Web Bluetooth Design Doc (2014)

Publicly visible document.
2014-07-22
jyasskin@chromium.org, scheib@chromium.org

Use Cases

- Allow web pages to communicate to a wide variety of devices:
  - Let Fitbit write a website instead of a native app.
  - Heart rate monitor
  - Barcode scanner
  - BT security tags
  - BT plant sensors

    - BT lightbulbs
    - Camera shutter remote (this is probably a HID device)
    - BT weather station
    - BT kitchen scale
    - BT enabled dive logger
    - Robotic car
    - The Physical Web initiative

Non-goals

- "Beacons" are out of scope because of the pairing problem: for security, we want users to explicitly pair individual devices with individual websites, but beacons need to pair a whole class of devices. Similarly, other BLE devices that don't require pairing with a computer may still require pairing with a website, to mitigate tracking/privacy concerns.
- Bluetooth speakers/microphones probably don't need this API because you pair them with the OS and then use navigator.getMediaDevices to use them the same as non-bluetooth devices.

Tentative Decisions

Bluetooth 4 Low Energy & GATT Initially

Bluetooth 4 protocols differ considerably from previous versions. Initial design will focus on the low energy protocol for simplicity and to solve the use cases perceived to be most in demand. The trade off is delaying support for many older devices. BTLE also appears more attractive from a security standpoint because the values written tend to follow very simple formats, which makes it more likely that their parsers will be secure. BT 2.1 uses a full socket/byte-stream protocol, which implies parsers in the devices that may not have been secured against malicious clients.

OS	Minimum version supporting BTLE
Android	Android 4.3 (Jellybean.3) (API Level 18) July 24, 2013
ChromeOS	M37 Late summer 2014
Windows	Windows 8; headers might not work until 8.1; 8.1 provides extra libraries; Chrome-driven pairing might not work until 9?
Mac	10.9/iOS 6 for Peripheral; iOS 5/10.? for Central
Linux	None yet

Central rather than Peripheral initially

Central and Peripheral devices act as transmitters and receivers, in a master and slave configuration respectively. Central is more useful for a general-purpose computer, although of course Peripheral will be useful eventually. Of the platforms that support BLE at all, only recent (as of 2014) versions of Mac/iOS and Android support peripheral.

Security and Privacy Risks

See the specification for updated analysis and mitigations, and security model post.

Device Selection / Pairing UI

Devices may already be paired, or may not yet have been paired. [Discuss needs for UI to pair new devices, select devices]
See USB getUserSelectedDevices Chrome API Proposal's proposal for a modal dialog to select devices.

Other APIs in this area

Chrome Apps

The Chrome Bluetooth API (design doc) is an API of free functions and events, where each function is passed enough IDs to recover the state it needs to operate.

```
bluetooth.startDiscovery() -> bluetooth.onDeviceAdded(function(Device){...}) event
bluetooth.getDevices() -> [Device]

bluetooth.Device {
dictionary Device {
   { address, name, deviceClass }
   { vendorIdSource, vendorId, productId, deviceId }
   type;  // Summarizes deviceClass.
   { paired, connected }
   uuids;  // All protocols, profiles, and services.
}
bLE = bluetoothLowEnergy;
bLE.connect(device.address, function(){...})
bLE.getServices(device.address, function([service]){...})
bLE.getCharacteristics(service.instanceId, function([characteristic]){...})
bLE.getDescriptors(characteristic.instanceId, function([descriptor]){...})

// Notifies when details of a Service changes on a connected device. This happens
asynchronously after the .connect() calls its callback.
bLE.onService{Added,Changed,Removed}(function(service){...})
bLE.readCharacteristicValue(characteristic.instanceId, function(characteristic){
  characteristic.value : ArrayBuffer;
})
bLE.writeCharacteristicValue(characteristic.instanceId, arraybuffer, function(){...})
bLE.readDescriptorValue(descriptor.instanceId, function(descriptor){...})
bLE.writeDescriptorValue(descriptor.instanceId, arraybuffer, function(){...})
bLE.startCharacteristicNotifications(characteristic.instanceId)
bLE.onCharacteristicValueChanged(function(characteristic){...})
bLE.stopCharacteristicNotifications(characteristic.instanceId)
bLE.onDescriptorValueChanged(function(descriptor){...})  // But no notifications.
```

There's IDL available at extensions/api/bluetooth.idl and extensions/api/bluetooth_low_energy.idl.

Mozilla

BootToGecko has a WebBluetooth spec for classic, not LE, bluetooth. THIS SPEC DOES NOT ALLOW COMMUNICATING WITH DEVICES, just administering them. Mozilla does appear interested in publishing a spec to allow socket access, they just haven't done so yet.

"Addresses" are probably 48-bit bluetooth hardware IDs formatted like "00:11:22:AA:BB:CC". "UUIDs" are probably 128-bit UUIDs.

```
navigator.bluetooth = {
  defaultAdapter : BluetoothAdapter
  getAdapters() -> sequence<BluetoothAdapter>
}

BluetoothAdapter {
  state ∈ {  "disabled",  "disabling",  "enabled",  "enabling" }
  { address, name, discoverable, discovering }
  enable();
  disable();
  setName(DOMString aName);
  setDiscoverable(boolean aDiscoverable);
  startDiscovery() -> ~Stream<BluetoothDevice>;
  stopDiscovery();
  pair(DOMString aAddress);
  unpair(DOMString aAddress);
  getPairedDevices() -> Promise<BluetoothDevice[]>;
  ondevicepaired -> BluetoothDevice
  ondeviceunpaired -> address
}
BluetoothDevice {
  { address, cod:BluetoothClassOfDevice, name, paired }
  uuids : String[];  // Bluetooth services
  fetchUuids();  // Updates uuids.
}
```

Android

Android 4.3 added BTLE support. The L release adds Peripheral support and support for filtering scans and setting their power level. (Reference download; not online yet)

Discovery:
((BluetoothManager)getSystemService(BLUETOOTH_SERVICE)).getAdapter() -> BluetoothAdapter
BluetoothAdapter.getBluetoothLeScanner().startScan(filters : [ScanFilter], settings, ScanCallback)

Settings:

- Mode: LOW_POWER, BALANCED, or LOW_LATENCY.

Filtering (android.bluetooth.le.ScanFilter.Builder):

- device mac address
- manufacturerData, possibly masked
- local name
- rssi range (min-max)
- service data, possibly masked
- service uuid, possibly masked

```
ScanCallback provides a stream of ScanResults {
  BluetoothDevice
  rssi
  scan record {
    int advertising flags indicating the discoverable mode and capability of the device.
    string local name of the BLE device.
    int manufacturer identifier, which is a non-negative number assigned by Bluetooth SIG.
    byte[] manufacturer specific data which is the content of manufacturer specific data field.
    byte[] service data.
    uuid of the service that the service data is associated with.
    uuid[] list of gatt services.
    int transmission power level of the packet in dBm
  }
  timestamp
}

BluetoothDevice {
  { address, name, BluetoothClass }
  pairing control
  BluetoothGatt connectGatt(Context context, boolean autoConnect, BluetoothGattCallback
callback)
```

```
  // The BluetoothGatt object is used to control the connection, while `callback` is used to receive results and completion notifications.
}

BluetoothGatt {
  connect(); disconnect(); close();

  discoverServices() -> onServicesDiscovered(BluetoothGatt, status) -> getServices() : [BluetoothGattService]

  readCharacteristic(BluetoothGattCharacteristic) ->  onCharacteristicRead(BluetoothGatt, BluetoothGattCharacteristic, status) -> characteristic.getValue()
  readDescriptor(BluetoothGattDescriptor) ->  onDescriptorRead(BluetoothGatt, BluetoothGattDescriptor, status) -> descriptor.getValue()
  readRemoteRssi() -> onReadRemoteRssi(BluetoothGatt, rssi, status)
  setCharacteristicNotification(BluetoothGattCharacteristic, enable) -> onCharacteristicRead()
  writeCharacteristic(BluetoothGattCharacteristic) -> onCharacteristicWrite(BluetoothGatt, BluetoothGattCharacteristic, status)
  writeDescriptor(BluetoothGattDescriptor) -> onDescriptorWrite(BluetoothGatt, BluetoothGattDescriptor, status)


  // Transaction support. Batches writeCharacteristic() ->  onCharacteristicWrite() calls to be applied atomically.
  // Applications are responsible for checking that onCharacteristicWrite reports the expected value.
  beginReliableWrite() …  executeReliableWrite() |  abortReliableWrite()

}

BluetoothGattService {
  getIncludedServices() : [BluetoothGattService]
  getCharacteristics() : [BluetoothGattCharacteristic]
  getInstanceId()  // Distinguishes multiple instances of the same service on one device.
  getUuid()
}

BluetoothGattCharacteristic {
  getDescriptors() : [BluetoothGattDescriptor]
  getService()
  getUuid()
  int getPermissions()
  getValue() : byte[]; setValue(byte[])
  // Several more convenience functions to decode/encode bytes at a particular offset in the
value with formats {float, float16, {u,s}int{8,16,32}
}

BluetoothGattDescriptor {
  getCharacteristic()
  int getPermissions()
  getUuid()
  getValue() : byte[]; setValue(byte[])
  // No convenience functions.
}
```

Apple

Core Bluetooth provides both Central and Peripheral support, including from background apps.

Key Classes:
CBCentralManager
    local central communicated with CBPeripheral, CBService, CBCharacteristic
CBPeripheralManager
    local peripheral communicating with remote CBCentral
    creates services CBMutableService, CBMutableCharacteristic

```
[myCentralManager scanForPeripheralsWithServices:nil options:nil]; // can filter by service
->:didDiscoverPeripheral:advertisementData:RSSI: // called for each device
[myCentralManager stopScan];
[myCentralManager connectPeripheral:peripheral options:nil];
->:didConnectPeripheral: // called on connect

[peripheral discoverServices:nil];
->:didDiscoverServices:
[peripheral discoverCharacteristics:nil forService:interestingService];
->:didDiscoverCharacteristicsForService:error:

// Subscribing to characteristic's values
[peripheral setNotifyValue:YES forCharacteristic:interestingCharacteristic];
->:didUpdateNotificationStateForCharacteristic:error:  // handle errors
->:didUpdateValueForCharacteristic:error:

// Writing the Value of a Characteristic
[peripheral writeValue:dataToWrite forCharacteristic:interestingCharacteristic
      type:CBCharacteristicWriteWithResponse]; // or WithoutResponse
->:didWriteValueForCharacteristic:error:
```

to be continued, from Background Processing

Windows

Windows 8.1 provides a set of classes under Windows.Devices.Bluetooth.GenericAttributeProfile. See Supporting Bluetooth Devices.

Regarding APIs

> Kiran Pathakota <kipathak@microsoft.com>, Program Manager at Microsoft and from what I understand we'll have to switch to WinRT APIs to support requestDevice and requestLEScan for Windows 10+.
>
> Here are some quotes from our conversation:
>
> *Unfortunately, those* [Windows Driver APIs] *APIs will not have support for discovery of unpaired devices. An example of the WinRT APIs (that you can still call from a Win32 app) are shown here: https://github.com/kpathakota/Build2016BluetoothCodeSamples/tree/master/BluetoothInAppGATT*
>
> *These APIs were first delivered in Windows 10 so unfortunately there isn't any downlevel support.*
>
> *Discovery of unpaired devices has already been added to WinRT APIs (as of Windows 10 Build 10586 – shipped October last year). See this sample for more details (Scenario 8 and 9 show how to query for unpaired Bluetooth devices):https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/DeviceEnumerationAndPairing/cs*

Also,

- https://github.com/Microsoft/Windows-universal-samples/tree/master/Samples/DeviceEnumerationAndPairing
- https://github.com/urish/win-ble-cpp

Tizen

See here. Not BTLE.

```
var adapter = tizen.bluetooth.getDefaultAdapter();
adapter.createBonding("35:F4:59:D1:7A:03", function(device){...},
onErrorCallback);
device.connectToServiceByUUID(serviceUUID, function(socket){...},
                              onSocketError);
var length = socket.writeData(sendtextmsg);
var data = socket.readData();
```

Bluez
A Linux stack I haven't investigated yet; appears to be controlled over DBus; probably not worth investigating for the web.

GATT REST API
Published by Bluetooth's Internet WG in April 2014. Describes a JSON format for describing nodes, characteristics, etc. Leaves pairing out of the spec, but provides full access to

Proposed API
https://github.com/WebBluetoothCG/web-bluetooth
- Use Cases and Security Requirements
- Explainer, showing how a site might use this API to satisfy the use cases
- Specification

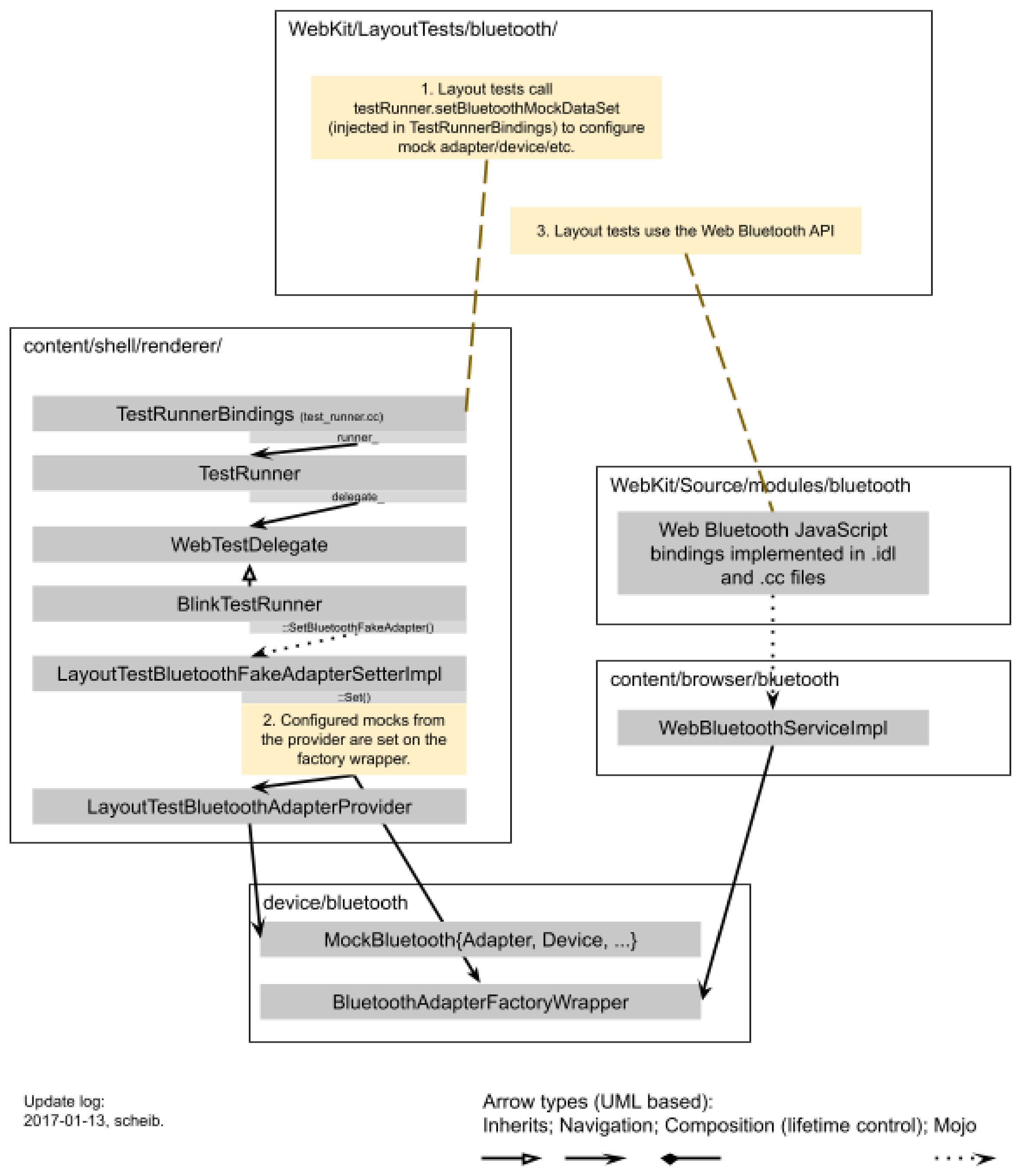
Outline
Discovery still TBD.

Start from the chrome.bluetoothLowEnergy API to represent services, characteristics, and descriptors. Switch everything to Promises. Where the Chrome API function takes an instanceID to represent an object, make the function into a method on that object, to be more consistent with the rest of the web. Figure out whether lookup methods should take UUIDs instead, or whether we'll still need instanceIDs to distinguish the same characteristic on multiple services.

Chrome & Blink API Implementation
The Web Bluetooth API is implemented by bridging the Bluetooth classes in src/device/bluetooth through Content and into a Blink module. Security is implemented in the browser process of the Content module. Here is an overview of the code components:
- **Bluetooth device implementation**
  - src/device/bluetooth
  - Contains cross platform abstractions for interacting with bluetooth devices.
- **Content Module**
  - src/content/browser/bluetooth
    - WebBluetoothServiceImpl
  - src/content/common/bluetooth
  - Bridges between the **Bluetooth device implementation** and the **Blink Platform API**, on the Browser and Renderer processes respectively.
  - Validation of security concerns are made on the Browser side, e.g. validating which services of a device may be accessed by a site.
  - **Testing** is exposed to layout tests by testRunner.SetBluetoothMockDataSet which configures different test data sets using the src/device/bluetooth/test mocks.

- **Blink Platform API**
    - src/third_party/WebKit/public/platform/modules/bluetooth
    - Expose interfaces for blink to access bluetooth functionality
- **Javascript API**
    - src/third_party/WebKit/Source/modules/bluetooth
    - Exposes Javascript interface via IDL and C++ implementation.
- **LayoutTests**
    - src/third_party/WebKit/LayoutTests/bluetooth
    - **Functional tests, span the javascript bindings, blink public API, mojo IPCs, finally to the src/device/bluetooth classes.**
    - Based on testharness, but many not portable to other browsers as they rely on testRunner.SetBluetoothMockDataSet (injected in Content module)
        - Current plan is to develop tests then formalize the testing dependencies and add them to the Web Bluetooth specification.

UML for how layout tests control the bluetooth mock:


WebKit/LayoutTests/bluetooth/
1. Layout tests call testRunner.setBluetoothMockDataSet (injected in TestRunnerBindings) to configure mock adapter/device/etc.
3. Layout tests use the Web Bluetooth API
content/shell/renderer/
TestRunnerBindings (test_runner.cc)
runner_
TestRunner
delegate_
WebTestDelegate
BlinkTestRunner
::SetBluetoothFakeAdapter()
LayoutTestBluetoothFakeAdapterSetterImpl
::Set()
2. Configured mocks from the provider are set on the factory wrapper.
LayoutTestBluetoothAdapterProvider
WebKit/Source/modules/bluetooth
Web Bluetooth JavaScript bindings implemented in .idl and .cc files
content/browser/bluetooth
WebBluetoothServiceImpl
device/bluetooth
MockBluetooth{Adapter, Device, ...}
BluetoothAdapterFactoryWrapper
Update log:
2017-01-13, scheib.
Arrow types (UML based):
Inherits; Navigation; Composition (lifetime control); Mojo

Mock Device Function Test Plan

The Web Bluetooth implementation is primarily ‘plumbing’ with only small amounts of logic at the various layers. Testing will primarily focus on functional testing from the JavaScript layer down to the src/device/bluetooth classes, which in turn have unit and browser tests.

Functional layout tests use testRunner.SetBluetoothMockDataSet() to specify a deterministic set of mock devices, and then Web Bluetooth API calls will be made with expected results verified.

Mock devices are implemented in src/device/bluetooth, and configured in content/shell/browser/layout_test/layout_test_bluetooth_adapter_provider.cc.

Mock Devices Design Doc

Implementation and Design Notes

Android LE Scan only

Cross platform BluetoothDiscoveryFilter supports TRANSPORT_DUAL = (TRANSPORT_CLASSIC | TRANSPORT_LE), however Android system only supports scanning for Classic or LE, but not both. Actual adapters have this limitation as well, but operating systems, such as ChromeOS|BlueZ, manage timesharing internally. Our implementation of Android Bluetooth will only perform LE scans. (scheib, armansito)

navigator.bluetooth.requestDevice LE-only scan

There isn't much support for GATT over BR/EDR from neither platforms nor devices so performing a Dual scan will find devices that the API is not able to interact with. To avoid wasting power and confusing users with devices they are not able to interact with, navigator.bluetooth.requestDevice only performs an LE Scan.

Device Selection User Interface Design

An example user flow provisioning a device:

1. User purchases a new appliance Widget for the home that can be configured using bluetooth. The device packaging invites the user to navigate to a site example.com/setup.
2. example.com/setup displays a welcome message
   - “It’s easy to setup your Widget using a phone, tablet, or computer with bluetooth. Click Here to connect to your Widget!”
3. User clicks, and the browser presents a dialog with text and a list of devices that is populated over the next few seconds as the browser scans for bluetooth devices.: (Strings are example and need to be refined:)
   - example.com requests permission control a device, select one:
   - [List of devices, showing as much information as we can about them, but filtered to only devices offering the service that the website requests, in most situations this will be a very short list that starts empty and then finds one item]
     - Widget Thing
   - [Connect][Connect And Remember][No]
4. [Presuming a device selected] Dialog is removed and website changes display to configure the Widget.

Standardized bluetooth services exist. In the event that a site requests devices for only standardized services we may choose to display that information directly. E.g. if a site asks for “Heart rate” and “Cycling power” the message presented for requestDevices dialog may be “example.com requests perfmission to access Heart rate and Cycling power from: [list of devices]”

Website has already paired a device, and then needs to add an additional service.

- e.g. a website has been given access to a heart rate monitor service. Then, later it determines that it needs access to an additonal service such as heart rate history.

Website is revisited after having previously been approved to access services on a device

- Would like user to be able to have device permissions persist for a website.

Website requests access for services that are not currently available on any device in proximity. If a device comes into proximity later, a user is prompted that websites A, B, C are interested in accessing that device.

-

Development

On Linux with stubs

- GYP_DEFINE chromeos=1
- Use the stubs (enabled by default)
- Get a fake heart rate service there, which you can use for development
    - Add more stubs, modify the fake_bluetooth_gatt_*_client.h|cc files

On Linux with real devices

- GYP_DEFINE chromeos=1
- --dbus-unstub-clients=bluetooth
- bluez from source
    - apply Chrome OS patches 5.24
        - git clone git://git.kernel.org/pub/scm/bluetooth/bluez.git
        - git checkout 5.24
        - README for
            - ./configure && make
                - probably don't install on a desktop that has 4, it'll break.
                - stop the system, and run the built,
                    - stop bluetoothd
                    - run local version with -n (no deamon)

On ChromeOS

- building-chromium-browser for chromeos
    - use a test image

Reference

- USB getUserSelectedDevices Chrome API Proposal
- W3C NFC proposal: this spec should try to support the "handover" mentioned in that spec, without blocking on that spec's implementation.
- https://github.com/dontcallmedom/web-bluetooth <- Dominique Hazaël-Massieux's research.