DeviceScript, TypeScript for Microcontrollers

Created by Peli de Halleux

https://learn.adafruit.com/devicescript

Last updated on 2023-08-29 04:57:37 PM EDT
# Table of Contents

**What is DeviceScript?**
- Blinky
- Supported Hardware
- Guides

**Development Experience**
- Console Logging
- Simulation
- Debugging
- Running on Hardware
- Remote workspaces

**Flash Firmware**
- Flash Firmware
- Board configuration

**Using Sensors**
- Temperature and Humidity sensing

**Settings and Secrets**
- .env files
- Using Settings

**Publish to Adafruit IO**
- Post data to Adafruit IO
- devicescript-adafruit-io package
- Final snippet on QT Py ESP32-C3
What is DeviceScript?

DeviceScript is a project that aims at bringing a TypeScript development experience to microcontrollers.

TypeScript is JavaScript with syntax for types. DeviceScript supports a subset of TypeScript. The TypeScript sources are compiled to a compact bytecode which is executed on the microcontroller by a native runtime. It is in many ways similar to CircuitPython, but for TypeScript instead.

DeviceScript is an incubation project from Microsoft Research by the same team that started MakeCode.

Blinking Example

To get started, this little program blinks the onboard LED on your device. It can be run on a simulator or on a physical device. Follow this guide to learn more.

```javascript
// A blinking animation on the onboard LED
import { delay } from "@devicescript/core"
import { setStatusLight } from "@devicescript/runtime"

// start an interval time every 10ms
setInterval(async () => { // async, await!
    // turn off
    await setStatusLight(0)
    await delay(1000)
    // turn on
    await setStatusLight(0x0f0f0f)
    await delay(1000)
}, 10)
```
Supported Hardware

DeviceScript is current supported on ESP32 and RP2040 (full list). In particular, DeviceScript already provides the pinout configuration for various Adafruit boards such as the QT Py ESP32-C3 WiFi. More can be added...

Guides

This guide will build a program that collects sensor data (temperature) and uploads it to Adafruit IO. A few aspects of DeviceScript are covered that are relevant to Adafruit hardware, and Adafruit IO.

For more in-depth information, refer to the DeviceScript documentation.

Development Experience

The best developer experience is found by installing the extension in Visual Studio Code. A command line interface is also available, but a lot of work was done with the extension to improve the developer experience.

The extension bundles the compiler, firmware flasher, debugger, simulators, console log and other goodies.

Console Logging

You can use `console` to log message as in your usual JavaScript projects. DeviceScript will automatically collect the message sent to `console.log` (and `debug`, ...) and show them in the Visual Studio Code terminal.
console.debug("debug")
console.log("log")
console.warn("warn")
console.error("error")

DeviceScript also introduce a new logging function, `console.data`, that specializes on logging and collecting numerical values. Data streams collected with `console.data` can be exported to CSV files and Jupyter notebooks.

```javascript
const temp = 20
const humi = 60
console.data({ temp, humi })
```

Simulation

Before testing your code on hardware, you can use simulators. Simulators will run on your computer (even on containers) and can greatly help with the development experience.
In Visual Studio Code, when you try to start this sample, DeviceScript will first try to use any connected microcontrollers. If it does not find any, it will launch a **device simulator**. The device simulator runs the DeviceScript native runtime compiled to **Web assembly**. Simulated sensors can also be spun up using the simulator dashboard so that the entire system runs virtually on your computer (this also works if you are developing from a container).

When you start running a program, the DeviceScript simulator view will start. If it detects that you need a service that is not available, for example you need a temperature sensor but it is not detected, the view will suggest to start one for you. Make sure to click AUTO START to spin off a temperature sensor since you need one to run this program.

```javascript
import { Temperature } from '@devicescript/core'
// needs a simulator
const temperature = new Temperature()

setInterval(async () => {
    // blocks until a temperature server is found
    const temp = await temperature.reading.read()
    console.data({ temp })
}, 1000)
```
Once the simulated sensors are running, you will typically have a slider or other interaction UI to change the values.

Debugging

You can set breakpoints, step and inspect variables in the debugger... As long as your program is not too time sensitive, this can aid with debugging your programs.

You can also use console.log. DeviceScript also provides `console.data` which is a specialized logging function for sensor data, which can be exported to a Jupyter Notebook for later analysis.

Running on Hardware

You can run and debug the code running on the physical device as well. Click on the connect (plug) button in the DeviceScript view and select Serial. The device will be automatically discovered and the simulator will be stopped. Make sure to flash the firmware before this step.
The developer experience is pretty much the same from there. You can set breakpoints, use `console.log`, etc... Unlike the simulated sensors, you cannot drag a slider to change the sensor data... you might have to blow on it or put it in the fridge!

Remote workspaces

DeviceScript also supporting developing in remote workspaces, like Docker, GitHub Codespaces, Windows WSL or other virtualization frameworks.

The Visual Studio Code extension runs in the container and does not have access to serial or USB on the host operating system; so it launches a separate browser page to handle the connection to the physical device through WebSerial/WebUSB. Other than that, the experience is pretty much the same.
Flash Firmware

Before being able to download DeviceScript programs to your microcontrollers, you will need to flash the native runtime. The native runtime is currently available for ESP32 and RP2040.

Of course, reality is a bit more complicated at the support table gives you a rundown of which features is enable for the various SKUs of those chipsets.

Adafruit QT Py ESP32-C3 WiFi Dev Board with STEMMA QT
What's life without a little RISC? This miniature dev board is perfect for small projects: it comes with our favorite connector - the...
https://www.adafruit.com/product/5405
Flash Firmware

In Visual Studio Code, you will run the Flash Firmware... command that will drive through choosing the correct board, downloading the firmware and launching the appropriate tool to flash it. This is typically a one-time experience, until you have a need to update the runtime.

For ESP32, you will need to have esptool installed. For RP2040, it is a UF2 file copy to the bootloader drive so you don't need any tool.

Board configuration

After this step, you will need to import the board configuration to setup the pins, I2C and SPI services. DeviceScript maintains a list of known devices for which it has a configuration, including the QT Py ESP32-C3 or the Feather ESP32-S2. It is also possible to fork pinout configurations (and contribute them back).

The configuration is loaded by imported the module for your board.

```python
import { pins, board } from '@dsboard/adafruit_qt_py_c3'
```

Using Sensors

Temperature and Humidity sensing

DeviceScript provides a driver for the SHTC3 temperature and humidity sensor.
The driver handles the low-level I2C communication protocol to the hardware and exposes a temperature and humidity service to DeviceScript. The services are the abstraction layer to interact with hardware.

```javascript
import { startSHTC3 } from "@devicescript/drivers"
// temperature and humidity are clients for the sensor servers
const { temperature, humidity } = await startSHTC3()
```

Adafruit Sensirion SHTC3 Temperature & Humidity Sensor

Sensirion Temperature/Humidity sensors are some of the finest & highest-accuracy devices you can get. And finally, we have some that have a true I2C interface for easy...

https://www.adafruit.com/product/4636

A temperature service has a reading register that can be read to retrieve the temperature in Celsius, regardless of the underlying hardware. Registers, like the servo angle, are also writeable.

In the example below, we read the temperature and print it to the console. The console output will automatically be displayed.

```javascript
import { startSHTC3 } from "@devicescript/drivers"
const { temperature, humidity } = await startSHTC3()

// run every 5 seconds
setInterval(async () => {
  // read data from temperature sensor
  const value = await temperature.reading.read()
  // print sensor value
  console.log({ value })
}, 5000)
```
Settings and Secrets

Before we start uploading data to the cloud, we need a way to store the connection secrets (WiFi password, Adafruit IO key) to the device... without hardcoding it into our source code. This is done with settings files (.env) in DeviceScript.

.env files

Click on the configure icon (wand) and select Add Settings to add two files to the project:

- **.env.default**, contains public settings such as your Adafruit username. This file can be added to your source control system.
- **.env.local**, contains your secrets. This file should NOT be added to your source control system

```
# env.default
# public settings, commit to source control
WIFI_SSID=your-wifi-ssid
IO_USERNAME=your_adafruit_io_user
IO_FEED=your_adafruit_io_feed

# env.local
# secrets, don't commit to source control
IO_KEY=your_secret_key
WIFI_PWD=your_wifi_password
```
Using Settings

When DeviceScript uploads your program to the device, it will also communicate and send the settings to be stored in flash. The settings can be read on the device using `readSetting`.

The `WIFI_SSID` and `WIFI_PWD` are automatically used by the native WiFi service when trying to connect.

```javascript
import { readSetting } from '@devicescript/settings'

const key = await readSetting("IO_KEY")
const user = await readSetting("IO_USERNAME")
const feed = await readSetting("IO_FEED")
```

---

Publish to Adafruit IO

Next is to add code to upload the sensor data to an Adafruit IO feed. DeviceScript supports the HTTPS and MQTT endpoints.

Post data to Adafruit IO

Adafruit IO supports a [POST HTTPS endpoint](https://io.adafruit.com/channel/2) to add a data entry to a feed.

First the program collects the Adafruit IO feed information `user` and `feed`, the Adafruit.io key `key` and reads the temperature from the sensor `value`; then to use the `fetch` function to issue an HTTP POST request to the secure endpoint of Adafruit.io.

```javascript
import { fetch } from '@devicescript/net'

// collect connection info and data
const user = ...
const feed = ...
const key = ...
const value = await temperature.reading.read()

// craft Adafruit.io payload and send POST request
await fetch(`https://io.adafruit.com/api/v2/${user}/feeds/${feed}/data`, {
  method: "POST",
  headers: { "X-AIO-Key": key, "Content-Type": "application/json" },
  body: JSON.stringify({ value })
})
```
devicescript-adafruit-io package

The devicescript-adafruit-io package on GitHub may be used to simplify using Adafruit IO.

DeviceScript supports sharing code and libraries through npm or GitHub. It also supports MQTT.

Use the command below to install the package in your project.

```bash
npm install --save pelikhan/devicescript-adafruit-io#v0.0.4
```

The package provides `createData` which wraps reading the settings and secrets, crafting a POST request and analyzing the results.

```javascript
import { createData } from "devicescript-adafruit-io"
const value = await temperature.reading.read()
const status = await createData(value)
console.log({ status })
```

Final snippet on QT Py ESP32-C3

On the QT Py ESP32-C3 board, the full sample looks as follows.

There is also a `schedule` helper that starts an interval where the first delay is configurable (great for debugging).

```javascript
import { pins, board } from "@dsboard/adafruit_qt_py_c3"
import { startSHTC3 } from "@devicescript/drivers"
import { schedule } from "@devicescript/runtime"
import { createData } from "devicescript-adafruit-io"

const { temperature, humidity } = await startSHTC3()

schedule(async () => {
    const value = await temperature.reading.read()
    await createData(value)
}, { timeout: 1000, interval: 60000 })
```
Adafruit QT Py ESP32-C3 WiFi Dev Board with STEMMA QT
What's life without a little RISC? This miniature dev board is perfect for small projects: it comes with our favorite connector - the...
https://www.adafruit.com/product/5405

Adafruit Sensirion SHTC3 Temperature & Humidity Sensor
Sensirion Temperature/Humidity sensors are some of the finest & highest-accuracy devices you can get. And finally, we have some that have a true I2C interface for easy...
https://www.adafruit.com/product/4636