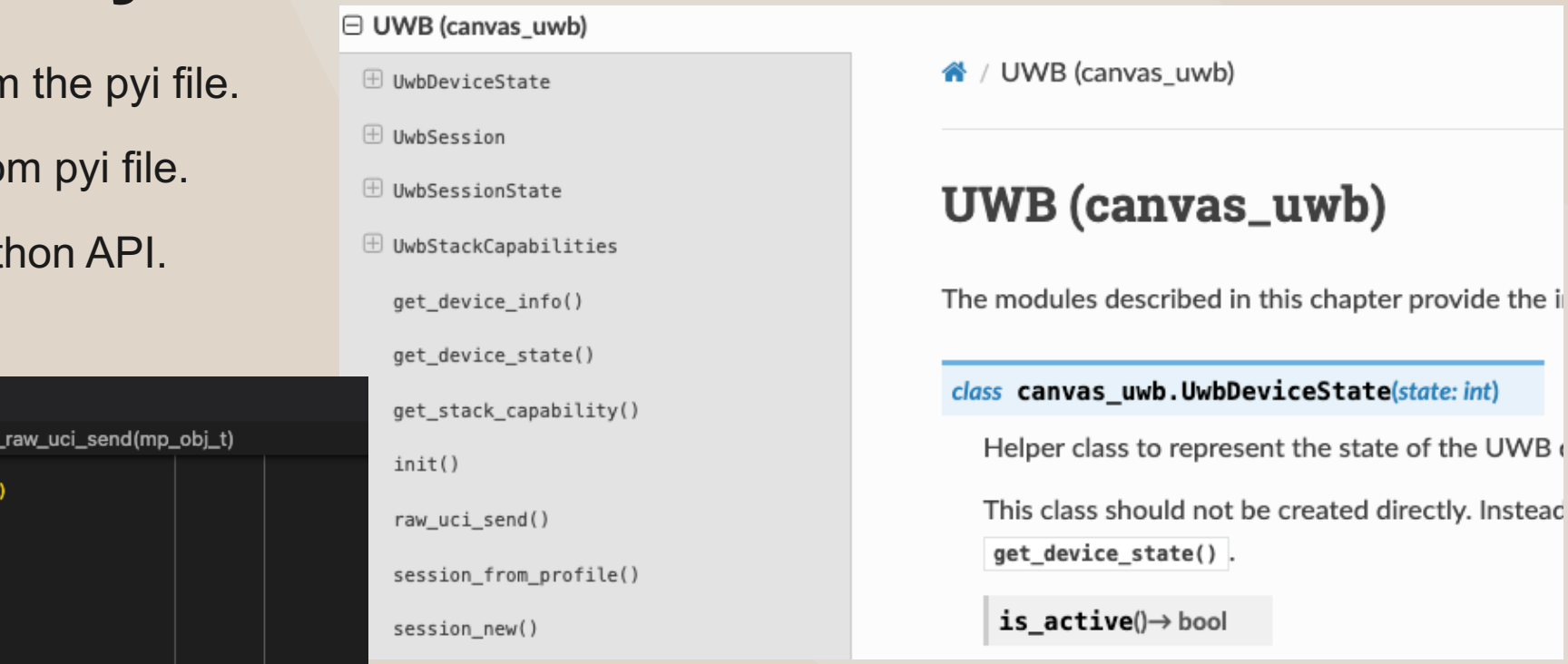
laird_connect > li_mp_uwb > canvas_uwb.pyi > ...

```
372   def init():
373       """
374       Initialize the UWB stack. This method must be called before any other
375       UWB stack methods are called.
376       """
377       pass
378
379   def shutdown():
380       """
381       Shutdown the UWB stack. This method must be called when the UWB stack
382       is no longer needed. To use the UWB stack again, the `init()` method.
383       """
384       pass
385
386   def get_stack_capability() -> UwbStackCapabilities:
387       """
388       Get the capabilities of the UWB stack.
389
390       :return: The capabilities of the UWB stack as a
391         :py:class:`UwbStackCapabilities` object.
392       """
393       pass
394
395   def get_device_state() -> UwbDeviceState:
396       """
397       Get the state of the UWB device.
398
399       :return: The state of the UWB device as a :py:class:`UwbDeviceState`
400         object.
401       """
402       pass
```

C APIs and MicroPython Cont.

- Python scripts to generate C from the pyi file.
- User Python API is generated from pyi file.
- C developer implements C to python API.





Canvas Python API:
https://lairdcp.github.io/canvas_python_docs/

Connectivity First

Zephyr provides the connectivity we need!

- Zephyr Bluetooth LE stack is full featured and SIG qualified
- Networking: TCP, UDP, CoAP, LwM2M, MQTT, HTTP
- Security: TLS, secure boot
- Drivers: Modem, Ethernet, UWB, USB
- LoRa and LoRaWAN

Production Ready with Firmware Updates!

- mcuboot bootloader
- Wired/wireless updates with mcumgr
  - USB/UART, Bluetooth LE
- Cloud
  - LwM2M/HTTP(s)

Firmware Updates With Bluetooth LE

- SMP over Bluetooth LE
  - Image management – core firmware
  - File system management – update python apps
  - SMP GATT service is always present in addition to user's python app
  - Default script runs if no user app

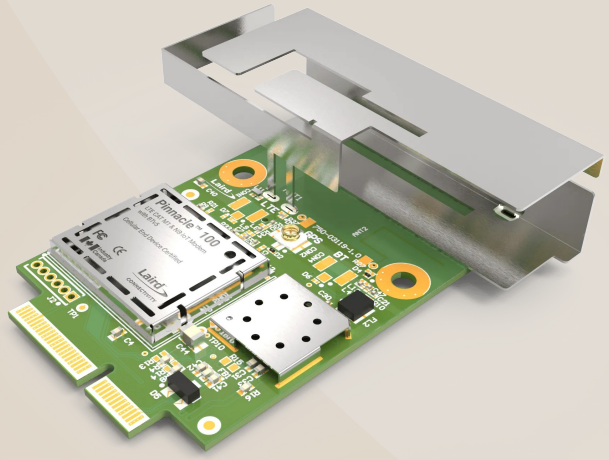
```
# Enable the Bluetooth mcumgr transport (unauthenticated).
CONFIG_MCUMGR_TRANSPORT_BT=y
CONFIG_MCUMGR_TRANSPORT_BT_AUTHEN=n
CONFIG_MCUMGR_TRANSPORT_BT_CONN_PARAM_CONTROL=y
```

```
#define MP_DEFAULT_SCRIPT                                                    \
    "import canvas_ble\n\n"                                                  \
    "def ble_connect(conn):\n"                                               \
    "    print('Connected to', canvas_ble.addr_to_str(conn.get_addr()))\n\n" \
    "def ble_disconnect(conn):\n"                                            \
    "    print('Disconnected from', canvas_ble.addr_to_str(conn.get_addr()))\n" \
    "    adv.start()\n\n"                                                    \
    "canvas_ble.init()\n"                                                    \
    "canvas_ble.set_periph_callbacks(ble_connect, ble_disconnect)\n"         \
    "adv = canvas_ble.Advertiser()\n"                                        \
    "adv.set_interval(250, 300)\n"                                           \
    "adv.add_ltv(canvas_ble.AD_TYPE_FLAGS, bytes([0x06]), False)\n"          \
    "addr = canvas_ble.addr_to_str(canvas_ble.my_addr())[12:14]\n"           \
    "adv.add_tag_string(canvas_ble.AD_TYPE_NAME_COMPLETE, \"Canvas\" + addr, False)\n" \
    "adv.add_smp_uuid(False)\n"                                              \
    "adv.add_canvas_data(0, 0, True)\n"                                      \
    "print('Advertising with SMP UUID')\n"                                   \
    "adv.start()\n"
```

Firmware Updates From the Cloud

Watch my ZDS 2023 presentation on device management! https://youtu.be/KB8TJAK_F7Y?si=lPkr6fpsNT8sroy2

- Use Zephyr LwM2M client
  - Firmware Update (Object 5)
    - Update core firmware
  - Software Management (Object 9)
    - HL7800 modem firmware
    - Python app



Zero configuration sample:
https://github.com/LairdCP/canvas_python_samples/tree/main/networking/lwm2m

Connects to public Leshan server:
https://leshan.eclipseprojects.io/#/clients

LwM2M Firmware Update Cont.

- User can control LwM2M from python
- Start LwM2M and firmware handles update functions
  - User doesn't need to worry about implementing complicated update logic

```python
def event_cb(evt):
    print("LwM2M event: {}".format(LWM2M_EVENTS[evt]))

# Configure the LWM2M client
lwm2m = Lwm2m(event_cb)
lwm2m.set_endpoint_name(endpoint_name)
lwm2m.set((lwm2m.OBJ_SECURITY, 0, 0), server_url)
lwm2m.set((lwm2m.OBJ_SECURITY, 0, 1), bootstrap)
lwm2m.set((lwm2m.OBJ_SECURITY, 0, 2), security_mode)

if security_mode == lwm2m.SECURITY_PSK:
    lwm2m.set((lwm2m.OBJ_SECURITY, 0, 3), psk_id)
    lwm2m.set((lwm2m.OBJ_SECURITY, 0, 5), psk)

if bootstrap == 0:
    lwm2m.set((lwm2m.OBJ_SECURITY, 0, 10), 101)
    lwm2m.create((lwm2m.OBJ_SERVER, 0))
    lwm2m.set((lwm2m.OBJ_SERVER, 0, 0), 101)
    lwm2m.set((lwm2m.OBJ_SERVER, 0, 1), 60)

# Configure the device object
lwm2m.create((lwm2m.OBJ_DEVICE, 0, 0), 32)
lwm2m.set((lwm2m.OBJ_DEVICE, 0, 0), "Laird Connectivity")
lwm2m.create((lwm2m.OBJ_DEVICE, 0, 3), 32)
lwm2m.set((lwm2m.OBJ_DEVICE, 0, 3), os.uname().release)
lwm2m.create((lwm2m.OBJ_DEVICE, 0, 17), 32)
lwm2m.set((lwm2m.OBJ_DEVICE, 0, 17), os.uname().machine)
lwm2m.set_exec_handler((lwm2m.OBJ_DEVICE, 0, 4), reboot_exec_cb)

# Wait for network to come up
print("Waiting for network")
net = NetHelper(None)
net.wait_for_ready()

print("Starting LwM2M wih endpoint name: {}".format(endpoint_name))
# Start the LwM2M client
lwm2m.start(bootstrap != 0)
```

Update Python App with LwM2M

Package your python app with our tool:
https://github.com/LairdCP/canvas_software_packager/blob/main/canvas_packager.py

- Python tool to package a python app and version it
- The package contains a manifest of all files with SHA256 for validation
- Use LwM2M object 9 to download and install the package

Device Management

- LwM2M is a great protocol for device management
  - Our partner, EdgeIQ, offers a platform for managing your fleet of devices
  - The open-source Eclipse Leshan project can be used for testing or starting your own server

Reliability – Automated Test Platform

- Automated testing with Robot Framework
- MicroPython REPL gives a standard way to interact with all devices

```
Connect Central
    [Arguments]    ${primary_board}    ${board_adv_name}    ${phy}

    ${resp}=    User REPL Send    ${primary_board}    required_filter_name = "${board_adv_name}"
    ${resp}=    User REPL Send    ${primary_board}    required_phy = ${phy}
    ${resp}=    Run Script on Board    ${primary_board}    ${CENTRAL_SCAN_SCRIPT}
    ${resp}=    Convert To String    ${resp}
    Should Contain    ${resp}    ${CENTRAL_SCAN_SCRIPT_START_RESP}

    ${total_time}=    Set Variable    ${20}
    ${result}=    Set Variable    ${False}
    WHILE    $total_time > ${0}
        ${resp1}=    User REPL Send    ${primary_board}    print(found)
        ${resp1}=    Convert To String    ${resp1}
        ${resp1}=    Replace String    ${resp1}    \r\n    ${EMPTY}
        IF    ${resp1} == True
            ${result}=    Set Variable    ${True}
            BREAK
        ELSE
            Sleep    1s
            ${total_time}=    Evaluate    ${total_time} - 1
        END
    END
    IF    ${result} == False    Fail    Failed to connect
```

A Look at Writing Python

BLE Peripheral

BleAd.py 1

apps > bl654_usb > BleAd > BleAd.py > ...

scottlederer-lci, 2 months ago | 1 author (scottlederer-lci)

```python
import canvas_ble as ble

ble_name = "Canvas BLE"
service_uuid = "11223344-5566-7788-99aa-bbccddeeff00"
char_uuid = "00000001-5566-7788-99aa-bbccddeeff00"

def cb_con(conn): # Connect callback
    print("Connected")

def cb_discon(conn): # Disconnect callback
    global advert
    print("Disconnected")
    advert.start()

def cb_write(event_object):
    print('Received message: ' + event_object.data.decode())

def uuidToBytes(uuid): # Convert the 128 bit UUID string to bytes
    uuid_bytes = list(bytes.fromhex(uuid.replace("-", "")))
    uuid_bytes.reverse()
    return bytes(uuid_bytes)

# GATT table definition for a service with a writable "message" characteristic
gatt_table = {
    "Service 1":{
        "Name": "My BLE Service",
        "UUID": service_uuid,
        "Characteristic 1":{
            "Name": "message",
            "UUID" : char_uuid,
            "Length" : 20,
            "Read Encryption" : "None",
            "Write Encryption" : "None",
            "Capability" : "Write",
            "Callback" : cb_write
        }
    }
}
```

```python

# Initialize the BLE stack and set the connection and disconnection callbacks
ble.init()
ble.set_periph_callbacks(cb_con, cb_discon)

# Start advertising our primary service by the 128 bit UUID
advert = ble.Advertiser()
advert.stop()
advert.clear_buffer(False)
advert.add_ltv(1, bytes([6]), False)
advert.add_ltv(7, uuidToBytes(service_uuid), False)
advert.add_tag_string(9, ble_name, False)
advert.set_phys(ble.PHY_1M, ble.PHY_2M)
advert.set_properties(True, False, True)
advert.set_interval(240, 250)
advert.start()

# Define the gatt server and callbacks and start the GATT server
my_gattserver = ble.GattServer()
my_gattserver.build_from_dict(gatt_table)
my_gattserver.start()
```

Writing Python

BLE Central

BleScan.py 1

apps > bl654_usb > BleScan > BleScan.py > ...

scottlederer-lcl, 2 months ago | 1 author (scottlederer-lcl)

```python
import canvas_ble as ble
import binascii
import time

service_uuid = "11223344-5566-7788-99aa-bbccddeeff00"
char_uuid = "00000001-5566-7788-99aa-bbccddeeff00"
peripheral_device_address = None
connection = None
gatt_client = None

def uuidToBytes(uuid): # Convert the 128 bit UUID string to bytes
    uuid_bytes = list(bytes.fromhex(uuid.replace("-", "")))
    uuid_bytes.reverse()
    return bytes(uuid_bytes)

def scan_init():
    global scanner
    scanner = ble.Scanner(scan_cb)
    # Filter ads that contain manufacturer ID 0x0077
    scanner.filter_add(ble.Scanner.FILTER_UUID, uuidToBytes(service_uuid))
    scanner.set_timing(100, 100) # Set scan to 100% duty cycle

def scan_cb(sr):
    global peripheral_device_address
    print("DID={},RS={},AD={}".format(
        binascii.hexlify(sr.addr).decode(),
        sr.rssi, binascii.hexlify(sr.data).decode()))
    peripheral_device_address = sr.addr

def send(message):
    global gatt_client
    global char_uuid
    gatt_client.write(char_uuid, message)

def connection_cb(conn):
    global connection
    global gatt_client
    print("Connection established with {}".format(
        ble.addr_to_str(conn.get_addr())))
    connection = conn
    gatt_client = ble.GattClient(conn)
    gatt_client.discover()
    print('Discovering services...')
    send(b'Hello, BLE!')
    print('Writing to characteristic...')

def disconnection_cb(conn):
    print("Connection closed with {}".format(
        ble.addr_to_str(conn.get_addr())))

ble.init()
scan_init()
scanner.start(1)
print('Scanning for peripheral devices for 2 seconds...')
time.sleep_ms(2000)
scanner.stop()

if peripheral_device_address is not None:
    print("Client found, attempting connection to", ble.addr_to_str(peripheral_device_address))
    ble.connect(peripheral_device_address, ble.PHY_2M, connection_cb, disconnection_cb)
```

Writing Python

BLE to the Cloud!

apps > mg100 > XbitTextAppGateway > main.py > MqttClient > __init__

```
44
45  def scan_cb(evt):
46      global BLE_LED, num_tracked_devices
47
48      # Get manufacturer-specific data from the advertisement
49      msg = canvas_ble.find_ltv(0xff, evt.data)
50      if msg is None:
51          return
52      msg = msg[4:].decode()
53
54      name = canvas_ble.find_ltv(9, evt.data)
55      if name is None:
56          return
57      name = name.decode()
58
59      addr = binascii.hexlify(evt.addr).decode()
60      # Check if we've seen this device before
61      if addr not in devices:
62          try:
63              devices[addr] = {'deviceId': addr}
64              devices[addr]['timestamp'] = time.time()
65              devices[addr]['message'] = ''
66              devices[addr]['name'] = name
67              num_tracked_devices += 1
68              print("Tracking device: {} ({})".format(addr, num_tracked_devices))
69          except:
70              print("Error tracking {}. {} tracked.".format(
71                  addr, num_tracked_devices))
72              try:
73                  del devices[addr]
74              except:
75                  pass
76              return
77
78      now = time.time()
79      if now - devices[addr]['timestamp'] < DEVICE_MSG_LIMIT_TIME_SEC:
80          return
81
82      last_message = devices[addr]['message']
83      if last_message != msg and len(publish_list) < (MAX_MSG_PUBLISH - 1):
84          BLE_LED.flash()
85          devices[addr]['name'] = name
86          devices[addr]['timestamp'] = time.time()
87          devices[addr]['message'] = msg
88          print("Received: {}".format(devices[addr]))
89          publish_list.append(devices[addr].copy())
```

```
92  def mqtt_publish_timer(event):
93      publish = False
94      msg = {
95          'message': {
96              'messages': []
97          }
98      }
99      while len(publish_list) > 0:
100         publish = True
101         device = publish_list.pop(0)
102         msg['message']['messages'].append(device)
103
104     if publish:
105         msg_str = json.dumps(msg)
106         print("Publishing: {}".format(msg_str))
107         client.publish(msg_str)
```

Future Features

- MicroPython threading with Zephyr threads
- More flexible firmware updates
  - User can download images via a transport of their choice
- Encrypted file system
- Bluetooth LE secure connections
- Memfault – Crash analytics and device metrics

Summary

- Development is faster with virtually no setup time.
- Production level code without writing in C.
- Putting all the firmware pieces together is hard. Customers no longer need to worry.

Questions?

Ryan Erickson

Ryan.Erickson@ezurio.com

Ezurio Discord Channel!