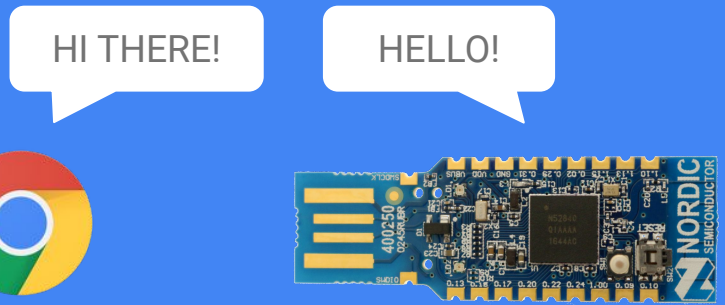
Web USB & Web Bluetooth with the nRF52840 Dongle

https://github.com/danielbank/web-nrf52-dongle

Daniel Bank / @DanielPBank
IoT DevFest 2020, 2020.01.25

Thanks to
Lars Knudsen / @denladeside



Who am I?

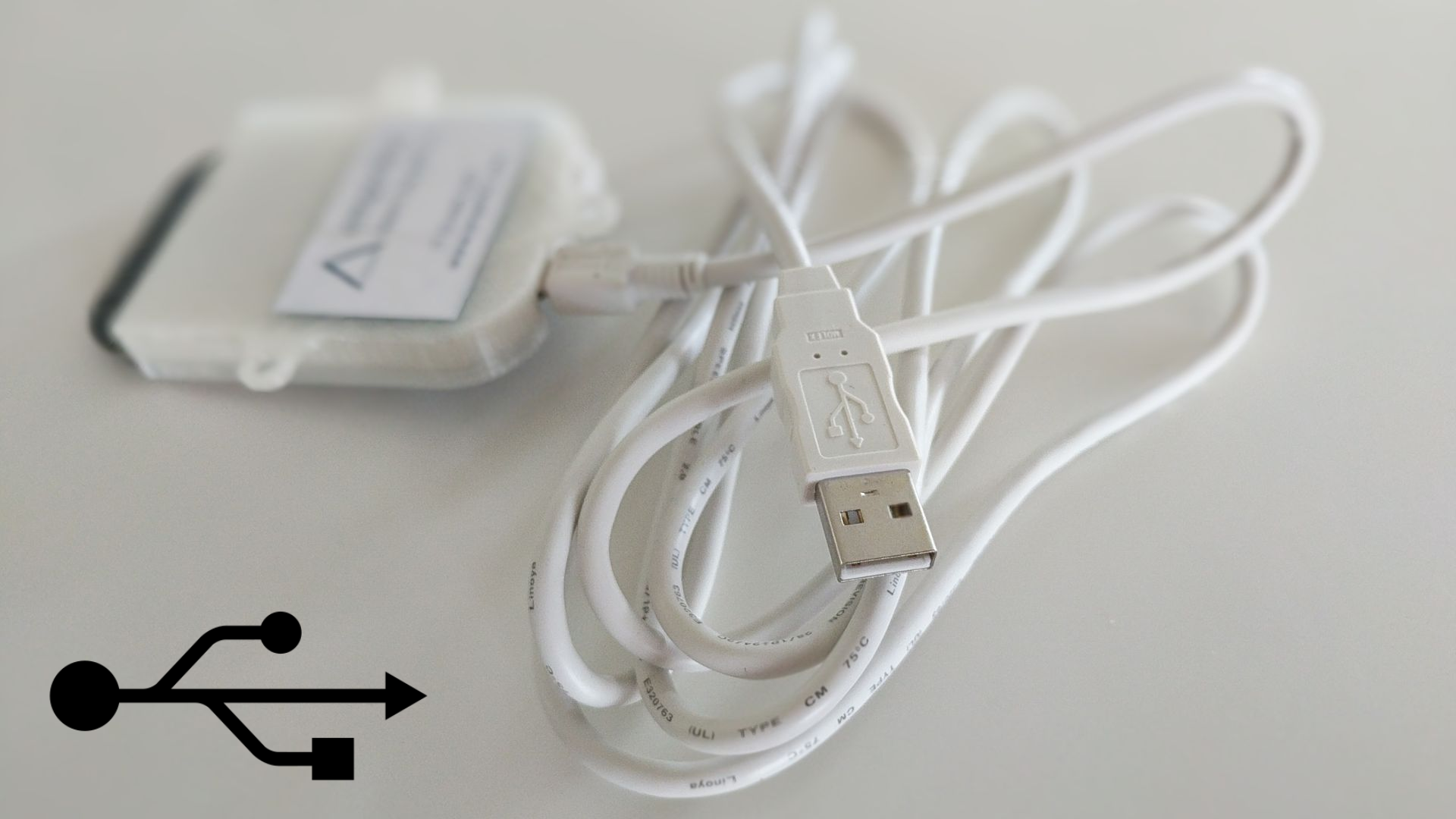
JavaScript / React Developer with a nascent interest in Rust

❤️ ...scuba diving, AI/ML, web, IoT, statistics, games

Daniel Bank

Twitter: @DanielPBank

GitHub: danielbank


E320763 (UL) TYPE CM 75°C

Identifying USB Hardware

In order be able to easily and consistently load the right drivers for the right hardware, a unique **Vendor** and **Product ID** must be assigned (VID/PID).

A vendor, having obtained an official Vendor ID (2 byte value) can assign individual Product IDs (also 2 bytes) to manufactured hardware.

The VID/PID combination is used as a key in e.g. Windows `*.inf` driver definition files.

Obtaining Vendor IDs

Vendor IDs are governed by usb.org: http://www.usb.org/developers/vendor/

Obtaining a VID costs ~ $5000 per year

Strict rules against selling/giving PIDs under the VID for 3rd parties

Registered vendors can redelegate a PID for minor productions/prototypes

Note: Many generic USB Serial chips in their hardware and stick with the generic VID/PID combo for those chips, ***making it impossible to classify*** *from those alone.*

Obtaining Vendor IDs for Open Source Projects

There is a Vendor ID that is open for Open Hardware Projects: http://pid.codes

Source code / schematics have to be public and licensed under a recognized open source license.

Submit a PR to their GitHub Repo and get your PID.

Common USB classes

Many standard classes exists, here are a few common ones:

- **Human Interface Device** (HID/03h), *e.g. keyboards, mice & joysticks*
- **Mass Storage Device** (MSD/08h), *e.g. memory sticks*
- **Comm. Device Class** (CDC/02h&0Ah), *e.g. modems & USB serial ports*

Devices not belonging to any standard class can use the ‘Custom’ class (FFh)

List: https://en.wikipedia.org/wiki/USB#Device_classes

Plug and Play...sometimes

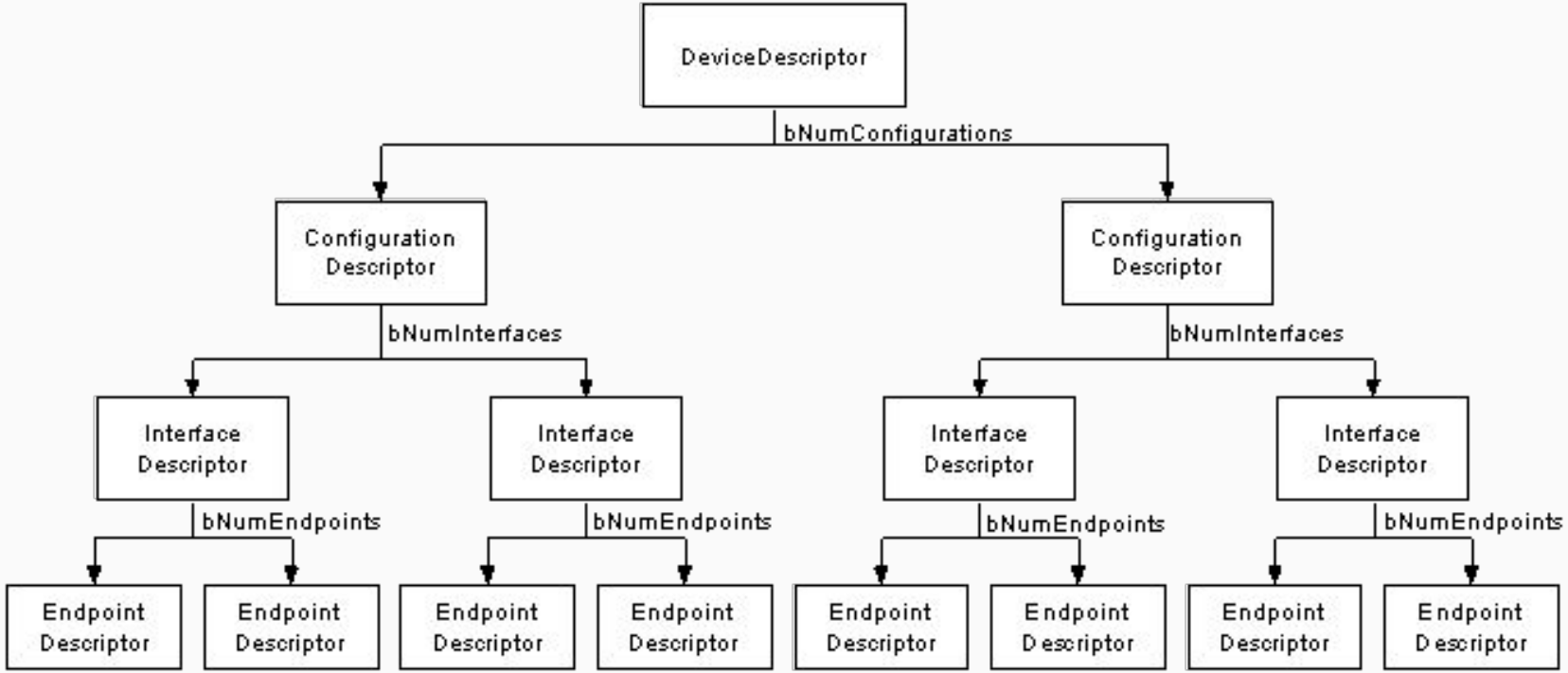
Imagine if you would need drivers for every new keyboard, mouse or memory stick….

Device descriptors are sent from the device during the initial handshake, potentially enabling **‘driverless’ PnP**.

Linux and OSX normally handles this properly, while **Windows is a bit behind** (although USB CDC works in Win10).

On Windows (for non standard classes), either a signed `*.inf` file must be installed or **MS specific descriptors** must be added in the device firmware.

Device, Configurations, Interfaces and Endpoints

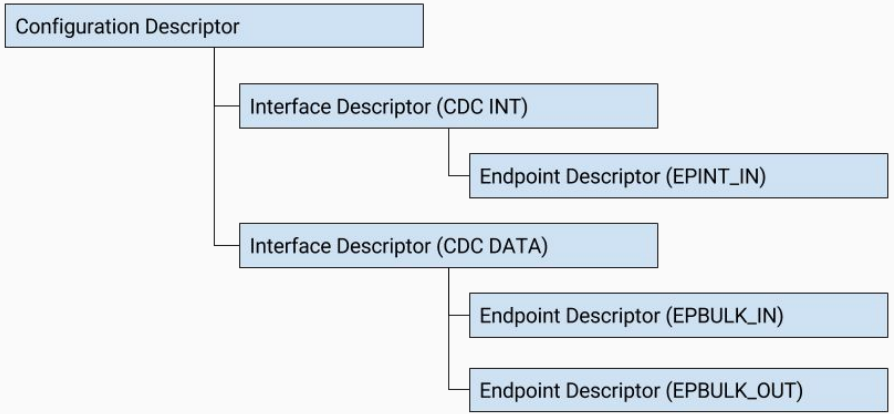


See: http://www.beyondlogic.org/usbnutshell/usb5.shtml

USB CDC (Serial)

Requires a combination of

- CDC control interface (baud rate, parity, etc.)
- CDC data interface (bulk in/out)
- IAD associating the two



USB CDC to Browser communication

Options include:

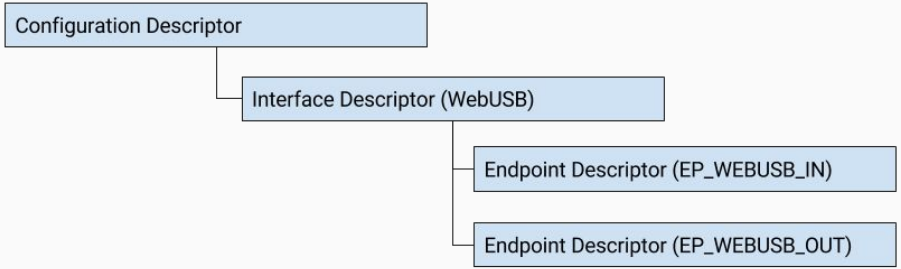
- Using chrome.serial (now deprecated for non-ChromeOS)
- Using the Serial API, now in origin trial on Chrome M80-82
- Creating a proxy using…
  - WebSockets (HTTP/HTTPS mixed content prevention=> **can't use for production**)
  - Native Messaging extension (Chrome, Firefox and Edge)

Web USB

Does not require any installation of any driver on the host OS

Generally available in stable since **Chrome M61** (released September, 2017)

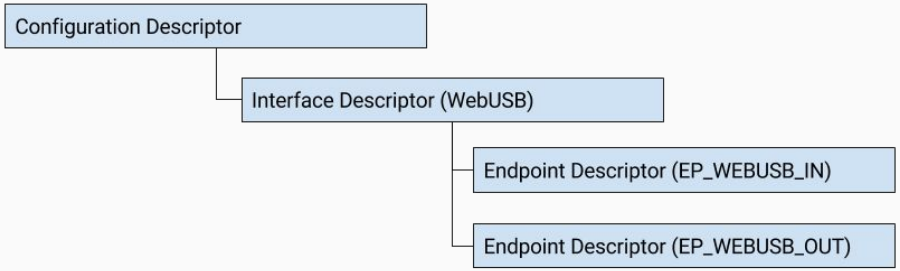
Clear benefits for hardware vendors, **bringing down maintenance cost**



Web USB

The device descriptor must include a data interface with bulk in/out (one that will not be claimed by the system OS).

Optionally, a **binary object store** (BOS) descriptor, with Web USB (and WINUSB info is needed and a few extra request handlers for true PnP on Windows)



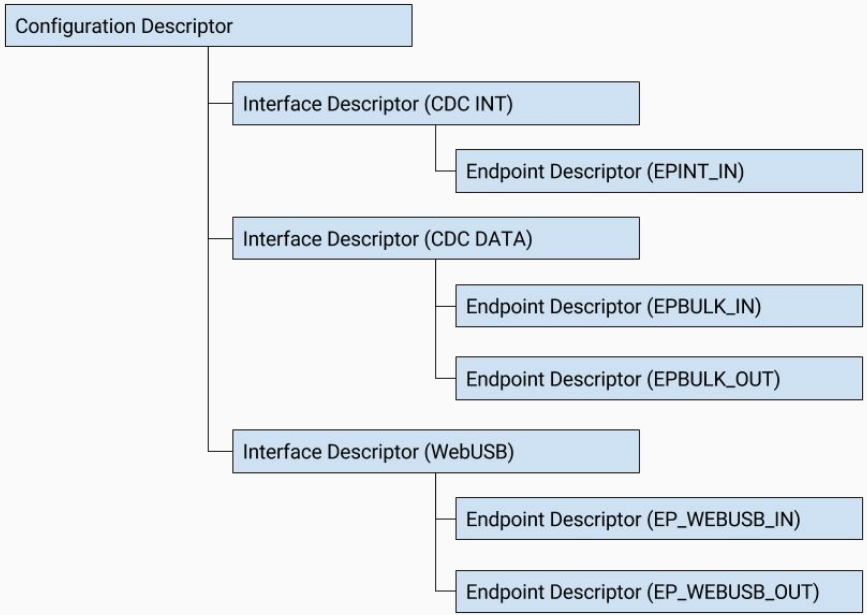

```
uint8_t * WebUSBCDC::urlIlandingPage() {
    static uint8_t urlIlandingPageDescriptor[] = {
        0x16,                   /* bLength */
        WEBUSB_URL,             /* bDescriptorType */
        WEBUSB_URL_SCHEME_HTTPS,/* bScheme */
        'e','m','p','i','r','i','k','i','t','.','g','i','t','h','u','b','.','i','o',
    };
    return urlIlandingPageDescriptor;
}
```

See: https://wicg.github.io/webusb/#webusb-descriptors-and-requests for more information

Composite USB CDC + Web USB

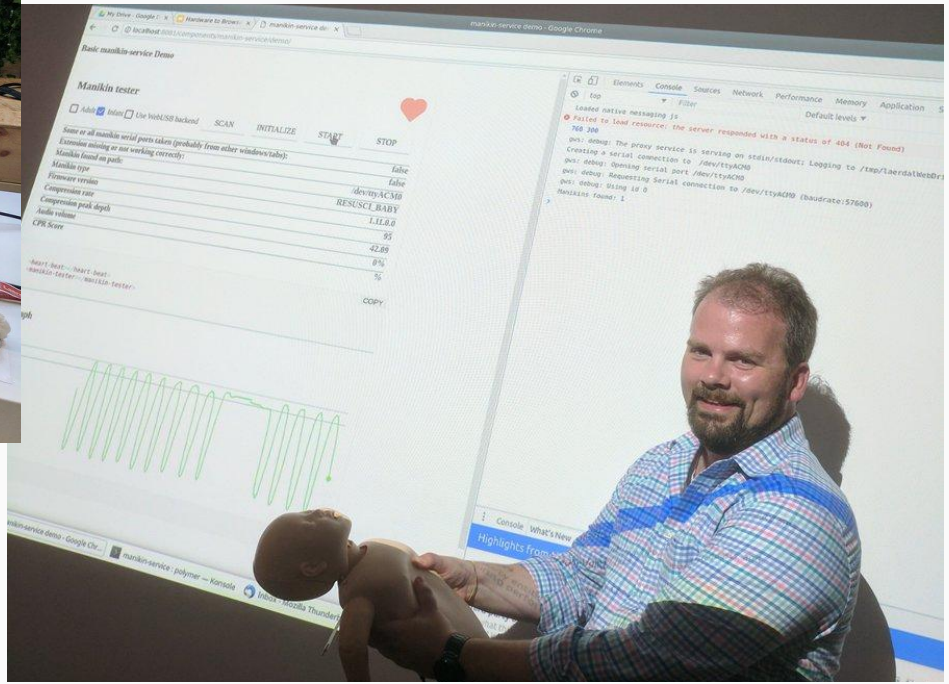
Composite devices appear as multiple (virtual) devices in the host OS. An example could be a secure external storage device (MSD interface) that also includes e.g. a CDC interface for control - or a modern headset, including both audio in/out as well as a HID interface for buttons.

USB CDC + Web USB can be a practical combination for **adding Web USB capabilities to legacy CDC hardware** while keeping backwards compatibility.



QCPR Manikin as game controller
(Shown at the Polymer Summit 2017)

First successful WebUSB implementation
In a Laerdal QCPR Manikin (Aug, 2017)



Laerdal Medical QCPR Manikin with WebUSB support

Quirks on Windows

- Requires MS OS descriptors
  - V 1.0 for Windows 8.0 and below
  - V 2.0 for Windows 8.1 and 10
- Requires a DeviceInterfaceGUIDs section & notifications are off
  - Until the new USB backend is in place in Chrome

USB to Browser Links

Follow on Twitter: **@denladeside @reillyeon @kennethrohde**

https://developers.google.com/web/updates/2016/03/access-usb-devices-on-the-web

https://wicg.github.io/webusb/

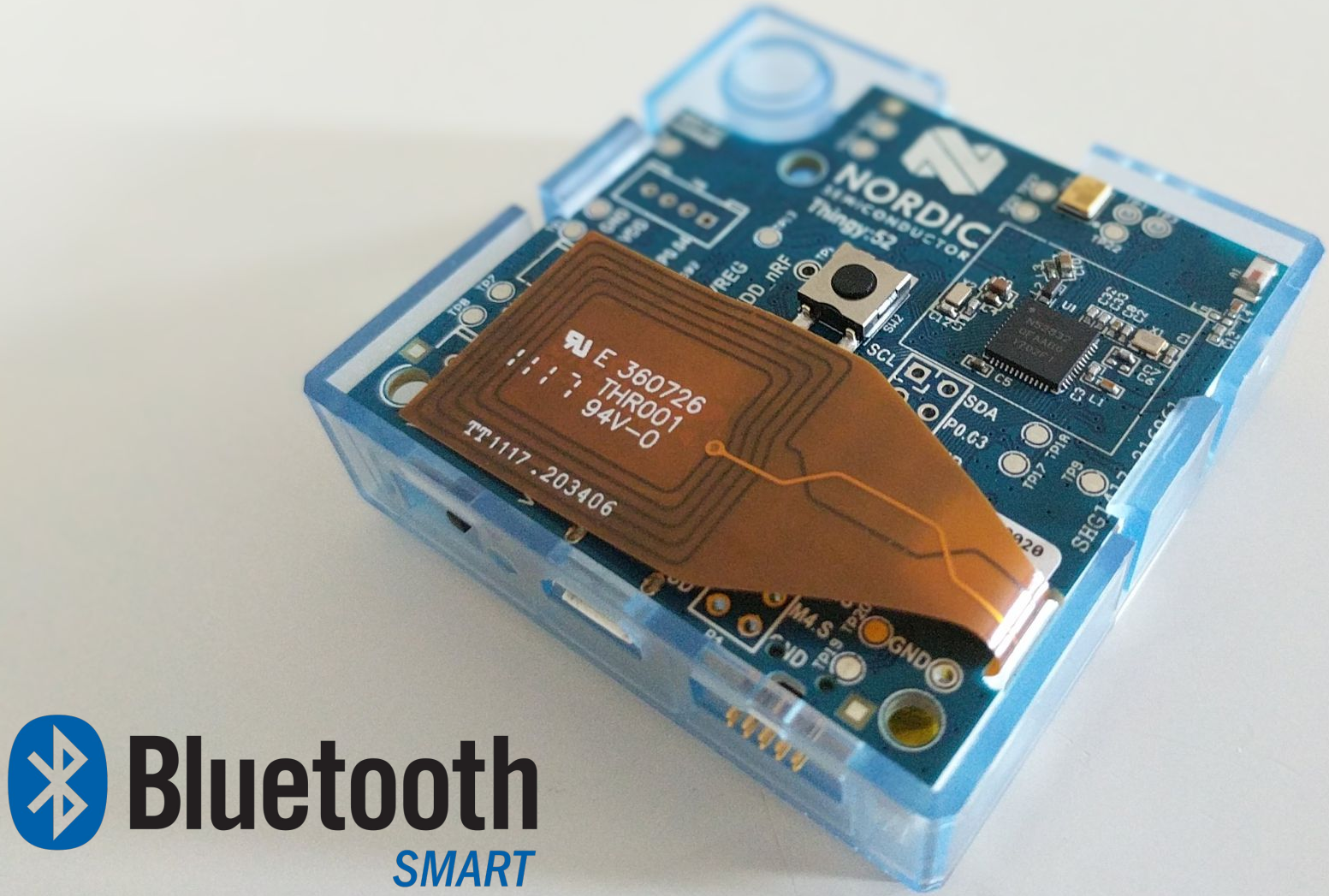
https://medium.com/@larsgk/web-enabling-legacy-devices-dc3ecb9400ed#.r41zt9a29

Examples of use with Web Components:

Web USB: https://github.com/empirikit

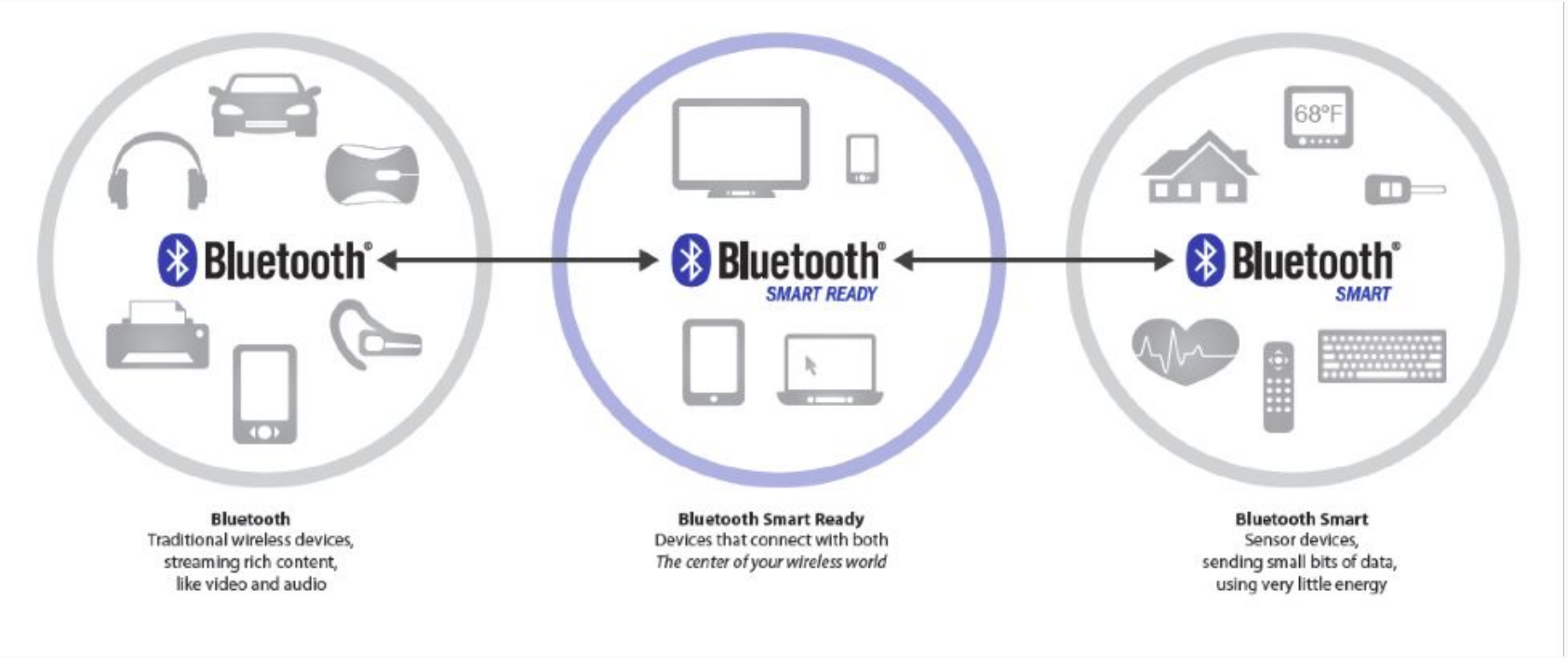
Web MIDI: https://github.com/larsgk/windmidi-service

Gamepad API: https://www.webcomponents.org/element/larsgk/gamepad-service



Bluetooth
SMART

Bluetooth Low Energy (BLE) == Bluetooth SMART

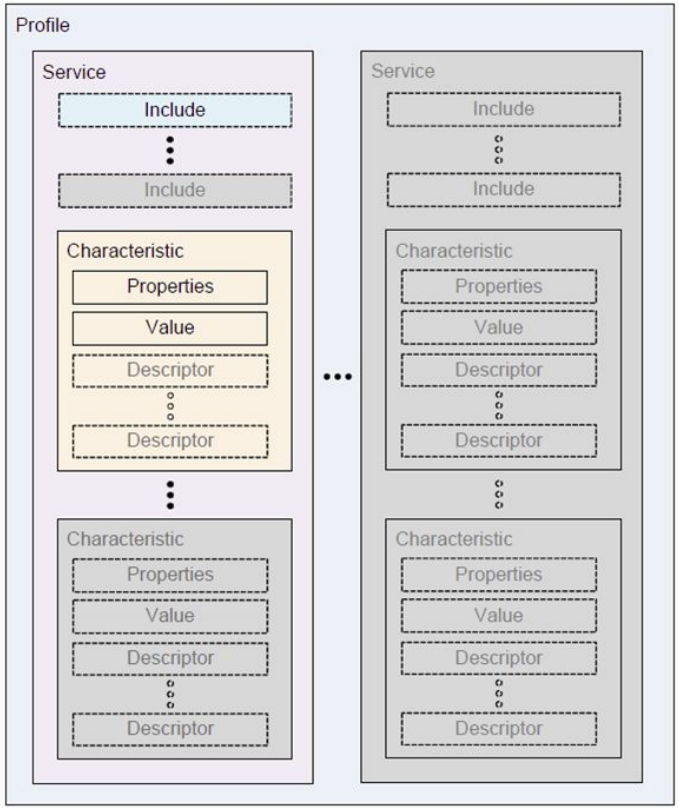


Source: Bluetooth SIG

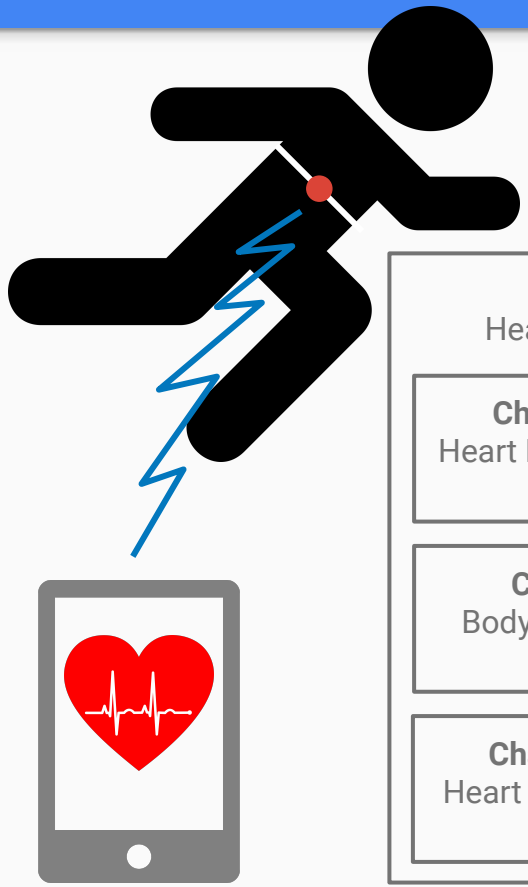
A few key features

- Advertising packets can contain a lot of info, which is utilized for e.g. beacons, where the listener **never has to connect**
- **Pairing is optional**, allowing for much more interesting public space hardware interaction
- Connections are **exclusive but fairly low latency** so units could be ‘time shared’ if it makes sense for the solution
- Using standard **GATT** identifiers allows users access from different vendors’ apps

Generic Attribute Profile (GATT)



Source: Bluetooth SIG



Service
Heart Rate Service (0x180D)

Characteristic (mandatory)
Heart Rate Measurement (0x2A37)
[notify]

Characteristic (optional)
Body Sensor Location (0x2A38)
[read]

Characteristic (conditional)
Heart Rate Control Point (0x2A39)
[write]

OS Support for BLE

- Windows: only recently added to Windows 10 with the Creators Update
- Linux: BlueZ 5.42 (late 2016) [GATT DBUS API out of experimental]
  - Initial support much earlier though (~ 2011)
- OSX (~ 2011)
- Android since 4.3 (mid 2013)
- iOS since iPhone 4S (late 2011)

Web Bluetooth basics

Can only be initiated by a user gesture and over HTTPS

Connect and read battery level

```
navigator.bluetooth.requestDevice({ filters: [{ services: ['battery_service'] }] })
.then(device => device.gatt.connect())
.then(server => {
  // Getting Battery Service...
  return server.getPrimaryService('battery_service');
})
.then(service => {
  // Getting Battery Level Characteristic...
  return service.getCharacteristic('battery_level');
})
.then(characteristic => {
  // Reading Battery Level...
  return characteristic.readValue();
})
.then(value => {
  console.log('Battery percentage is ' + value.getUint8(0));
})
.catch(error => { console.log(error); });
```

```
navigator.bluetooth.requestDevice({ filters: [{ services: ['heart_rate'] }] })
.then(device => device.gatt.connect())
.then(server => server.getPrimaryService('heart_rate'))
.then(service => service.getCharacteristic('heart_rate_measurement'))
.then(characteristic => characteristic.startNotifications())
.then(characteristic => {
  characteristic.addEventListener('characteristicvaluechanged',
                                  handleCharacteristicValueChanged);
  console.log('Notifications have been started.');
})
.catch(error => { console.log(error); });

function handleCharacteristicValueChanged(event) {
  var value = event.target.value;
  console.log('Received ' + value);
  // TODO: Parse Heart Rate Measurement value.

}
```

Connect and monitor heart rate

DEMO Thingy:52

Thingy:52

"...a compact, power-optimized, multi-sensor development kit. It is an easy-to-use development platform, designed to help you build IoT prototypes and demos, without the need to build hardware or write firmware." - https://www.nordicsemi.com/Software-and-tools/Prototyping-platforms/Nordic-Thingy-52

Core: nRF52832 Bluetooth® 5 SoC, ARM Cortex-M4F, 512kB flash + 64kB RAM

Sensors and actuators: Sound (mic), button, temperature, humidity, pressure, air quality, color, light, accelerometer, gyroscope and compass (mag), RGB LED, speaker and NFC

USBTronica

https://larsgk.github.io/usbtronica/

https://github.com/larsgk/usbtronica

Web Bluetooth Links

https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web

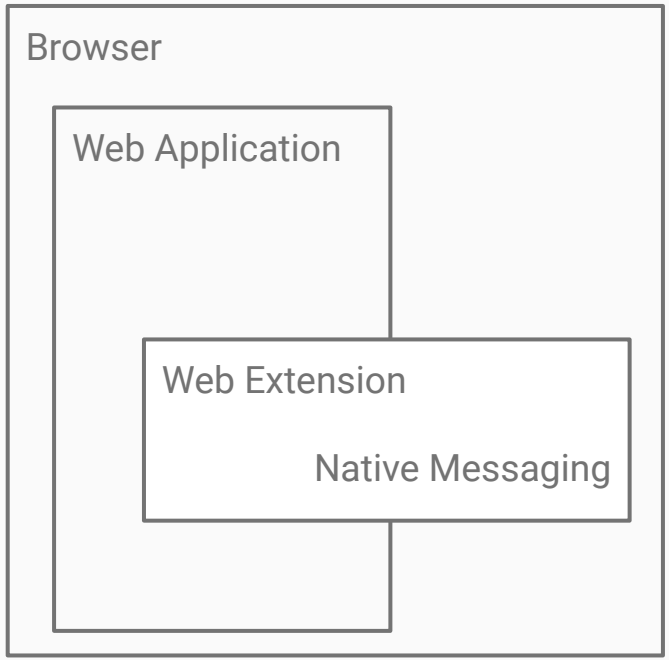
https://webbluetoothcg.github.io/web-bluetooth/

https://larsgk.github.io/webbluetooth-tester/index.html

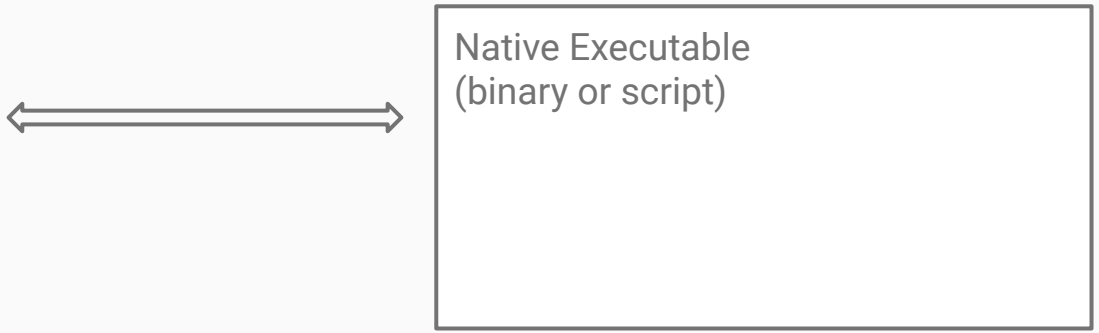
https://github.com/larsgk/webbluetooth-tester

Native Messaging

What is it?



“Native messaging enables an extension to exchange messages with a native application installed on the user's computer. This enables native applications to provide a service to extensions without needing to be reachable over the web.” - MDN web docs



Browser support and differences

- Chrome M29+ (mid 2013, Lin/Win/Mac)
  - Any executable (binary or script)
  - Comm: length prefixed JSON through stdin/stdout pipes
- Firefox 50 (late 2016, Lin/Win/Mac)
  - Very small differences from chrome
- Edge (based on Chromium - just released)
  - Should support the same as Chrome
  - Old EDGE requires a UWP extension.

Note: Safari 10 on OS X 10.12+ has it's own custom solution called Safari App Extensions

Native Messaging vs Web USB or Web Bluetooth

PROS

- Custom security/control mechanism
- More than just USB and BLE possible
- Possibly easier to migrate legacy code to extension binary than pure web
- Currently better browser support (but hopefully not for long)

CONS

- Requires local installation
- Fragmented update mechanisms
- Likely higher maintenance cost
- Native code on client's machines a potential security risk

Links

https://developer.chrome.com/extensions/nativeMessaging

https://developer.mozilla.org/en-US/Add-ons/WebExtensions/Native_messaging

https://docs.microsoft.com/en-us/microsoft-edge/extensions/guides/native-messaging

Recap (as of 2020.01)

For good coverage and good usability, traditional industry migrating legacy solutions could consider a combination of Web USB (or Web Bluetooth) for zero-install/zero-conf and a fallback to Native Messaging, where needed

** The old Edge has been phased out and is replaced with a new Chromium based one, which should have the same support as Chrome!!!

Desktop browsers	Web USB	Web Bluetooth	Native Messaging	Serial
Chrome (usage ~ 65%)	✔ (M61)	✔ (M56) (M70/Win)	✔ (M29)	OriginTrial M80
Firefox (usage ~ 10%)	Undecided	Considering	✔ (50)	Nothing public
Edge ** (usage unknown)	Should be available on launch	Should be available on launch	Extensions should work as in Chrome	Might come after Chrome Origin Trial
Safari (usage ~ 4%)	✗ Not Considering	✗ Not Considering	~ (10) similar solution	No info

Let’s build something!

Q & A

Daniel Bank

Twitter: @DanielPBank

@IoTDevFest

GitHub: danielbank

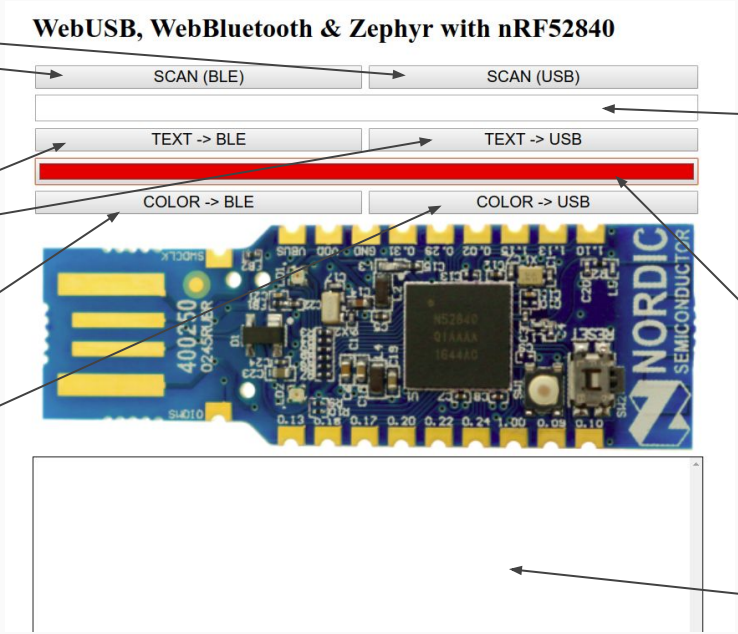
WORKSHOP

Repo WebApp usage - live: https://larsgk.github.io/web-nrf52-dongle/

Scan for BLE or connected USB devices

Send message text to all connected BLE or USB devices

Set RGB LED on connected BLE or USB devices to selected color



Message text to send (the firmware passes through messages to 'the other side', TX(USB)->RX(BLE) and TX(BLE)->RX(USB)

Select color

Connection status and messages received - prefixed with USB or BLE + unique IDs

Repo for This Workshop

Live Demo App:

https://larsgk.github.io/web-nrf52-dongle/

Repo:

https://github.com/danielbank/web-nrf52-dongle