ICS 変換基板

ICS ライブラリ

for　Arduino

第 2.11 版

近藤科学株式会社

2021 年 9 月 13 日　第 2.11 版

目 次

目次 1
はじめに 3
諸注意 3
免責事項 3
お問い合わせ 3
準備するもの 4
ICS 変換基板 4
製品の構成 4
Arduino について 4
Arduino 対応機器 5
開発環境 5
ICS 機器 6
Dual USB アダプターHS 6
電源 6
ICS Library for Arduino 8
接続方法 9
Arduino との接続方法 9
シールド基板を用いる方法 11
プログラムの書き込み方法 12
ICS Library for Arduino について 13
ライブラリの概要 13
ライブラリの入手方法 13
ライブラリ圧縮ファイルの中身 13
ライブラリのインポート 14
ライブラリの保存場所 15
ライブラリの削除 15
サンプルプログラムについて 16
サンプルプログラムの呼び出し 16
サンプルプログラムの種類 16
KrsServo1 の詳細 17
ソースコード 17
ソースコード解説 17
SoftwareSerial を使用する場合 18
複数の UART を用いる場合 19
ICS Library for Arduino の詳細 20
ICS 機器と通信するための最初設定 20

ICS クラスを宣言する(HardwareSerial 版)　IcsHardSerialClass () 20
ICS クラスを宣言する(SoftwareSerial 版)　IcsSoftSerialClass() 21
ICS クラスを開放する　~ IcsHardSerialClass ()/~IcsSoftSerialClass() 22
通信を開始する　begin() 22
個別にコマンドを送受信する場合の関数 24
コマンドを使って送受信する　synchronize () 24
固定パラメーター覧 25
サーボモータを動かす 26
サーボモータを動かす　setPos () 26
サーボをフリーにする　setFree () 27
サーボモータのパラメータを書き込む 28
ストレッチ書き込み　setStrc () 28
スピード書き込み　setSpd () 28
電流値書き込み　setCur () 29
温度値書き込み　setTmp () 30
サーボモータのパラメータを取得する 31
ストレッチ読み込み　getStrc () 31
スピード読み込み　getSpd () 31
現在の電流値を読み込む　getCur () 32
現在の温度値を読み込む　getTmp () 33
現在のポジションデータを読み込む　getPos () 33
ID の管理 35
接続されている機器の ID を読み込む　getID () 35
接続されている機器の ID を書き込む　setID () 35
ポジションデータと角度(degree)を変換する 37
角度(degree)をサーボのポジションデータを変換する　degPos () 37
ポジションデータを角度に変換する　posDeg () 37
100 倍した角度をポジションデータに変換する　degPos100 () 38
ポジションデータを 100 倍した角度に変換する　posDeg100 () 39
KRR（受信機）の値を取得する 41
enum KRR_BUTTON：unsigned short 41
ボタンデータを取得する　getKrrButton () 42
アナログデータを取得する　getKrrAnalog () 43
すべてのデータを一括で取得する　getKrrAllData () 43
改訂履歴 45

はじめに

このたびは ICS 変換基板をお買い求めいただき、誠にありがとうございます。本製品をご使用の前に、本マニュアルを熟読いただきますよう、お願いいたします。

本マニュアルは ICS 変換基板を用いて Arudino 系マイコンボードから弊社製 ICS 機器を制御するための手順、ライブラリ、サンプルプログラムについて解説します。

著作権などの法的権利については近藤科学株式会社が有します。またマニュアルに記載の会社名、商品名およびロゴマークはそれぞれの会社の商標または登録商標ですので、無断で使用することはできません。

本マニュアルおよび付属品に掲載された一切の情報の流用による結果についての責任は負いかねます。

また内容は予告無く内容が変更される場合があります。ご理解の上ご使用なさいますよう、お願いいたします。

諸注意

- 本製品を濡らしたり、湿度が高い場所で使用しないでください。
- 本製品は基板上の各端子が露出しているため、ショートの危険性があります。ショートを起こしたり端子の誤接続をすると発熱/発火の恐れがありますのでご注意下さい。金属に接した状態での使用はおやめください。
- 本製品の日本国内以外での使用に関しましてはサポートいたしかねます。
- ケーブル類はプラグ部分をつかんで挿抜していただき、差し込むときは、向きを間違えないようにしてください。
- お客様によるハンダ付け不良等による不具合に関しましては保証対象外となります。
- 異常（発熱・破損・異臭など）を感じたらすぐにバッテリや電源を切るようにしてください。
- 問題が発生した場合は直ちに使用をやめ、弊社サービス部へご相談ください。

免責事項

本リファレンスおよび内容に関する一切の権利は近藤科学株式会社が有しますが、このリファレンスは参考資料として公開されるものです。このリファレンスを使用したときの障害や損害につきましては、近藤科学株式会社は一切保証いたしませんので、使用者の責任においてご利用ください。

本マニュアルに関する著作権、ロゴや一部のアイコンのデザイン、その他法的な諸権利の一切は近藤科学株式会社が有します。

本マニュアルには 2018 年 1 月現在における対応状況を記載しております。

本マニュアルの内容につきましては、予告なく変更する場合があります。

本マニュアルおよびソフトェアのあらゆる形態での不特定多数への再配布は禁止します。また著作権者に無断で販売、貸与およびリースなども出来ません。

近藤科学株式会社は、ソフトェアのインストールや使用の結果によって生じたあらゆる損害に対して、一切の責任を負いません。

お問い合わせ

本製品ならびに付属品についてのお問い合わせは弊社サポート窓口までご連絡下さい。

〒116-0014　東京都荒川区東日暮里 4-17-7

近藤科学株式会社　サービス部　TEL　03-3807-7648　（サービス直通）

土日祝祭日を除く 9:00 ～ 12:00　13:00 ～ 17:00

製品についての告知及びアップデータ等は弊社ウェブサイト http://www.kondo-robot.com に掲載されます。E-mail でのお問い合わせにつきましては、support@kondo-robot.com にて承りますが、回答までお時間を頂く場合がございます。

※Arduino 使用方法やプログラムの詳細等お問い合わせはお答えできない場合がありますので、あらかじめご了承ください。

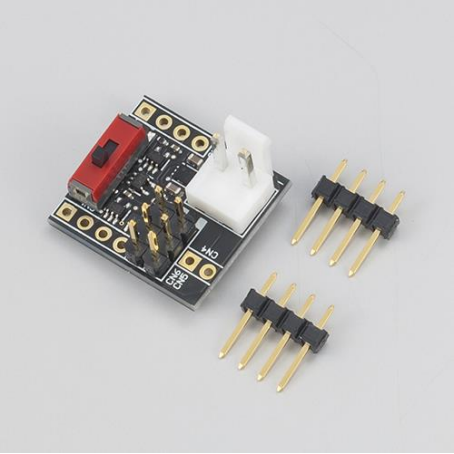
準備するもの

ICS 変換基板

弊社 ICS 機器を市販のマイコンボードのシリアル端子（UART）に接続するための変換基板です。Tx、Rx などの通信線や電源などの回路を用意する手間が省け、接続するだけで簡単に ICS 機器の制御が可能となります。

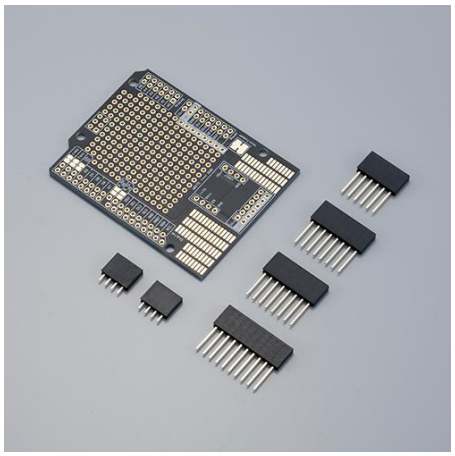
また、別売りの専用シールド基板を用意するとより、Arduino UNO への配線も省き簡単に接続することが可能です。

※本製品は、ヘッダピンのはんだ付け作業が必要です。



ICS 変換基板

No.03121　￥1,800（税別）



KSB シールド 2

No.03149　￥1,200（税別）

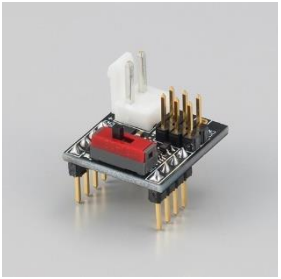
製品の構成

ICS 機器と Arduino などマイコンボードとの間に ICS 変換基板を中継することにより、回路を自作することなく手軽に通信を行うことができます。

ICS 機器

KRS サーボ、KRR-5FH など

⇔



ICS 変換基板

⇔

Arduino

Arduino について

Arduino はオープンソースハードウェアで、対応したマイコンボードが一般販売されています。

本製品は Arduino のシリアル端子(UART、HardwareSerial)を利用し、ICS 機器と通信を行います。また、Arduino UNO などシリアル端子が少ないボードでは、デジタル I/O 端子を使用した SoftwareSerial での通信も可能です。

HardwareSerial と SoftwareSerial

Arduino などマイコンボードに実装されているシリアル端子(UART)を使用して通信することを HardwareSerial と呼びます。通常は HardwareSerial を使用することで安定した通信を行いますが、マイコンによってはシリアル端子が別の用途に使用されていたり、実装自体がない

場合があります。その場合、デジタル I/O 端子をソフトウェア上でシリアル端子のように使用します。この方法を SoftwareSerial と呼びます。SoftwareSerial は、本来ボードが持っている機能ではないため、不安定な場合があり通信が途切れる可能性があります。可能な限り HardwareSerial を使用することをお勧めします。

Arduino 対応機器

弊社にて動作確認ができている対応機器は以下の通りになります。(2020 年 2 月現在)

※Arduino IDE を使用できる前提になります。

●HardwareSerial

Arduino UNO	Arduino 101	Arduino ZERO
ArduinoMEGA 2560	~~Arduino Due~~	Arduino MICRO
Arduino MKR ZERO		
Arduino Nano Every		
M5StackC		
Tenssy3.6	Tenssy4.0	

●SoftwareSerial

Arduino UNO、Arduino Mega 2560、Arduino mini

※Arduino の種類によって使用できるデジタル I/O 端子が異なります。SoftwareSerial の対応端子を使用してください

※HardwareSerial、SoftwareSerial ともに Arduino Due は対応していません

開発環境

開発環境は、Arduino IDE を用います。下記のアドレスの公式ウェブサイトからダウンロードしてご利用ください。

https://www.Arduino.cc/en/Main/Software

PC の対応等は上記公式ウェブサイトにてご確認ください。

本マニュアル作成時、弊社にて確認している Arduino IDE のバージョンは 1.8.10 です。（2020 年 2 月現在）

ICS 機器

ICS3.5 および 3.6 に対応した機器をご用意ください。

弊社 KRS シリーズのサーボモータ、無線コントローラ(KRC-5FH/KRR-5FH)が使用できます。

ICS とは

ICS とは「Interactive Communication System」の略で、近藤科学独自のデータ通信規格です。従来は PWM でサーボに角度の指令を送っていましたが、特性の変更や拡張などをより細かく制御するため、規格をシリアル通信に変更しました。

対応機器（2020 年 2 月現在）

●HV(9~12V)用サーボモータ

KRS-6003RHV ICS	KRS-6003R2HV ICS	KRS-6104HV		
KRS-4034HV ICS	KRS-4033HV ICS	KRS-4032HV ICS	KRS-4031HV ICS	
KRS-5054HV ICS H.C	KRS-5053HV ICS H.C	KRS-5034HV ICS	KRS-5033HV ICS	KRS-5032HV ICS
KRS-2552RHV ICS	KRS-2572HV ICS	KRS-2542HV ICS		

●LV(6.0～7.4V)用サーボモータ

KRS-3304 ICS	KRS-3204 ICS	KRS-3304R2 ICS	KRS-3301 ICS	KRS-3302 ICS

●無線コントローラ(6.0~12V)

KRR-5FH

ICS は、各デバイスに個別の ID 番号が割り振られていますので、デバイス同士をつないでデイジーチェーン（マルチドロップ）接続が可能です。デイジーチェーン接続で配線すれば、一つの端子から複数のデバイスをつなげることができますので、配線を最小限に多数のデバイスを制御することができます。また、各パラメータの設定変更やサーボの現在値の取得（ICS3.6 のみ）など多彩な機能が備わっています。ICS3.5 および 3.6 の仕様に関しては、ソフトウェアマニュアルをご覧ください。

サーボは瞬間的に大電流を消費することがありますので、KRS サーボを使う場合はデイジーチェーン 1 列に対して 8 個程度を目途に接続してください。それ以上接続する場合は、別途ハブ基板を用意し接続を根元で分岐してください。

接続する場合は、サーボの電源電圧に注意する必要があります。LV サーボ（6～7.4V 対応）と HV サーボ（9～12V 対応）では電源が共存できませんので注意してください。

Dual USB アダプターHS

ICS 機器と PC を接続するための USB アダプタです。モードを切り替えることで RCB-4（HV/mini）を PC に接続することも可能です。これとサーボマネージャを利用して ICS 機器の ID 番号や通信速度、各種パラメータを設定することができます。ライブラリでは、各サーボに設定された個別の ID 番号で対象機器を指定しますので、ICS 機器を使用する場合はこの USB アダプタをご用意ください。

『ICS3.5/3.6 Manager software R1.1.0.0』ダウンロードページ

電源

電源は、ご利用の ICS 機器に対応した電圧、電流容量を用意してください。サーボは瞬間的に大電流を消費することがありますので、充電式のバッテリを使用するか電源容量に余裕のある安定化電源、または AC アダプタをご用意ください。

弊社では、ロボット用にバッテリを販売しておりますので、そちらもご検討ください。また、各バッテリに対応した充電器もありますので、詳しくは弊社ウェブサイトにてご確認ください。

HV（9～12V）対応

バッテリ（リチウムフェライトタイプ）：ROBO パワーセル F3-850/1450/2100（Li-Fe）（3 セル/9.9V）

充電器：BX-20L

AC アダプター(12V5A)

※AC アダプター（12V5A）は、KRS-2552RHV を 17 個搭載した KHR-3HV が歩行できる程度の容量です。

ご利用するサーボの目安としてご参考にしてください。

LV(6V～7.4V)対応

バッテリ（リチウムフェライトタイプ）：ROBO パワーセル F2-850/1450（Li-Fe）（2 セル/6.6V）

充電器：BX-31LF/BX-20L（いずれもリチウムフェライト専用）

AC アダプター(6V2A)

※AC アダプター（6V2A）は、KRS-3301 ICS を 16 個搭載した KXR-L2 が歩行できる程度の容量です。

ご利用するサーボの目安としてご参考にしてください。

バッテリ（ニッケル水素タイプ）：ROBO パワーセル E タイプ 6N-800mAh(Ni-MH)

充電器：BX-32MH（ニッケル水素専用）

【ご注意ください】

弊社で取り扱っているバッテリは、Li-Fe(リチウムフェライト(リフェ)）タイプと Ni-MH(ニッケル水素)タイプの 2 種類あります。Li-Fe タイプは瞬間的な電源変動に強くなりますが、電源管理を徹底して行う必要があります。最悪の場合発煙、発火する可能性がありますので、取扱いに注意が必要です。Li-Fe タイプをご利用の場合は、以下のリンクをご一読ください。

■サポート情報『Li-Fe バッテリのメリットと注意事項』

LV サーボでそれほどパワーを使わない場合は、Ni-MH タイプのバッテリをお勧めします。

※バッテリは、種類のよって対応する充電器が異なります。必ず対応バッテリと充電器を組み合わせてご利用ください。

ICS Library for Arduino

ICS 機器を Arduino から動かすためのライブラリおよびサンプルプログラムです。

詳細は後述いたしますので、そちらを参照してください。

接 続 方 法

Arduino との接続方法

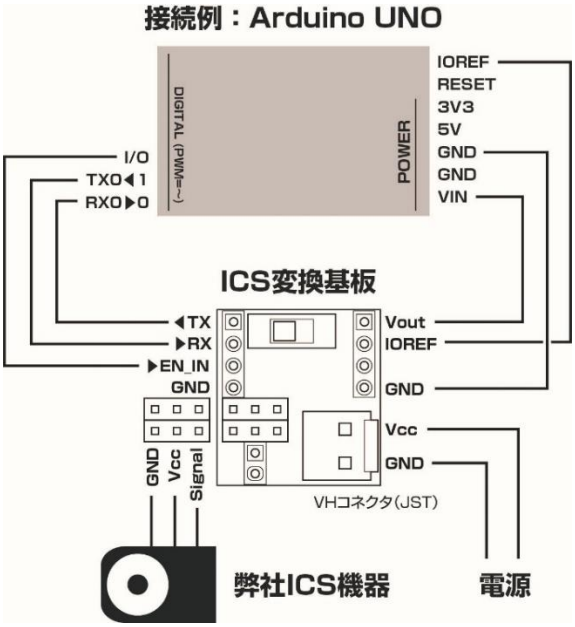
ICS 変換基板を Arduino に接続する場合、下図のように各端子を接続します。

Arduino Uno は、Serial(HardwareSerial)端子を使用して PC と通信し、プログラムの書き込みやデータの送受信を行います。また、ICS 機器も同様に Serial(HardwareSerial)端子を使用します。そのため、サーボと通信するために ICS 変換基板を Serial(HardwareSerial)端子に接続するとプログラムの書き込みの際に PC 間の情報が ICS 機器間の情報と混在してしまう場合があります。これを解決するために ICS 変換基板には、PC との通信、ICS 機器との通信を切り替えるためのスイッチが実装されていますので、状態により切り分けることができます。

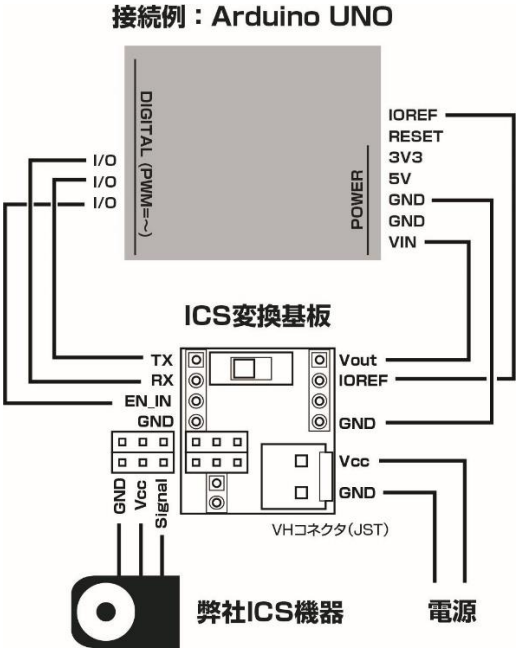
ただし、PC と ICS 機器を同時に通信することはできませんのでので、ICS 機器からの情報を PC に送信したり、切り替えの手間を省く場合は、ICS 機器との配線を Digital 端子に接続して SoftwareSerial をご利用ください。

接続に関しては、KSB シールド 2 を使用すると便利です。シールドの解説は、次項の「シールド基板を用いる方法」をご参照ください。

●配線図



HardwareSerial の場合



SoftwareSerial の場合

●MCU 通信用端子

情報を送受信するための端子を接続します。

【HardwareSerial の場合】

Arduino		ICS 変換基板(CN4)
Serial(UART)　RX	⇔	TX
Serial(UART)　TX	⇔	RX
Digital	⇔	EN_IN(送受信切替え端子)

【SoftwareSerial の場合】

Arduino		ICS 変換基板(CN4)
Digital	⇔	TX

Digital	⇔	RX
Digital	⇔	EN_IN(送受信切替え端子)

SoftwareSerial で使用できる Digital 端子は決まっています。Arduino の種類により端子が異なりますので確認してご利用ください。

サンプルプログラムでは、ICS 機器の RX を D8 ピン、TX を D9 ピンに接続しています。

また、送信と受信を切り替えるピン(EN_IN)は D2 ピンに接続されています。

以下の電源供給は、HardwareSerial、SoftwareSerial の配線は共通です。

●Artuino 電源供給

バッテリから Arduino へ電源を供給するための配線を行います。

Arduino		ICS 変換基板(CN3)
Vin	⇔	Vout
IOREF	⇔	IOREF
GND	⇔	GND

●ICS 機器接続端子　(CN5/CN6・2.54mm ピッチ・3pin)

端子	種類	説明
1pin	Signal	ICS 機器と通信を行ないます。
2pin	Vcc	ICS 機器に電源を供給します
3pin	GND	-

ICS 機器の接続端子、電源は下記の仕様になっています。接続には極性がありますので十分にご注意ください。

接続ケーブルに装着されたコネクタの爪がある方が Signal（1pin）です。

●電源端子　(CN2・VH コネクタ・2pin)

端子	種類	説明
1pin	Vcc	全体の電源を供給します。サーボの種類によって供給する電源が異なります。 ・HV サーボ：9～12V ・LV サーボ：6～7.4V
2pin	GND	

ICS 変換基板への接続には、サーボコネクタの接続ケーブルを使用してください。コネクタの爪があるほうが Signal（信号）線です。

※接続の向きを間違えるとデバイス破損の原因になります。接続する向きには十分ご注意ください。

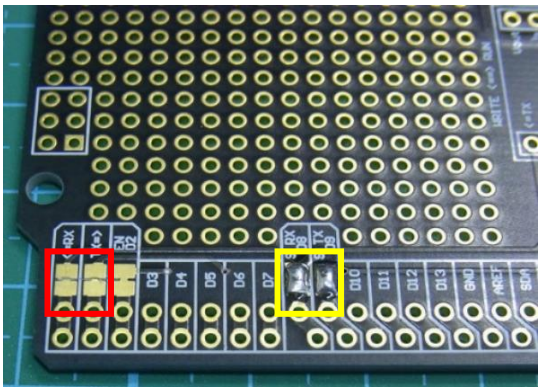
シールド基板を用いる方法

ユニバーサル基板などで上記図の配線を行うと手間がかかりますが、ICS 変換基板と Arduino を接続できる KSB シールド 2 を使用すれば、ヘッダピンを立てるだけですぐにご利用できます。シールド基板には、Arduino UNO に直接取り付けられるピン配置になっており、下記ピンが結線されています。

・Serial(もしくは Serial1)の TX、RX
・デジタル I/O　D8、D9（SoftwareSerial 用）（はんだ付けによるジャンパが必要です）
・EN_IN にデジタル 2 番ピン
・電源（Vout、IOREF、GND）の配線

シールド基板は裏と表があり、ロゴが入っている側が裏面になります。内側はユニバーサル基板を配置し、面実装部品が取り付けられるランドも用意しましたので必要に応じてお使いください。

HardwareSerial と SoftwareSerial は同じシールド上で使用できますが、使用する端子が異なりますのではんだ付けによるジャンパにて接続先を切り替える必要があります。工場出荷時は HardwareSerial に接続されていますので、SoftwareSerial で使用する場合は、以下のようにジャンパを切り替えてください。



KSB シールド 2 を SoftwareSerial で使用するための準備

画像のように、赤枠の RX、TX 端子のパターンの細い部分をカッターなどでカットし、黄色枠の S_RX(D8)、S_TX(D9)端子のパターンをはんだで接続します。EN(D2)はカットしませんのでご注意ください。

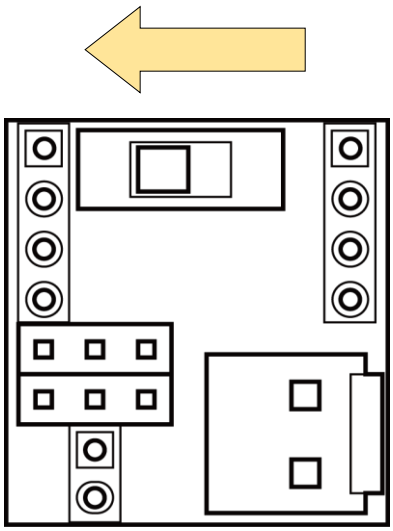
プログラムの書き込み方法

Arduino UNO など一部のボードは UART が 1 系統しかなく、USB 通信と兼用になっていますので HardwareSerial で ICS 機器を接続した場合に情報が混在して通信がうまくいかなくなる可能性があります。そこで、ICS 変換基板には、PC との通信時と ICS 機器との通信時で回路を切り替えできるようスイッチを付加しました。

SoftwareSerial の場合は、スイッチを切り替える必要がありません。ICS 機器とつながった状態の「実行モード」でご利用ください。また、Arduino Mega など UART 機能が複数あり、書き込みに使用しない場合も、スイッチは「実行モード」のままで構いません。

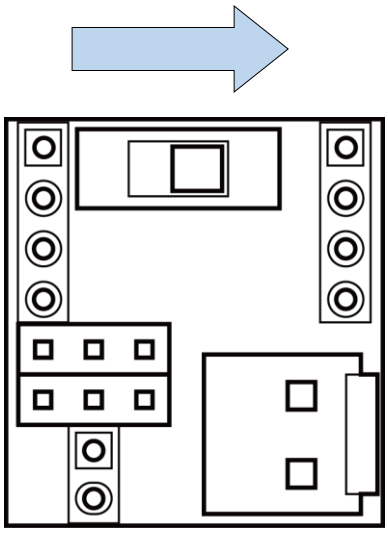
【プログラム書込モード】

プログラム書き込み時（PC との通信時）は『書込』側にスイッチを切り替えてください。



【プログラム実行モード】

プログラムを書きこんだ後、実行する時（ICS 機器との通信時）は逆方向『実行』側にスイッチを切り替えてください。



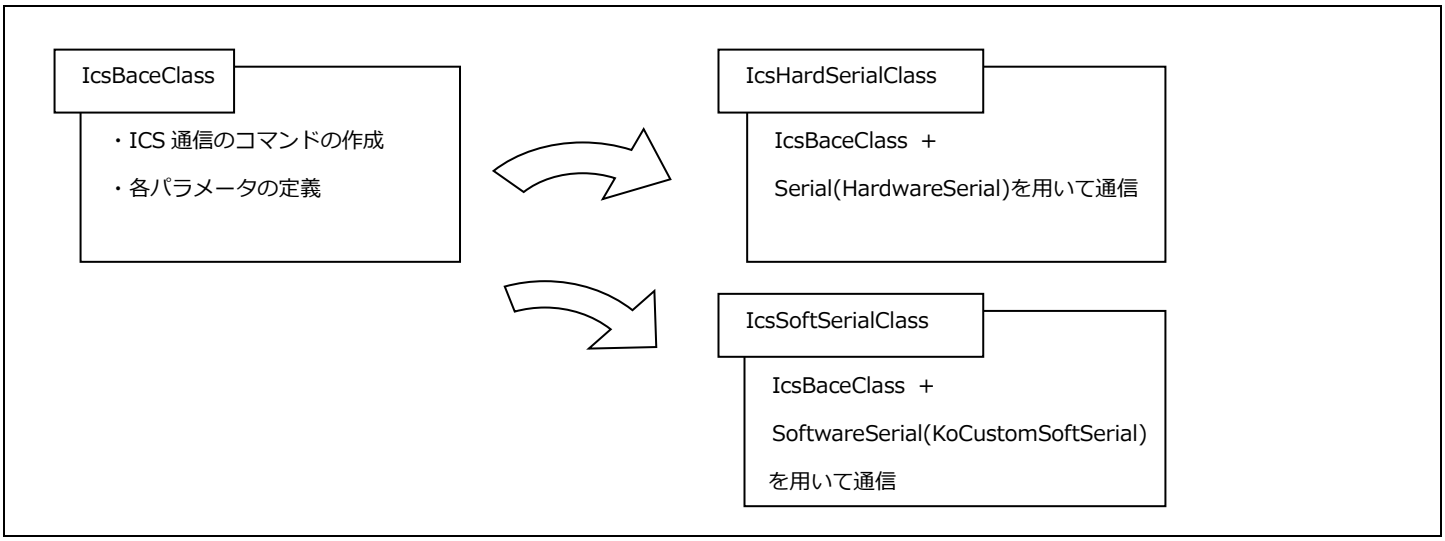
スイッチを実行モードにしてプログラムを書き込みますと、プログラムが正常に書けないためエラーになります。また、スイッチを書込モードのままプログラムを実行した場合、ICS 機器からの通信が返ってこないためエラーになり、意図しない動きになりますので注意が必要です。

ICS Library for Arduino について

ライブラリの概要

ICS 機器を Arduino から簡単に制御できるようにライブラリを用意しました。KRS サーボを ID 番号で指定し、ポジション（角度情報）を送信することで簡単に角度制御を行うことができます。また、プログラム中にサーボのスピードやストレッチを変更することも可能です。

ICS 機器を使用するための関数は、「IcsBaseClass」をベースのクラスにし HardwareSerial を使用するための「IcsHardSerialClass」と SoftwareSerial を使用するための「IcsSoftSerialClass」が用意されています。



IcsSoftSerialClass は、Arduino 標準の SoftwareSerial クラスを使用よう検証しましたが、通信速度が足りず通信ができなかったため別途「KoCustomSoftSerial」を作成しました。IcsSoftSerialClass を使用する場合は、「KoCustomSoftSerial」もインポートしてください。

ライブラリの入手方法

弊社ウェブサイトよりダウンロード可能です。

http://kondo-robot.com/faq/ics-library-a2　（圧縮ファイルですべてを 1 つにまとめています）

ライブラリ圧縮ファイルの中身

圧縮ファイルの中には、Arduino でライブラリをインポートしやすいように ZIP で圧縮したライブラリがあります。その中に Arduino 用のサンプルプログラムも用意しています。

```
ICS_Library_for_Arduino_V2_1/
        ├IcsClassV210.zip                (IcsBaceClass,IcsHardSerialClass)
        ├IcsSoftSerialClassV210.zip      (IcsSoftSerialClass)
        ├KoCustomSoftSerial.zip          (KoCustomSoftSerial)
        └取扱説明書など
```

KoCustomSoftSerial は、Rcb4SoftSerialClass と共通で使用します。すでにインポート済みの場合は、再度インポートする必要はありません。

IcsBaseClass は IcsHardSerialClassV200.zip の中に入っています。

IcsSoftSerialClassV200 の中には入っていませんので、SoftwareSerial を使用する場合は IcsHardSerialClassV200 もインポートしてください。

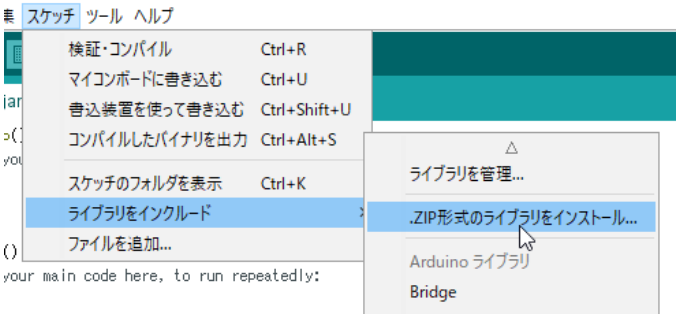
ライブラリのインポート

※インポート方法は Arduino　IDE のバージョン 1.8.3 を例にします。

①　弊社ウェブサイトよりライブラリ一式をダウンロードし、解凍します。生成されたフォルダ内の zip ファイルは解凍しないでください。

②　Arduino IDE を起動します

③　メニューバーの「スケッチ」　→　「ライブラリをインクルード」
→「 .zip 形式のライブラリをインストール…」 を選択します



④『インストールするライブラリを含む ZIP ファイルまたはフォルダを指定してください』となり、ファイル指定ダイアログが表示されますので、ダウンロードした zip ファイルの場所を指定します。

・HardwareSerial の場合：IcsClassVxxx.zip

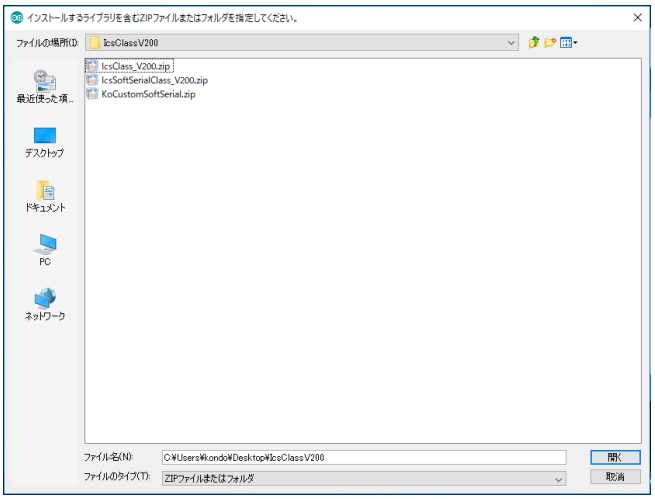
・SoftwareSerial の場合：IcsSoftSerialClassVxxx.zip

KoCustomSoftSerialVxxx.zip

(SoftwareSerial で使用する場合でも

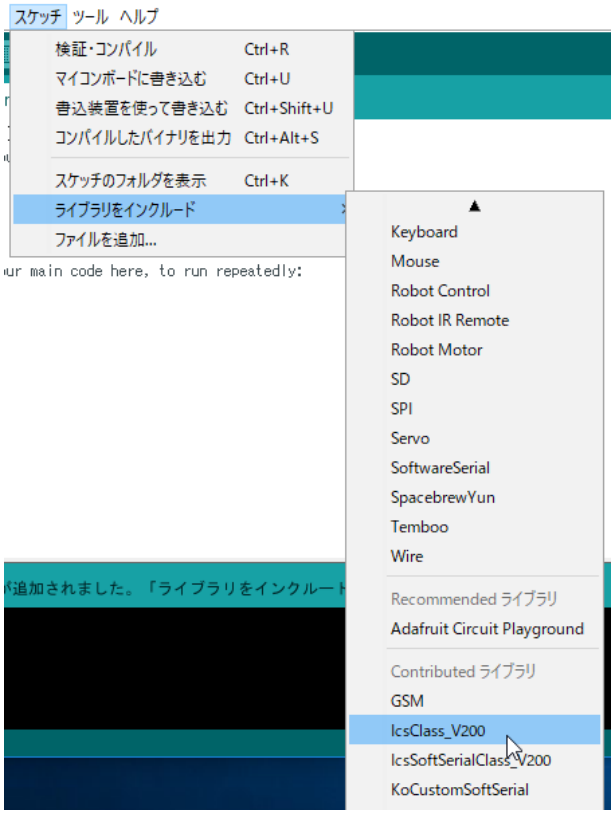
IcsClassVxxx.zip もインストールしてください)

(xxx はバージョンにより異なります)



⑤　エラーでなければ、ライブラリがインポートされます。手順③で表示した「ライブラリをインクルード」リスト内に「IcsClass_Vxxx」または、「IcsSoftSerialClassVxxx」の表示がありましたら成功です。

※右の画像は IcsClass のバージョン 200 をインポートした場合です



ライブラリの保存場所

インポートされたライブラリのフォルダは、デフォルト設定では、「ドキュメント¥Arduino¥libraries」にデータが置かれます。詳細に関してはArduinoIDE のマニュアルをご参照ください。

ライブラリの削除

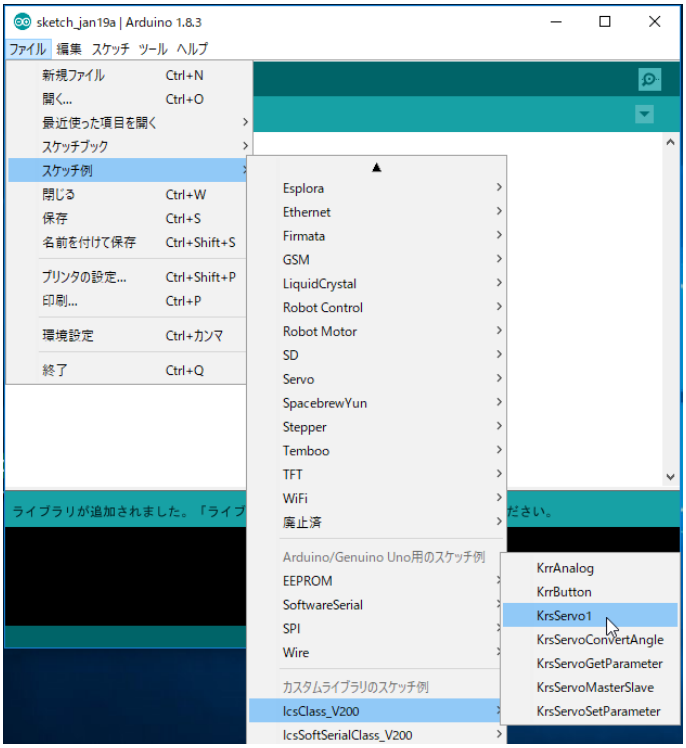
インポートしたライブラリを消したい場合は、「ドキュメント¥Arduino¥libraries」のフォルダを消してください。なお、フォルダを消した場合はArduino IDE の再起動が必要です。

サンプルプログラムについて

サンプルプログラムの呼び出し

ライブラリをインポートすると同時にサンプルプログラムがインポートされます。

呼び出し方法は、「ファイル」→「スケッチの例」→「IcsClass_Vxxx」または「IcsSoftSerialClassVxxx」でサンプルを呼び出します。



サンプルプログラムの種類

サンプルプログラムは下記 8 種類あります。HardwareSerial と SoftwareSerial で機能は共通ですが、宣言方法など異なる箇所がありますのでご注意ください。

サンプルプログラム名	ICS 接続機器	概要
KrsServo1	サーボ x1(ID:0)	サーボを一定間隔で動かします。
KrsServoConvertAngle	サーボ x1(ID:0)	サーボの角度(degree)を指定して動かします。
KrsServoMasterSlave	サーボ x2(ID:0 ,1)	ID:1 のサーボのポジションデータを取得し ID:0 に送ります。
KrsServoGetParameter	サーボ x1(ID:0)	電流値、温度値、ポジションデータ(ICS3.6 のみ)を取得します。
KrsServoSetParameter	サーボ x1(ID:0)	ストレッチ、スピードの設定をします。
KrrButton	サーボ x1(ID:0) KRR-5FH	KRR が受信したボタンデータをもとにサーボを動かします。
KrrAnalog	サーボ x1(ID:0) KRR-5FH	KRR が受信したアナログデータをもとにサーボを動かします。
KrsServoCheckID	サーボ x1	接続されたサーボモータの ID を変更します (V210 から追加)

※　サンプルプログラムを動かすためにはそれに対応した機器をご用意ください。

※　ICS 機器の通信速度は 115200bps に設定します。Aruduino のシリアル端子の通信速度に上限があるため 1250000bps には設定できません。

※　受信機 KRR-5FH のサンプルプログラムを動作するためには、ペアリングしてある送信機 KRC-5FH が必要です。また、アナログデータを KRR-5FH から取得する場合、KRC-5FH にアナログ出力デバイスが接続されている必要があります。

KrsServo1 の詳細

基本となる KrsServo1 を例に説明します。このサンプルは、サーボを一定間隔で左右に動かします。先に HardwareSerial を使用したプログラムを解説し、後に SoftwareSerial での違いを解説します。

ソースコード

```
#include <IcsHardSerialClass.h>

const byte EN_PIN = 2;
const long BAUDRATE = 115200;
const int TIMEOUT = 1000;                    //通信できてないか確認用にわざと遅めに設定
IcsHardSerialClass krs(&Serial,EN_PIN,BAUDRATE,TIMEOUT);   //インスタンス＋EN ピン(2 番ピン)および UART の指定

void setup() {
  krs.begin();   //サーボの通信初期設定
}

void loop() {
    krs.setPos(0,7500);       //位置指令　ID:0 サーボを 7500 へ 中央
    delay(500);                //0.5 秒待つ
    krs.setPos(0,9500);       //位置指令　ID:0 サーボを 9500 へ 右
    delay(500);                //0.5 秒待つ
    krs.setPos(0,7500);       //位置指令　ID:0 サーボを 7500 へ 中央
    delay(500);                //0.5 秒待つ
    krs.setPos(0,5500);       //位置指令　ID:0 サーボを 5500 へ 左
    delay(500);
}
```

ソースコード解説

ここでは HardwareSerial のソースコードの詳細を解説します。SoftwareSerial については後述の『SoftwareSerial を使用する場合」をご参照ください。Arduino のコードの詳細に関しては、ArduinoIDE のマニュアルを参照してください。

```
#include <IcsHardSerialClass.h>
```

HardwareSerial で通信を行う IcsHardSerialClass を使えるようにします。

```
const int EN_PIN = 2;
const long BAUDRATE = 115200;
const int TIMEOUT = 1000;
IcsHardSerialClass krs(&Serial,EN_PIN,BAUDRATE,TIMEOUT);   //インスタンス＋EN ピン(2 番ピン)および UART の指定
```

IcsClass クラスを krs という名前で宣言し初期化します。

●シリアルポートの設定 => Serial

UNO を使用する場合は UART が一つしかありませんので"Serial"を記述してください。Serial の前に必ず「＆」が必要です。

UART が複数ある場合は後述します。

●送受信切り替えピン(EN_IN) =>デジタル 2 番ピン

ICS は 1 つの通信線で送信受信を切り替えて通信を行います。その切替えに 1 つデジタルピンを使用します。

サンプルプログラムでは、デジタル 2 番ピンを使用していますが、他のデジタルピンも使用することができます。

●通信速度(baudrate) => 115200bps　※1250000bps には設定できません。

●読み込みのタイムアウト(timeout) => 1000ms

ICS 機器から返信データが返ってくるまでの待ち時間を設定します。設定した時間を過ぎると次の処理に移ります。

この箇所は、変数を宣言せず直接記述することもできます。

```
IcsHardSerialClass krs(&Serial,2,115200,1000);
```

```
krs.begin();   //サーボの通信初期設定
```

宣言した krs を初期化し、通信を開始します。

ID0 のサーボを動かすためには以下のように記述します。

```
krs.setPos(0,7500);        //位置指令　ID:0 サーボを 7500 へ (中央)
krs.setPos(0,9500);        //位置指令　ID:0 サーボを 9500 へ (右)
krs.setPos(0,5500);        //位置指令　ID:0 サーボを 5500 へ (左)
```

krs で設定したポートに接続されているサーボにポジションを送り、動作させます。

また、ID:3 のサーボを 3500(-135 度)に動作させたい場合は、

```
krs.setPos(3,3500);        //位置指令　ID:3 サーボを 3500 へ (左端)
```

と記述します。

角度をポジションに変換する関数 degPos()も用意していますので、必要な場合は「KrsServoConvertAngle」のサンプルプログラムをご覧ください。

SoftwareSerial を使用する場合

```
#include <IcsSoftSerialClass.h>

const byte S_RX_PIN = 8;
const byte S_TX_PIN = 9;

const byte EN_PIN = 2;
const long BAUDRATE = 115200;
const int TIMEOUT = 200;     //softSerial は通信失敗する可能性があるため短めに
IcsSoftSerialClass krs(S_RX_PIN,S_TX_PIN,EN_PIN,BAUDRATE,TIMEOUT);   // IcsSoftSerialClass の定義、softSerial 版
```

SoftwareSerial を使用する場合は、宣言する箇所が異なります。setPos()などの各関数は同じ使い方ができます。

```
#include <IcsSoftSerialClass.h>
```

インクルードファイルの名称が異なります。

```
const byte S_RX_PIN = 8;
const byte S_TX_PIN = 9;
```

SoftwareSerial で使用する端子を宣言します。8、9 はそれぞれ Arduino のデジタル I/O ピンである D8 と D9 です。Arduino によっては RX ピンが制限されますので、Arduino のマニュアルでご確認ください。

```
const byte EN_PIN = 2;
const long BAUDRATE = 115200;
const int TIMEOUT = 200;     //softSerial は通信失敗する可能性があるため短めに
```

この箇所は、HardwareSerial と共通です。

```
IcsSoftSerialClass krs(S_RX_PIN,S_TX_PIN,EN_PIN,BAUDRATE,TIMEOUT);   // IcsSoftSerialClass の定義、softSerial 版
```

IcsSoftSerialClass クラスを krs という名前で宣言し初期化します。初期化には、この一文の前で各変数に代入した値を使用しています。

複数の UART を用いる場合

サンプルでは、IcsHardSerialClass クラスを krs いう名前で宣言をしていましたが、krs_1 や krs_2 などの任意名前でも宣言できます。

```
IcsHardSerialClass krs_1(&Serial1,EN_PIN,BAUDRATE,TIMEOUT);   //Serial1 およびデジタル 2 番ピンを使用する
IcsHardSerialClass krs_2(&Serial2,EN_PIN,BAUDRATE,TIMEOUT);   //Serial2 およびデジタル 3 番ピンを使用する
```

名前が同じでなければ、同時に複数の IcsHardSerialClass を宣言することができます。

UART ごとに IcsHardSerialClass を宣言し Serial ポートを変えることで複数の UART を使うことができますので、接続する ICS 機器の数を増やすことも可能です。ただし、各 IcsHardSerialClass の Serial ポートや EN_PIN が競合しないように注意してください。

また、複数のシリアル端子を使い分けて Rcb4Class と共存することも可能です。

上記の内容は、SoftwareSerial でも同様です。ただし、SoftwareSerial は接続する機器が増えることで通信量が多くなると動作が不安定になる場合がありますのでご注意ください。

宣言した場合は通信等の初期設定をそれぞれ行う必要があります。

```
Krs_1.begin();   // krs_1 で宣言した IcsHardSerialClass の通信初期設定
krs_2.begin();   // krs_2 で宣言した IcsHardSerialClass の通信初期設定
```

各 ICS 機器を動かす場合は下記のように記述できます。

```
Krs_1.setPos(3,9500);        //位置指令   krs_1 で設定した Serial(Serial1)を使って KRS サーボ ID3 に命令
Krs_2.setPos(5,6500);        //位置指令   krs_2 で設定した Serial(Serial2)を使って KRS サーボ ID5 に命令
```

ICS Library for Arduino の詳細

ICS Library for Arduino の内容は、下記クラス、定義で構成されています。

クラス/定義名	内容
IcsBaseClass	ICS のコマンド生成や定義をまとめたクラス
IcsHardSerialClass	IcsBaseClass から派生し、Serial(HardwareSerial)を用いて ICS 機器と通信をする
IcsSoftSerialClass	IcsBaseClass から派生し、SoftwareSerial(KoCustomSoftSerial)を用いて ICS 機器と通信をする
KoCustomSoftSerial	115200bps で通信できるように Arduino の SoftSerial を改造
KRR_BUTTON	KRR のボタンデータを定義しています

下記からは、ICS 機器を動かすために必要な関数を列挙しています。記載されてない関数もありますので、詳細に関しては解凍フォルダ内の html 版_Vxxx もしくは FunctionList Vxxx(フォルダの中の index.html をご覧ください。（xxx はバージョンにより異なります。）

説明文の例のところで krs.xxxx と出てきますが、krs という名前で IcsHardSerialClass/ IcsSoftSerialClass を宣言しています。

ICS 機器と通信するための最初設定

ICS クラスを宣言する(HardwareSerial 版)　IcsHardSerialClass ()

【書式】

IcsHardSerialClass (HardwareSerial *icsSerial, int enpin)

IcsHardSerialClass (HardwareSerial *icsSerial, int enpin, long baudrate, int timeout)

【概要】

コンストラクタ

クラスを宣言(インスタンス)、初期化するための関数

【引数】

種類	名称	説明	設定範囲
in	*icsSerial	ICS に設定する UART(HardwareSerial 型のポインタ)	UNO：Serial Mega：Serial1 ～　Serial3
in	enpin	送受信切替えピンのピン番号	空いている任意のデジタルピン
in	baudrate	ICS の通信速度	115200bps のみ　※
in	timeout	受信タイムアウト(ms)	～32767ms

※Arduino の種類のよっては,625000,1250000(1.25M)bps を使用できる場合があります。

【使用例】

```
IcsHardSerialClass krs(&Serial,2,115200,5);   //Serial およびデジタル 2 ピンを使用して 115200bps 通信設定、通信待ち時間 5ms
```

【説明】

HardwareSerial を使用して ICS 機器と通信を行う「IcsHardSerialClass」を宣言する関数です。C 言語とは違い宣言をすると同時に初期設定もできます。通信を開始する場合は、別途通信開始用の関数(begin())を使います。

HardwareSerial 型は、Arduino 内で使用している UART のクラスになります。Arduino の環境の中で、UART1 を HardwareSerial 型で Serial と宣言されています。HardwareSerial では宣言された Serial(Mega の場合は Serial1 や Serial2)を直接使いますので、初期設定ではそのポインタを渡します。

enpin は、ICS 通信の送信と受信を切り替える EN_IN ピンを設定します。シールド基板ではデジタル 2 番ピンを固定として使用しますが、他の空いているデジタルピンを使用することも可能です。

baudrate は、ICS 機器と通信する通信速度を設定できます。ICS 機器は、最大で 1.25Mbps まで通信することが可能ですが、Arduino の通信速度は 1.25Mbps まで対応していないため、通常は 115200bps を使用します。

timeout は返信データを待つ時間です。ICS 機器が正常に接続されている場合はデータが返ってきますが、接続されていなかったりした場合は待ち続けることになりますので、それを打ち切るための時間です。

ICS クラスを宣言する(SoftwareSerial 版)　IcsSoftSerialClass()

【書式】

IcsSoftSerialClass(byte rxPin,byte txPin,byte enpin)

IcsSoftSerialClass(byte rxPin,byte txPin,byte enpin, long baudrate, int timeout)

IcsSoftSerialClass(KoCustomSoftSerial *koSoftSerial,byte enpin)

IcsSoftSerialClass(KoCustomSoftSerial *koSoftSerial,byte enpin, long baudrate, int timeout)

【概要】

コンストラクタ

クラスを宣言(インスタンス)、初期化するための関数

【引数】

種類	名称	説明	設定範囲
in	rxPin	データを受信するピン番号	Arduino により使えるピンが違うので注意
in	txPin	データを送信するピン番号	空いている任意のデジタルピン
in	enpin	送受信切替えピンのピン番号	空いている任意のデジタルピン
in	baudrate	ICS の通信速度	115200bps のみ　※
in	timeout	受信タイムアウト(ms)	～32767ms
in	* koSoftSerial	ICS 通信に設定する Soft(KoCustomSoftSerial 型のポインタ)	UNO：Serial Mega：Serial1 ～　Serial3

※SoftSerial は 115200bps のみとなります。

【使用例】

```
IcsSoftSerialClass krs(8,9,2,115200,5);
//インスタンス+RX ピン(8 番ピン)、TX ピン(9 番ピン)、EN ピン(2 番ピン)と通信速度(115200bps)、タイムアウト(5ms)を設定
```

【説明】

SoftSerial を使用して ICS 機器と通信を行う「IcsSoftSerialClass」を宣言する関数です。C 言語とは違い宣言をすると同時に初期設定もできます。

Arduino 純正の SoftSerial を使用すると高速で通信ができないため SoftSerial を改造した KoCustomSoftSerial を用意しました。

通信を開始する場合は、別途通信開始用の関数(begin())を使います。

KoCustomSoftSerial は、SoftSerial とほぼ同じように使えます。rxPin、txPin でピンを指定すれば使うことができますが、Arduino によっては RX ピンに使えないピンがありますので、Arduino の SoftSerial を確認してください。

enpin、baudrate、timeout に関しては、IcsHardSerialClass と同じです。

ICS クラスを開放する　~ IcsHardSerialClass ()/~IcsSoftSerialClass()

【書式】　なし

【概要】

デストラクタ

宣言しているクラスを解放します。

関数が終わり、使わない場合は自動で呼ばれます。

通信を開始する　begin()

【書式】

bool begin ()

bool begin(long baudrate,int timeout)

bool begin (HardwareSerial *serial, int enpin, long baudrate, int timeout　long baudrate, int timeout)　//(IcsHardSerialClass)

bool begin(KoCustomSoftSerial *serial,byte enpin,long baudrate,int timeout)　//(IcsSoftSerialClass)

【概要】

通信の初期設定、通信を開始します。

【引数】

種類	名称	説明	設定範囲
in	*serial	ICS に設定する UART のポインタ (HardwareSerial もしくは KoCustomSoftSerial のポインタ)	

in	enpin	送受信切替えピンのピン番号	空いている任意のデジタルピン
in	baudrate	ICS の通信速度	115200bps のみ　※
in	timeout	受信タイムアウト(ms)	～32767ms

【戻り値】

値	説明
true	通信設定完了
false	通信設定失敗

【使用例】

```
krs.begin();
```

【説明】

通信の初期設定、通信を開始します。

この関数を記述した後から Arduino は ICS 機器との通信を開始します。各関数を記述する前に必ず宣言してください。

通常は引数を持たない関数を使用しますが、初期設定をしなおしたい場合は引数有りの関数を使用してください。

コマンドを使って送受信する　synchronize ()

【書式】

bool synchronize (byte *txBuf, byte txLen, byte *rxBuf, byte rxLen)

【概要】

ICS 通信の送信、受信を行います。

【引数】

種類	名称	説明
in,out	*txBuf	送信データ格納バッファ
in	txLen	送信データ数
out	*rxBuf	受信データ格納バッファ
in	rxLen	受信データ数

【戻り値】

値	説明
true	通信成功
false	通信失敗

【注意】

送信データ数、受信データ数はコマンドによって異なりますので注意してください。

【使用例】

```
byte   txCmd[3] = {0x80,0x3A,0x4C};   //ID0 に 7500 を送る直接のコマンドの配列
byte   rxCmd[3];                      //受信のデータ配列
krs. synchronize (txCmd, sizeof (txCmd), rxCmd, sizeof (rxCmd));
```

【説明】

各 ICS 機器に直接コマンドを送りたい場合はこの関数を使用します。この関数は、以下で説明する setPos()など各関数内で使用しています。

ICS3.5/3.6 のソフトウェアマニュアルを参照しながら自分でコマンドを送受信したい場合にご利用ください。

固定パラメーター覧

【概要】

サーボに使われる値の最大値や通信エラー時の値を定義しています。

サーボパラメータの範囲判定や通信エラーの判定時にお使いください。

【固定値】

名称、型	値	説明
static const int MAX_ID	31	サーボ ID の最大値
static const int MIN_ID	0	サーボ ID の最小値
static const int MAX_POS	11500	サーボのポジション最大値
static const int MIN_POS	3500	サーボのポジション最小値
static const int ICS_FALSE	-1	ICS 通信等々で失敗したときの値

【使用例】

```
krs.setPos(0, IcsHardSerialClass::MAX_POS);        //位置指令　ID:0 サーボを最大値(MAX_POS)まで動かす
krs.setPos(0, krs.MAX_POS);                //↑と同じ。krs が宣言されていると左記述のようにも使える
```

サーボモータを動かす　setPos ()

【書式】

int setPos (byte id, unsigned int pos)

【概要】

ポジションデータでサーボの角度を指定し動かします。

【引数】

種類	名称	説明	設定範囲
in	id	サーボの ID 番号	0～31　※
in	pos	ポジションデータ	3500　～　7500（センター）　～　11500

※KRR-5FH は ID31 に固定されているため、受信機が接続されている場合サーボは ID31 を使用できません。以下の関数でも同様です。

【戻り値】

種類	値	説明
成功の場合	3500～115000	ポジションデータが返ってきます
失敗の場合	-1	範囲外、通信失敗（**ICS_FALSE**）

【使用例】

```
krs.setPos(3,3500);        //位置指令　ID:3 サーボを 3500 へ (左端)

//ポジションコマンドで現在値を取得する場合
int pos;
pos =   krs.setPos(0,7500);        //位置指令　ID:0 サーボを 7500 へ (中央)
if(pos == HardwareSerial::ICS_FALSE)        //通信が失敗したかの確認
{
  //失敗の処理
}
```

【説明】

ID で指定したサーボを任意の角度に動作させる関数です。角度は、ポジションデータ 3500～11500 の数値で指定します。7500 を指定するとニュートラル位置（センター）に移動します。通信が成功した場合は、戻り値としてサーボの現在値が返ってきますが、失敗した場合は-1(**ICS_FALSE**)が返ってきます。これを利用して通信が失敗した場合の処理もプログラムすることが可能です。

サーボをフリーにする　setFree ()

【書式】

int setFree (byte id)

【概要】

サーボをフリー(脱力)状態にします。

【引数】

種類	名称	説明	設定範囲
In	id	サーボの ID 番号	0～31

【戻り値】

種類	値	説明
成功の場合	3500～115000	ポジションデータ
失敗の場合	-1	範囲外、通信失敗（**ICS_FALSE**）

【使用例】

```
krs.setFree(3);        //フリー指令　ID:3 をフリー状態に
```

【説明】

ID で指定したサーボをフリー（脱力）状態にするための関数です。setPos()と同じく通信が成功するとサーボの現在地が、失敗すると-1 が戻り値として返ってきます。

再びホールド（力が入った）状態に戻すには setPos()を使用してください。setFree()で現在地を取得し、その値を setPos()で指定すると、ロボットの各関節を現状の角度ままホールド状態に戻すことができます。

ストレッチ書き込み　setStrc ()

【書式】

int setStrc (byte id, unsigned int strc)

【概要】

サーボのストレッチ（保持力）の値を書き込みます。

【引数】

種類	名称	説明	設定範囲
in	id	サーボの ID 番号	0～31
in	strc	ストレッチのデータ	（Soft）1～127（Hard）

【戻り値】

種類	値	説明
成功の場合	1～127	書き込んだストレッチのデータ
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
krs.setStrc(0,90);   //ID0 のストレッチを 90 にします
```

【説明】

ID で指定したサーボのストレッチの値を書き込むための関数です。ストレッチとは、軸の保持力を指します。1 に近づけるごとに保持力が弱くなり、127 に近づけるごとに強くなります。ストレッチを程よく下げることで軸の動作が滑らかになったり、ハンチングが起こり始めた時に下げると効果的です。また、普段は 60 程度に設定しておいて踏ん張りが必要なタイミングで 127 に上げるような使い方もできます。

スピード書き込み　setSpd ()

【書式】

int setSpd (byte id, unsigned int spd)

【概要】

サーボのスピード（出力）を変更します。

【引数】

種類	名称	説明	設定範囲
in	id	サーボの ID 番号	0～31
in	spd	スピードのデータ	（Slow）1 ～ 127 (Fast)

【戻り値】

種類	値	説明
成功の場合	1～127	書き込んだスピードのデータ
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
krs.setSpd(0,90);  //ID0 のスピードを 90 にします
```

【説明】

ID で指定したサーボのスピードの値を書き換えるための関数です。1 に近づけるごとに遅くなり、127 に近づけるごとに早くなります。

電流値書き込み　setCur ()

【書式】

int setCur (byte id, unsigned int curlim)

【概要】

サーボの電流リミット（電流制限）値を変更します。

【引数】

種類	名称	説明	設定範囲
in	id	サーボの ID 番号	0～31
in	curlim	電流値のデータ	(Low)1～63(High)

【戻り値】

種類	値	説明
成功の場合	1～63	サーボの電流リミット値
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
krs.setCur(0,20);　//ID0 の電流リミットを 20 にします
```

【説明】

ID で指定したサーボの電流制限値を書き換えるための関数です。サーボがロックした状態になると電流値が上昇しますが、サーボ内で検出した電流値が設定した閾値を超えるとサーボが脱力状態になります。閾値を下回れば復帰します。1 の時は 0.1A、20 の時は 2.0A の閾値になります。出荷状態では 40 に設定されています。

温度値書き込み　setTmp ()

【書式】

int setTmp (byte id, unsigned int tmplim)

【概要】

サーボの温度リミット（温度制限）値を変更します。

【引数】

種類	名称	説明	設定範囲
in	id	サーボの ID 番号	0～31
in	tmplim	温度値のデータ	(High)1～127(Low)

【戻り値】

種類	値	説明
成功の場合	1～127	サーボの温度リミット値
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
krs.setTmp(0,20);　//ID0 の温度リミットを 20 にします
```

【説明】

ID で指定したサーボの温度制限値を書き換えるための関数です。サーボを使用し続けて温度が上昇し、サーボ内で検出した温度が設定した閾値を超えるとサーボが脱力状態になります。閾値を下回れば復帰します。1 に近づけるごとに高い温度での設定値になり、127 に近づけるごとに低い温度での設定値になります。30 で 100℃、75 で 70℃の設定です。初期設定では 75 に設定されて言います。

ストレッチ読み込み　getStrc ()

【書式】

int getStrc (byte id)

【概要】

現在のサーボのストレッチ（保持力）の値を読み込みます。

【引数】

種類	名称	説明	設定範囲
in	id	サーボの ID 番号	0～31

【戻り値】

種類	値	説明
成功の場合	1～127	読み込んだストレッチのデータ　（Soft）1～127（Hard）
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
int st;
st = krs.getStrc(0);   //ID0 のストレッチを読み取ります
```

【説明】

ID で指定したサーボのストレッチの値を読み込むための関数です。

スピード読み込み　getSpd ()

【書式】

int getSpd (byte id)

【概要】

現在のサーボのスピード（出力）を読み込みます。

【引数】

種類	名称	説明	設定範囲
in	id	サーボの ID 番号	0～31

【戻り値】

種類	値	説明
成功の場合	1～127	読み込んだスピードのデータ　（Slow）1 ～ 127 (Fast)
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
int sp;
sp= krs.getSpd(0);   //ID0 のスピードを読み取ります
```

【説明】

ID で指定したサーボのスピードの値を読み込むための関数です。

現在の電流値を読み込む　getCur ()

【書式】

int getCur (byte id)

【概要】

サーボの現在の電流値を読み込みます。

【引数】

種類	名称	説明	設定範囲
in	id	サーボの ID 番号	0～31

【戻り値】

種類	値	説明
成功の場合	1～127	サーボの現在の電流値　(CW)1～63、(CCW)64～127
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
int cur;
cur = krs.getCur(0);   //ID0 の電流値を読み取ります
```

【説明】

ID で指定したサーボの現在の電流値を読み込むための関数です。

※電流制限値ではありません。

現在の温度値を読み込む　getTmp ()

【書式】

int getTmp (byte id)

【概要】

サーボの現在の温度値を読み込みます。

【引数】

種類	名称	説明	設定範囲
in	id	サーボの ID 番号	0～31

【戻り値】

種類	値	説明
成功の場合	1～127	サーボの現在の温度値　(High)1～127(Low)
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
int tmp;
tmp = krs.getTmp(0);   //ID0 の温度値を読み取ります
```

【説明】

ID で指定したサーボの現在の温度値を読み込むための関数です。

※温度制限値ではありません。

現在のポジションデータを読み込む　getPos ()

【書式】

int getPos (byte id)

【概要】

サーボの現在のポジションデータを読み込みます。

※ICS3.6 から有効です。ICS3.5 のサーボに送った場合は返事を返しません 。

【引数】

種類	名称	説明	設定範囲
in	id	サーボの ID 番号	0～31

【戻り値】

種類	値	説明
成功の場合	3500 ～ 11500	指定した ID の現在のポジションデータ
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
int pos
pos = krs.getPos(0);   //ID0 の現在のポジションデータを読み取ります
if(pos == IcsHardSerialClass:: ICS_FALSE)          //通信が失敗したかの確認
{
   //失敗の処理
}
```

【説明】

ID で指定したサーボの角度を読み込むための関数です。

ID の管理

接続されている機器の ID を読み込む　getID ()

【書式】

int getID ()

【概要】

接続されているサーボモータの ID を取得します

V210 から追加

【引数】

なし

【戻り値】

種類	値	説明
成功の場合	0　～　31	接続された機器の ID 番号
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
int id
id = krs.getID();　//接続されている機器の ID を読み取ります
```

【説明】

接続された機器の ID を取得するための関数です。

１対１で通信を行ってください。複数接続されているとすべての機器から ID が返ってくるため意図しない値になります。

接続されている機器の ID を書き込む　setID ()

【書式】

int setID (byte id)

【概要】

接続されている機器の ID を変更します。

V210 から追加

【引数】

種類	名称	説明	設定範囲
in	id	書き込む ID 番号	0～31

【戻り値】

種類	値	説明

成功の場合	0～31	変更された機器の ID 番号
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
krs.setID(1);　//接続機器の ID を 1 番に変更
```

【説明】

接続された機器の ID を変更するための関数です。

１対１で通信を行ってください。複数の機器を接続されていると意図しない値が返ってきて ID が正常に変わらない可能性があります。

ポジションデータと角度(degree)を変換する

角度(degree)をサーボのポジションデータを変換する　degPos ()

【書式】

static int degPos (float deg)

【概要】

角度(degree) (float 型)をポジションデータに変換します。

【引数】

種類	名称	説明	設定範囲
in	deg	角度(degree)(float 型)	-135.0 ~ 135.0

【戻り値】

種類	値	説明
成功の場合	3500 ~ 11500	ポジションデータ
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
int pos = krs. degPos(-90.0);     //-90.0deg(度)をポジションデータに変換します
krs.setPos (0,pos);               //変換したデータを ID:0 のサーボに送ります
```

【説明】

サーボの角度をポジションデータに変換するための関数です。サーボの動作角は 270°ですので、センターを 0°とし左右-135.0°～135.0°まで変換できます。引数として角度を渡すと、戻り値でポジションデータが返ってきます。

ポジションデータを角度に変換する　posDeg ()

【書式】

static float posDeg (int pos)

【概要】

ポジションデータを角度(degree) (float 型)に変換します。

【引数】

種類	名称	説明	設定範囲
in	pos	ポジションデータ	3500 ～　11500

【戻り値】

種類	値	説明
成功の場合	-135.0 ~ 135.0	角度(degree)(float 型)
失敗の場合	9999.9	正方向範囲外（ANGLE_F_FALS)
	-9999.9	負方向範囲外（-ANGLE_F_FALS)

【使用例】

```
int pos;
float deg;
pos = krs.getPos(0);   //ID0 の現在のポジションデータを読み取ります
deg = krs. posDeg(pos);    //読み取ったポジションデータを角度に変換します
```

【説明】

degPos()とは逆にポジションデータを角度に変換するための関数です。この関数のみ、失敗した場合は 9999.9（正転）、-9999.9（逆転）が戻り値として返ってきます。

100 倍した角度をポジションデータに変換する　degPos100 ()

【書式】

static int degPos100 (int deg)

【概要】

角度(degree) x 100(int 型)をポジションデータに変換します。

【引数】

種類	名称	説明	設定範囲
in	deg	角度(degree　x 100)(int 型)	-13500 ～　13500

【戻り値】

種類	値	説明
成功の場合	3500 ～ 11500	変換されたポジションデータ
失敗の場合	-1	範囲外

【使用例】

```
int pos = krs. degPos100(-9050);    //-90.5deg(度)をポジションデータに変換します
krs.setPos (0,pos);                 //変換したデータを ID:0 のサーボに送ります
```

【説明】

-135.0～135.0 を 100 倍した-13500～13500 をポジションデータへ変換する関数です。少数を使用した計算を避けたい場合にご利用ください。

ポジションデータを 100 倍した角度に変換する　posDeg100 ()

【書式】

static int posDeg100 (int pos)

【概要】

ポジションデータを角度(degree) x 100(int 型)に変換します。

【引数】

種類	名称	説明	設定範囲
in	pos	ポジションデータ	3500 ～ 11500

【戻り値】

種類	値	説明
成功の場合	-13500 ～ 13500	サーボの温度リミット値
失敗の場合	0x7FFF (32767)	正方向範囲外 (ANGLE_I_FALSE)
	0x8000 (-32768)	負方向範囲外 (-ANGLE_I_FALSE)

【使用例】

```
int pos;
int deg100;
```

```
pos = krs.getPos(0);   //ID0 の現在のポジションデータを読み取ります
deg100   = krs. posDeg100(pos);    //読み取ったポジションデータを角度に変換します
```

【説明】

ポジションデータをもとに角度を 100 倍にした数値（-13500～13500）に変換するための関数です。この関数のみ、失敗した場合は 0x7FFF（正転）、0x8000（逆転）が戻り値として返ってきます。

KRR（受信機）の値を取得する

enum KRR_BUTTON : unsigned short

【書式】

enum KRR_BUTTON : unsigned short

【概要】

KRR が受信するボタンデータを unsigned short 型で値を割り当て定義しています。

同時押しの場合は各データの論理和をとります。

【列挙値】

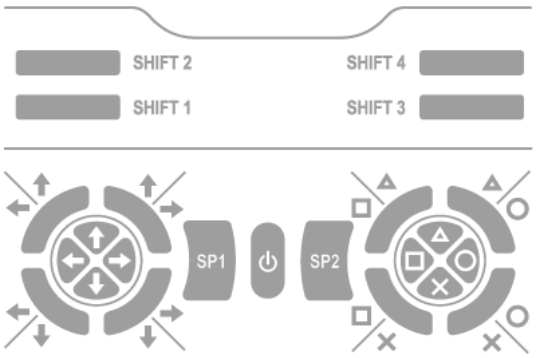
名称	数値	対応ボタン
KRR_BUTTON_NONE	0x0000	何も押されていない
KRR_BUTTON_UP	0x0001	↑
KRR_BUTTON_DOWN	0x0002	↓
KRR_BUTTON_RIGHT	0x0004	→
KRR_BUTTON_LEFT	0x0008	←
KRR_BUTTON_TRIANGLE	0x0010	△
KRR_BUTTON_CROSS	0x0020	×
KRR_BUTTON_CIRCLE	0x0040	○
KRR_BUTTON_SQUARE	0x0100	□
KRR_BUTTON_S1	0x0200	シフト 1 左手前
KRR_BUTTON_S2	0x0400	シフト 2 左奥
KRR_BUTTON_S3	0x0800	シフト 3 右手前
KRR_BUTTON_S4	0x1000	シフト 4 右奥
KRR_BUTTON_FALSE	0xFFFF	エラー値(受信失敗等)

【使用例】

```
unsigned short S1_Up =   KRR_BUTTON_UP | KRR_BUTTON_S1;      //S1 と↑ボタンを同時押しした時の値
```

【備考】

KRC-5FH のボタン配置は下記図になります。



【説明】

無線コントローラ KRC-5FH の各ボタンには数字が割り当ててあります。KRR-5FH は、これらの数字で受信したデータを通知しますので、上記の表を参考にどのボタンが押されたかを判断してください。ボタンは組み合わせることも可能です。組み合わせは論理和となりますので、↑（0x0001）と→（0x0004）同時押しされた場合は、0x0005 が通知されます。

ボタンデータを取得する　getKrrButton ()

【書式】

unsigned short getKrrButton ()

【概要】

KRR が受信しているボタンのデータを取得します。

【引数】

なし

【戻り値】

種類	値	説明
成功の場合	-	KRR_BUTTON で定義されているボタンデータが返ってきます。 同時押しの場合は、その論理和をとります。
失敗の場合	0xFFFF	通信失敗（**KRR_BUTTON_FALSE**）

【使用例】

```
unsigned short buttonData;
buttonData = krs.getKrrButton();
if(buttonData == KRR_BUTTON_UP)      //↑ボタンのみが押されている時
{
  //↑ボタンが押されている時の処理
```

```
}
```

【説明】

受信したボタンデータを読み込むための関数です。

※ID は特に指定しませんが、受信機 KRR-5FH は ID31 で固定されています。

アナログデータを取得する　getKrrAnalog ()

【書式】

int getKrrAnalog (int paCh)

【概要】

KRR が受信している指定したアナログポート(PA)のデータを取得します。

【引数】

種類	名称	説明	設定範囲
in	paCh	アナログポートの番号	1～4

【戻り値】

種類	値	説明
成功の場合	0～127	指定したアナログポートのデータ
失敗の場合	-1	通信失敗（**ICS_FALSE**）

【使用例】

```
int adData;
adData = krs. getKrrAnalog(1);          //PA1 のデータを取得する
```

【説明】

KRC-5FH に実装されているアナログ端子からの値を読み込むための関数です。本体に刻印されている PA1 が getKrrAnalog(1)に該当します。

KRC-5FH のアナログポートは PA1～4 ですので、それ以外の値を渡すとエラーになって返ってきます。

すべてのデータを一括で取得する　getKrrAllData ()

【書式】

bool getKrrAllData (unsigned short *button, int adData[4])

【概要】

KRR が受信しているボタンデータおよびアナログポート(PA)のデータをすべて取得します。

【引数】

種類	名称	説明	設定範囲
out	*button	KRR_BUTTON で定義されているボタンデータが返ってきます。 同時押しの場合は、その論理和をとります。	ポインタを渡します。 成功すれば渡した変数に値が代入されます。
out	adData	4ch 分のアナログデータ	配列のサイズは 4 固定です

【戻り値】

種類	値	説明
成功の場合	ture	通信成功
失敗の場合	false	通信失敗

【使用例】

```
int addata[4];
unsigned short buttonData;
krs. getKrrAllData (&buttonData,addata);      //すべてのデータを取得する(ポインタを渡し、中身を書き換える)
if(buttonData == KRR_BUTTON_UP)      //↑ボタンのみが押されている時
{
   //↑ボタンが押されている時の処理
}
```

【説明】

KRR-5FH が受信したすべてのデータを一括で読み込むための関数です。

改 訂 履 歴

バージョン	日付	詳細
1.0	2017/02/23	第 1 版を作成。
2.0	2018/01/19	第 2 版を作成
2.1	2020/02/25	第 2.1 版を作成 ・getID および setID の追加 ・最新機種の追加