ANDROID BLUETOOTH MULTIPLAYER

by

LOST POLYGON

v2.0.0

Contents

General information

Android Bluetooth Multiplayer gives you an ability to add Bluetooth multiplayer to your Android game using a simple API, similar to that of Unity Network component. It is also fully compatible with Unity built-in networking. This means you can easily reuse your networking code for Internet and local gaming with minimal changes, or use any of existing tutorials about Unity built-in networking.

Plugin overrides built-in Unity activity and adds new permissions to AndroidManifest.xml in order to function. AndroidManifest.xml is generated automatically in case it is not present. You can also do that manually. To do that, in case your project doesn't uses any plugins that modify AndroidManifest.xml, use

Tools → Lost Polygon → Android Bluetooth Mutiplayer → Generate AndroidManifest.xml

menu item, otherwise try

Tools → Lost Polygon → Android Bluetooth Mutiplayer → Patch existing AndroidManifest.xml

That will work for most cases. In case of any problems, refer to *"Configuring AndroidManifest.xml and extending Activities"* section of this documentation.

All the plugin code resides in `LostPolygon.AndroidBluetoothMultiplayer` namespace.

AndroidBluetoothMultiplayer class wraps all interactions between Unity and Java. All methods are declared static, so no object instantiation is required. This component, attached to a GameObject, is also required to successfully receive callbacks from Java side. It will be created automatically when you use it, so you don't have to do that manually.

Do not forget to unregister the event listeners upon destruction of objects that use the events (for example, at `MonoBehaviour.OnDestroy()` or `MonoBehaviour.OnDisable()`), as not doing that may lead to memory leaks and undefined behavior.

Do not rename the *AndroidBluetoothMultiplayer* class, as that will break Java callbacks.

It is important to call `Init(string uuid)` before any other interaction with the plugin. UUID is an identifier that must be unique for every Bluetooth application.

Bluetooth connection can only be established if the connecting and host device have the same UUID. You can generate random unique UUID for your game by using

Component → Lost Polygon → Android Bluetooth Mutiplayer → UUID generator

The package has three demo scenes included (see "AndroidBluetoothMultiplayer /Demos/"). First one shows simple Bluetooth multiplayer interaction. Second one is an example of how to discover nearby Bluetooth devices. Third demo demonstrates usage of RPC in order to transfer large blocks of data over Bluetooth.

It is highly recommended running demos on your device firsts to see if plugin is working okay. The Demos directory and its contents can be safely deleted for production, or just if you don't need it anymore.

Note: as Android's Bluetooth implementation guarantees data delivery, it is highly discouraged to use reliable state synchronization for NetworkView's, as that may lead to delays and stuttering.

Note: Device discovery is a heavyweight procedure. New connections to remote Bluetooth devices should not be attempted while discovery is in progress, and existing connections will experience limited bandwidth and high latency. Because of that, `StopDiscovery()` is always called automatically when connecting to server.

Use the logcat while testing the plugin and debugging your game — some debug information is available only in the log, and most of it also requires calling `setVerboseLog(true)`.

Plugin is tested in Unity 4.x - 5.x. Both free and Pro versions are supported.

AndroidBluetoothMultiplayer methods overview

`bool`	`Initialize(string uuid)` Initializes the plugin and set the Bluetooth service UUID.
`bool`	`StartServer(ushort port)` Starts the server, listening for incoming Bluetooth connections.
`bool`	`Connect(string address, ushort port)` Connects to a Bluetooth device.
`bool`	`Stop()` Stops all Bluetooth connectivity.
`bool`	`StartListening()` Starts listening for new incoming connections.
`bool`	`StopListening()` Stops listening for new incoming connections.
`bool`	`ShowDeviceList(bool showAllDeviceTypes)` Shows the Bluetooth device picker dialog.
`bool`	`RequestEnableBluetooth()` Opens a dialog asking user to enable Bluetooth.
`bool`	`EnableBluetooth()` Enables the Bluetooth adapter, if possible.
`bool`	`DisableBluetooth()` Disables the Bluetooth adapter, if possible.
`bool`	`RequestEnableDiscoverability(int discoverabilityDuration)` Opens a dialog asking user to make device discoverable on Bluetooth.
`BluetoothMultiplayerMode`	`GetCurrentMode()` Returns the current plugin mode.
`bool`	`GetIsBluetoothEnabled()` Returns `true` if Bluetooth is currently enabled and ready for use.
`bool`	`GetIsBluetoothAvailable()` Returns `true` if Bluetooth is available on the device.
`BluetoothDevice`	`GetCurrentDevice()` Returns current Bluetooth device.
`bool`	`StartDiscovery()` Starts discovery of nearby discoverable Bluetooth devices.
`bool`	`StopDiscovery()` Stops discovery of nearby discoverable Bluetooth devices.

bool	`GetIsDiscovering()` Returns `true` if Bluetooth device discovery is going on.
bool	`GetIsDiscoverable()` Returns `true` if device is discoverable by other devices.
BluetoothDevice[]	`GetBondedDevices()` Returns an array of bonded (paired) Bluetooth devices.
BluetoothDevice[]	`GetNewDiscoveredDevices()` Returns an array of Bluetooth devices discovered during current discovery session.
BluetoothDevice[]	`GetDiscoveredDevices()` Returns an array of bonded (paired) Bluetooth devices and Bluetooth devices discovered during current discovery session.
bool	`SetRawPackets(bool doEnable)` Enables or disables raw packets. Use only if you know what you are doing.
void	`SetVerboseLog(bool doEnable)` Enables or disables verbose logging.

AndroidBluetoothMultiplayer methods

```
public static bool Initialize(string uuid)
```

Initializes the plugin and sets the Bluetooth service UUID.

Parameters

uuid	Bluetooth service UUID. Must be different for each game.

Returns

`true` on success, `false` if UUID format is incorrect.

```
public static bool StartServer(ushort port)
```

Starts the server that listens for incoming Bluetooth connections. Must be called before `Network.InitializeServer`.

Throws `BluetoothNotEnabledException` if called when Bluetooth was not enabled.

Parameters

port	Server port number. Must be the same as passed to `Network.InitializeServer`.

Returns

`true` on success, `false` on error.

```
public static bool Connect(string hostDeviceAddress, ushort port)
```

Connects to a Bluetooth device. Must be called before `Network.Connect`.

Throws `BluetoothNotEnabledException` if called when Bluetooth was not enabled.

Parameters

hostDeviceAddress	Address of host Bluetooth device to connect to.
port	Server port number. Must be the same as passed to `Network.Connect`.

Returns

`true` on success, `false` on error.

```
public static bool Stop()
```

Stops all Bluetooth connections. Client will disconnect from the server. Server will break connection with all the clients and then halt.

Returns

`true` on success, `false` on error.

```
public static bool StartListening()
```

Starts listening for new incoming connections if listening was disabled by `StopListening()`.For example, you should listen for connections while in game lobby, and stop listening when the actual game has started to make sure no new device could connect. Available only in Server mode.

Returns

`true` on success, `false` on error.

```
public static bool StopListening()
```

Stops listening for new incoming connections. For example, you should listen for connections while in game lobby, and stop listening when the actual game has started to make sure no new device could connect. Available only in Server mode.

Returns

`true` on success, `false` on error.

```
public static bool ShowDeviceList(bool showAllDeviceTypes = false)
```

Shows the Bluetooth device picker dialog. *Note:* this method may fail some on exotic Android modifications like Amazon Fire OS.

Throws `BluetoothNotEnabledException` if called when Bluetooth was not enabled.

Parameters

showAllDeviceTypes	Whether to show all types or devices (including headsets, keyboards etc.) or only data-capable.

Returns

`true` on success, `false` on error.

```
public static bool RequestEnableDiscoverability (int discoverabilityDuration = 120)
```

Opens a dialog asking user to make device discoverable on Bluetooth for `discoverableDuration seconds`. This will also request the user to turn on Bluetooth if it was not enabled.

On Android 4.0 and higher, setting the parameter to 0 allows making device discoverable “forever” (until discoverability is disabled manually or Bluetooth is disabled).

Parameters

discoverableDuration	The desired duration of discoverability (in seconds). Default value 120 seconds.

Returns

`true` on success, `false` on error.

```
public static BluetoothMultiplayerMode GetCurrentMode()
```

Returns the current plugin mode (None, Server, or Client).

Returns

Current plugin mode on success, `BluetoothMultiplayerMode.None` on error.

```
public static bool RequestEnableBluetooth ()
```

Opens a dialog asking the user to enable Bluetooth. It is recommended to use this method instead of `EnableBluetooth()` for more native experience.

Returns

`true` on success, `false` on error.

```
public static bool EnableBluetooth()
```

Enables the Bluetooth adapter, if possible.

Do not use this method unless you have provided a custom GUI acknowledging user about the action. Otherwise use `RequestBluetoothEnable()`.

Returns

`true` on success, `false` on error.

```
public static bool DisableBluetooth()
```

Disables the Bluetooth adapter, if possible.

Returns

`true` on success, `false` on error.

```
public static bool GetIsBluetoothEnabled()
```

Returns `true` if Bluetooth is currently enabled and ready for use.

Returns

`true` if Bluetooth connectivity is available and enabled, `false` otherwise.

```
public static bool GetIsBluetoothAvailable()
```

Returns whether the Bluetooth is available. Bluetooth can be unavailable if no Bluetooth adapter is present, or if some error occurred.

Returns

`true` if Bluetooth connectivity is available, `false` otherwise.

```
public static BluetoothDevice GetCurrentDevice()
```

Returns `BluetoothDevice` of current device the application runs on.

Returns

`BluetoothDevice` if Bluetooth connectivity is available and enabled, `null` otherwise or on error.

```
public static bool StartDiscovery()
```

Starts the process of discovering nearby discoverable Bluetooth devices.

The process is asynchronous and is usually held for 10-30 seconds in time. Note that performing device discovery is a heavy procedure for the Bluetooth adapter and will consume a lot of its resources and drain battery power.

Returns

`true` if Bluetooth connectivity is available and enabled, `false` otherwise.

```
public static bool StopDiscovery()
```

Stops the process of discovering nearby discoverable Bluetooth devices.
Because discovery is a heavyweight procedure for the Bluetooth adapter, this method is called automatically when connecting to the server.

Returns
`true` if Bluetooth connectivity is available and enabled and the discovery was going on, `false` otherwise.

```
public static bool GetIsDiscovering()
```

Returns whether the local Bluetooth adapter is currently in process of device discovery.

Returns
`true` if Bluetooth connectivity is available and enabled and device discovery is currently going on, `false` otherwise.

```
public static bool GetIsDiscoverable()
```

Returns wheter the local Bluetooth adapter can be discovered by other devices.

Returns
`true` if Bluetooth connectivity is available and enabled and device is currently discoverable by other devices, `false` otherwise.

```
public static BluetoothDevice[] GetBondedDevices()
```

Returns `BluetoothDevice[]` of bonded (paired) devices. This method is available even without starting the discovery process.

Returns
`BluetoothDevice[]` if Bluetooth connectivity is available and enabled, `null` otherwise or on error.

```
public static BluetoothDevice[] GetNewDiscoveredDevices()
```

Returns `BluetoothDevice[]` of devices discovered during the last or current discovery session. This list is not cleared after the discovery ends.

Returns
`BluetoothDevice[]` if Bluetooth connectivity is available and enabled and device discovery is currently going on, `null` otherwise or on error.

```
public static BluetoothDevice[] GetDiscoveredDevices()
```

Returns `BluetoothDevice[]` of bonded (paired) devices *and* devices discovered during the ongoing discovery session. This list is not cleared after the discovery ends.

Returns

`BluetoothDevice[]` if Bluetooth connectivity is available and enabled and device discovery is currently going on, `null` otherwise or on error.

```
public static void SetVerboseLog(bool isEnabled)
```

Enables or disables verbose logging. Useful for testing and debugging.

Parameters

`isEnabled`	The new state of verbose logging.

```
public static bool SetRawPackets(bool isEnabled)
```

Enables or disables raw packets mode. Could only be called when no Bluetooth networking is going on. This option can be used if you want to exchange raw data with a generic Bluetooth device (like an Arduino with a Bluetooth Shield). *Use this only if you know what you are doing.*

Parameters

`isEnabled`	The new state of raw packets mode.

Returns

`true` if no Bluetooth networking is going on, `false` otherwise.

AndroidBluetoothMultiplayer events

```
// Fired when server is started and waiting for incoming connections
public static event Action ListeningStarted;

// Fired when listening for incoming connections
// was stopped by AndroidBluetoothMultiplayer.StopListening()
public static event Action ListeningCanceled;

// Fired when Bluetooth was enabled
public static event Action AdapterEnabled;

// Fired when request to enabled Bluetooth failed for some reason
// (user did not authorized to enable Bluetooth or an error occured)
public static event Action AdapterEnableFailed;

// Fired when Bluetooth was disabled
public static event Action AdapterDisabled;

// Fired when Bluetooth discoverability was enabled
// Provides discoverability period duration.
public static event Action<int> DiscoverabilityEnabled;

// Fired when request to enabled Bluetooth discoverability failed for some reason
// (user dismissed the request dialog or an error occured)
public static event Action DiscoverabilityEnableFailed;

// Fired when Bluetooth client successfully connected to the Bluetooth server
// Provides BluetoothDevice of the server device
public static event Action<BluetoothDevice> ConnectedToServer;

// Fired when Bluetooth client failed to connect to the Bluetooth server.
// Provides BluetoothDevice of the server device
public static event Action<BluetoothDevice> ConnectionToServerFailed;

// Fired when Bluetooth client disconnected from the Bluetooth server.
// Provides BluetoothDevice of the server device.
public static event Action<BluetoothDevice> DisconnectedFromServer;

// Fired on Bluetooth server when an incoming Bluetooth
// client connection was accepted.
// Provides BluetoothDevice of the connected client device
public static event Action<BluetoothDevice> ClientConnected;

// Fired on Bluetooth server when a Bluetooth client had disconnected.
// Provides BluetoothDevice of the disconnected client device
public static event Action<BluetoothDevice> ClientDisconnected;

// Fired when user selects a device in the device picker dialog.
// Provides BluetoothDevice of the picked device
public static event Action<BluetoothDevice> DevicePicked;

// Fired when Bluetooth discovery is started
public static event Action DiscoveryStarted;

// Fired when Bluetooth discovery is finished
public static event Action DiscoveryFinished;

// Fired when a new device was found during Bluetooth discovery procedure.
// Provides BluetoothDevice of the found device
public static event Action<BluetoothDevice> DeviceDiscovered;
```

Migrating from 1.x to 2.0

Android Bluetooth Multiplayer 1.x had two classes – `BluetoothMultiplayerAndroid` contained methods, and `BluetoothMultiplayerAndroidManager` containing events. *Android Bluetooth Multiplayer* 2.0 replaces them with a single `AndroidBluetoothMultiplayer` class that merges functionality of the two.
Some methods and events were renamed to make them cleaner and less confusing to a programmer, and to adhere to better code standards. Also, all code now resides in `LostPolygon.AndroidBluetoothMultiplayer` namespace.

Renamed methods:

Old name	*New name*
`Init`	`Initialize`
`InitializeServer`	`StartServer`
`Disconnect`	`Stop`
`StopListen`	`StopListening`
`RequestBluetoothEnable`	`RequestEnableBluetooth`
`BluetoothEnable`	`EnableBluetooth`
`BluetoothDisable`	`DisableBluetooth`
`RequestDiscoverable`	`RequestDiscoverability`
`CurrentMode`	`GetCurrentMode`
`IsBluetoothEnabled`	`GetIsBluetoothEnabled`
`IsBluetoothAvailable`	`GetIsBluetoothAvailable`
`CurrentDevice`	`GetCurrentDevice`
`IsDiscovering`	`GetIsDiscovering`
`IsDiscoverable`	`GetIsDiscoverable`

Names of the remaining methods are unchanged.

Renamed events:

Old name	*New name*
`onBluetoothListeningStartedEvent`	`ListeningStarted`
`onBluetoothListeningCanceledEvent`	`ListeningStopped`
`onBluetoothAdapterEnabledEvent`	`AdapterEnabled`
`onBluetoothAdapterEnableFailedEvent`	`AdapterEnableFailed`
`onBluetoothAdapterDisabledEvent`	`AdapterDisabled`
`onBluetoothDiscoverabilityEnabledEvent`	`DiscoverabilityEnabled`
`onBluetoothDiscoverabilityEnableFailedEvent`	`DiscoverabilityEnableFailed`
`onBluetoothConnectedToServerEvent`	`ConnectedToServer`

`onBluetoothConnectToServerFailedEvent`	`ConnectionToServerFailed`
`onBluetoothDisconnectedFromServerEvent`	`DisconnectedFromServer`
`onBluetoothClientConnectedEvent`	`ClientConnected`
`onBluetoothClientDisconnectedEvent`	`ClientDisconnected`
`onBluetoothDevicePickedEvent`	`DevicePicked`
`onBluetoothDiscoveryStartedEvent`	`DiscoveryStarted`
`onBluetoothDiscoveryFinishedEvent`	`DiscoveryFinished`
`onBluetoothDiscoveryDeviceFoundEvent`	`DeviceDiscovered`

Configuring AndroidManifest.xml and extending Activities

Plugin overrides built-in Unity activity and adds new permissions to AndroidManifest.xml in order to function.

Added permissions are:

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

Activity classes for this plugin, overriding the built-in Unity activities, are:

```
com.lostpolygon.unity.bluetoothmediator.player.BluetoothUnityPlayerProxyActivity
com.lostpolygon.unity.bluetoothmediator.player.BluetoothUnityPlayerActivity
com.lostpolygon.unity.bluetoothmediator.player.BluetoothUnityPlayerNativeActivity
```

You must override `BluetoothUnityPlayerNativeActivity` and `BluetoothUnityPlayerActivity` to implement your custom functionality (for example, adding some other plugin). Refer to

http://docs.unity3d.com/Documentation/Manual/PluginsForAndroid.html

for more details.

Activities source code can be found in “Assets/Plugins/Android/AndroidBluetoothMultiplayer_Integration/PlayerActivitiesSource.zip”. It is recommended to use Eclipse for building the code.

Integration with Vuforia

Plugin includes a custom Activity for easy integration with Vuforia.

1. Setup Vuforia and make sure that everything works.
2. Import Android Bluetooth Multiplayer package.
3. Extract “AndroidBluetoothMultiplayerVuforia.jar” file from “Assets/Plugins/Android/ AndroidBluetoothMultiplayer_Integration/VuforiaCompatible.zip” archive and place it into “Assets/Plugins/Android/” directory.
4. Open “Assets/Plugins/Android/AndroidManifest.xml” file. In that file, find “`com.qualcomm.QCARUnityPlayer.QCARPlayerNativeActivity`” and replace it with “`com.lostpolygon.unity.bluetoothmediator.player.vuforia.BluetoothUnityPlayerNativeActivity`”.
5. You should be able to use both Vuforia and Bluetooth Multiplayer for Android now.

Integration with Prime31’s Android Activity Sharing

Plugin includes a custom Activity for easy integration with Prime31’s plugins.

1. Setup Prime31’s plugins and make sure that everything works.
2. Import Android Bluetooth Multiplayer package.
3. Extract “AndroidBluetoothMultiplayerPrime31.jar” and “Prime31UnityActivity.jar” files from “Assets/Plugins/Android/ AndroidBluetoothMultiplayer_Integration/ Prime31ActivitySharing.zip” archive and place them into “Assets/Plugins/Android/” directory.
4. Open “Assets/Plugins/Android/AndroidManifest.xml” file.
5. In that file, find “`com.unity3d.player.UnityPlayerNativeActivity`” and replace it with “`com.prime31.UnityPlayerNativeActivity`”.
6. In the same file, add the following line to the <application> section

```
<meta-data
android:name="com.lostpolygon.unity.bluetoothmediator.player.prime31.BluetoothUnityPlayerPrime31Proxy"
android:value="UnityPlayerActivityProxy"/>
```

7. You should be able to use both Prime31’s plugins and Android Bluetooth Multiplayer now.

You can also reference to the Android Activity Sharing documentation: https://gist.github.com/prime31/0908e6100d7e228f1add

Contact

e-mail: contact@lostpolygon.com
Skype: serhij.yolkin

Changelog

2.0.0:

- Major code refactoring and standardization.
- Added `StartListening()` method.
- Improved demos code.
- Added basic RPC file transfer demo.
- Fixed incorrect handling of disconnected clients that sometimes lead to crashes.
- Implemented comparison methods for `BluetoothDevice`.
- Improved compatibility with Unity 5.

1.3.3:

- Improved Android 5 compatibility.

1.3.2:

- Added `onBluetoothDiscoverabilityEnabled` and `onBluetoothDiscoverabilityEnableFailed` events.

1.3.1:

- Fixed an issue when `GetDiscoveredDevices()` returned empty array before starting the discovery process.

1.3:

- New `IsDiscoverable()` method.
- Added a parameter to `ShowDeviceList()` for showing only data-capable devices.
- Added detection of the Bluetooth device class.
- Improved integration with other Android plugins.
- Improved demo scenes.
- Code clean-up.

1.2.3:

- Fixed an issue when client wasn’t disconnecting from server.

1.2:

- New `GetCurrentDevice()` method to get current Bluetooth device information.
- BluetoothMultiplayerAndroidManager is now instantiated automatically, no need to add prefab.
- New `SetRawPackets()` method to transmit data as raw (for advanced usage).

1.1:

- Add device discovery API (with demo usage example).
- Fix manifest generation on Mac OS X.
- Minor fixes.

1.0:

- Initial release.