

---

# RCB-4 変換基板

---

## RCB-4 ライブラリ

for Arduino

第 2.0 版

---

近藤科学株式会社

---



# 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
RCB-4 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
RCB-4 もしくは RCB-4 搭載の弊社製ロボット .....	6
HeartToHeart4.....	7
Dual USB アダプターHS.....	7
電源.....	7
ICS 機器.....	9
RCB-4Library for Arduino.....	9
接続方法.....	10
Arduino との接続方法 .....	10
シールド基板を用いる方法 .....	12
プログラムの書き込み方法 .....	13
RCB-4 Library for Arduino について.....	14
ライブラリの概要.....	14
ライブラリの入手方法 .....	14
ライブラリ圧縮ファイルの中身.....	14
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 .....	17
モーションの再生について .....	17
①直接モーションにジャンプする(Rcb4MotionPlay).....	17
②KRR のボタンデータを指定する(Rcb4SetKrrButton) .....	17
Rcb4MotionPlay2 の詳細 .....	18
ソースコード .....	18
ソースコード解説 .....	20
複数の UART を用いる場合 .....	22
Rcb4SetKrrButton の詳細 .....	24

Rcb4MotionPlay2 のソースコード相違と解説 .....	24
RCB-4 Library for Arduino の詳細 .....	25
RCB-4 と通信するための最初設定 .....	25
RCB-4 のクラスを宣言する Rcb4HardSerialClass::Rcb4HardSerialClass() .....	25
通信を開始する Rcb4HardSerialClass:: open() .....	26
固定パラメーター一覧 .....	27
KRR ボタンの固定値 KRR_BUTTON .....	27
SIO の定義、割り当て値 SioNum .....	28
RCB-4 のコンフィグデータの型 ConfigData .....	28
サーボモータをひとかたまりにした型 ServoData .....	29
公開変数 .....	29
RCB-4 のコンフィグデータ configData .....	29
基本通信関数 .....	31
コンフィグデータを取得 getConfig() .....	31
RCB-4 と接続できているか確認 checkAcknowledge() .....	31
モーションに関する関数 .....	33
指定したモーションを再生 motionPlay() .....	33
KRR のデータを転送する関数 .....	35
KRR に割り当てられているボタンデータを転送 setKrrButtonData() .....	35
KRR に割り当てられているアナログデータを転送 setKrrAdData() .....	35
PIO を制御する関数 .....	37
PIO の状態を取得 getPio () .....	37
PIO の状態を設定 setPio() .....	37
PIO の入出力の変更 setPioMode() .....	38
AD データを取得する関数 .....	40
AD 値を取得 getAdData() .....	40
サーボモータ関連の関数 .....	41
単体サーボモータを動作 setSingleServo() .....	41
単体サーボモータをフリーに setFreeSingleServo() .....	41
単体サーボモータにトルクをかける setHoldSingleServo() .....	42
複数のサーボモータを動かす setServoPos() .....	43
複数のサーボモータをフリーにする setFreePos() .....	44
複数のサーボモータのトルクをかける setHoldPos() .....	45
指定したサーボモータの角度を取得する getSinglePos() .....	46
複数のサーボモータのスピード値を変更 setServoSpeed() .....	46
複数のサーボモータのストレッチ値を変更 setServoStretch() .....	47
改訂履歴 .....	49

## はじめに

このたびは RCB-4 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 RCB-4 変換基板を用いて Arduino 系マイコンボードから弊社製 RCB-4 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、ショートの大危険性があります。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ください。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ー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 2025 年 1 月現在における対応状況を記載しております。Arduino 側のバージョンの変更によりプログラムの動作に影響がある場合があります。

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ウ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、ソフトウ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

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 <https://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](mailto: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

### RCB-4 変換基板

弊社ロボット用コントロールボード RCB-4 (HV/mini) 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 RCB-4 の制御が可能となります。

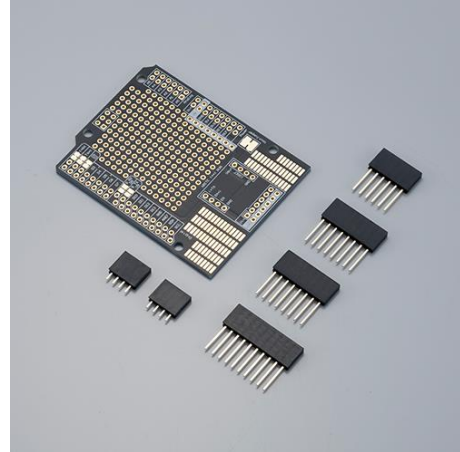
また、別売りの専用シールド基板を用意するとより、Arduino UNO R3/R4 への配線も省き簡単に接続することが可能です。

※本製品は、ヘッダピンのはんだ付け作業が必要です。



RCB-4 変換基板

No.03144 ￥2,600 (税別)



KSB シールド 2

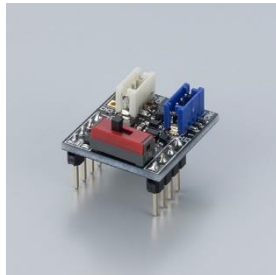
No.03149 ￥1,200 (税別)

### 製品の構成

ロボットと Arduino などマイコンボードとの間に RCB-4 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。



ロボット  
(RCB-4 搭載)



RCB-4 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、RCB-4 と通信を行います。また、Arduino UNO R3 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機能 (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ポート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 をソフトウェア上でシリアル機能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ます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。

※RCB-4 Library for Arduino Ver.2.0 より SoftwareSerial のサポートを終了しました。SoftwareSerial を使用する場合は、[ライブラリ Ver.1.1](#) をご利用ください。SoftwareSerial は最新の環境での動作確認をしていません。

## 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。(2025 年 1 月現在)

※Arduino IDE を使用できる前提になります。

### ●HardwareSerial

Arduino UNO R3	ArduinoMEGA 2560	Arduino MKR ZERO
Arduino UNO R4 Minima	Arduino Nano Every	
M5Stack Gray	M5Stack Basic V2.6	M5StackC
Tenssy3.6	Tenssy4.0	Tenssy4.1

### 【ご注意ください】

Arduino UNO R3 など一部のボードは、シリアルポート（UART）が USB と共通になっているため、サーボなど ICS 機器からの返事を PC（シリアルモニタ）に表示することはできません。PC（シリアルモニタ）に表示する必要がある場合は、Arduino UNO R4 Minima や Arduino Nano Every などシリアルポート（UART）と USB が別になっているボードをご利用ください。

## 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 2.3.4 です。(2025 年 3 月現在)

### 【ご注意ください】

Arduino UNO R4 を使用する場合は、下記のバージョン以上の環境でご利用ください。

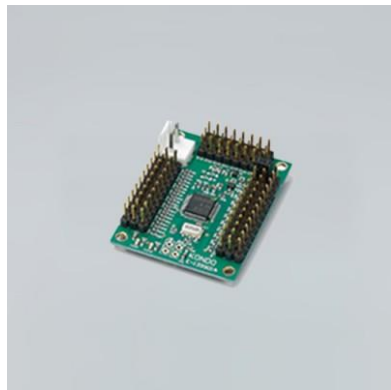
- ・ Arduino IDE のバージョン : 2.3.4 以上
- ・ ボードマネージャ(ArduinoUNO R4 Boards)のバージョン : 1.3.2 以上

これ以前のバージョンではシリアル通信が正常に動作しない場合があります。

## RCB-4 もしくは RCB-4 搭載の弊社製ロボット

RCB-4 は弊社 KRS サーボを搭載したロボットを動かすためのコントロールボードです。2 系統の ICS 機器用シリアル端子を搭載し、サーボを最大 35 個制御できます。また、ICS 対応の受信機を接続することで簡単に無線コントロールすることが可能です。ジャイロセンサや加速度センサなどアナログセンサを接続できる AD 端子や、デジタル I/O として利用できる PIO 端子も実装しています。（ボードの種類により端子の数が異なります）モーション作成ソフト HeartToHeart4 と組み合わせることで、容易にロボットの動きを作ることができます。HeartToHeart4 について詳しくは、下記『[HeartToHeart4](#)』の項をご覧ください。

### RCB-4HV



#### <スペック>

- 寸法：45×35×13mm(突起部除く)
- 重量：12g
- 電源電圧：6V～12V
- 対応サーボ：シリアル専用
- 通信規格：ICS3.0/3.5/3.6
- 通信速度：115200/625000/1250000
- 信号レベル：TTL
- ICS 最大接続数：36
- 接続方法：デジチェーン
- SIO 端子：8（2 系統）
- AD 端子：10
- PIO：10
- COM ポート：1
- 端子：サーボコネクタ

KHR-3HV、KMR-M6/P4 に付属するコントロールボードです。サーボ端子に 2.54 ピッチのヘッダピンを採用しているため、電流を多く消費する KRS-9000 シリーズ、5000 シリーズを使用した大型ロボットにも使用できます。

■商品情報『RCB-4HV』 <http://kondo-robot.com/product/03076>

### RCB-4mini



#### <スペック>

- 寸法：35×30×12mm(突起部除く)
- 重量：7.6g
- 電源電圧：6V～12V
- 対応サーボ：シリアル専用
- 通信規格：ICS3.0/3.5/3.6
- 通信速度：115200/625000/1250000
- 信号レベル：TTL
- ICS 最大接続数：36
- 接続方法：デジチェーン
- SIO 端子：6（2 系統×3 口）
- AD 端子：5
- PIO：0
- COM ポート：1
- 端子：ZH コネクタ

小型のロボットに搭載することを目的にした RCB-4HV の小型版です。RCB-4 と同じく HeartToHeart4 を使用したモーション作成が可能です。センサ用の AD 端子が 5 ポートに省略され、デジタル I/O 端子は実装していません。サーボ用の端子が ZH 端子のため、小型サーボ KRS-3300 シリーズや KRS-2552/2542HV でのご利用に最適です。

■商品情報『RCB-4mini』 [http://kondo-robot.com/product/RCB-4\\_mini](http://kondo-robot.com/product/RCB-4_mini)

### RCB-4 を搭載した弊社製ロボット（2025 年 1 月現在）

- ・ KXR シリーズ
- ・ KHR-3HV Ver.1/2/3/3.1
- ・ KMR-M6/P4 Ver.1.0/2.0
- ・ カメ型ロボット 02 Ver.1.0/1.5

## HeartToHeart4

---

HeartToHeart4（以下 HTH4）は、ロボット用ビジュアルプログラミングソフトウェアです。

ほとんどの作業をマウス操作で行うことができ、プログラミングの知識がない方でも簡単にモーション作成が可能です。また、ロボットの姿勢の調整、ジャイロや加速度などオプションセンサーの設定といったソフトウェアの作業をすべてカバーします。HTH4 がもつたくさんの機能を使って自由に自分だけのロボットを作り上げることができます。詳しくは下記ウェブサイトをご覧ください。

■商品情報『HeartToHeart4』 <http://kondo-robot.com/product/hearttoheart4>

**ライブラリを使用する場合は、HTH4 の ver2.4 以降をご利用ください。**それ以前のバージョンはアドレス等が違うため誤動作する原因となります。

RCB-4 内部のサーボモータの設定やロボットのモーションの作成も HTH4 を使用してください。以下の解説では、サンプルプロジェクトなど予め RCB-4 にプロジェクトが書き込まれ、各モーションが動作できる状態であることを前提に説明します。

## Dual USB アダプターHS

---



RCB-4 と PC を接続するための USB アダプタです。「シリアルモード」で使用します。これを使用して、HTH4 で作成したモーションデータやサーボモータの設定情報を RCB-4 に書き込みます。モードを「ICS モード」に切り替えることで KRS サーボや KRR-5 受信機など ICS 機器を直接 PC に接続することも可能です。

弊社各ロボットキットに付属しています。ただし、RCB-4 単品販売には付属していませんので、別途ご準備ください。

## 電源

---

電源は、ご利用の RCB-4 に接続されているサーボモータの電源電圧に対応した電源をご利用ください。

ロボットは、サーボモータを多く使用するため、瞬間的に大電流を流すことがあります。充電式のバッテリー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。

弊社ロボットキットには、充電式のバッテリー、または AC アダプタが付属されています。各製品を個別にご用意いただく場合は、単品販売のロボット用に バッテリー をご検討ください。また、各バッテリー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



バッテリー（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-2552R3HV 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



バッテリー（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(5.9V2A)

※ACアダプター（5.9V2A）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。



バッテリー（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ーは、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。最悪の場合燃え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 バッテリー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、ニッケル水素（Ni-MH）タイプのバッテリーをお勧めします。

※[バッテリーは、種類によって対応する充電器が異なります。必ず対応バッテリー・充電器を組み合わせご利用ください。](#)

## ICS 機器

RCB-4 は、弊社独自の通信規格 ICS を採用した KRS サーボシリーズが使用できます。ICS 準拠のコマンドにより、角度制御や各種パラメータの設定が可能です。また、ICS 対応の受信機を接続し無線化することもできます。この説明書では、KRS サーボや受信機など ICS 対応機器をまとめて「ICS 機器」と表記する場合があります。

弊社ロボットキットでは、この KRS サーボが採用されています。RCB-4 と HeartToHeart4 の組み合わせにより、プログラミングの知識がない方でも、簡単に KRS サーボを制御することができ、自由にモーションを作成することができます。

RCB-4 に接続できる ICS 機器については前述の『[RCB-4 について](#)』の解説をご覧ください。

## RCB-4Library for Arduino

RCB-4 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、[そちらを参照にしてください。](#)

## 接 続 方 法

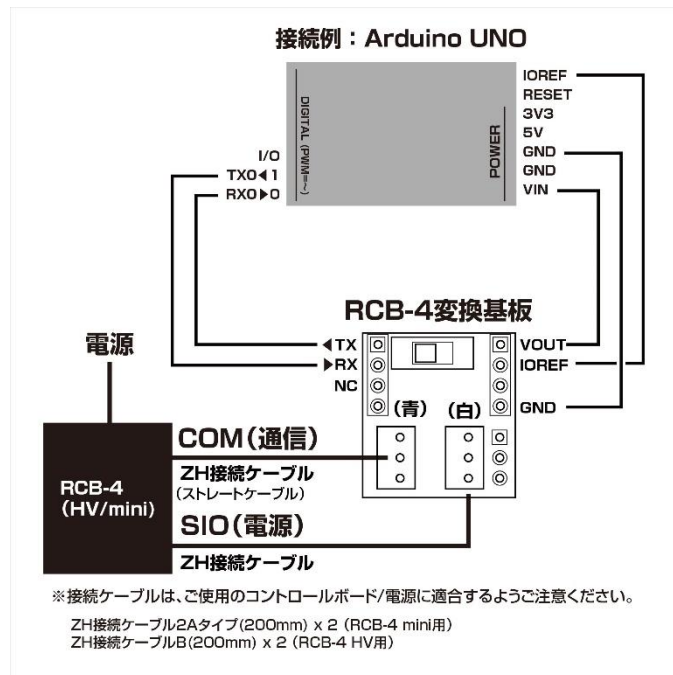
### Arduino との接続方法

RCB-4 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 と RCB-4 は、シリアル機能（UART）を使用して通信しデータを送受信します。下記の図を参考にボードの TX、RX ピン、その他必要なピンを ICS 変換基板に接続してください。

Arduino UNO R3/R4 を使用する場合は、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



#### ●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。

Arduino		RCB-4 変換基板(CN4)
Serial(UART) RX	⇔	TX
Serial(UART) TX	⇔	RX

#### ●Arduino 電源供給

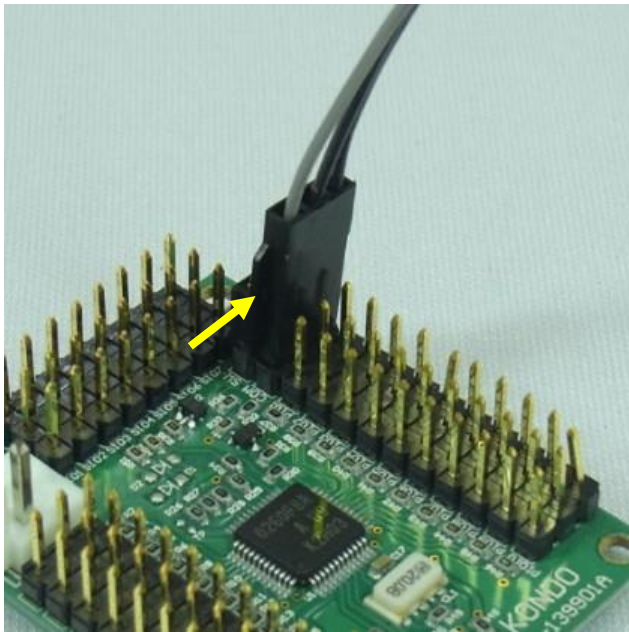
RCB-4 から Arduino へ電源を供給するための配線を行います。

Arduino		RCB-4 変換基板(CN3)
Vin	⇔	Vout
IOREF	⇔	IOREF
GND	⇔	GND

●RCB-4 通信用(COM)(ZH コネクタ 3pin、青色)

端子	RCB-4 COM		RCB-4 変換基板(CN6)
1pin	TX(RCB-4 からデータの送信)	⇔	RX
2pin	RX	⇔	TX(変換基板からデータの送信)
3pin	GND	⇔	GND

RCB-4 と RCB-4 変換基板の COM を付属の ZH 接続ケーブルで接続します。RCB-4 変換基板側は、青い ZH コネクタに接続します。RCB-4 の種類によって端子が異なります。端子に合わせて接続ケーブルを使い分けてください。



【RCB-4HV の場合】

ZH 接続ケーブル B を使用します。RCB-4 側のコネクタは、爪がボードの内側になるように接続してください。

※コネクタの接続向きを間違える破損する可能性があるのでご注意ください

【RCB-4mini の場合】

ZH 接続ケーブル A を使用します。一方向にしか刺さりませんので、コネクタの爪の向きを確認して接続してください。

●電源供給端子(CN1(ZH コネクタ、ナチュラル色)/CN2(2.54mm ピッチパットのみ) )

端子	種類	説明
1pin	SIO	RCB-4 の信号が流れている端子ですが、特に使用しません
2pin	Power	RCB-4 から電源を供給します
3pin	GND	GND

RCB-4 から RCB-4 変換基板に電源を供給するための端子です。ZH 接続ケーブルで RCB-4 の空いている SIO 端子に接続します。この端子を通して、Arduino へ電源が供給されます。



#### 【RCB-4HV の場合】

ZH 接続ケーブル B を使用します。RCB-4 側のコネクタは、爪がボードの内側になるように接続してください。

※コネクタの接続向きを間違える破損する可能性があるのでご注意ください

#### 【RCB-4mini の場合】

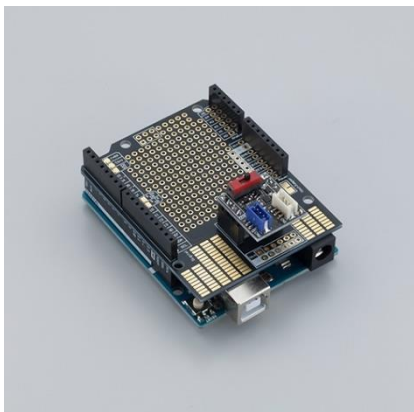
ZH 接続ケーブル A を使用します。

RCB-4 変換基板上の電源端子として使用する ZH コネクタの隣にあるパターンは、このコネクタと同じように配線されています。ICS 変換基板との組み合わせや、その他機器と同時に使用する場合、電源の供給用としてご利用ください。

RCB-4 に接続する場合は COM と SIO ピン、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、RCB-4 変換基板と Arduino を接続できる KSB シールド 2 を使用すれば、各ヘッダピン、ソケットをはんだ付けするだけですぐにご利用できます。シールド基板には、Arduino UNO R3/R4 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
- 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。

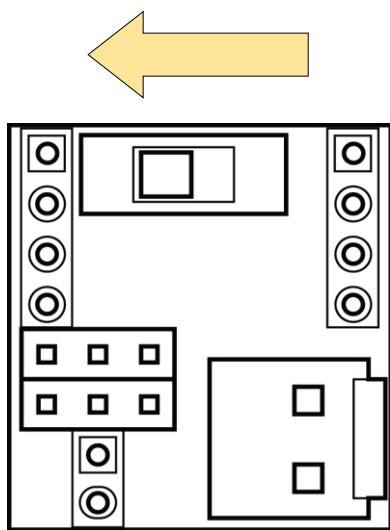
## プログラムの書き込み方法

Arduino UNO R3 など一部のボードはシリアルポート（UART）が 1 系統しかなく、USB 通信と兼用になっていますので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、RCB-4 変換基板には、PC との通信時と RCB-4 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。

Arduino UNO R4、Arduino Nano Every、M5Stack 、Arduino Mega などシリアルポート（UART）と USB が別になっている場合は、常にスイッチを「実行モード」にしてご利用ください。

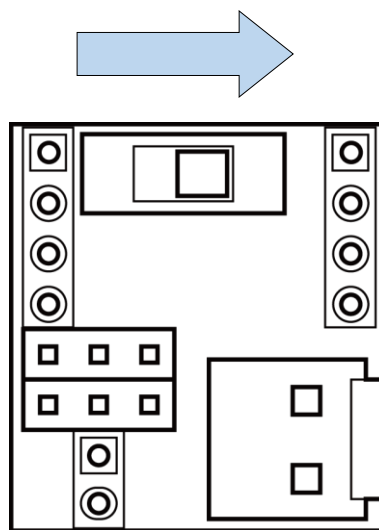
### 【プログラム書き込みモード】

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（RCB-4 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。

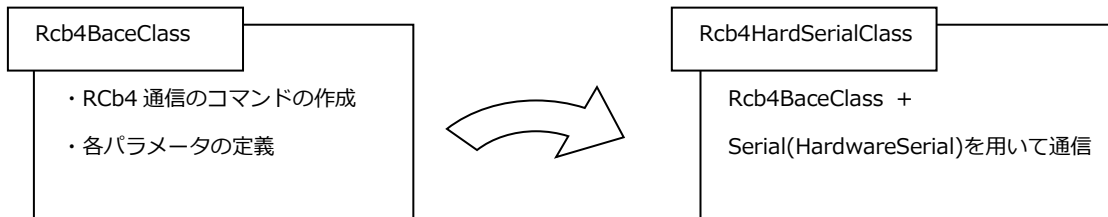


# RCB-4 Library for Arduino について

## ライブラリの概要

RCB-4 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。RCB-4 に登録されたモーションの再生や、KRS サーボを個別に制御したり、センサの値を取得することも可能です。

RCB-4 を使用するための関数は、「Rcb4BaseClass」をベースのクラスにし HardwareSerial を使用するための「Rcb4HardSerialClass」が用意されています。



## 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。

[https://kondo-robot.com/fag/rcb4-library-a2\\_0](https://kondo-robot.com/fag/rcb4-library-a2_0) (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。

RCB4\_Library\_for\_Arduino\_V2/

└Rcb4ClassV200.zip (Rcb4BaseClass, Rcb4HardSerialClass)

└取扱説明書など

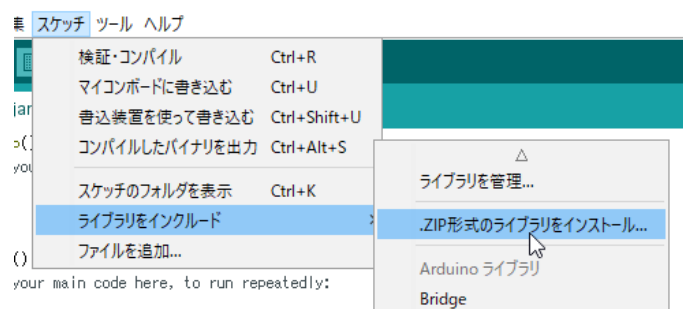
## ライブラリのインポート

※インポート方法は Arduino IDE のバージョン 2.3.4 を例にします。

① 弊社ウェブサイトよりライブラリー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

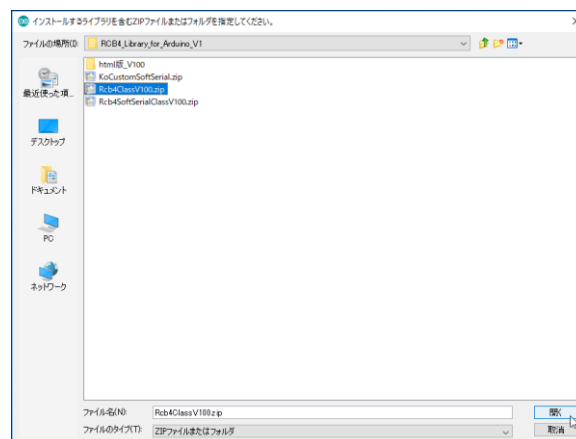




④ 『インストールするライブラリを含む ZIP ファイルまたはフォルダを指定してください』となり、ファイル指定ダイアログが表示されますので、ダウンロードした zip ファイルの場所を指定します。

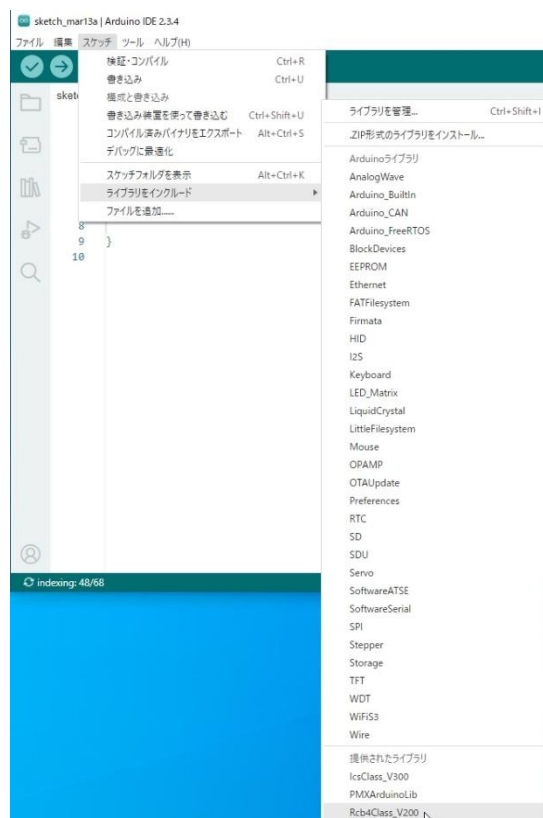
・ HardwareSerial の場合 : Rcb4ClassVxxx.zip

(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「Rcb4Class\_Vxxx」の表示がありましたら成功です。

※右の画像は Rcb4Class 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。

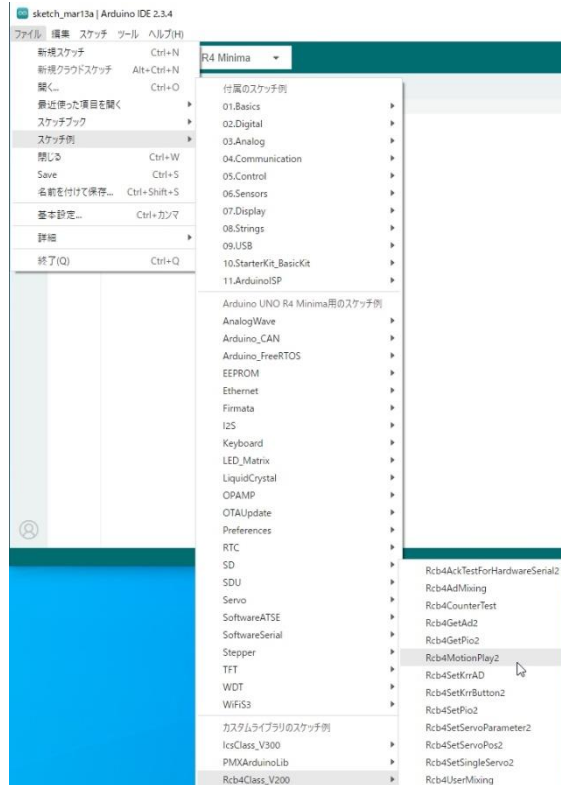


# サンプルプログラムについて

## 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呼び出し方法は、「ファイル」→「スケッチの例」→「Rcb4Class\_Vxxx」で呼び出します。

ます。  
サンプル



## サンプルプログラムの種類

サンプルプログラムは下記 13 種類あります。赤字のサンプルは Ver.2.0 で追加されたサンプルです。

サンプルプログラム名	概要
Rcb4MotionPlay2	RCB-4 に登録されたモーションを、モーション番号を指定して再生します。
Rcb4SetKrrButton2	RCB-4 の各モーションに登録されたボタンデータによりモーションを再生します。
Rcb4SetKrrAD	KRR からの AD 値を疑似的に Arduino から RCB-4 へ送ります。
Rcb4AckTestForHardwareSerial2	ACK コマンドが返ってくる(接続できている)か確認します。
Rcb4GetAd2	AD ポート 0 に接続されている電源電圧のデータを取得し、電圧に応じて Arduino の D4 ポートを ON/OFF させます。
Rcb4GetPio2	RCB4 の PIO1 のデジタル値(H/L)を Arduino の D3 ポートに出力(H/L)します。
Rcb4SetPio2	RCB-4 の PIO を 0.1 秒毎順番に H にします
Rcb4SetSingleServo2	接続されているサーボモータ 1 個ずつ動かします。サンプルでは、SIO1-4 に接続された ID1、SIO5-8 の ID2 のサーボを動かします。
Rcb4SetServoPos2	HTH4 の POS コントロールと同じように指定したサーボモータを複数個一括で動かします。
Rcb4SetServoParameter2	指定したサーボモータのパラメータ(ストレッチ及びスピード)を変更します。
Rcb4AdMixing	AD 値をもとにサーボヘミキシングをかけます。
Rcb4CounterTest	ユーザーカウンターの値を読み書きします。
Rcb4UserMixing	ユーザー変数の値をもとにサーボヘミキシングをかけます。

- ※ 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。
- ※ ボードにより対応する通信速度が異なります。RCB-4 の COM 通信速度はボードに合わせて設定し、ご利用ください。
- ※ サーボモータ単体で使用する場合、RCB-4 にサーボを認識させる必要があります。方法は、HTH4 の「ウィンドウ」メニューから「システム設定」ウィンドウを開き、対象のサーボにチェックを入れます。チェックを入れたら「プロジェクト設定ウィンドウ」から「ROM にすべて保存」ボタンで設定を書き込んでください。詳細は『HeartToHeart4 ユーザーズマニュアル』p.50 をご参照ください。作業が完了したら必ずプロジェクトを保存してから閉じてください。
- ※ このサンプルを使用する場合は、RCB-4 に予め使用するロボットに合わせたプロジェクトを登録した状態で使用してください。

## モーションの再生について

モーションの再生方法は 2 種類あります

### ①直接モーションにジャンプする(Rcb4MotionPlay)

Rcb4MotionPlay は、RCB-4 に登録されたモーションのモーション番号で指定します。これは、HTH4 の「モーション一覧」ウィンドウに表示されている番号です。

この方法は、他のモーションの再生中でも強制停止させ違うモーションになりますので、モーションのつなぎ目とか注意が必要です

### ②KRR のボタンデータを指定する(Rcb4SetKrrButton)

Rcb4SetKrrButton は、モーションに登録された無線コントローラ用のボタンデータを指定してモーションを再生します。これにより、モーション内で使用しているボタン分岐にも対応することができます。歩行し続けたり、しゃがみ続けるなど、ボタンによる動作の変化を指定することができます。サンプルモーションのボタンの登録で動くモーションを動かすには非常に便利です。

ただし、1 度ボタンデータを送るとそのボタンのままやり続けますので、そのモーションを終わるには NONE(押されていない状態)を送る必要があ

ります。

また、モーションによってはボタン分岐後にホームポジションに戻るような処理が入っていたりしますので、動くまでタイムラグがある場合があります。

## Rcb4MotionPlay2 の詳細

基本となる Rcb4MotionPlay を例に説明します。このサンプルは、RCB-4 に登録されているモーション番号 1 と 2 を再生します。

※モーション番号は、HTH4 の「モーション一覧」ウィンドウに登録したリストの番号です。

## ソースコード

```
#include <Rcb4HardSerialClass.h>

const long BAUDRATE = 115200; //RCB-4 の通信速度
const int TIMEOUT = 100;      //RCB-4 とのシリアル通信に設定する応答待ち時間

//インスタンス+EN ピン(2 番ピン)および UART の指定
//ボードに接続しているシリアルポートに合わせて&Serial を変更してください。
//(&Serial / &Serial1 / &Serial2...)
Rcb4HardSerialClass rcb4(&Serial1,BAUDRATE,TIMEOUT);

void setup() {

    Serial.begin(9600); //PC との通信を開始する
    delay(1000);        //通信が可能になるまで待つ

    rcb4.configData.word = 0xffff; //ConfigData は間違えないように 0xFFFF で初期化

    while(rcb4.open() == false)    //Config をとれないとモーション再生ができないため、取得できるまでループ
    {
        Serial.println("RCB-4 Open False");
        delay(500);
    }
    Serial.println("RCB-4 OPEN: True");

    if(rcb4.checkAcknowledge() == true) //RCB-4 と通信ができていますか確認
    {
        Serial.println("RCB-4 Check Ack: True");
    }
    else
    {
        Serial.println("RCB-4 Check Ack: False");
    }
}
```

```

}

void loop() {

    if(rcb4.motionPlay(1) == true)                //モーション 1 を再生
    {
        //通信に成功した場合
        Serial.println("Motion:1 play");
    }
    else
    {
        //通信に失敗した場合
        Serial.println("Motion:1 False");
    }

    //現在再生しているモーションの番号を取得する
    int num = rcb4.getMotionPlayNum();
    Serial.print("Now Motion Num: ");
    Serial.println(num);

    delay(2000);

    if(rcb4.motionPlay(2) == true)                //モーション 2 を再生
    {
        //通信に成功した場合
        Serial.println("Motion:2 play");
    }
    else
    {
        //通信に失敗した場合
        Serial.println("Motion:2 False");
    }

    // モーションが終わるまで確認し、終わったところでループを抜ける
    while(true)
    {
        int playNum = rcb4.getMotionPlayNum();
        if(playNum == 0)
        {
            // モーション再生が終わって待機中(Motion:0)
            Serial.println("Motion finish");
            break;
        }
    }
}

```

```

    }

    else if(playNum < 0)
    {
        // モーションの取得に失敗した
        Serial.println("Communication failed");
        break;
    }

    Serial.print("NowMotion>>>");
    Serial.println(playNum);
}

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。Serial.println()使ってエラー出力結果をシリアルモニタに表示できるようにしています。詳しくは ArduinoIDE のマニュアルをご参照ください。

```
#include <Rcb4HardSerialClass.h>
```

Rcb4HardSerialClass を使えるようにします。

```

const long BAUDRATE = 115200;
const int TIMEOUT = 100;

Rcb4HardSerialClass rcb4(&Serial1,BAUDRATE,TIMEOUT); //RCB4HardSerialClass の定義

```

Rcb4HardserialClass クラスを rcb4 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 => Serial1

ボードで使用するシリアルポートを指定します。Serial の前に必ず「&」が必要です。ボードによりシリアルポートの番号が異なります。接続先のシリアルポートを確認して設定してください。(&Serial、&Serial1、&Serial2…など)

### ●通信速度(BAUDRATE) => 115200bps

RCB-4 に設定されている「COM 通信速度」と同じ通信速度を指定します。RCB-4 は 115200bps、625000bps、1250000bps に切り替えることができます。ボードにより対応する通信速度が異なりますので確認してからご利用ください。※「ICS 通信速度」ではありませんのでご注意ください。

### ●読み込みのタイムアウト(TIMEOUT) => 100ms

RCB-4 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。モーションがうまく再生できないときは、タイムアウトを長めにとってください。

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。

```
Rcb4HardSerialClass rcb4(&Serial1,115200,100);
```

## void setup()内のソースコード

```
Serial.begin(9600); //PC との通信を開始する
```

```
delay(1000);      //通信が可能になるまで待つ
```

Arduino IDE のシリアルモニタに結果を表示するため、Arduino と PC 間の通信を開始します。

```
rcb4.configData.word = 0xffff;      //ConfigData は間違えないように 0xFFFF で初期化
```

rcb4.configData は RCB4 内のコンフィグデータを保存しておく変数です。モーションを動かす際に RCB4 内のコンフィグデータが必要となります。この後の OPEN 時に取得しに行くのですが、初期値は不明ですので、0xFFFF で初期化しておきます。なお、rcb4.configData.word では short 型ですべて直接アクセスできるようになっています。

直接値の変更もできます。(例: rcb4.configData.bit. GreenLED = 1;(RCB-4 の緑色 LED を点灯させます))

configData を変更しても RCB4 内のコンフィグデータに書かないと変更はされません。

```
while(rcb4.open() == false)    //Config をとれないとモーション再生ができないため、取得できるまでループ
{
    Serial.println("RCB-4 Open False");
    delay(500);
}
Serial.println("RCB-4 OPEN: True");
```

rcb4.open() で宣言した rcb4 を初期化し宣言した通信速度、タイムアウトで通信をできるようにします。もし、データが返ってこない場合は false が返ってきます。Config データを取得できないとモーションの再生ができません。false で返ってくる間はずっと繰り返すので、Config データを取得できるまで繰り返し続けます。

処理が完了したらシリアルモニタに結果を表示します。

```
if(rcb4.checkAcknowledge() == true) //RCB-4 と通信ができているか確認
{
    Serial.println("RCB-4 Check Ack: True");
}
else
{
    Serial.println("RCB-4 Check Ack: False");
}
```

rcb4.checkAcknowledge() は RCB4 とつながっているか確認します。(正確には ACK が返ってるかを確認しています)

true が返ってきた場合は、RCB4 から返事が返ってきて通信が確立されています。

### void loop()内のソースコード

```
if(rcb4.motionPlay(1) == true)    //モーション 1 を再生
{
    //通信に成功した場合
    Serial.println("Motion:1 play");
}
else
{
    //通信に失敗した場合
```

```
Serial.println("Motion:1 False");
}
delay(2000);
```

rcb4.motionPlay(1)で RCB-4 に登録されたモーション番号 1 を再生します。rcb4.motionPlay(1)は、通信が成功すると true を返り値として返しますので、これを確認しシリアルモニタに結果を表示します。

引数である ( ) 内の数字を変更することで任意のモーションを再生することができます。センサの値によって if 文や switch 文で分岐し、モーションを再生することで簡単な自律ロボットを作成することができます。

モーションが終了するまで delay()で処理を中断します。または、下記の方法でモーション再生終了を確認し、次の処理に移動します。

```
while(true)
{
    int playNum = rcb4.getMotionPlayNum();
    if(playNum == 0)
    {
        // モーション再生が終わって待機中(Motion:0)
        Serial.println("Motion finish");
        break;
    }
    else if(playNum < 0)
    {
        // モーションの取得に失敗した
        Serial.println("Communication failed");
        break;
    }

    Serial.print("NowMotion>>>");
    Serial.println(playNum);
}
```

getMotionPlayNum()で現在再生しているモーションの番号を取得することができます。この関数は、モーションを再生していない場合は 0 を返します。これにより、モーション再生中は 1 以上の数字、モーションが終了すると 0 になりますので、上記のようにモーションが終了するまでループで処理を中断し、モーション終了と同時に次の処理へ移動することができます。

## 複数の UART を用いる場合

サンプルでは、Rcb4HardSerialClass クラスを rcb4 という名前で宣言をしていましたが、rcb4\_1 や rcb4\_2 などの任意名前でも宣言できます。

```
Rcb4HardSerialClass rcb4_1(&Serial1,115200,100); //Serial1 を使用,115200bps,timeout100
Rcb4HardSerialClass rcb4_2(&Serial2,115200,100); //Serial2 を使用,115200bps,timeout100
```

名前が同じでなければ、同時に複数の Rcb4HardSerialClass を宣言することができます。

UART ごとに Rcb4HardSerialClass 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 RCB-4 の数を増やすことも可能です。ただし、各 Rcb4HardSerialClass の Serial ポート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 IcsClass と共存することも可能です。

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。

```
rcb4_1.open(); // rcb4_1 で宣言した Rcb4HardSerialClass の通信初期設定  
rcb4_2.open(); // rcb4_2 で宣言した Rcb4HardSerialClass の通信初期設定
```

各 RCB-4 を動かす場合は下記のように記述できます。

```
rcb4_1.motionPlay(1);    //位置指令  rcb4_1 で設定した Serial(Serial1)を使って 1 枚目の RCB-4 のモーション番号 1 を再生  
rcb4_2.motionPlay(3);    //位置指令  rcb4_2 で設定した Serial(Serial2)を使って 2 枚目の RCB-4 のモーション番号 3 を再生
```



## Rcb4SetKrrButton の詳細

先ほどまでは、Rcb4MotionPlay の解説を行ってきましたが、モーションを動かすもう一つの方法の Rcb4SetKrrButton を解説します。この関数を使用することにより、KRC-5FH など無線コントローラでロボットを操縦しているようにモーションを再生することが可能です。ボタンによるモーション分岐にも対応していますので、ボタンの組み合わせや、ボタンが押され続けている間連続歩行、などの命令が可能になります。

※KRR（受信機）は、送信機からの情報がない場合は常に NONE を出力しているため、RCB-4 に KRR が接続された状態だと連続歩行などができません。Rcb4SetKrrButton を使用する場合は、KRR を外してください。初期設定は Rcb4MotionPlay と同じになりますので省略します。相違部分は loop() 処理内です。

## Rcb4MotionPlay2 のソースコード相違と解説

```
void loop() {
    // put your main code here, to run repeatedly:

    rcb4.setKrrButtonData (rcb4.KRR_BUTTON_UP);    // ↑ボタンの指示
    delay(2000);                                    // 2s 待つが、その間 ↑ボタンのまま
    rcb4.setKrrButtonData (rcb4.KRR_BUTTON_DOWN);  // ↓ボタンの指示
    delay(2000);                                    // 2s 待つが、その間 ↓ボタンのまま

    if(rcb4.setKrrButtonData (rcb4.KRR_BUTTON_NONE) == true) // 送れたかどうかの合否判定付の書き方(何も押していない状態を送信)
    {
        Serial.println("Communication: True");
    }
    else
    {
        Serial.println("Communication: False");
    }
    delay(2000);
}
```

上記 loop{} 内のプログラムは、

↑ボタンを送る => 2 秒待つ(この間は↑ボタンのモーション再生) => ↓ボタンを送る => 2 秒待つ(この間は↓ボタンのモーション再生) => NONE(何も押していない(停止))を送る => 判定でシリアルモニタに結果を表示 => 2 秒待つ(この間は何も再生れない) => ↑ボタンを送る…(以降ループ) という処理を行っています。

ボタンデータは、rcb4.setKrrButtonData() で送ることができます。引数は Rcb4baceClass 内でボタン定義(KRR\_BUTTON)を使用できます。各ボタンの割り当て、表記など詳しくは [ライブラリの解説部分](#) をご覧ください。

例 : rcb4.KRR\_BUTTON\_DOWN                      // ↓ボタン

ボタンの同時押しの場合は、定義したものの論理和(or)をとることで実現できます。

例 : rcb4.setKrrButtonData(rcb4.KRR\_BUTTON\_DOWN | rcb4.KRR\_BUTTON\_S1 );    // ↓ボタンと S1 ボタンを押したものを送ります。

また、rcb4.setKrrButtonData は送信が成功したかどうかの結果を返り値で取得することができます。

# RCB-4 Library for Arduino の 詳 細

RCB-4 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
Rcb4BaseClass	Rcb4 のコマンド生成や定義をまとめたクラス
Rcb4HardSerialClass	Rcb4BaseClass から派生し、Serial(HardwareSerial)を用いて RCB4 と通信をする

この章では RCB-4 にコマンドを送るために必要な関数を列挙しています。関数や定義が多数ありますのでサンプルプログラムで使用する部分のみ列挙します。記載されていない関数の詳細に関しては、解凍フォルダ内の html 版\_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 rcb4.xxxx と出てきますが、rcb4 という名前で Rcb4HardSerialClass を宣言しています。実際の Class 内とは異なり、わかりやすいように順番を入れ替えて解説していますのでご了承ください。

途中でクラスの指定していない関数の部分がありますが、Rcb4BaseClass の関数、変数、型になります。

## RCB-4 と通信するための最初設定

### RCB-4 のクラスを宣言する Rcb4HardSerialClass::Rcb4HardSerialClass()

【書式】

Rcb4HardSerialClass::Rcb4HardSerialClass (HardwareSerial \* rcb4Serial, long baudrate, int timeout)

【概要】

コンストラクタ (HardwareSerial 版)

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	* rcb4Serial	RCB-4 に接続する UART(HardwareSerial 型のポインタ)	Serial / Serial1 / Serial2…※
in	baudrate	RCB-4 の COM 通信速度	115200bps / 625000bps / 1250000bps
in	timeout	受信タイムアウト(ms)	～32767ms

※ボードに接続したシリアルポートの番号を指定します。

※ボードの種類のよって対応する通信速度が異なります。確認してからご利用ください。通常は 115200bps を使用します。

【使用例】

```
const long BAUDRATE = 115200;
const int TIMEOUT = 100;

Rcb4HardSerialClass rcb4(&Serial,BAUDRATE,TIMEOUT); //RCB4HardSerialClass の定義
```

【説明】

Rcb4HardSerialClass 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 open 関数を使います。

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。Rcb4Class 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。

BAUDRATE は、RCB-4 と通信する通信速度を設定できます。RCB-4 は、最大で 1.25Mbps まで通信することが可能ですが、ボードによって対応する通信速度が異なります。通常は 115200bps を使用します。

TIMEOUT は、返信データを待つ時間です。RCB-4 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。

**通信を開始する Rcb4HardSerialClass:: open()**

【書式】

bool Rcb4HardSerialClass:: open()

【概要】

通信の初期設定、通信を開始します。

【引数】

なし

【戻り値】

値	説明
true	通信設定完了
false	通信設定失敗

【使用例】

```
rcb4.open();  
while(rcb4.open() == false); //Config を取得できるまでループ
```

【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 RCB-4 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。

内部では、初期設定以外にも ACK で通信確認、RCB4 内の Config データの取得を行います。

## 固定パラメーター一覧

### KRR ボタンの固定値 KRR\_BUTTON

#### 【書式】

enum KRR\_BUTTON : unsigned short

#### 【概要】

KRR のボタンデータに相当する値を 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】

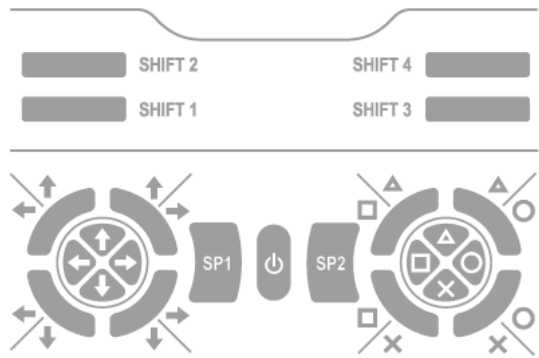
名称	数値	対応ボタン
KRR_BUTTON_NONE	0x0000	何も押されていない
KRR_BUTTON_UP	0x0001	↑
KRR_BUTTON_DOWN	0x0002	↓
KRR_BUTTON_RIGHT	0x0004	→
KRR_BUTTON_LEFT	0x0008	←
KRR_BUTTON_TRIANGLE	0x0010	△
KRR_BUTTON_CROSS	0x0020	×
KRR_BUTTON_CIRCLE	0x0040	○
KRR_BUTTON_SQUARE	0x0100	□
KRR_BUTTON_S1	0x0200	シフト1 左手前
KRR_BUTTON_S2	0x0400	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、上記の表を参考にと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。

SIO の定義、割り当て値    SioNum

【書式】

enum SioNum : byte

【概要】

Sio コネクタの番号の定義

【列挙値】

値	説明
SIO1_4	SIO1-4 を示す値
SIO5_8	SIO5-8 を示す値

RCB-4 のコンフィグデータの型    ConfigData

【書式】

union ConfigData

【概要】

RCB-4 内のコンフィグデータにアクセスしやすいように型定義をします。

【詳細】

```
typedef union {  
    /// ビットフィールド構造体  
    struct {
```

```

byte EnableSio :1;          //!< b0:ICS スイッチ
byte EnableRunEeprom :1;    //!< b1: EEPROM プログラム実行スイッチ
byte EnableServoResponse :1; //!< b2: 補間動作終了メッセージスイッチ
byte EnableReferenceTable :1; //!< b3:ベクタジャンプスイッチ
byte Frame :2;              //!< b4, b5 出力周期レジスタ[4:5]
byte Baudrate :2;           //!< b6, b7 COM ポーレートレジスタ[6:7]
byte ZeroFlag :1;          //!< b8 ゼロフラグ
byte CarryFlag :1;         //!< b9 キャリーフラグ
byte ProgramError :1;      //!< b10 プログラムエラーフラグ
byte RFU :1;               //!< b11,b12 未使用
byte IcsBaudrate :2;        //!< b13,b14 ICS スイッチポーレートレジスタ[13:14]
byte GreenLED :1;          //!< b15 LED レジスタ

} bit;

unsigned short word;        //!< 16 ビット WORD データ(ConfigData を一度にアクセスできるように)
} ConfigData;

```

## サーボモータをひとかたまりにした型 **ServoData**

### 【書式】

```
typedef struct ServoData
```

### 【概要】

サーボモータの ID および SIO と、ポジションなど任意の 2BYTE データを格納します。

### 【格納変数】

型	名前	説明
byte	Id	サーボモータの ID 値を格納します
byte	Sio	<a href="#">SIO</a> 値を格納します
int	Data	ポジションデータやストレッチ等を格納します

### 【詳細】

サーボモータの ID および SIO と、ポジションなど任意の 2BYTE データを格納します。

複数サーボモータを動かしたりパラメータを変えたりするときに使用します。

## 公開変数

### RCB-4 のコンフィグデータ **configData**

#### 【書式】

[ConfigData](#) configData

#### 【概要】

RCB-4 のコンフィグデータを格納しておく変数

【使用例】

```
rcb4.configData.bit. GreenLED = 1;          //(RCB-4 の緑色 LED を点灯させます)
rcb4.configData.word = 0xffff;              //ConfigData は間違えないように 0xFFFF で初期化
```

【注意】

configData は変数として格納しておくだけなので、RCB-4 内部のコンフィグデータを書き換える場合はこの変数を RCB-4 内部の RAM の ConfigRamAddress に送る必要があります。

最初の文字が大文字の場合は型になりますので、注意が必要です。

基本通信関数

コンフィグデータを取得 getConfig()

【書式】  
  
bool getConfig ()

【概要】  
  
RCB-4 のコンフィグデータを読み取る。

【引数】  
  
なし

【戻り値】

値	説明
true	通信完了 、configData を取得
false	通信失敗

【説明】  
  
RCB-4 のコンフィグデータを読みとります。  
  
読み取りが成功した場合、class 内の configData にデータが入ります。

RCB-4 と接続できているか確認 checkAcknowledge()

【書式】  
  
bool checkAcknowledge ()

【概要】  
  
ack コマンドを送信

【引数】  
  
なし

【戻り値】

値	説明
true	通信完了 (RCB-4 と接続が確立している)
false	コマンドが返ってこなかった時および NCK が返ってきたとき

【使用例】

```
if(rcb4.checkAcknowledge() == false)
{

```



```
//ACK が返ってこなかった時の処理
```

```
}
```

【説明】

ACK コマンドを送って、返答があるかどうか見ます。

モーションに関する関数

指定したモーションを再生 motionPlay()

【書式】

bool motionPlay (int motionNum)

【概要】

モーション番号で指定したモーションを再生します。

【引数】

種類	名称	説明	設定範囲
in	<i>motionNum</i>	再生させるモーション番号	最大は 120 です HTH4 の「モーション一覧」ウィンドウに登録したリストの番号

【戻り値】

値	説明
true	モーション設定完了
false	通信失敗

【使用例】

```
rcb4. motionPlay (1);      //1 番のモーションを再生します。
```

【説明】

指定したモーションを再生します。

suspend() => resetProgramCounter() => setMotionNum() => resume()

一時中断 => プログラムカウンタをリセット => 動かすモーションアドレスにジャンプ => モーションの再開 の手順でモーションを再生します。

現在、他のモーションを再生中でも強制的に止めてから再生します。

再生中のモーション番号を取得 getMotionPlayNum()

【書式】

bool getMotionPlayNum ()

【概要】

モーション番号で指定したモーションを再生します。

【引数】

なし

【戻り値】

値	説明
1～	再生中のモーション番号
0	待機中
-1	通信失敗
-2	再生している場所が異常

【使用例】

```
int playNum = rcb4.getMotionPlayNum();
```

【説明】

再生中のモーション番号を取得します。例えば、モーション番号 1 のモーションを再生した場合、再生中は 1 が返ってきます。モーションが終了し、待機状態になると 0 になります。これにより、モーションの終了を知ることができます。

## KRR のデータを転送する関数

### KRR に割り当てられているボタンデータを転送 `setKrrButtonData()`

#### 【書式】

`bool setKrrButtonData (unsigned short buttonData)`

#### 【概要】

KRR の領域にボタンデータを送信する。

#### 【引数】

種類	名称	説明	設定範囲
in	<i>buttonData</i>	ボタンデータ	ボタンデータは <a href="#">KRR_BUTTON</a> を参照

#### 【戻り値】

値	説明
true	通信完了
false	通信失敗

#### 【使用例】

```
rcb4.setKrrButtonData (rcb4.KRR_BUTTON_UP);    // ↑ ボタンの指示
```

#### 【説明】

この関数を使用することにより、KRC-5FH など無線コントローラでロボットを操縦しているようにモーションを再生することが可能です。ボタンによるモーション分岐にも対応していますので、ボタンの組み合わせや、ボタンが押され続けている間連続歩行、などの命令が可能になります。

#### 【注意】

※KRR（受信機）は、送信機からの情報がない場合は常に NONE を出力しているため、RCB-4 に KRR が接続された状態だと連続歩行などができません。Rcb4SetKrrButton を使用する場合は、KRR を外してください。

### KRR に割り当てられているアナログデータを転送 `setKrrAdData()`

#### 【書式】

`bool setKrrAdData (byte paPort, byte adData)`

#### 【概要】

KRR の領域にアナログデータを送信する。

#### 【引数】

種類	名称	説明	設定範囲
----	----	----	------

in	<i>paPort</i>	指定する AD ポート(1～4)	KRC に接続されている PA ポート番号になります
in	<i>adData</i>	AD データ(7bit)	KRC から送られてくるデータは 7bit になります

#### 【戻り値】

値	説明
true	通信完了
false	通信失敗

#### 【使用例】

```
rcb4. setKrrAdData(1,100);    //KRR の PA1 の領域に 100 を書き込む
```

#### 【説明】

KRR に受信されている AD の領域にデータにおくります。

KRR の AD 値を使ってモーションを変更している場合に外部から変更できます

#### 【注意】

※KRR（受信機）は、送信機からの情報がない場合は常に NONE を出力しているため、RCB-4 に KRR が接続された状態だと連続歩行などができなくなります。Rcb4SetKrrButton を使用する場合は、KRR を外してください。

## PIO を制御する関数

### PIO の状態を取得 `getPio ()`

【書式】

unsigned short getPio ()

【概要】

RCB-4 にある PIO の現在の状態を取得します。

【引数】

なし

【戻り値】

値	説明
	現在の PIO の状態(2byte を含めすべて)
<i>0xFFFF</i>	通信失敗

【使用例】

```
unsigned short pio = rcb4.getPio(); //RCB-4 の PIO の取得
```

【説明】

Move 命令を使って現在の PIO の H か L の状態を取得します。

【注意】

最低 1byte 単位でしか取得できないため、1bit ずつ判定できません。

### PIO の状態を設定 `setPio()`

【書式】

bool setPio (unsigned short pioData)

【概要】

RCB-4 にある PIO の状態を設定します。

【引数】

種類	名称	説明	設定範囲
in	<i>pioData</i>	PIO 全体のデータ	

【戻り値】

値	説明
true	通信完了
false	通信失敗

#### 【使用例】

```
rcb4.setPio(0x0001);    //PIO1 のみ H にする
```

#### 【説明】

Move 命令を使って現在の PIO を H か L の状態にします。

#### 【注意】

1bit ずつ変更できないので 2byte 単位で変更します。

### PIO の入出力の変更 setPioMode()

#### 【書式】

```
bool setPioMode (unsigned short pioModeData)
```

#### 【概要】

RCB-4 にある PIO の入出力状態を変更します。

#### 【引数】

種類	名称	説明	設定範囲
in	<i>pioModeData</i>	PIO 全体の入出力データ	

#### 【戻り値】

値	説明
true	通信完了
false	通信失敗

#### 【使用例】

```
unsigned short setModeData = 0xFFFE;    //1 : 出力、0:入力  0b111111 11111110   PIO1 のみ入力
while(rcb4.setPioMode(setModeData));    //漏れがないように PIO の入出力データが書き込めるまで繰り返す
```

#### 【説明】

Move 命令を使って現在の PIO 入力と出力を変更します。

対応した bit の 1:出力 0:入力

【注意】

1bit ずつ変更できないので、変更する場合はすべての PIO の値を変更する。



AD データを取得する関数

AD 値を取得     getAdData()

【書式】

unsigned short getAdData (int adPort)

【概要】

RCB-4 にある指定した AD（アナログ端子）の現在の値を取得します。

【引数】

種類	名称	説明	設定範囲
in	<i>adPort</i>	AD ポートの指定	0~10 まで 0 は、RCB-4 の電源電圧を取得できます

【戻り値】

値	説明
	指定したポートの AD 値(分解能 10bit)
0xFFFF	通信失敗

【使用例】

```
unsigned short ad = rcb4.getAdData(0); //RCB-4 の電源電圧の AD 値を取得
```

【説明】

Move 命令を使って指定した現在の AD 値を取得します。

AD0 を取得すると現在の RCB-4 に入力されている電源電圧値を分圧した値(約 1/5、10/49)が読めます。

サーボモータ関連の関数

単体サーボモータを動作 `setSingleServo()`

【書式】  
`bool setSingleServo (byte id, byte sio, int pos, byte frame)`

【概要】  
指定したサーボモータ単体を動作させます。

【引数】

種類	名称	説明	設定範囲
in	<i>id</i>	サーボモータの ID 番号	0~31
in	<i>sio</i>	SIO の値( <a href="#">SioNum</a> )	
in	<i>pos</i>	サーボモータの目標ポジション値	3500-11500
in	<i>frame</i>	フレーム数(多いほどゆっくり動く)	1-255

【戻り値】

値	説明
true	通信完了
false	通信失敗

【使用例】

`rcb4.setSingleServo (2, rcb4.SIO5_8,9500,50);    //SIO5-8 の ID:2 を 50 フレームで 9500 にする`

【説明】  
指定したサーボモータ単体を ID,SIO,POS,Frame 数を使用して動かします。  
モーション再生中に指定した場合は、モーションの角度データが優先されます。  
実際には、RCB-4 の内部のトリムの値が反映されるので注意する。

単体サーボモータをフリーに `setFreeSingleServo()`

【書式】  
`bool setFreeSingleServo (byte id, byte sio)`

【概要】  
指定したサーボモータ単体をフリー（脱力した状態）にします。

【引数】

種類	名称	説明	設定範囲
in	<i>id</i>	サーボモータの ID 番号	0~31
in	<i>sio</i>	SIO の値( <i>SioNum</i> )	

【戻り値】

値	説明
true	通信完了
false	通信失敗

【使用例】

```
rcb4.setFreeSingleServo (2, RCB-4.SIO5_8);      //SIO5-8 の ID:2 をフリーに
```

【説明】

ID,SIO で指定したサーボモータをフリーにします。

【注意】

Free を指定すると Hold のコマンドが発行されるまで Free 状態を保ち続けます。

## 単体サーボモータにトルクをかける `setHoldSingleServo()`

【書式】

bool setHoldSingleServo (byte id, byte sio)

【概要】

指定したサーボモータ単体にトルクをかけます。(ホールドした状態にします)

【引数】

種類	名称	説明	設定範囲
in	<i>id</i>	サーボモータの ID 番号	0~31
in	<i>sio</i>	SIO の値( <i>SioNum</i> )	

【戻り値】

値	説明
true	通信完了

false	通信失敗
-------	------

#### 【使用例】

```
rcb4.setHoldSingleServo (2, rcb4.SIO5_8);      //SIO5-8 の ID:2 をホールドに
```

#### 【説明】

ID,SIO で指定したサーボモータにトルクをかけます。Free 状態からトルクをかける状態に復帰します。  
Free を指定すると Hold のコマンドが発行されるまで Free 状態を保ち続けます。

### 複数のサーボモータを動かす `setServoPos()`

#### 【書式】

```
bool setServoPos (ServoData servoDatas[], byte servoCount, byte frame)
```

#### 【概要】

指定したサーボモータ複数を実作させます。

#### 【引数】

種類	名称	説明	設定範囲
in	<i>servoDatas[]</i>	サーボモータのデータをひとかたまりにしたものを複数個	
in	<i>servoCount</i>	servoDatas のサーボの数	1~35
in	<i>frame</i>	フレーム数(多いほどゆっくり動く)	1~255

#### 【戻り値】

値	説明
true	通信完了
false	通信失敗

#### 【使用例】

```
Rcb4BaseClass::ServoData servoData[2];  //サーボのデータ群

servoData[0].Id = 2;
servoData[0].Sio =rcb4.SIO5_8;
servoData[0].Data = 7500;

servoData[1].Id = 1;
servoData[1].Sio =rcb4.SIO1_4;
servoData[1].Data = 7500;
```

```
rcb4.setServoPos (servoData,2,50);    //指定したサーボモータの角度を変更する  ServoData 配列、配列数、フレーム数
```

#### 【説明】

servoData に入っている複数のサーボモータの情報をもとに Frame 数を使用して動かします。

#### 【注意】

モーション再生中に指定した場合はモーションの角度データが優先されます。

### 複数のサーボモータをフリーにする `setFreePos()`

#### 【書式】

```
bool setFreePos (ServoData servoDatas[], byte servoCount)
```

#### 【概要】

指定したサーボモータ単体をフリー（脱力した状態）にします。

#### 【引数】

種類	名称	説明	設定範囲
in	<i>servoDatas[]</i>	サーボモータのデータをひとかたまりにしたものを複数個	
in	<i>servoCount</i>	servoDatas のサーボの数	1~35

#### 【戻り値】

値	説明
true	通信完了
false	通信失敗

#### 【使用例】

```
Rcb4BaseClass::ServoData servoData[2];    //サーボのデータ群

servoData[0].Id = 2;
servoData[0].Sio =rcb4.SIO5_8;
servoData[1].Id = 1;
servoData[1].Sio =rcb4.SIO1_4;

rcb4.setFreePos(servoData,2);    //指定したサーボモータをフリーにする  ServoData 配列、配列数
```

#### 【説明】

servoDatas[]で指定したサーボモータをフリーにします。

【注意】

Free を指定すると Hold のコマンドが発行されるまで Free 状態を保ち続けます。

servoDatas の Data が 0x8000 に書き換わります。

複数のサーボモータのトルクをかける `setHoldPos()`

【書式】

bool setHoldPos (ServoData servoDatas[], byte servoCount)

【概要】

指定した複数のサーボモータにトルクをかけます。(ホールドした状態にします)

【引数】

種類	名称	説明	設定範囲
in	<i>servoDatas[]</i>	サーボモータのデータをひとかたまりにしたものを複数個	
in	<i>servoCount</i>	servoDatas のサーボの数	1~35

【戻り値】

値	説明
true	通信完了
false	通信失敗

【使用例】

```
Rcb4BaseClass::ServoData servoData[2];    //サーボのデータ群
servoData[0].Id = 2;
servoData[0].Sio =rcb4.SIO5_8;
servoData[1].Id = 1;
servoData[1].Sio =rcb4.SIO1_4;

rcb4.setHoldPos(servoData,2);    //指定したサーボモータにトルクをかける  ServoData 配列、配列数
```

【説明】

servoDatas[]で指定したサーボモータをフリーにします Free 状態からトルクをかける状態に復帰します。

Free を指定すると Hold のコマンドが発行されるまで Free 状態を保ち続けます。

【注意】

servoDatas の Data が 0x7FFF に書き換わります。

指定したサーボモータの角度を取得する      getSinglePos()

【書式】

int getSinglePos (byte id, byte sio)

【概要】

指定したサーボモータのポジションデータを取得します

【引数】

種類	名称	説明	設定範囲
in	<i>id</i>	サーボモータの ID 番号	0~31
in	<i>sio</i>	SIO の値( <a href="#">SioNum</a> )	

【戻り値】

値	説明
	指定したサーボモータのポジションデータ
-1	通信失敗

【使用例】

```
int servoPos = getSinglePos(2, rcb4.SIO5_8);    //SIO5-8 の ID2 のポジションデータを取得する
```

【説明】

ID,SIO で指定したサーボモータのポジションデータを取得します。

【注意】

この値は、RCB-4 からポジションデータを送信したときのサーボからの返事に含まれるデータです。サーボがすでに移動している場合があるため、正確な現在値ではない場合があります。

複数のサーボモータのスピード値を変更      setServoSpeed()

【書式】

bool setServoSpeed (ServoData servoDatas[], byte servoCount)

【概要】

指定したサーボモータのスピードを変更します。

【引数】

種類	名称	説明	設定範囲
in	<i>servoDatas[]</i>	サーボモータのデータをひとかたまりにしたものを複数個	
in	<i>servoCount</i>	servoDatas のサーボの数	1~35

【戻り値】

値	説明
true	通信完了
false	通信失敗

【使用例】

```
Rcb4BaseClass::ServoData servoData[2];    //サーボのデータ群

servoData[0].Id = 2;
servoData[0].Sio =rcb4.SIO5_8;
servoData[0].Data = 127;
servoData[1].Id = 1;
servoData[1].Sio =rcb4.SIO1_4;
servoData[1].Data = 64;

rcb4.setServoSpeed(servoData,2);           //指定したサーボモータのスピードを変更する  ServoData 配列、配列数
```

【説明】

servoDatas[]で指定したサーボモータのスピード(出力の上限値)を変更します。

servoDatas の Data にはスピード値を入れます。スピード値は、1~127 で指定できます。数字が大きい方が早くなります。

【注意】

スピードを落とした場合、トルクも減少しますので、移動速度を変更する場合はフレームを変更することをお勧めします。

**複数のサーボモータのストレッチ値を変更    setServoStretch()**

【書式】

bool setServoStretch (ServoData servoDatas[], byte servoCount)

【概要】

指定したサーボモータのストレッチを変更します。



【引数】

種類	名称	説明	設定範囲
in	<i>servoDatas[]</i>	サーボモータのデータをひとかたまりにしたものを複数個	
in	<i>servoCount</i>	<i>servoDatas</i> のサーボの数	1~35

【戻り値】

値	説明
true	通信完了
false	通信失敗

【使用例】

```
Rcb4BaseClass::ServoData servoData[2];    //サーボのデータ群

servoData[0].Id = 2;
servoData[0].Sio =rcb4.SIO5_8;
servoData[0].Data = 127;
servoData[1].Id = 1;
servoData[1].Sio =rcb4.SIO1_4;
servoData[1].Data = 64;

rcb4.setServoStretch(servoData,2);    //指定したサーボモータのスピードを変更する  ServoData 配列、配列数
```

【説明】

*servoDatas[]* で指定したサーボモータのストレッチ(出力軸の柔らかさ)を変更します。

*servoDatas* の *Data* にはストレッチ値を入れます。1~127 の間で指定できます。数字が大きいくほど出力軸は固くなります。

ホームポジションなど、通常の姿勢の場合はストレッチを半分ほどに設定するとサーボへの負荷を減らすことができます。また、ハンチング（サーボのブルブルとした症状）を抑える効果もあります。

## 改訂履歴

バージョン	日付	詳細
1.0	2018/02/01	第 1 版を作成。
2.0	2025/1/10	第 2 版を作成。