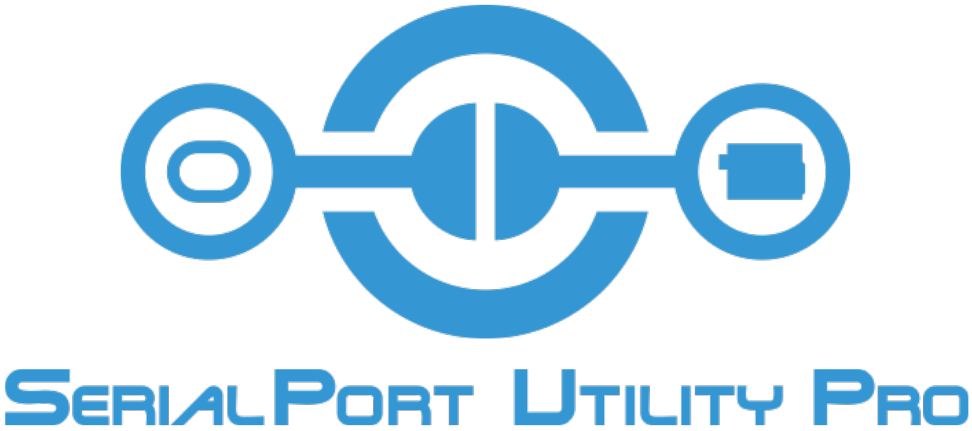


Unity3D 拡張プラグイン

パワフルなシリアル通信ユーティリティ

Android, Linux, Mac OS, Windows 向け

https://portutility.com/

バージョン　2.14

2019 年 8 月 14 日公開

Contents

1 Introduction

Unityで使える強力なシリアルポート通信ユーティリティです。Android、Linux、MacOS、Windowsのマルチプラットフォームに対応し、開発者に使いやすいパワフルな通信ソリューションを提供するためのUnity用拡張プラグインです。

PCやマイコンとのデバイス通信をより簡単に実装でき、あなたのアイデアを最大限にサポートします。
応用: M2M, IoT, PLC, 施設制御, ホビーデバイス, ディープラーニング, FPGA UART.

1.1 Features

- Unityで簡単にシリアルポート通信を利用できます。
- PCとマイコン(Arduino, Ftdi, Microchip, Cypress, Silicon Labs, etc.)の通信を簡単に行えます。
- このプラグインではデバイス特有情報を指定して使用するデバイス直接的に選択できます。
- Windows、MacOS、Linux、Androidなどのマルチプラットフォームに対応。
- このプラグインは、ネイティブ処理で実行しUnityに最適化しています。よって低負荷で動作します。
- イベント駆動型の受信システムを採用し、分かりやすいプログラム構造になり、バグを生み出さないシステムを構築できます。
- JSONフォーマットに対応し、クラスの変数に直接データを入れ込むシステムを搭載しています。
- このプラグインでは、受信した文字列を自動的に処理して List<string>やDictionary<key, string>にコンバートします。
- システムはネイティブコードで実装されており、処理が重く遅い.NET標準のSystem.IO.Portsを使用しません。
- デバイスとの物理的切断が原因による終了のエラー検知。
- USB, PCI, 組み込み系UARTなどの物理的通信をサポート。
- Bluetooth SPP (Virtual COMPort)をサポート。
  TCP Serial Port エミュレータ(サーバーモード及びクライアントモード)をサポート。

1.2 Free Version

製品版を購入する前にプロジェクトに合うかどうかテストするためのバージョンです。
トライアルバージョンの制約は「3.1 Trial Version Limitations」を確認ください。それ以外は製品同様に利用可能です。
トライアルバージョンはここからダウンロードできます。
ダウンロード ： https://portutility.com/

2 System Requirements

- Unity 5.5 以降, Unity 2017.x, Unity 2018.x (Personal and Proの両方で利用可能)
- Windows 7 以降 (32-bit and 64-bit)
- Mac OS X 10.10 以降(64bit)
- Linux OS 4.x
- Android 4.1(Jelly Bean) 以降
- Serial Port デバイス(USB, RS-232C, RS-485, etc).
- Bluetooth デバイス
- Ethernet, Wi-Fi (TCP/IPv4)

3 Installation

3.1 Free Version Limitations

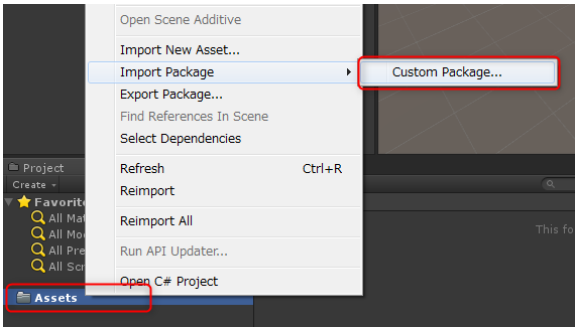
無料トライアルバージョンでは、以下制約があります。

- いずれかのデバイスのデータ転送(送受信)が**合計1MBまで使用可能**です。超過後は自動的に通信が切断されアプリケーションを再起動するまで使用できません。
- Windows及びMacOSのみサポートされています。
- 無料バージョンを使用したアプリケーションの配布はできません。

＜インストール方法＞

① サイトより「spup_v2.unitypackage」をダウンロードします。

② UnityエディタのAssetsを右クリックして「Import Package」→「Custom Package...」を選択して先ほどダウンロードしたパッケージをインポートします。



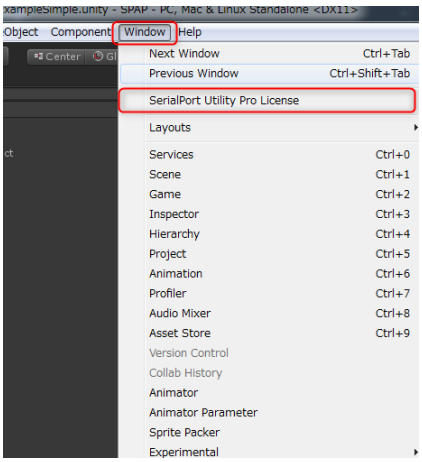
3.2 Upgrade to the licensed software.

製品版への移行はUnity Asset Storeから購入、アップグレードできます。製品購入後、下記よりパッケージを適用してください。

＜ライセンス適用方法＞

① Unity Asset Storeにアクセスし、「Serial Port Utility Pro」と検索します。**直接URL** http://u3d.as/1h5h

② 指示された手順にて購入してください。ダウンロードしたunitypackageを上書きでインポートすることも可能です。

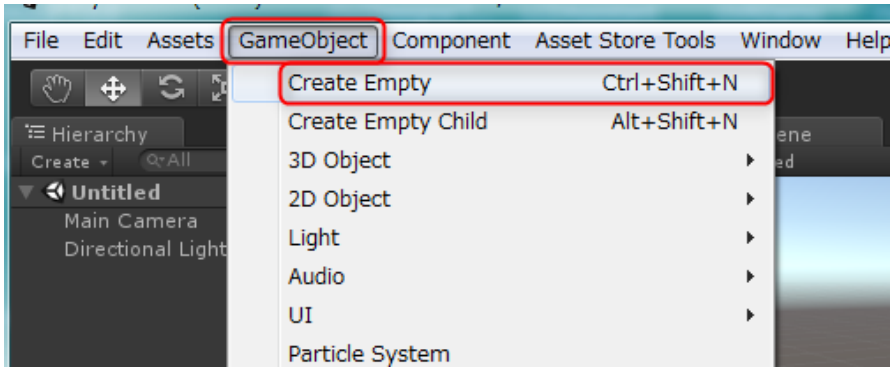


4 Using the Asset

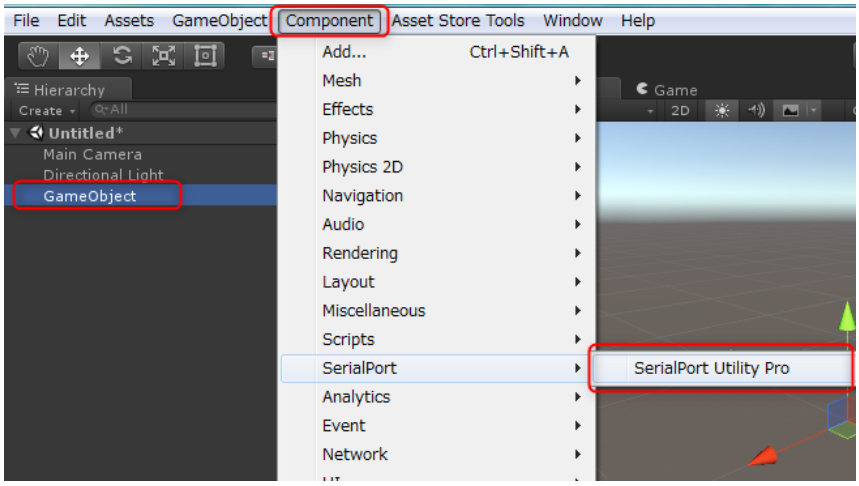
4.1 Getting Started

① プラグインがプロジェクトにインポートされているのを確認します。

② メニューバーのGameObjectからCreate Emptyを選択してシーンにオブジェクトを追加します。



③ 先ほどシーンに追加したオブジェクトを選択して、ComponentからSerialPort Utility Proを選択してオブジェクトにコンポーネントを追加します。



④ シーンにあるオブジェクトにコンポーネントが追加され機能として利用可能なります。

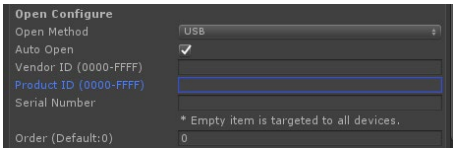
★もし詳しくビデオで確認する場合はこちらにアクセスしてください。

4.2 Supported Device Types

- **USB**

  Arduino社、Microchip社などのボード基板に搭載されているUSBデバイスや、USBとRS-232C、RS-485などのコンバータケーブルを使用して通信を行います。

  USBのVendor ID、Product ID、シリアルナンバーを指定して特定のポートにアクセスできます。

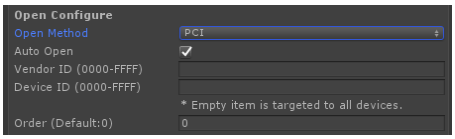


- **PCI**

  RS-232C、RS-485ポート用のPCI拡張ボードを指定して通信を行います。

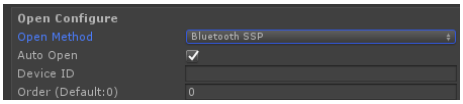
  PCI拡張ボードのDevice ID、Product IDを指定して特定のポートにアクセスできます。

  ※Windows及びLinuxのみ対応しています。



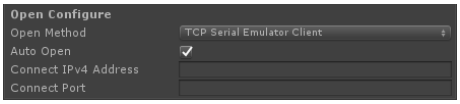
- **Bluetooth SPP**

  Bluetoothの仮想シリアルポートとAndroidに搭載されるBluetooth SPP機能を使って通信を行います。予め仮想のシリアルポートをOSに登録している必要があります。

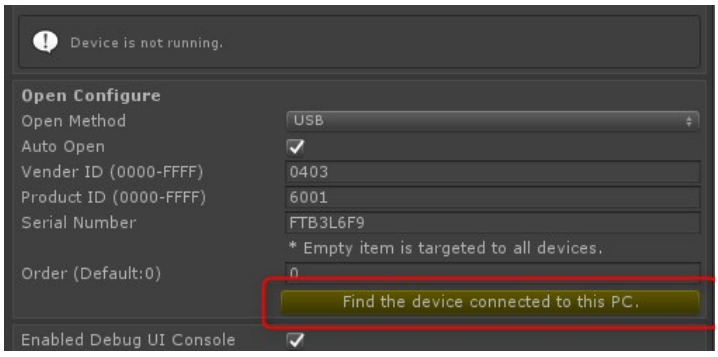


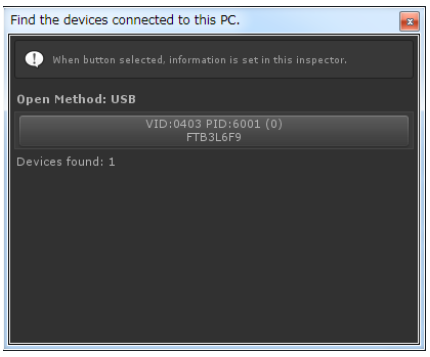
- **TCP Emulator Client & TCP Emulator Server**

  シリアルポートで通信を行うようにTCPを使って通信をエミュレートできます。使用時にはクライアントかサーバーとして通信するかを選択します。



Unityエディタで作業中のPCに接続されているデバイスの一覧を表示し、自動で入力してくれる機能です。一覧を表示するには下記のボタンをクリックしリストウィンドウを開きます。





4.3 How to Configurations

ゲームオブジェクトにComponentを追加すると、スクリプトの項目が設定できます。

一部の項目は、プログラミングによって動的に変更可能です。

SerialPort Status

このスクリプトのポート状態が表示されています。

Open Configure

ポートをオープンする方法と必要な情報を入力します。オープン方法によって入力できる項目が異なります。また、PCに接続されているデバイスを見つけることができます

Enabled Debug UI Console

デバッグ用のUIコンソールを有効にするかどうかを決定します。一度Enableにするとスクリプトからすることはできません。

Enabled External Config

外部ファイルを用いて、オープン前にクラスの変数に代入することができます。詳しくは「4.8 Using the External File Conifg」をご確認ください。

Enabled Transmission

データの送受信が可能にするかどうかを決定します。

Communication Structure

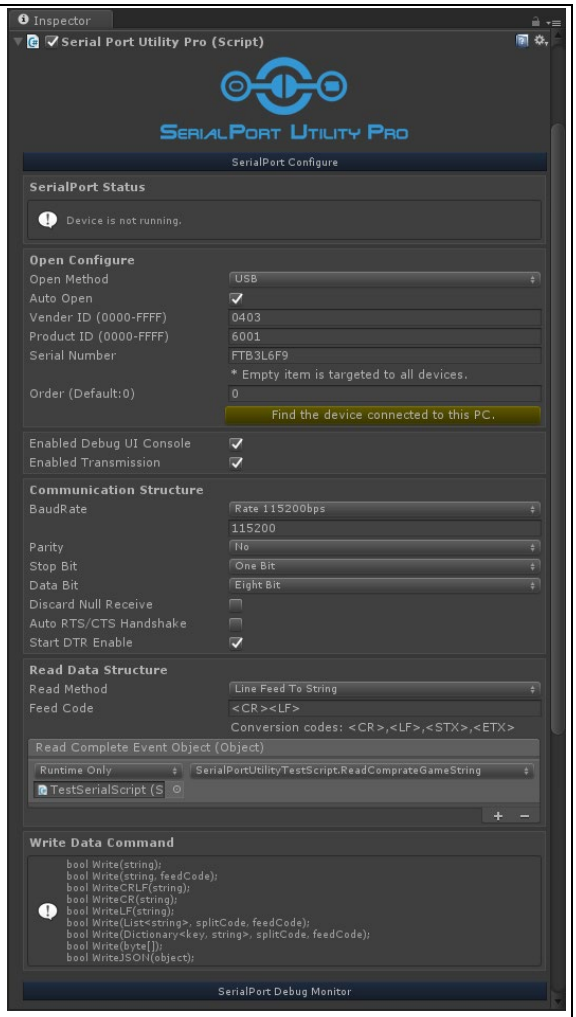
USB、PCIのシリアル通信の通信構造を設定します。

Read Data Structure

シリアル通信の受信を行う方法と、その処理の設定に関する必要な情報を入力します。

Write Data Command

シリアル通信の送信を行うためのスクリプト関数群が表示されています。この項目には入力はありません。



4.4 Receiving Data

ポートに受信されたデータは、MethodSystem ReadProtocolに指定した受信方法で処理をされたのち、SPUPEventObject ReadCompleteEventObjectで指定した関数が呼ばれます。タイミングは、必ず各フレーム(MonoBehaviour.Update関数)の最初かつ同期的に呼ばれます。

関数の引数には、objectで渡されるために関数内で必ず変換(as キャスト)が必要です。

- ReadProtocol: Streaming or BinaryStreaming

  受信したデータを特殊な処理せず、そのまま垂れ流しを行います。遅延は少し発生します。
  連続データは一時的に貯め、受信関数には一定のデータの塊として渡されます。

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

- ReadProtocol: LineFeedDataToString or ToBinary

  指定したFeedコードを1つの区切りとして受信関数が呼ばれます。デフォルト:<CR><LF>

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

- ReadProtocol: FixedLenghDataToString or ToBinaryData

  連続したデータの指定した固定数ごとに受信関数が呼ばれます。デフォルト12

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

- ReadProtocol: SplitStringToArray

  指定したFeedコードを1つの区切りとして受信関数が呼ばれます。また、指定したSplitコードごとに分割したデータでList配列(Array)に変換されます。

0000ms -> AAA,BBB,CCC,DDD<CR><LF> 0010ms -> EEE,FFF,GGG,HHH<CR><LF> 1000ms -> III,JJJ,KKK,LLL<CR><LF>	var DAT = data as List<string>; DAT[0] -> AAA, DAT[2] -> BBB, DAT[3] -> CCC, DAT[4] -> DDD DAT[0] -> EEE, DAT[2] -> FFF, DAT[3] -> GGG, DAT[4] -> HHH DAT[0] -> III, DAT[2] -> JJJ, DAT[3] -> KKK, DAT[4] -> LLL

- ReadProtocol: SplitStringToDictionary

  指定したFeedコードを1つの区切りとして受信関数が呼ばれます。また、指定したSplitコードごとに分割したデータで辞書配列(Dictionary)に変換されます。関連付けは「=」で示します。

0000ms -> AAA=BBB,CCC=DDD<CR><LF>	var DAT = data as Dictionary<string, string>;

0010ms -> EEE=FFF,GGG=HHH<CR><LF> 1000ms -> III=JJJ,KKK=LLL<CR><LF>	DAT["AAA"] -> BBB, DAT["CCC"] -> DDD DAT["EEE"] -> FFF, DAT["GGG"] -> HHH DAT["III"] -> JJJ, DAT["KKK"] -> LLL

- ReadProtocol: SplitStringToGameObject

  指定したFeedコードを1つの区切りとして受信関数が呼ばれます。また、指定したSplitコードごとに分割したデータで辞書配列(Dictionary)に変換されます。関連付けは「=」で示します。

0000ms -> AAA=BBB,CCC=DDD<CR><LF> 0010ms -> EEE=FFF,GGG=HHH<CR><LF> 1000ms -> III=JJJ,KKK=LLL<CR><LF> ※GameObjというゲームオブジェクトを指定	ゲームオブジェクトの変数に格納されます。 GameObj.AAA = "BBB", GameObj.CCC = "DDD" GameObj.EEE = "FFF", GameObj.GGG = "HHH" GameObj.III = "JJJ", GameObj.KKK = "LLL"

- ReadProtocol: JSONToClassObject

  JSON形式の文字列をクラスオブジェクトに変換します。

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

- ReadProtocol: ModbusASCII

  Modbusフォーマットに対応したASCII文字列を処理します。

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

4.5 Sending Data

ポートから送信したいデータは、SerialPortUtilityProクラスのWrite関数で行えます。また、予めOpen関数によって正常にポートが開かれている必要があります。

Splitコード及びFeedコード用引数には、特殊制御コード<CR>,<LF>,<STX>,<ETX>に対応しています。

```
SerialPortUtility.SerialPortUtilityPro port;
//Componentでportを取得する
port = this.GetComponent<SerialPortUtility.SerialPortUtilityPro>();
//書き込み
port.Write( “Hello World!” );
port.WriteCRLF( “Hello!” ); // Write( “Hello!¥r¥n” );と等価です
port.WriteCR( “Hello!” ); // Write( “Hello!¥r” );と等価です
port.WriteLF( “Hello!” ); // Write( “Hello!¥n” );と等価です
port.Write( “Hello!” ,” <CR><LF>” ); // Feedコードは文字列の最後に追記します。
```

文字列ではなくバイト列を送る場合は同じWrite関数によって送信できます。
Stringと違うところは、NULLなど文字として表現されないデータも取り扱いできます。

```
byte[] dataByte = new byte[256]; //256byte
dataByte[0] = 0x00;
dataByte[1] = 0x20;
//必要に応じてデータを処理.
port.Write(dataByte);
```

ListクラスやDictionaryクラスも、指定したSplitコードとFeedコードを使って送信文字列を自動生成して送信することができます。

```
List<string> dataArray = new List<string>();
dataArray.Add("AAA");
dataArray.Add("BBB");
dataArray.Add("CCC");
port.Write(dataArray, ",","<CR><LF>");
//AAA,BBB,CCC<CR><LF>として送信されます。
```

```
Dictionary<string, string> dataArray = new Dictionary<string, string>();
dataArray.Add("AAA", "BBB");
dataArray.Add("CCC", "DDD");
port.Write(dataArray, ",", "<CR><LF>");
//AAA=BBB,CCC=DDD<CR><LF>として送信されます。
```

実体のあるクラスをJSONフォーマットの文字列に自動生成して送信することも可能です。

```
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);
}
```

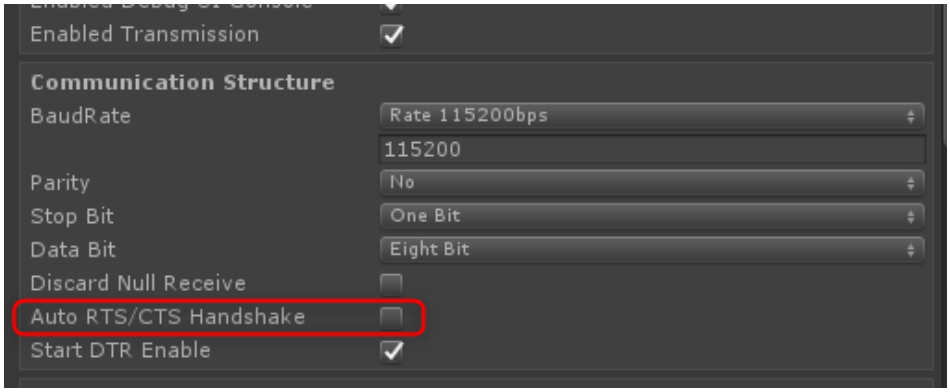
クラスオブジェクトを渡すことが出来るので、非常にスムーズなデータ交換が可能となります。また、送信した内容については、Debug UI Consoleにも反映されます。

4.6 About control functions

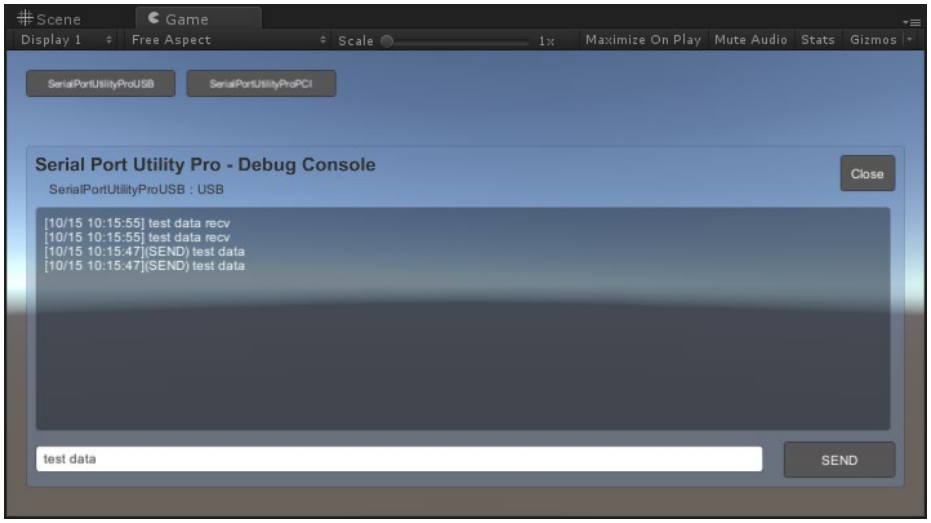
OpenMethodがBluetoothSPP及びTCP/IP以外では、シリアルポートのライン制御用の関数が用意されています。また、制御可能なラインは、DSR、CTS、DTR、DTSで、「6 Scripting Reference」項目の関数群をご参照ください。

これらの状態を制御するには、予めOpen関数によって正常にポートが開かれている必要があります。

Unityのアプリケーションの状態によって自動で切り替える機能「Auto RTS/CTS Handshake」もあります。これはUnityアプリケーションがフォアグラウンドで動作しているかどうかなどを検知し、自動的にRTS/CTSの設定を切り替える機能です。

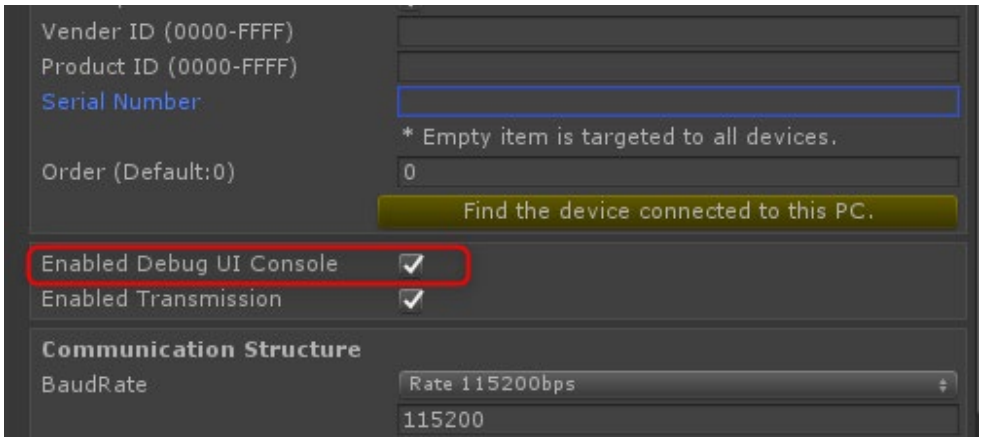


4.7 Using the Debug UI Console



Serial Port Utility Proには、デバッグ用のUIコンソールが実装しています。すぐにシリアル通信のデータを可視化できます。また、デバッグにチェックが入っている全てのコンポーネントがボタンとして表示され、ボタンをクリックするとシリアルモニタとして使用できます。

デバッグコンソールを有効にするには、「Enabled Debug UI Console」にチェックを入れます。

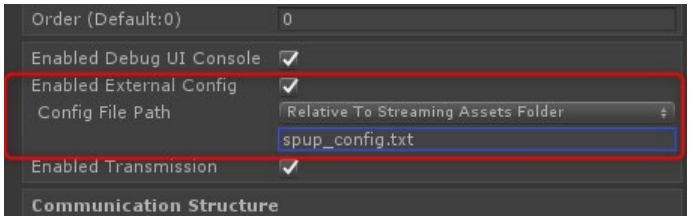


製品ビルド時には、このモードをfalseにしておく方がよいでしょう。

4.8 Using the External File Conifg

Serial Port Utility Proには、ビルド後の実行ファイルに関して、外部ファイルによる設定変更が実装しています。ビルドされた実行ファイルに対して設定を変更する必要がある場合、チェックを入れておくと外部設定ファイルより、デバイスのオープン前にクラスのメンバー変数を変更します。

これによって、ビルド後でも容易に環境に応じて設定を変えることが出来ます。



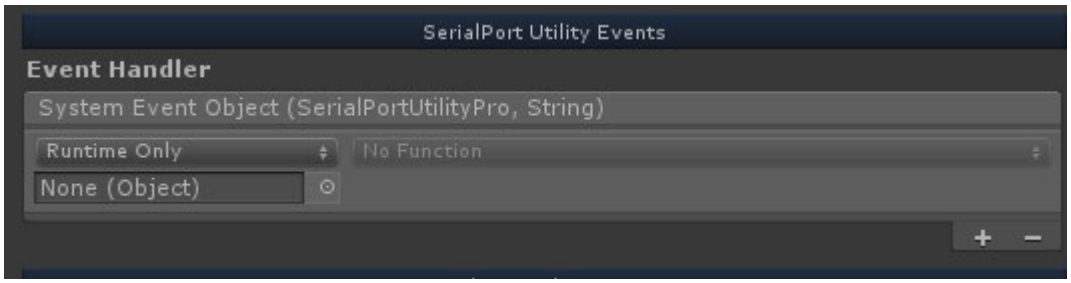
Serial Port Utility Proクラスにあ るPublic変数名を「=」を使って代入します。複数ある場合は、改行して次の変数を指定します。例えば、spup_config.txtというファイルを作り、下記のようなデータを入れます。

```
VendorID=1234       # コメント
BaudRate=115200
Parity=0             # 0=No
StartEnableDTR=1    # 1=true 0=false
```

予期せぬデータが入力された場合、実行ファイルがクラッシュする場合があります。データの見直しを行ってください。

4.9 Utility Events :ユーティリティイベントとは

ユーティリティイベントは、クラスがオープンしたとき、クローズしたとき、さらにはエラーが発生したときなどにInvokeで呼び出されるイベントハンドラーです。関数の引数には、呼び出したSerialPort UtilityProのクラス、呼び出した種類の文字列が返ります。



標準ではグループで閉じてあるので、「SerialPort Utility Events」タブをクリックして開いてください。

5 Quick Start Examples

5.1 Quick Start Programming

AssetのExampleScenesのシーンを用いて解説をしていきます。シーンを確認するには、ExampleScensesフォルダにあるシーンを開いてください。

- **ExampleSimple から ExampleSimpleBinary**

Serial Port Utility Proを使うときの基本的なサンプル例です。まずは、SPUPTestScriptSimple.csを参考に下記受信用の関数を作成します。リスト形式となったデータとして受信したい場合は、

```
public void ReadComprateList(object data) {
  var text = data as List<string>;
  for(int i=0; i < text.Count; ++i)
    Debug.Log(text[i]);
}
```

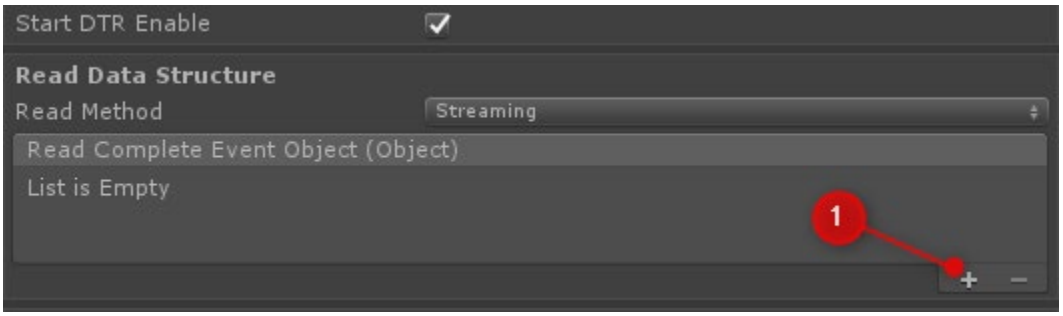
辞書形式データとして受信し処理したい場合は、

```
public void ReadComprateDictionary(object data) {
  var text = data as Dictionary<string, string>;
  Debug.Log(text["test"]);
}
```

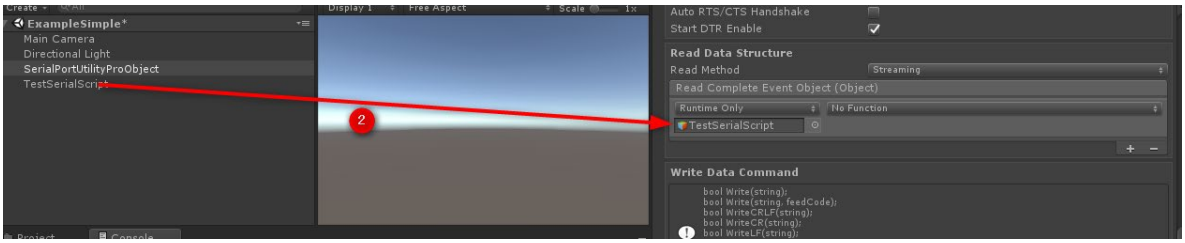
単純に一定の文字列として受信し処理したい場合は、

```
public void ReadComprateString(object data) {
  var text = data as string;
  Debug.Log(text);
}
```

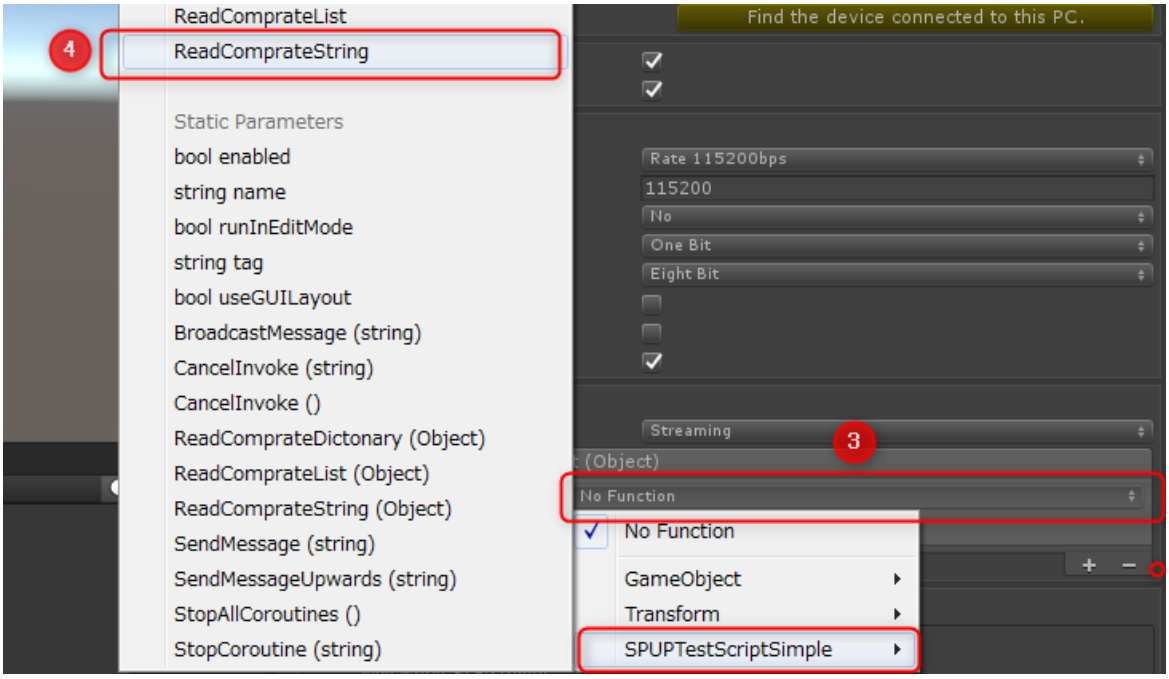
というふうに関数を作成します。「Read Data Structure」に下記のように受信する関数をセットします。



シーンオブジェクトからインスペクターに追加します。



シーンオブジェクトにあるコンポーネントの関数を呼び出すことが出来ます。



今回は、ReadCompreateStringをセットしています。ここでは必ずDynamic Objectタイプの関数をセットしてください。

そして、データを送信するにはオープンしているSerialPortUtilityProクラスを取得し、そのメンバー関数である「Write」を使用します。

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

- ExampleGameObject

  シーンのヒエラルキーに存在するGameObjectを指定して、そのスクリプトに対してデータを受信し代入するサンプル例です。GameObjectに関連付けされているクラスとメンバー変数全てに適用されます。

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

  というクラスに対して、

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

  というデータを受信した場合は、指定したGameObjectに関連付けしているSPUPTestScriptGObj.Item1とSPUPTestScriptGObj.Item2の変数それぞれにTESTDATA、TESTDATA2が代入されます。

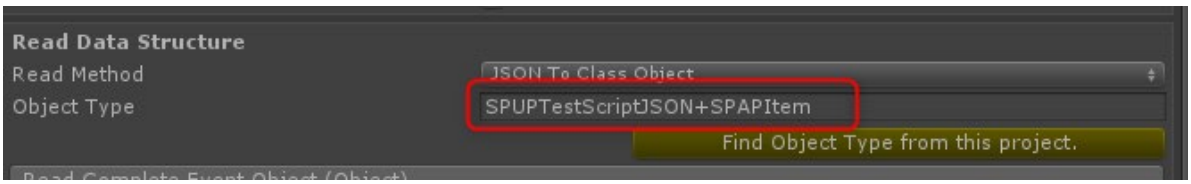
- ExampleJSON

  JSONフォーマット形式でデータの通信を行うことができるサンプル例です。JSONフォーマットから変換したobject型を明確的にキャストする必要があります。

```
public void ReadComprateJSON(object data) {
  var item = data as SPAPItem;
}
```

  送信するときには、WriteJSONを使用してJSONフォーマットに変えて送信することが出来ます。

```
SerialPortUtility.SerialPortUtilityPro serialPort = XXX;
serialPort.WriteJSON(itobj);
```



  Read Data Structureを設定するときには、Object Typeを明確的に設定する必要があります。「Find Object Type from this project」から開いているプロジェクトに存在するクラスの一覧を表示し選択できます。

- ExampleProgramming

  プログラム内で完結する方法のサンプル例です。「SPUP Programming.cs」が生成するので、停止時にはシーン上にSerialPortUtilityProオブジェクトは存在しません。

```
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;
```

ここで、注意が必要なのは、オブジェクトの生成方法です。

```
serialPort = new SerialPortUtility.SerialPortUtilityPro();  //不可能
```

としてSerialPortUtilityProオブジェクトの生成はできません。

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

のようにGameObjectによって管理される必要があります。親クラスにMonoBehaviourがあるからです。また、さらに各種項目の設定後にOpen関数を使用して手動で通信を開始する必要があり、終了時にはClose関数を呼び出し明確的にオブジェクトを終了する必要があります。

5.2 Troubleshooting

- **Windows使用時にspap.dllのロードエラーとなる。**

  〈解決策〉

  下記マイクロソフト社のURLからVC++ 2017ランタイムをインストールする

  https://go.microsoft.com/fwlink/?LinkId=746572

- **Linux使用時にlibspap.soのロードエラーとなる。**

  Libspap.soは、libudevを使用しています。udevを別途インストールする必要があります。

  〈解決策〉

  1. udev をインストールする

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

- **Linux使用時にシリアルポートが存在するのに、オープンできない場合。**

  デバイスは、権限がスーパーユーザーに設定しているため、使用ユーザーに権限を設定しないと使用できません。

  〈解決策〉

  1. スーパーユーザーとしてアプリケーションを起動する。もしくは、

```
$ sudo ./unity_application
```

  2. OS起動時にデバイスに権限を一括で変更する。もしくは、

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

3. “dialout”グループに使用しているユーザーを追加します。

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

この場合、最初追加したら2度目からは設定不要になります。

- **AndroidOSのUSBシリアルを使用時にパーミッション許可ダイアログが出ないようにする。**

1. Assetフォルダにある「Plugin」フォルダ→「Android」フォルダを開きます
2. [spap_devicelist.zip]ファイルを解凍します。
3. 「res」フォルダ→「xml」フォルダ内の「device_filter.xml」をテキストエディタで開きVendorIDとProductIDを10進数で登録します。

```
<resources>
  <usb-device vendor-id="1027" product-id="24577" />
  <usb-device vendor-id="9025" product-id="67" />
</resources>
```

4. 解凍され変更されたファイルを含むフォルダをZIPしてください。
5. このフォルダの[spap_devicelist.zip]を[spap_devicelist.aar]に変更すると有効になります。

5.3 Tutorial Videos

https://www.youtube.com/watch?v=HOtF3tdDKA4&list=PL6GxgTGZovAsVwecl1mLnIGasvSjU7eHm

6 Scripting Reference

```
namespace SerialPortUtility;
    class SerialPortUtility. SerialPortUtilityPro (MonoBehaviour)
    <Method>
        void Open()
```

Description: 設定した項目でデバイスをオープンを試みます。

Parameter: none

Return: none

```
        void Close()
```

Description: 設定した項目でデバイスをクローズを試みます。

Parameter: none

Return: none

bool IsOpened()

Description: デバイスがオープンしているかどうかを確認します。

Parameter: none

Return: 戻り値がtrueの場合は、既にオープンしていることを示します。

bool IsErrorFinished()

Description: デバイスがオープンしていたにもかかわらず、何らかのエラー原因で終了してしまった場合trueがセットされます。

Parameter: none

Return: 戻り値がtrueの場合は、エラー終了を示します。

bool Write(string)

Description: 開かれたシリアルポートに対して書き込み(送信)を行います。

Parameter: 送信する文字列

Return: 戻り値がtrueの場合、書き込みに成功。

bool Write(byte[] or byte)

Description: 開かれたシリアルポートに対してバイトデータを書き込み(送信)します。

Parameter: 送信するbyte配列かbyte型

Return: 戻り値がtrueの場合、書き込みに成功。

bool Write(string, string)

Description: 開かれたシリアルポートに対してFeedコードを付加して書き込み(送信)を行います。Feedコードは、制御用コード<CR>,<LF>,<STX>,<ETX>に対応しています。

Parameter: 1. 送信する文字列, 2.末尾に付加するFeed文字コード

Return: 戻り値がtrueの場合、書き込みに成功。

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

Description: 開かれたシリアルポートに対してList化されたデータをSplitコード及びFeedコードを用いて書き込み(送信)を行います。Split、Feed文字コードは、制御用コード<CR>,<LF>,<STX>,<ETX>に対応しています。

Parameter: 1.Listクラス, 2.Splitコード,3.Feedコード

Return: 戻り値がtrueの場合、書き込みに成功。

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

Description: 開かれたシリアルポートに対してDictionary化されたデータをSplitコード及びFeedコードを用いて書き込み(送信)を行います。Split、Feedコードは、制御用コード
<CR>,<LF>,<STX>,<ETX>に対応しています。
Parameter: 1.Dictionaryクラス, 2.Splitコード,3.Feedコード
Return: 戻り値がtrueの場合、書き込みに成功。

bool Write(UnityEngine.GameObject, string, string)

Description: 開かれたシリアルポートに対して、GameObjectに加されたスクリプトのメンバー変数をSplit及びFeedを用いて書き込み(送信)を行います。Split、Feed文字コードは、制御用コード
<CR>,<LF>,<STX>,<ETX>に対応しています。
Parameter: 1.ヒエアルキーにあるGameObject, 2.Splitコード,3.Feedコード
Return: 戻り値がtrueの場合、書き込みに成功。

bool Write(SPUPMudbusData mudbus_data, bool binaryMode)

Description: Mudbusデータ形式でデータを送信します。チェックデジットも自動的に付加します。
Parameter: 1.Mudbusデータクラス, 2.ASCIIモード及びRTUモード選択
Return: 戻り値がtrueの場合、書き込みに成功。

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

Description: 開かれたシリアルポートに対してCRやLFコードを付加して書き込み(送信)を行います。
Parameter: 送信する文字列
Return: 戻り値がtrueの場合、書き込みに成功。

bool WriteJSON(object)

Description: 開かれたシリアルポートに対して引数のデータをJSON形式で書き込み(送信)を行います。
Parameter: 送信するオブジェクト、JSONフォーマットに変換されます。
Return: 戻り値がtrueの場合、書き込みに成功。

bool CtsHolding()

Description: 開かれたシリアルポートのCTSの状態を取得します。
Parameter: none

Return: CTSの状態がHIかLOWで返ります。

bool DsrHolding()

Description: 開かれたシリアルポートのDSRの状態を取得します。

Parameter: none

Return: DSRの状態がHIかLOWで返ります。

bool DtrEnable(bool)

Description: 開かれたシリアルポートのDTRの状態を変更します。

Parameter: trueの場合はHIになります。

Return: 戻り値がtrueの場合、変更に成功。

bool DtrGetStatus()

Description: 開かれたシリアルポートのDTRの設定された状態を取得します。

Parameter: none

Return: DTRの状態がHIかLOWで返ります。

bool RtsEnable(bool)

Description: 開かれたシリアルポートのRTSの状態を変更します。

Parameter: trueの場合はHIになります。

Return: 戻り値がtrueの場合、変更に成功。

bool RtsGetStatus()

Description: 開かれたシリアルポートのRTSの設定された状態を取得します。

Parameter: none

Return: RTSの状態がHIかLOWで返ります。

bool IsConnected()

Description: **DsrHolding()関数と同じ挙動です。「TCP Emulator Server」でクライアントの接続確認などの時に使用します。**

Parameter: none

Return: RTSの状態がHIかLOWで返ります。

bool SetBreakSignal(bool)

Description: 開かれたシリアルポートに対してブレーク信号のON、OFFを行います。

Parameter: trueでブレーク信号中、falseでブレーク信号解除となります。

Return: ブレーク信号の変更が成功した場合はtrueが返ります。

bool SerialDebugAddString(string)

Description: デバッグ用のウィンドウに表示に指定した文字列を書き込みます。シリアルポートに対しては、送信されません。

Parameter: 送信する文字列

Return: 戻り値がtrueの場合、書き込みに成功。

void ReadUpdate()

Description: 受信システムを更新します。通常UnityのUpdate()にて呼ばれますが、Manualモード時にはこの関数を使用して手動で更新する必要があります。

Parameter: none

Return: none

void GetConnectedDeviceList (OpenSystem OpenMethod)

Description: 接続されているデバイスのリストを表示します。

Parameter: デバイスリストを表示するデバイスの種類

Return: DeviceInfo[] デバイスリスト

<Property>

bool IsAutoOpen = true;	Trueの場合、スクリプトが有効(OnEnable)になった時に自動的にオープンされます。デフォルトはtrueです。
OpenSystem OpenMethod = OpenSystem.USB;	オープンするデバイスの種類を選択します。 OpenSystem列挙から選択できます。
MethodSystem ReadProtocol = MethodSystem.SplitStringToArray;	受信時の処理の方法を選択します。 MethodSystem列挙から選択できます。
string VendorID	オープンするデバイスのVendorIDです。
string ProductID	オープンするデバイスのProductIDです。
string SerialNumber	オープンするデバイスのシリアルナンバーです。 有効なのはOpenMethod がUSBの時のみです。
string IPAddress	オープンするIPアドレスです。 有効なのはOpenMethod がTCP Serial Emulator Clientの時のみです。
int Port	オープンするインターネットのポート番号です。

	有効なのはOpenMethod がTCP Serial Emulator Client及びServerの時のみです。
string DeviceName	オープンするBluetooth SPPのデバイス名です。 有効なのはOpenMethodがBluetooth SPPの時のみです。
int BaudRate = 9600;	デバイス通信の速度ボーレートを設定します。
ParityEnum Parity = ParityEnum.No;	デバイス通信のパリティチェックを設定します。
StopBitEnum StopBit = StopBitEnum.OneBit;	デバイス通信のストップビットを設定します。
DataBitEnum DataBit = DataBitEnum.EightBit;	デバイス通信のデータビット数を設定します。
bool RecvDiscardNull= false;	Trueの場合、NULL文字を受信前に破棄します。
bool AutoRTSCTSHandshake = false;	Trueの場合、UnityによってRTSCTSを制御します。
bool StartEnableDTR = true;	Trueの場合、スタート時にDTRを1にします。
bool DtrEnabled = false;	Trueの場合、DTRを1にします。
bool RtsEnabled = false;	Trueの場合、RTSを1にします。
int Skip = 0;	同一のデバイスが接続されている場合に判別するための値です。
string FeedCode = "<CR><LF>"	送りをするための文字コード(読み込み完了及びInvolveが入ります)
string SplitCode = ","	分割用の文字コードを設定します。
int FixedFeedCount = 10	ReadProtocolがFixedCharactersToStringで設定されている場合のみ有効です。FixedSplitCountで設定した分割数に達した時、送りをします。
UpdateMethod UpdateProcessing = Update	ReadUpdate関数の呼ぶタイミングを指定します。
bool DiscardSpaceTabChar = false	Trueの場合、スペース、タブ文字を受信前に破棄します。
SPUPEventObject ReadCompleteEventObject = new SPUPEventObject()	処理されたデータの受信完了時に呼ばれる関数を指定します。
string ReadCompleteEventObjectType = ""	ReadCompleteEventObjectで設定の引数
GameObject ReadClassMembersObject = null	ReadProtocolがSplitStringToGameObjectで設定されている場合のみ有効です。設定しているGameObjectのコンポーネントクラスのメンバー変数に直接反映させます。
string GetSerialDebugString	現在のデバッグ表示用の文字列を取得します。編集はできません。

enum SerialPortUtility.SerialPortUtilityPro.MethodSystem

FixedLengthDataToString	固定文字数を区切りに1つの文字列として処理します。
FixedLengthDataToBinaryData	固定データ数を区切りに1かたまりのとして処理します。

JSONString	JSON文字列をオブジェクトとして変更します。
LineFeedDataToString	Feed文字ごとに文字列を読み込みます。
LineFeedDataToBinary	Feed文字ごとに文字列をバイナリデータとして読み込みます。
SplitStringToArray	Feed文字ごとのSplit文字で分割された文字列を配列として処理します。
SplitStringToDictionary	Feed文字ごとのSplit文字で分割された文字列を辞書オブジェクトとして処理します。
SplitStringToGameObject	Feed文字ごとのSplit文字で分割された文字列をGameObjectのメンバー変数に直接入れて処理します。
Streaming	一定間隔で受信するテキストを処理せずそのまま受信します。
BinaryStreaming	一定間隔で受信するデータを処理せずそのまま受信します。

enum SerialPortUtility.SerialPortUtilityPro.OpenSystem

Bluetooth SSP	Bluetooth SPP を使って通信をします。
Number Order	USB->PCI->BluetoothSPPの順番で接続されている全てのデバイスを試みます。
PCI	PCI インターフェイスを使って通信をします。
TCP SerialEmulator Client	TCP/IPを使ってシリアル通信をエミュレートします。
TCP SerialEmulator Sever	TCP/IPを使ってシリアル通信をサーバーモードでエミュレートします。
USB	USB インターフェイスを使って通信をします。

enum SerialPortUtility.SerialPortUtilityPro.DataBitEnum

EightBit	データビット数8 bit
SevenBit	データビット数7 bit

enum SerialPortUtility.SerialPortUtilityPro.ParityEnum

Even	偶数としてパリティチェックします。
Mark	パリティチェックは常に1とします。
No	パリティチェックはしません。
Odd	奇数としてパリティチェックします。
Space	パリティチェックは常に0とします。

enum SerialPortUtility.SerialPortUtilityPro.StopBitEnum

OneBit	ストップビット1 bit
TwoBit	ストップビット2 bit

class SerialPortUtility.DebugConsole (MonoBehaviour)

It is a script for implementing an internal debugging console.

You cannot touch the code.

7 Revision History

- **v2.14 - August 13, 2019**
  1. Fixed bugs of Andriod Plugin and Mac OS Plugin.
  2. Added function to display device list.
- **v2.13 - June 14, 2019**
  1. Fixed not to close by the break-signal.
  2. Added AndroidManifest.xml template to hide AndroidOS USB permission dialog.
- **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 more 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 website.
- v1.0 - July 7, 2018
  1. First released in private.

8 License

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

- [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

WIZPAPLY 株式会社

OFFICE

Address: 552-0002 大阪府大阪市港区市岡元町3－7－10 KSビル5F

TEL: 06-4400-6308

E-MAIL: info@wizapply.com

- ご不明点ございましたらお気軽にご連絡くださいませ。