# SerialPort Utility Pro

Powerful serial port utility in Unity

for Android, Linux, Mac OS, and Windows

https://portutility.com/

Version 2.12

Released May 23, 2019

# Contents

# 1 Introduction

Powerful Serial port utility in Unity for Android, Linux, Mac OS, and Windows. This is extended plugin for Unity that gives developers an easy to use powerful computer communication solution.

When you use this plugin, you can spend time on important other works.

The connection of devices should become more straightforward.

Applications: M2M, IoT, PLC, Amusement Control, Hobby Device, Deep learning, FPGA UART.

## 1.1 Features

- Implement Computer-to- Computer communication in Unity easily via serial ports.
- Implement Computer-to-Microcontroller (Arduino, Ftdi, Microchip, Cypress, Silicon Labs, etc.) communication in Unity easily via serial ports.
- This plugin can specify the unique ID of a device and can run a device. The stable systems development is possible.
- Support cross-platform with Windows, Mac, and Android.
- This plugin does native processing asynchronously and is optimizing to Unity.
- An event-driven function for data reception. By this, the design of software becomes very easy.
- There is a system which puts a JSON type into the variable of a class directly.
- In this plugin, after carving a character string automatically, these are converted to List<string> or Dictionary<key, string>.
- The native code implements I/O. Late and heavy processing ".NET System.IO.Ports" is unnecessary.
- Error detection by physical disconnection of a device.
- USB, PCI, EmbeddedUART physical interface supported.
- Bluetooth SPP (Virtual COMPort) supported.
- TCP Serial Port Emulator (Server mode and Client mode) supported.

## 1.2 Free Version

This is the version for testing whether that suits your project, before purchasing the licensed version. The limitation of the FREE version should check "3.1 Free Version Limitations."

The Free(trial) version is downloadable from here.

**Download** ： https://portutility.com/

# 2 System Requirements

- Unity 5.5 and above, Unity 2017.x, Unity 2018.x (It can be used for both Personal and Pro.)
- Windows 7 and above (x86 and x64)
- Mac OS X 10.10 and above(64bit)
- Linux PC (x86 and x64)
- Android 4.1(Jelly Bean) and above
- SerialPort Device(USB, RS-232C, RS-485)
- Bluetooth Device
- Ethernet, Wi-Fi (TCP/IPv4)

# 3 Installation

## 3.1 Free Version Limitations

In the free trial version, there is limitation below.

- The data transfer (Send and receive) of devices can use it to a total of **1MB**.
- Support only Windows and MacOS.
- Please do not release to the public using the FREE version.

<< How to Import >>

1. Download "spup_v2.zip" from the site.

2. Please import the package which right-clicked Assets of the Unity editor, chose "Import-Package" -> "Custom Package ...", moreover, downloaded.
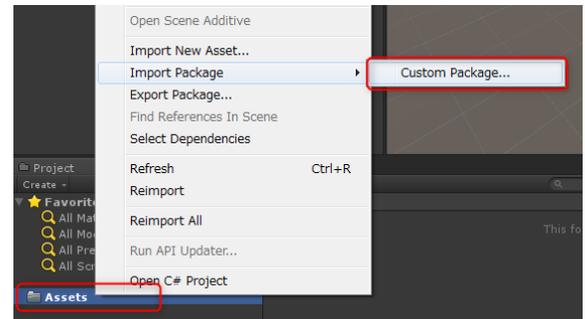
## 3.2 Upgrade to the licensed software.

The conversion to a release version can be purchased and upgraded from Unity Asset Store. Please apply a package from the following after product purchase.

<< The method of applying for the license >>

1. Access Unity Asset Store and search with "Serial Port Utility Pro."  **Direct URL:** http://u3d.as/1h5h

2. Please purchase in the directed Procedure. It is also possible to import downloaded "unitypackage" by overwriting.
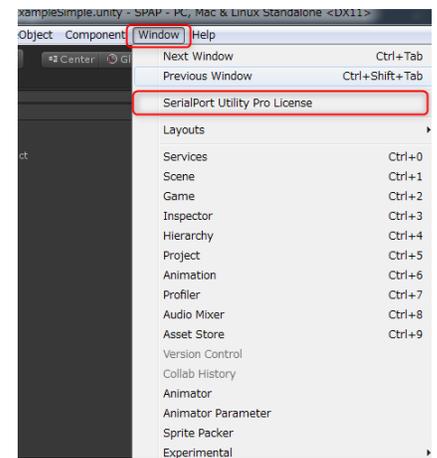
# 4 Using the Asset

## 4.1 Getting Started

1. Please check whether the plugin is imported at the project.

2. Choose Create Empty from GameObject of a menu bar, and add an object to a scene.
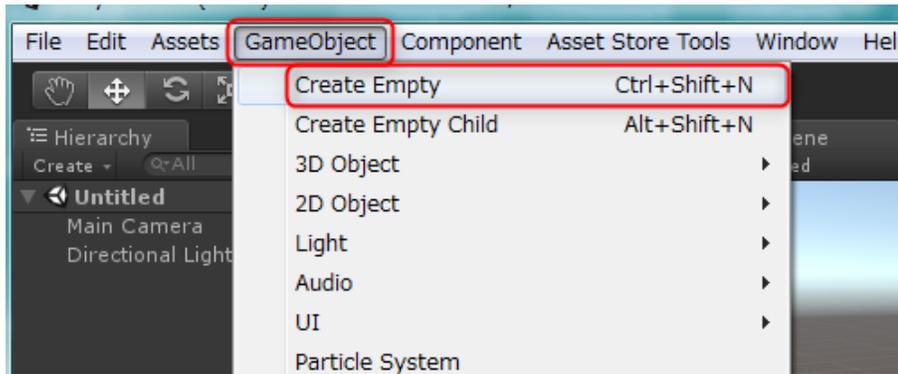


3. Choose the object added to the scene as the point, it chooses SerialPort Utility Pro from Component and adds a component to the object.
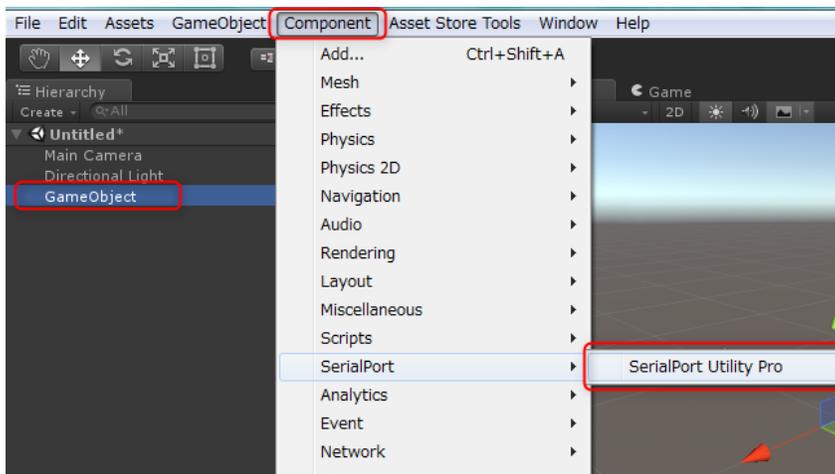


4. The component is added to the object, and use of it is attained.



**\* If you would like to check out the video, please visit HERE.**

## 4.2 Supported Device Types

- **USB**

  Communicate using converter cables, such as USB device, USB and RS-232C, and
  RS-485 which are mounted inboard boards such as Arduino and Microchip. Specify
  Vendor ID of USB, Product ID, and a serial number, and can access a specific port.

- **PCI**

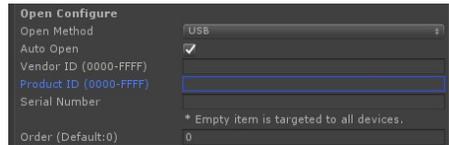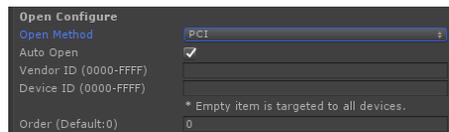  Communicate by specifying RS-232C and the PCI add-in board for RS-485 ports. Specify
  Device ID of a PCI add-in board, and Product ID, and can access a specific port.

  * Only Windows and Linux Supported.

- **Bluetooth SSP**

  Communicate using the Bluetooth SSP function mounted in the virtual serial port and
  Android of Bluetooth. Need to register the serial port of imagination into OS previously.
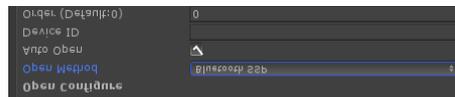
- **TCP Emulator Client & TCP Emulator Server**

  Communication can be emulated using TCP so that it may communicate with the serial
  port. Choose the Client or Server at the time of an activity.

  It is the function to display the list of the devices connected to PC under work and to input
  automatically by the UnityEditor. Click the following button to display a list and open a list
  window.

## 4.3 How to Configurations

If Component is added to a GameObject, the set list item of the script will be displayed.

Some list items can be dynamically modified by programming.

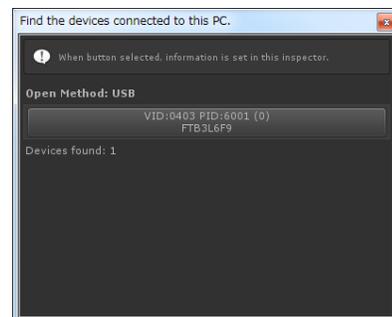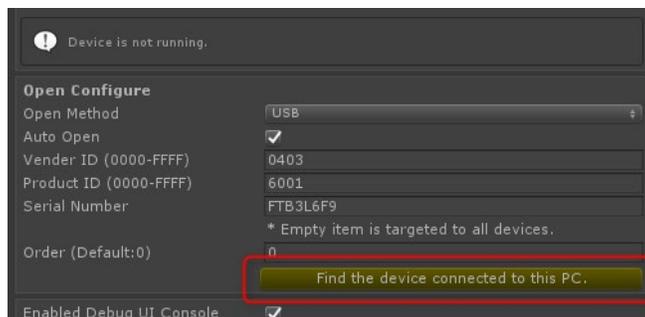| | |
|---|---|
| **SerialPort Status**<br><br>The port status of this script is displayed.<br><br>**Open Configure**<br><br>Input the methodology of opening a port, and required information. The list items which can be inputted by the open methodology differ. Moreover, the device connected to the PC can be found.<br><br>**Enabled Debug UI Console**<br><br>Decide whether to validate UI console for a debugging. <u>Once it uses Enable, it cannot Disable from the script.</u><br><br>**Enabled External Config**<br><br>It can substitute for the variable of a class before open using an external file. Please confirm "4.8 Using the External File Config" for details.<br><br>**Enabled Transmission**<br><br>Whether to enabled send and receive.<br><br>**Communication Structure**<br><br>Serial communication architecture of USB or PCI.<br><br>**Read Data Structure**<br><br>Input the methodology of receiving serial communication, and the required information about a set of the processing.<br><br>**Write Data Command**<br><br>The script set of functions for sending serial communication is displayed. This list item does not have an input. |  |
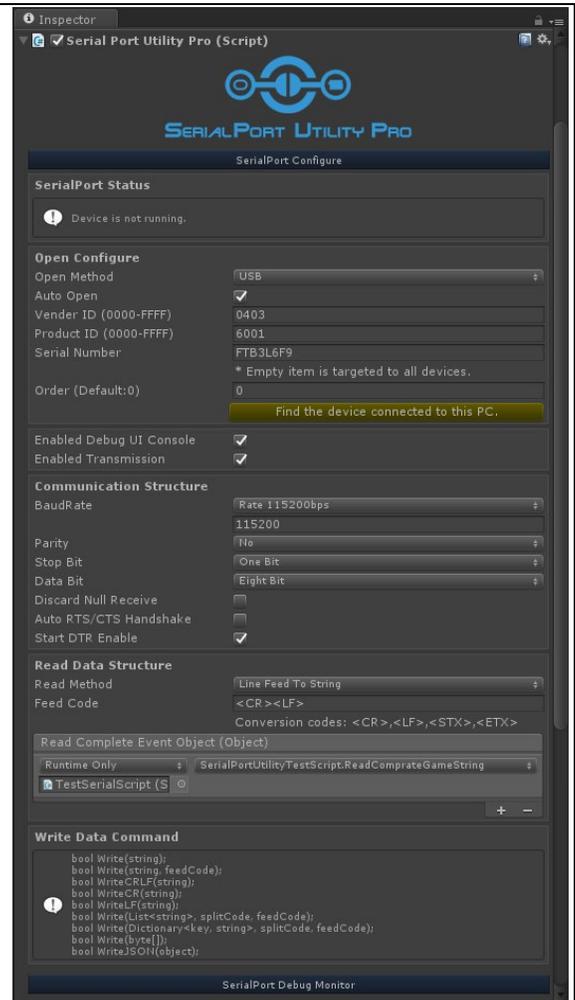
## 4.4 Receiving Data

- After being processed by the receiving methodology which specified the data received in the

port as MethodSystem ReadProtocol, the function specified by SPUPEventObject ReadCompleteEventObject is called. Timing is certainly called for the beginning of each frame (MonoBehaviour.Update function), and an asynchronous.    The argument of a function certainly requires a translation (as cast) within a function, because the object passes it.

● **ReadProtocol: Streaming or BinaryStreaming**

Receive the received data by a streaming, without special processing. A continuation data is collected temporarily and passed to a receiving function as fixed granularity.

| | |
|---|---|
| 0000ms -> AAA,BBB,CCC,DDD<CR><LF><br>0010ms -> EEE,FFF,GGG,HHH<CR><LF><br>1000ms -> III,JJJ,KKK,LLL<CR><LF> | var DAT = data as string; or or byte[];<br>AAA,BBB,CCC,DDD<br><CR><LF>EEE,FFF,GGG,HHH <CR><LF><br>III,JJJ,KKK,LLL <CR><LF> |

● **ReadProtocol: LineFeedDataToString or LineFeedDataToBinary**

The receiving function is called considering the specified Feed cord as one delimiter.

| | |
|---|---|
| 0000ms -> AAA,BBB,CCC,DDD<CR><LF><br>0010ms -> EEE,FFF,GGG,HHH<CR><LF><br>1000ms -> III,JJJ,KKK,LLL<CR><LF> | var DAT = data as string or byte[];<br>AAA,BBB,CCC,DDD<br>EEE,FFF,GGG,HHH<br>III,JJJ,KKK,LLL |

● **ReadProtocol: FixedLenghDataToString or ToBinaryData**

The receiving function is called for every number of the settlements which the continuous data specified.

| | |
|---|---|
| 0000ms -> AAA,BBB,CCC,DDD<CR><LF><br>0010ms -> EEE,FFF,GGG,HHH<CR><LF><br>1000ms -> III,JJJ,KKK,LLL<CR><LF> | var DAT = data as string; or byte[];<br>AAA,BBB,CCC,<br>DDD<CR><LF>EEE,FFF<br>,GGG,HHH<CR><LF>II<br>I,JJJ,KKK,LL<br>L<CR><LF> |

● **ReadProtocol: SplitStringToArray**

The receiving function is called considering the specified Feed cord as one delimiter. And, it is changed into a List array (Array) by the data divided for every specified Split cord. Use "=" for making it relate.

| | |
|---|---|
| 0000ms -> AAA=BBB,CCC=DDD<CR><LF><br>0010ms -> EEE=FFF,GGG=HHH<CR><LF><br>1000ms -> III=JJJ,KKK=LLL<CR><LF> | var DAT = data as Dictionary<string, string>;<br>DAT["AAA"] -> BBB, DAT["CCC"] -> DDD<br>DAT["EEE"] -> FFF, DAT["GGG"] -> HHH<br>DAT["III"] -> JJJ, DAT["KKK"] -> LLL |

- **ReadProtocol: SplitStringToDictionary**

  The receiving function is called considering the specified Feed cord as one delimiter. And, it is changed into a lexicon array (Dictionary) by the data divided for every specified Split cord. Use "=" for making it relate.

  | | |
  |---|---|
  | 0000ms -> AAA=BBB,CCC=DDD<CR><LF><br>0010ms -> EEE=FFF,GGG=HHH<CR><LF><br>1000ms -> III=JJJ,KKK=LLL<CR><LF> | var DAT = data as Dictionary<string, string>;<br>DAT["AAA"] -> BBB, DAT["CCC"] -> DDD<br>DAT["EEE"] -> FFF, DAT["GGG"] -> HHH<br>DAT["III"] -> JJJ, DAT["KKK"] -> LLL |

- **ReadProtocol: SplitStringToGameObject**

  The receiving function is called considering the specified Feed cord as one delimiter. And, it is changed into a lexicon array (Dictionary) by the data divided for every specified Split cord. Use "=" for making it relate.

  | | |
  |---|---|
  | 0000ms -> AAA=BBB,CCC=DDD<CR><LF><br>0010ms -> EEE=FFF,GGG=HHH<CR><LF><br>1000ms -> III=JJJ,KKK=LLL<CR><LF><br>* Specify a game object called "GameObj". | It is stored in the variable number of a GameObject.<br>GameObj.AAA = "BBB", GameObj.CCC = "DDD"<br>GameObj.EEE = "FFF", GameObj.GGG = "HHH"<br>GameObj.III = "JJJ", GameObj.KKK = "LLL" |

- **ReadProtocol: JSONToClassObject**

  Change the text of a JSON format into a class object.

  | | |
  |---|---|
  | 0000ms -> {"Name": "JSON", "AGE": 33} | var DAT = data as OriginalClass;<br>DAT.Name<br>DAT.AGE |

- **ReadProtocol: ModbusASCII**

  Process ASCII string corresponding to Modbus format.

  | | |
  |---|---|
  | 0000ms -> :0105040BFF00EC<CR><LF> | var DAT = data as SPUPMudbusData;<br>DAT.Address<br>DAT.Function<br>DAT.Data |

## 4.5 Sending Data

Write function of a SerialPortUtilityPro customer local area signaling service can do a data to send from a port. Moreover, the port needs to be previously opened normally by the Open function.　It responds to the argument for the Split cord and the Feed cord at the special control code<CR>, <LF>, <STX>, and <ETX>.

```
SerialPortUtility.SerialPortUtilityPro port;
// Get port by Component.
```

```
port = this.GetComponent<SerialPortUtility.SerialPortUtilityPro>();
//Write
port.Write("Hello World!");
port.WriteCRLF("Hello!"); // The equal in case of Write("Hello!¥r¥n");
port.WriteCR("Hello!"); // The equal in case of Write("Hello!¥r");
port.WriteLF("Hello!"); // The equal in case of Write("Hello!¥n");
port.Write("Hello!","<CR><LF>"); // Add the Feed cord to the last of a text.
```

If it sends not a text but a byte string, it can send with the same Write function.
The data which is not expressed as letters, such as NULL, is also dealt with, and can do the place different from String.

```
byte[] dataByte = new byte[256]; //256byte
dataByte[0] = 0x00;
dataByte[1] = 0x20;
// Process a data if needed.
port.Write(dataByte);
```

Using the Split cord and the Feed cord which were specified, a List customer local area signaling service and a Dictionary customer local area signaling service can also do the auto-creation of the transmit string and can be sent.

```
List<string> dataArray = new List<string>();
dataArray.Add("AAA");
dataArray.Add("BBB");
dataArray.Add("CCC");
port.Write(dataArray, ",","<CR><LF>");
//It is sent as AAA,BBB,CCC<CR><LF>
```

```
Dictionary<string, string> dataArray = new Dictionary<string, string>();
dataArray.Add("AAA", "BBB");
dataArray.Add("CCC", "DDD");
port.Write(dataArray, ",", "<CR><LF>");
//It is sent as AAA,BBB,CCC,DDD<CR><LF>
```

It is also possible to do the auto-creation of the customer local area signaling service with substance to the text of a JSON format and to send.

```
public class Item {
    public float Right;
    public float Left;
}
```

```
void send() {

    SPAPItem itobj = new SPAPItem();

    itobj.Right = 0.0f;

    itobj.Left = 0.0f;

    port.WriteJSON(itobj);

}
```

Because a class object can be passed, a very smooth data interchange becomes possible.

Moreover, it is also reflected in Debug UI Console about the content which sent.

## 4.6 About control functions

Except for Bluetooth SPP and TCP/IP, the function for the line controls of the serial port is prepared for OpenMethod. Moreover, controllable lines are data signaling rate, CTS, DTR, and DTS, and refer to the set of functions of the list item "6. Scripting Reference" for them.

To control these statuses, the port needs to be previously opened normally by the Open function. There is a facility "Auto RTS/CTS Handshake" automatically changed according to the status of the application of Unity. This is a facility in which Unity application detects an operation by foreground and changes a set of RTS/CTS automatically.
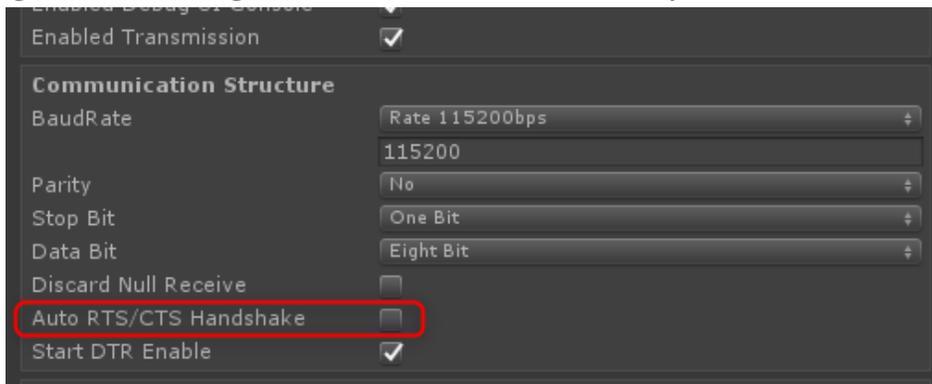


## 4.7 Using the Debug UI Console
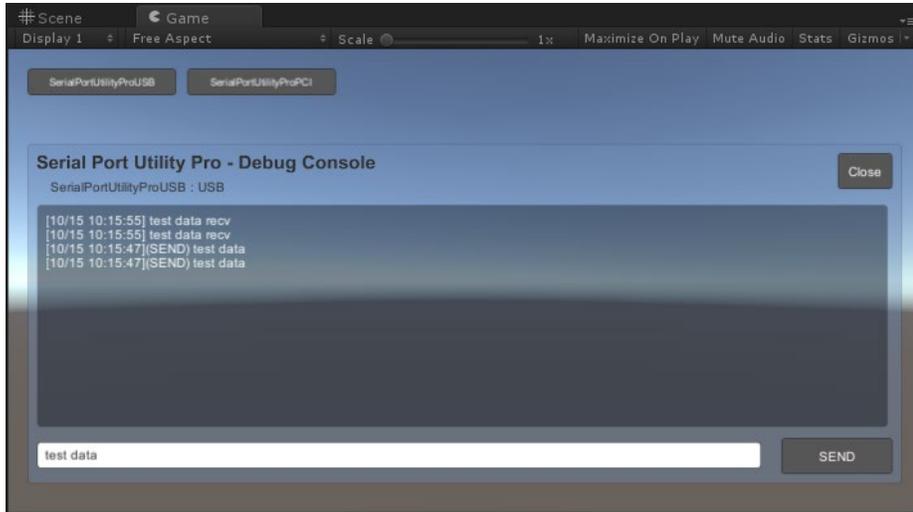
The UI console for debugging is implemented in Serial Port Utility Pro. The data on serial communication can be visualized immediately. Moreover, if all the components with which the check is included in the debugging are displayed as a button and click a button, it can be used as a serial monitor.

To validate a debugging console, put a check into "Enabled Debug UI Console."
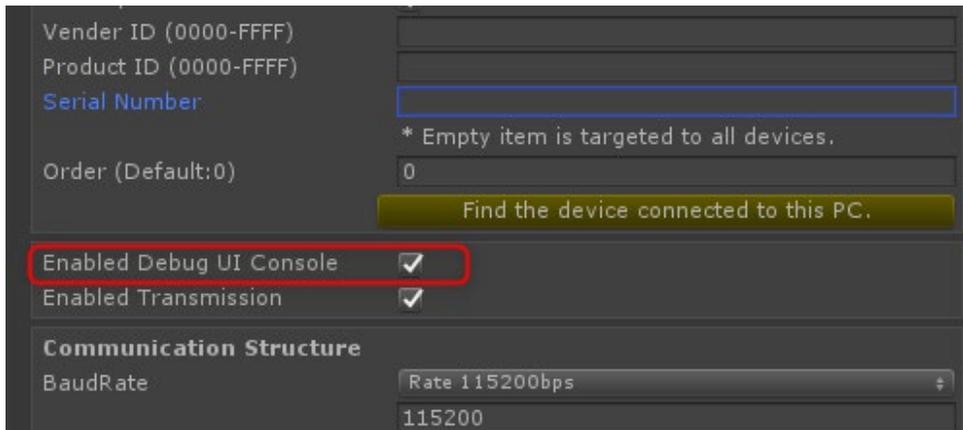


When you build, please set this wave mode to false.

## 4.8 Using the External File Config

The set varying by an external file implements to the executable file after an output. When it is necessary to change a setting to the built executable file, if the check is put in, the member variable of a class will be changed before open of a device from an external configuration file. By this, a set is easily changeable according to an environment also after an EXE output.
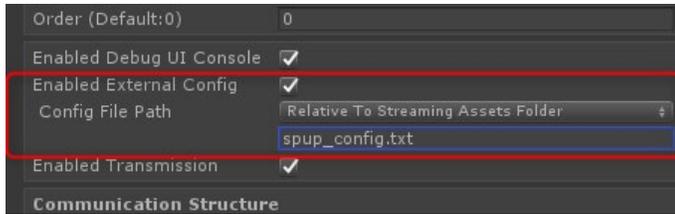
Substitute the Public variable name in a [Serial Port Utility Pro] class using "=." If there are more than one, start a new line and specify the next variable. For example, make a file called spup_config.txt and put in the following data.

| | |
|---|---|
| VendorID=1234 | # Comment |
| BaudRate=115200 | |
| Parity=0 | # 0=No |
| StartEnableDTR=1 | # 1=true 0=false |

If an unexpected data inputs, an executable file may crash. Please look over the input data again.

## 4.9 Utility Events

The Utility Event is an event handler called by Invoke, when class the open and close, and also when an error happens. The class called SerialPortUtilityPro and the text of the requested category return to the parameter of a function.
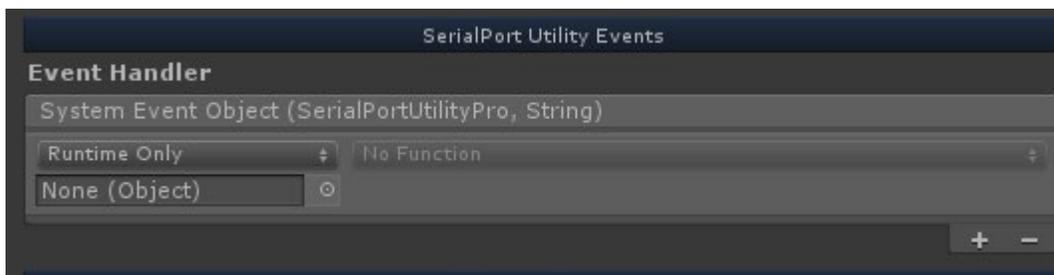


Because it has closed in the group in usual, please click and open the "SerialPort Utility Events" tab.

# 5 Quick Start Examples

## 5.1 Quick Start Programming

Explain using the scene of ExampleScenes of Asset. To check a scene, please open the scene in an ExampleScenses folder.

- **ExampleSimple or ExampleSimpleBinary**

  It is a fundamental example of a sample when spending Serial Port Utility Pro. First, make the function for the following reception for SPUPTestScriptSimple.cs to reference. If you

would like to receive as a data used as a list form,

```
public void ReadComprateList(object data) {

   var text = data as List<string>;

   for(int i=0; i < text.Count; ++i)

      Debug.Log(text[i]);

}
```

If you would like to receive as a lexicon formal data and to process,

```
public void ReadComprateDictionary(object data) {

   var text = data as Dictionary<string, string>;

   Debug.Log(text["test"]);

}
```
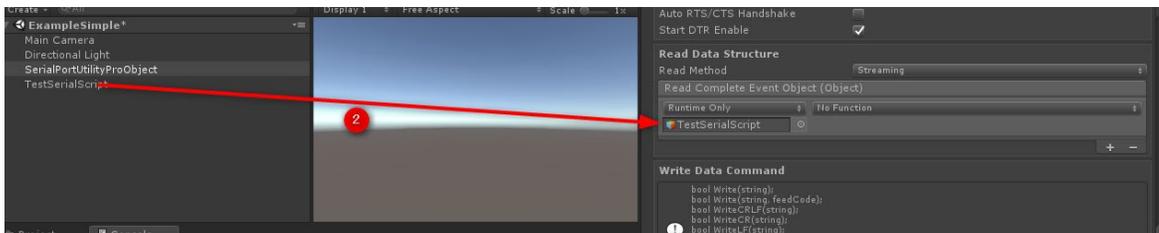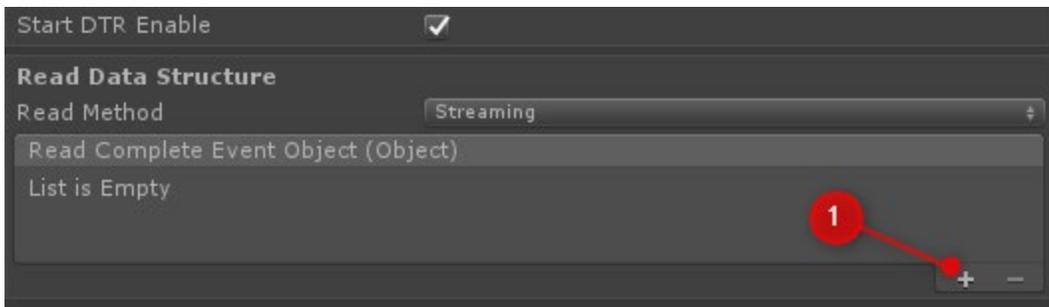
If you would like to receive as a fixed text simply and to process,

```
public void ReadComprateString(object data) {

   var text = data as string;

   Debug.Log(text);

}
```

Thus, make a function.

Next, set the function for receiving as follows to "Read Data Structure."
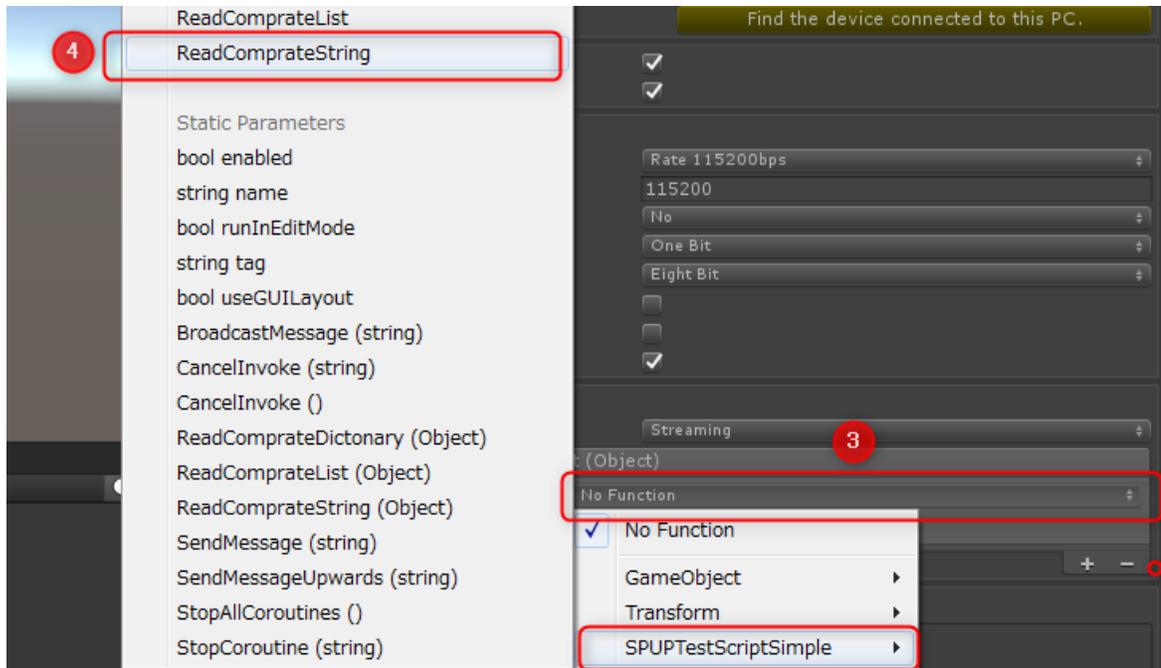
This time, ReadCompreateString is set. Here, please be sure to set a Dynamic Object type function.

Moreover, for sending a data, get the opened SerialPortUtilityPro customer local area signaling service, and use "Write" which is the member function.

```
SerialPortUtility.SerialPortUtilityPro serialPort = XXX;
serialPort.WriteCRLF("TestData");
```

- **ExampleGameObject**

  It is an example of a sample which specifies GameObject which exists in the hierarchy of a scene and receives and does substitution ト of the data to the script. It is applied to all the customer local area signaling service and member variables by which the association is done to GameObject.

```
public class SPUPTestScriptGObj: MonoBehaviour {
    //Item
    public string Item1;
    public string Item2;
    public string Item3;
}
```

In the above class,

```
SPUPTestScriptGObj.Item1,TESTDATA, SPUPTestScriptGObj.Item2,TESTDATA2 <CR><LF>
```

If the data to say is received, TESTDATA and TESTDATA2 substitutes for each variable of SPUPTestScriptGObj.Item1 and SPUPTestScriptGObj.Item2 which is associated and is set

to specified GameObject.

- **ExampleJSON**

  It is an example of a sample which can communicate a data in a JSON format. It is necessary to do in the clear the cast of the object type changed from the JSON format.
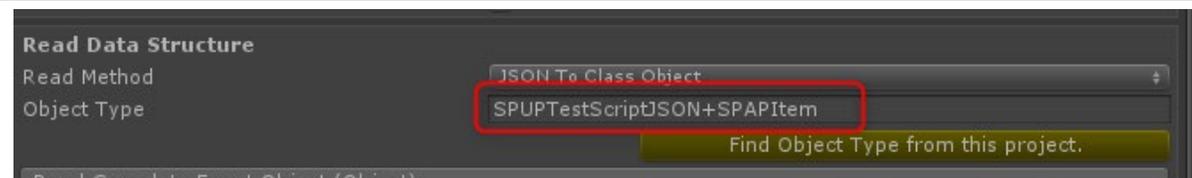
```
public void ReadComprateJSON(object data) {

    var item = data as SPAPItem;

}
```

  When sending, it can change and send to a JSON format using "WriteJSON."

```
SerialPortUtility.SerialPortUtilityPro serialPort = XXX;

serialPort.WriteJSON(itobj);
```



  Setting Object Type in clear when setting "Read Data Structure." A listing of the class which exists in a project open from "Show Object Type from this project" can be displayed and chosen.

- **ExampleProgramming**

  It is an example of a sample of the methodology of completing within a program. Because "SPUP_Programming.cs" generates, at the time of a shutdown, a SerialPortUtilityPro object does not exist on a scene.

```
serialPort = this.gameObject.AddComponent<SerialPortUtility.SerialPortUtilityPro>();

//config

serialPort.OpenMethod = SerialPortUtility.SerialPortUtilityPro.OpenSystem.USB;

serialPort.VendorID = "";

serialPort.ProductID = "";

serialPort.SerialNumber = "";

serialPort.BaudRate = 115200; //115200kbps

serialPort.ReadProtocol = SerialPortUtility.SerialPortUtilityPro.MethodSystem.Streaming;

serialPort.ReadCompleteEventObject.AddListener(this.ReadComprateString);        //read function

serialPort.RecvDiscardNull = true;
```

  Here, a thing to be warned is the generation methodology of an object.

```
serialPort = new SerialPortUtility.SerialPortUtilityPro();   // Impossible
```

  As the generation of SerialPortUtilityPro object cannot be done.

```
serialPort = this.gameObject.AddComponent<SerialPortUtility.SerialPortUtilityPro>();
```

It needs to be managed by GameObject as mentioned above. It is because a parent class has MonoBehaviour. Furthermore, it is necessary to start a communication manually after a set of sundry items using an Open function, to call a Close function at the time of a quit, and to end an object in the clear.

## 5.2 Troubleshooting

● **It becomes a loading error of spap.dll at the time of Windows usage.**

Install runtime -> https://go.microsoft.com/fwlink/?LinkId=746572

● **It becomes a loading error of libspap.so at the time of Linux usage.**

Libspap.so is using libudev. It is necessary to install udev separately.

<Solution>

1. Install udev.

```
$ sudo apt-get install udev
```

● **It becomes an open error although the serial port exists at the time of LinuxOS usage.**

Because setting the privilege of devices as a SuperUser, it may be unable to be used unless it assigns privilege to a usage Users.

<Solution>

1. Start application as a super user. OR,

```
$ sudo ./unity_application
```

2. Modify a privilege into devices by batch at OS during starting. OR,

```
$ find /dev ¥( -name 'ttyS*' -or -name 'ttyACM*' -or -name 'ttyUSB*' ¥) | xargs sudo chmod 666
```

3. Add your USER to the "dialout" group.

```
$ sudo adduser USERNAME dialout
$ reboot
```

Once it adds, it will become unnecessary from the 2nd times.

## 5.3 Programming with MCU

● Please watch tutorial videos. -> https://portutility.com/tutorial/

# 6 Scripting Reference

namespace SerialPortUtility;

**class** SerialPortUtility. SerialPortUtilityPro (MonoBehaviour)

<span style="color:red">&lt;Method&gt;</span>

### void Open()

Description: That tries opening for the device by it setting item.

Parameter: none

Return: none

### void Close()

Description: That tries closing for the device.

Parameter: none

Return: none

### bool IsOpened()

Description: That checks whether the device opens or not.

Parameter: none

Return: If a return value is true, that has already opened.

### bool IsErrorFinished()

Description: Although the device opened, true is set if it has ended by a certain cause of an error.

Parameter: none

Return: If a return value is true, that has already opened.

### bool Write(string)

Description: Write (send) to the opened serial port.

Parameter: Transmit string.

Return: If the return value is true, write succeeded.

### bool Write(byte[] or byte)

Description: Writes (transmits) byte data to the opened serial port.

Parameter: Transmit byte array. Or transmit byte.

Return: If the return value is true, write succeeded.

### bool Write(string, string)

Description: Write (transmit) with the feed code added to the opened serial port.

The Feed code corresponds to control codes <CR>,<LF>,<STX>,<ETX>.

Parameter: 1. Transmit string, 2. Feed character code appended to the end.

Return: If the return value is true, write succeeded.

## bool Write(System.Collections.Generic.List<string>, string, string)

Description: Write (send) the Listed data to the opened serial port using Split code and Feed code. The Feed code corresponds to control codes <CR>,<LF>,<STX>,<ETX>.

Parameter: 1. List class, 2. Split code, 3. Feed code.

Return: If the return value is true, write succeeded.

## bool Write(System.Collections.Generic.Dictionary<string,string>, string, string)

Description: Write (send) the Dictionary data to the opened serial port using Split code and Feed code. The Feed code corresponds to control codes <CR>,<LF>,<STX>,<ETX>.

Parameter: 1. Dictionary class, 2. Split code, 3. Feed code.

Return: If the return value is true, write succeeded.

## bool Write(UnityEngine.GameObject, string, string)

Description: We write (send) the member variable of the script added to GameObject to Split and Feed for the opened serial port. The Feed code corresponds to control codes <CR>,<LF>,<STX>,<ETX>.

Parameter: 1. GameObject in the scene, 2. Split code, 3. Feed code.

Return: If the return value is true, write succeeded.

## bool Write(SPUPMudbusData mudbus_data, bool binaryMode)

Description: Send data in Mudbus data format. The check digit is also added automatically.

Parameter: 1. Mudbus Data Class, 2. Ascii mode or RTU mode.

Return: If the return value is true, write succeeded.

## bool WriteCRLF(string)
## bool WriteCR(string)
## bool WriteLF(string)

Description: Writing (send) is performed by adding CR or LF code to the opened serial port.

Parameter: Transmit string

Return: If the return value is true, write succeeded.

## bool WriteJSON(object)

Description: Writes (sends) the argument data to the opened serial port in JSON format.

Parameter: Transmitted object, converted to JSON format.

Return: If the return value is true, write succeeded.

## bool CtsHolding()

Description: Gets the state of the CTS of the opened serial port.

Parameter: none

Return: The state of the CTS returns with HI or LOW.

## bool DsrHolding()

Description: Gets the state of the DSR of the opened serial port.

Parameter: none

Return: The state of the DSR returns with HI or LOW.

## bool DtrEnable(bool)

Description: Change the state of DTR on the opened serial port.

Parameter: Output is HI when true is set.

Return: If the return value is true, change succeeded.

## bool DtrGetStatus()

Description: Gets the state of the DTR of the opened serial port.

Parameter: none

Return: The state of the DTR returns with HI or LOW.

## bool RtsEnable(bool)

Description: Change the state of RTS on the opened serial port.

Parameter: Output is HI when true set.

Return: If the return value is true, change succeeded.

## bool RtsGetStatus()

Description: Gets the state of the RTS of the opened serial port.

Parameter: none

Return: The state of the RTS returns with HI or LOW.

## bool IsConnected()

Description: It is the same action as the DsrHolding(). Use this by splicing confirm of the CLIENT of "TCPEmulator Server.".

Parameter: none

Return: The state of the DSR returns with HI or LOW.

## bool SetBreakSignal(bool)

Description: Turn on / off the break signal to the opened serial port.

Parameter: If it is true, a break signal is in progress, false is set to break signal.

Return: It returns true if the break signal change succeeded.

## bool SerialDebugAddString(string)

Description: Write the character string specified for display in the debugging window. It is not sent to the serial port.

Parameter: Transmit string.

Return: If the return value is true, write succeeded.

## void ReadUpdate()

Description: Update the receiving system. It is usually called by Update() of Unity, However, in Manual mode it is necessary to update it manually using this function.

Parameter: none

Return: none

## <Property>

| bool IsAutoOpen = true; | If True, when the script becomes significance (OnEnable), it is opened automatically. The default is true. |
|---|---|
| OpenSystem OpenMethod = OpenSystem.USB; | Choose the category of device to open. It can choose from the OpenSystem citation. |
| MethodSystem ReadProtocol = MethodSystem.SplitStringToArray; | Choose the methodology of the processing at the time of the reception. It can choose from the "MethodSystem" citation. |
| string VendorID | VendorID of the device for open. |

| | |
|---|---|
| string ProductID | ProductID of the device for open. |
| string SerialNumber | The serial number of the device for open. It is effective only when setting the OpenMethod variable as a USB. |
| string IPAddress | IP address to open. |
| int Port | Only the time of OpenMethod being TCP Serial Emulator Client is effective. It is the port number of the internet to open Only the time of OpenMethod(s) being Transmission Control Protocol Serial Emulator Client and Server are effective. |
| string DeviceName | It is the device name of Bluetooth SSP to open. It is effective only at the time of "OpenMethod." |
| int BaudRate = 9600; | Set of the baud rate. |
| ParityEnum Parity = ParityEnum.No; | Set of the parity check. |
| StopBitEnum StopBit = StopBitEnum.OneBit; | Set of the stop bit. |
| DataBitEnum DataBit = DataBitEnum.EightBit; | Set of the data bit. |
| bool RecvDiscardNull= false; | If True, cancel before receiving the NULL letter. |
| bool AutoRTSCTSHandshake = false; | If True, control RTSCTS by Unity. |
| bool StartEnableDTR = true; | If True, set DTR to one at the time of the start. |
| bool DtrEnabled = false; | If True, set DTR to HI(1). |
| bool RtsEnabled = false; | If True, set RTS to HI(1). |
| int Skip = 0; | The value for distinguishing if the same device is connected. |
| string FeedCode = "<CR><LF>" | Set of the feedcode. |
| string SplitCode = "," | Set of the splitcode. |
| int FixedFeedCount = 10 | Valid only when ReadProtocol is set with FixedCharactersTo*. When it reaches the division number set by FixedSplitCount, it sends. |
| UpdateMethod UpdateProcessing = Update | Set the timing to call the ReadUpdate function. |
| bool DiscardSpaceTabChar = false | If True, cancel before receiving Space and a Tab character. |
| SPUPEventObject ReadCompleteEventObject = new SPUPEventObject() | Specify the function called at the time of the reception complete of the processed data. |
| string ReadCompleteEventObjectType = "" | ReadCompleteEventObject parameter. |
| GameObject ReadClassMembersObject = null | It is effective only if setting as |

| | "SplitStringToGameObject" to the ReadProtocol variable. It is directly reflected in the member variable of the component class of set GameObject. |
|---|---|
| string GetSerialDebugString | Get the text for the current debugging view. This variable cannot be edited. |

enum SerialPortUtility.SerialPortUtilityPro.MethodSystem

| | |
|---|---|
| FixedLengthDataToString | Process a fixed text length as one text to a delimiter. |
| FixedLengthDataToBinaryData | Process a fixed data length as one data to a delimiter. |
| JSONString | Change a JSON text as an object. |
| LineFeedDataToString | Read the text for every Feed letter. |
| LineFeedDataToBinary | Read the data for every Feed letter. |
| SplitStringToArray | Process as an array the text divided in the Split letter for every Feed letter. |
| SplitStringToDictionary | Process the text divided in the Split letter for every Feed letter as a lexicon object. |
| SplitStringToGameObject | Put directly the text divided in the Split letter for every Feed letter into the member variable of GameObject, and process it. |
| Streaming | Don't process the text received at a fixed spacing, but receive as it is. |
| BinaryStreaming | Don't process the data received at a fixed spacing, but receive as it is. |

enum SerialPortUtility.SerialPortUtilityPro.OpenSystem

| | |
|---|---|
| Bluetooth SSP | Communicate using bluetooth SSP. |
| Number Order | Try all the devices connected in order of universal serial bus->protocol-control-information->BluetoothSSP. |
| PCI | Communicate using a protocol-control-information interface. |
| TCP SerialEmulator Client | Emulate serial communication using TCP/IP. |
| TCP SerialEmulator Sever | Emulate serial communication in server mode using TCP/IP. |
| USB | Communicate using a universal serial bus interface. |

enum SerialPortUtility.SerialPortUtilityPro.DataBitEnum

| | |
|---|---|
| EightBit | The Data bit is 8 bit |
| SevenBit | The data bit is 7 bit |

enum SerialPortUtility.SerialPortUtilityPro.ParityEnum

| | |
|---|---|
| Even | Do a parity check as an even. |
| Mark | A parity check is always set to 1. |
| No | A parity check does not do. |
| Odd | Do a parity check as an odd. |

| Space | A parity check is always set to 0. |
| --- | --- |

**enum** SerialPortUtility.SerialPortUtilityPro.StopBitEnum

| OneBit | Stop bit is1 bit |
| --- | --- |
| TwoBit | Stop bit is2 bit |

**class** SerialPortUtility.DebugConsole (MonoBehaviour)

It is a script for implementing an internal debugging console.

You cannot touch the code.

# 7 Revision History

- **v2.12 – May 23, 2019**

  1. Add break-signal function on Android OS.

  2. Improving write system performance.

- **v2.11 - May 1, 2019**

  1. Fixed bugs of Andriod Plugin

  2. Supported Modbus protocol (ASCII mode)

- **v2.10-April 9, 2019**

  1. Fixed bugs of Andriod Plugin and Linux Plugin

  2. Improving read performance

- **v2.09- March 11, 2019**

  1. Fixed Bugs of Android Plugin

- **v2.07 - February 15, 2019**

  1. Renamed from ReadMethod to ReadProtocol.

  2. Fixed bug that 0x3F or over cannot be received.

  3. Fixed bugs in Linux build.

  4. Added binary data send/receive function.

  5. Added function to set update timing.

- **v2.06 - February 4, 2019**

  1. Fixed Bugs in Bluetooth.

  2. Stabilize "TCP Serial Emulator" and add an example.

- **v2.05 - February 1, 2019**

  1. Fixed Bugs of Android OS.

- **v2.04 - January 25, 2019**

  1. Fixed Bugs

  2. Support for high-speed data loading.

  3. Changed the Enum name of ReadProtocol.

- **v2.03 - January 11, 2019**

  1. Fixed Bugs

- **v2.02 - December 29, 2018**

  1. Compatible with Linux OS

- **v2.01 - December 19, 2018**

  1. Inspector UI was changed.

  2. Add the update of config by an external file.

3. Add the IsConnect() function which checks whether it is connected or not.

4. Add the event handle for the device status.

- **v2.0 – November 8, 2018**

  1. Fixed Bugs

  2. First released in our webpage.

- **v1.0 – July 7, 2018**

  1. First released in private.

# 8 License

If source code is required, we can indicate based on a license.

- [Android JAVA] mik3y - usb serial for android

  https://github.com/mik3y/usb-serial-for-android/blob/master/LICENSE.txt

- [Android JAVA] MacroYau - Blue2Serial

  https://github.com/MacroYau/Blue2Serial/blob/master/LICENSE.md


# 9 Support & About Wizapply

Wizapply Co., Ltd.

**OFFICE**

Address: KS Bld.5F, 3-7-10, Ichiokamotomachi, Minato-ku Osaka-shi, Osaka, 552-0002, Japan

TEL: +81.644006308

E-MAIL: info@wizapply.com

- Please let us know if you have any questions.