

Bluetooth LE plugin for Xamarin/MAUI, supporting Android, iOS, Mac, Windows

Apache-2.0 license

634 stars 276 forks

Star

Watch

Code Issues 225 Pull requests 5 Discussions Actions Projects Wiki Security Insights

master

...

janusw README: add some links ...

7 hours ago 772

View code

README.md



Bluetooth LE plugin for Xamarin & MAUI

Build status: ci/github-actions passing

Xamarin, MAUI and MvvmCross plugin for accessing the bluetooth functionality. The plugin is loosely based on the BLE implementation of Monkey Robotics.







Important Note: With the term "vanilla" we mean the non-MvvmCross version, i.e. the pure Xamarin or MAUI plugin. You can use it without MvvmCross, if you download the vanilla package.

Support & Limitations

Release Notes

Platform	Version	Limitations
Xamarin.Android	4.3	
Xamarin.iOS	7.0	
Xamarin.Mac	10.9 (Mavericks)	>= 2.1.0
Xamarin.UWP	1709 - 10.0.16299	>= 2.2.0
MAUI (all 4 OS)		>= 3.0.0

Nuget Packages

package	stable	beta	downloads
Plugin.BLE			
MvvmCross.Plugin.BLE			

Installation

Vanilla

```
// stable
Install-Package Plugin.BLE
// or pre-release
Install-Package Plugin.BLE -Pre
```

MvvmCross

```
Install-Package MvvmCross.Plugin.BLE
// or
Install-Package MvvmCross.Plugin.BLE -Pre
```

Android

Add these permissions to AndroidManifest.xml. For Marshmallow and above, please follow [Requesting Runtime Permissions in Android Marshmallow](#) and don't forget to prompt the user for the location permission.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Android 12 and above may require one or more of the following additional runtime permissions, depending on which features of the library you are using (see [the android Bluetooth permissions documentation](#))

```
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />
```

Add this line to your manifest if you want to declare that your app is available to BLE-capable devices **only**:

```
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
```

iOS

On iOS you must add the following keys to your Info.plist

```
<key>UIBackgroundModes</key>
<array>
```

```

    <!--for connecting to devices (client)-->
    <string>bluetooth-central</string>

    <!--for server configurations if needed-->
    <string>bluetooth-peripheral</string>
</array>

<!--Description of the Bluetooth request message (required on iOS 10, deprecated)-->
<key>NSBluetoothPeripheralUsageDescription</key>
<string>YOUR CUSTOM MESSAGE</string>

<!--Description of the Bluetooth request message (required on iOS 13)-->
<key>NSBluetoothAlwaysUsageDescription</key>
<string>YOUR CUSTOM MESSAGE</string>

```

MacOS

On MacOS (version 11 and above) you must add the following keys to your Info.plist :

```

<!--Description of the Bluetooth request message-->
<key>NSBluetoothAlwaysUsageDescription</key>
<string>YOUR CUSTOM MESSAGE</string>

```

UWP

Add this line to the Package Manifest (.appxmanifest):

```
<DeviceCapability Name="bluetooth" />
```

Sample app

We provide a sample Xamarin.Forms app, that is a basic bluetooth LE scanner. With this app, it's possible to

- check the ble status
- discover devices
- connect/disconnect
- discover the services
- discover the characteristics
- see characteristic details
- read/write and register for notifications of a characteristic

Have a look at the code and use it as starting point to learn about the plugin and play around with it.

Usage

Vanilla

```

var ble = CrossBluetoothLE.Current;
var adapter = CrossBluetoothLE.Current.Adapter;

```

MvvmCross

The MvvmCross plugin registers `IBluetoothLE` and `IAdapter` as lazy initialized singletons. You can resolve/inject them as any other MvvmCross service. You don't have to resolve/inject both. It depends on your use case.

```
var ble = Mvx.Resolve<IBluetoothLE>();  
var adapter = Mvx.Resolve<IAdapter>();
```

or

```
MyViewModel(IBluetoothLE ble, IAdapter adapter)  
{  
    this.ble = ble;  
    this.adapter = adapter;  
}
```

Please make sure you have this code in your `LinkerPleaseLink.cs` file

```
public void Include(MvvmCross.Plugins.BLE.Plugin plugin)  
{  
    plugin.Load();  
}
```

IBluetoothLE

Get the bluetooth status

```
var state = ble.State;
```

You can also listen for State changes. So you can react if the user turns on/off bluetooth on your smartphone.

```
ble.StateChanged += (s, e) =>  
{  
    Debug.WriteLine($"The bluetooth state changed to {e.NewState}");  
};
```

IAdapter

Scan for devices

```
adapter.DeviceDiscovered += (s,a) => deviceList.Add(a.Device);  
await adapter.StartScanningForDevicesAsync();
```

Scan Filtering

```
var scanFilterOptions = new ScanFilterOptions();  
scanFilterOptions.ServiceUuids = new [] {guid1, guid2, etc}; // cross platform filter  
scanFilterOptions.ManufacturerDataFilters = new [] { new ManufacturerDataFilter(1), new ManufacturerDataFilter(2) };  
scanFilterOptions.DeviceAddresses = new [] { "80:6F:B0:43:8D:3B", "80:6F:B0:25:C3:15", etc }; // android only filter  
await adapter.StartScanningForDevicesAsync(scanFilterOptions);
```

ScanTimeout

Set `adapter.ScanTimeout` to specify the maximum duration of the scan.

ScanMode

Set `adapter.ScanMode` to specify scan mode. It must be set **before** calling `StartScanningForDevicesAsync()`. Changing it while scanning, will not affect the current scan.

Connect to device

`ConnectToDeviceAsync` returns a `Task` that finishes if the device has been connected successful. Otherwise a `DeviceConnectionException` gets thrown.

```
try
{
    await _adapter.ConnectToDeviceAsync(device);
}
catch(DeviceConnectionException e)
{
    // ... could not connect to device
}
```

Connect to known Device

`ConnectToKnownDeviceAsync` can connect to a device with a given GUID. This means that if the device GUID is known, no scan is necessary to connect to a device. This can be very useful for a fast background reconnect. Always use a cancellation token with this method.

- On **iOS** it will attempt to connect indefinitely, even if out of range, so the only way to cancel it is with the token.
- On **Android** this will throw a GATT ERROR in a couple of seconds if the device is out of range.

```
try
{
    await _adapter.ConnectToKnownDeviceAsync(guid, cancellationToken);
}
catch(DeviceConnectionException e)
{
    // ... could not connect to device
}
```

Get services

```
var services = await connectedDevice.GetServicesAsync();
```

or get a specific service:

```
var service = await connectedDevice.GetServiceAsync(Guid.Parse("ffe0ecd2-3d16-4f8d-90de-e89e7fc396a5"));
```

Get characteristics

```
var characteristics = await service.GetCharacteristicsAsync();
```

or get a specific characteristic:

```
var characteristic = await service.GetCharacteristicAsync(Guid.Parse("d8de624e-140f-4a22-8594-e2216b84a5f2"))
```

Read characteristic

```
var bytes = await characteristic.ReadAsync();
```

Write characteristic

```
await characteristic.WriteAsync(bytes);
```

Characteristic notifications

```
characteristic.ValueUpdated += (o, args) =>
{
    var bytes = args.Characteristic.Value;
};

await characteristic.StartUpdatesAsync();
```

Get descriptors

```
var descriptors = await characteristic.GetDescriptorsAsync();
```

Read descriptor

```
var bytes = await descriptor.ReadAsync();
```

Write descriptor

```
await descriptor.WriteAsync(bytes);
```

Get System Devices

Returns all BLE devices connected or bonded (only Android) to the system. In order to use the device in the app you have to first call `ConnectAsync`.

- For iOS the implementation uses get [retrieveConnectedPeripherals\(services\)](#)
- For Android this function merges the functionality of the following API calls:
 - [getConnectedDevices](#)
 - [getBondedDevices\(\)](#)

```
var systemDevices = adapter.GetSystemConnectedOrPairedDevices();
```

```
foreach(var device in systemDevices)
```

```
{  
    await _adapter.ConnectToDeviceAsync(device);  
}
```

Caution! Important remarks / API limitations

The BLE API implementation (especially on **Android**) has the following limitations:

- *Characteristic/Descriptor Write*: make sure you call characteristic.**WriteAsync**(...) from the **main thread**, failing to do so will most probably result in a GattWriteError.
- *Sequential calls*: **Always** wait for the previous BLE command to finish before invoking the next. The Android API needs its calls to be serial, otherwise calls that do not wait for the previous ones will fail with some type of GattError. A more explicit example: if you call this in your view lifecycle (onAppearing etc) all these methods return **void** and 100% don't guarantee that any **await bleCommand()** called here will be truly awaited by other lifecycle methods.
- *Scan with services filter*: On **specifically Android 4.3** the *scan services filter does not work* (due to the underlying android implementation). For android 4.3 you will have to use a workaround and scan without a filter and then manually filter by using the advertisement data (which contains the published service GUIDs).

Best practice

API

- Surround Async API calls in try-catch blocks. Most BLE calls can/will throw an exception in certain cases, this is especially true for Android. We will try to update the xml doc to reflect this.

```
try  
{  
    await _adapter.ConnectToDeviceAsync(device);  
}  
catch(DeviceConnectionException ex)  
{  
    //specific  
}  
catch(Exception ex)  
{  
    //generic  
}
```

- **Avoid caching of Characteristic or Service instances between connection sessions**. This includes saving a reference to them in your class between connection sessions etc. After a device has been disconnected all Service & Characteristic instances become **invalid**. Always use **GetServiceAsync** and **GetCharacteristicAsync** to get a valid instance.

General BLE iOS, Android

- **Scanning**: Avoid performing ble device operations like Connect, Read, Write etc while scanning for devices. Scanning is battery-intensive.
 - try to stop scanning before performing device operations (connect/read/write/etc)
 - try to stop scanning as soon as you find the desired device
 - never scan on a loop, and set a time limit on your scan

How to build the nuget package

1. Build

Open a console, change to the folder "dotnet-bluetooth-le/.build" and run `cake`.

2. pack the nuget

```
nuget pack ../Source/Plugin.BLE/Plugin.BLE.csproj
```

```
nuget pack ../Source/MvvmCross.Plugins.BLE/MvvmCross.Plugins.BLE.csproj
```

Extended topics

- [How to set custom trace method?](#)
- [Characteristic Properties](#)
- [Scan Mode Mapping](#)
- [iOS state restoration \(basic support\)](#)

Useful Links

- [Android Bluetooth LE guideline](#)
- [iOS CoreBluetooth Best Practices](#)
- [iOS CoreBluetooth Background Modes](#)
- [MvvmCross](#)
- [Monkey Robotics](#)

How to contribute

We usually do our development work on a branch with the name of the milestone. So please base your pull requests on the currently open development branch.

Licence

[Apache 2.0](#)

Releases 11

 **2.1.3** Latest
on May 1, 2022

[+ 10 releases](#)

Packages

No packages published

Used by 57



Contributors 42



[+ 31 contributors](#)

Languages

● C# 98.2% ● PowerShell 1.8%