Information

Simple Bluetooth for android the plugin allows you to quickly and easily connect two devices on Android OS. Using this plugin, you can create a multiplayer game or chat.

In this asset you will find two examples. The first example contains buttons for calling plug-in methods and an area for displaying events and transmitted messages. The second example is a simple multiplayer game.

Also, you can see the work of the plugin on the example of a real game - https://play.google.com/store/apps/details?id=com.Servalstar.HitThePlane (the game is not included in the asset).

The plugin itself consists of the BTPluginAndroid.aar file that you need to put in the "\Assets\Plugins\Android" folder in your project and the BluetoothForAndroid.cs script.

Work with plugin

This section describes the minimum steps that must be taken to work with the plugin.

The first thing you need to do is add the *SVSBluetooth* namespace in your script that will work with the plugin. To do this, you need to write at the top of your script:

```
using SVSBluetooth;
```

Next, you need to subscribe to events that report receiving a message from another device:

```
private void OnEnable () {
        BluetoothForAndroid .ReceivedByteMessage += GetMessage;
}
private void OnDisable () {
        BluetoothForAndroid .ReceivedByteMessage -= GetMessage;
}
```

The *GetMessage (message)* method is your method that will be called when a message is received and accept this message as a parameter.

If you need to subscribe to other events, do it the same way.

Next, you must initialize the plugin. Please note that you can only initialize the method on an Android device. To do this, simply call the method:

```
BluetoothForAndroid .Initialize();
```

Now you can use plugin methods.

To connect two devices, one of them must be assigned by the server and the other by the client. Separation on the server and the client is needed only at the time of connection. After connecting the device will be absolutely equivalent.

On the first device, call the server creation method:

```
BluetoothForAndroid.CreateServer( "8b10d196-4efd-4d83-a942-63fc970e591b" );
```

Here *"8b10d196-4efd-4d83-a942-63fc970e591b "* is the UUID of your application. For each application you need to use your own UUID. In this case, the UUID of the server and the client must be the same. You can generate a UUID in free online services, for example https://www.uuidgenerator.net

Now on the second device you need to call the method to connect to the server:

```
BluetoothForAndroid.ConnectToServer( "8b10d196-4efd-4d83-a942-63fc970e591b" );
```

After calling the method, a standard Android window will open in which you will need to select which device you want to connect to.

In case of successful connection, the *BluetoothForAndroid*.DeviceConnected event will be triggered, so it makes sense to subscribe to it.

After connecting, you can start forwarding messages between devices. To do this, simply call the method:

```
BluetoothForAndroid.WriteMessage(message);
```

As the *message* argument you can pass data such as Int32 , float , string or byte array .

For example, on the first device, you call the method:

```
BluetoothForAndroid.WriteMessage ("Hello");
```

On the second device will be called the event *BluetoothForAndroid.ReceivedByteMessage* and method *GetMessage(string val);* where *val* is the *"Hello"* message.

Thus, you can transfer text strings, game events, game object positions, health and so on between devices.

To disconnect, simply call the method:

```
BluetoothForAndroid.Disconnect();
```

This example shows only the most basic methods and events that are needed to use the plugin. For better development and use of your application, I advise you to use additional methods and events that are described below.

Getting a list of paired devices and connecting to the device at its address

Basically, to connect to the server, you will only need the *ConnectToServer (string UUID)* method, which opens the standard Android window where you select the server you want to connect to. But you can also connect to the server directly if you know its mac address. This may come in handy in the following cases:

- if you want to make your own, beautifully designed window with a list of devices;

- if you want to connect to the server to which you were already connected the previous time;

- if you know in advance the mac address of the device you need;

- sometimes, if the devices cannot connect the first time, you can implement an automatic reconnection attempt;

- very rarely, on some devices the standard window for connecting to the server may not open;

The *BluetoothForAndroid*.*GetBondedDevices()* method returns a list of paired devices as an array of *BTDevice* objects with the fields *.name* and *.address*. If there are no paired devices or the bluetooth adapter is turned off, “none” will be in the name and address fields.

The *BluetoothForAndroid*.*DeviceSelected(string device)* event is raised when you select a server in the device selection window. The string device parameter contains the name and mac address of the selected device, separated by a comma. You can separate the name and address using the *Split()* method.

You can take the address of the desired device and call *BluetoothForAndroid*.*ConnectToServerByAddress(string UUID, string address)* to connect to the desired device.

Thus, if you know the address in advance or get the address through the *GetBondedDevices()* method or through the *DeviceSelected()* event, you can connect to the server through the *ConnectToServerByAddress()* method without calling the standard device selection window.

Please note that when you call the ConnectToServerByAddress() method on the client device, you need to remember to call the *CreateServer()* method on the server device.

Examples of obtaining a list of devices, connecting to the first device in the list and connecting to the last selected device are in the UsageExample scene.

Plugin methods

public static void Initialize() - method initializes the plugin on an Android device. This method should always be called first.

public static bool IsBTEnabled() - the method checks whether the Bluetooth module is enabled on the device. Returns true if enabled and false if disabled.

public static void EnableBT() - method enables the Bluetooth module on the device.

public static void DisableBT() - the method turns off the Bluetooth module on the device.

public static void CreateServer(string UUID) - the method creates a server with the specified UUID, which is waiting for the client to connect.

public static void StopServer() - is the method that stops the server. If there is a connection with the client, this method is called automatically and you do not need to call it manually.

public static void ConnectToServer(string UUID) - the method calls the standard Android window in which you will need to choose which device you want to connect to. When attempting to connect, the specified UUID will be used.

public static void Disconnect() - disconnects the connected device.

public static void WriteMessage(val) - the method sends a message to the connected device. The method takes the following parameters val : Int32, float, string, byte[].

public static void ConnectToServerByAddress(string UUID, string address) – connection to the server directly, without calling the standard device selection window.

public static BTDevice[] GetBondedDevices() – the method returns a list of paired devices in the form of objects of type BTDevice[]. The BTDevice[] type has the fields .name and .address.

Plugin events

public static event Action BtAdapterEnabled - bluetooth adapter turned on.

public static event Action BtAdapterDisabled - bluetooth adapter turned off.

public static event Action ServerStarted - server is running.

public static event Action ServerStopped - server stopped.

public static event Action AttemptConnectToServer - attempt connect client to server. Called on the client device.

public static event Action FailConnectToServer - not succeeded connect to server.

public static event Action DeviceConnected - device connected.

public static event Action DeviceDisconnected - device disabled.

public static event Action <int> ReceivedIntMessage - received the message. On this event, you need to sign methods with parameters of type Int32.

public static event Action <float> ReceivedFloatMessage - received the message. Methods with parameters of type float must be signed to this event.

public static event Action <string> ReceivedStringMessage - received the message. Methods with parameters of type string must be signed to this event.

public static event Action <byte[]> ReceivedByteMessage - received the message. For this event, you need to sign methods with parameters of type byte[].

public static event Action<string> DeviceSelected – device selected. The event is triggered when on the client device, in a standard window, the server for connection is selected. The event contains a parameter of type string, which contains the name and mac address of the selected device, separated by a comma. Methods with parameters of type string must be signed to this event.