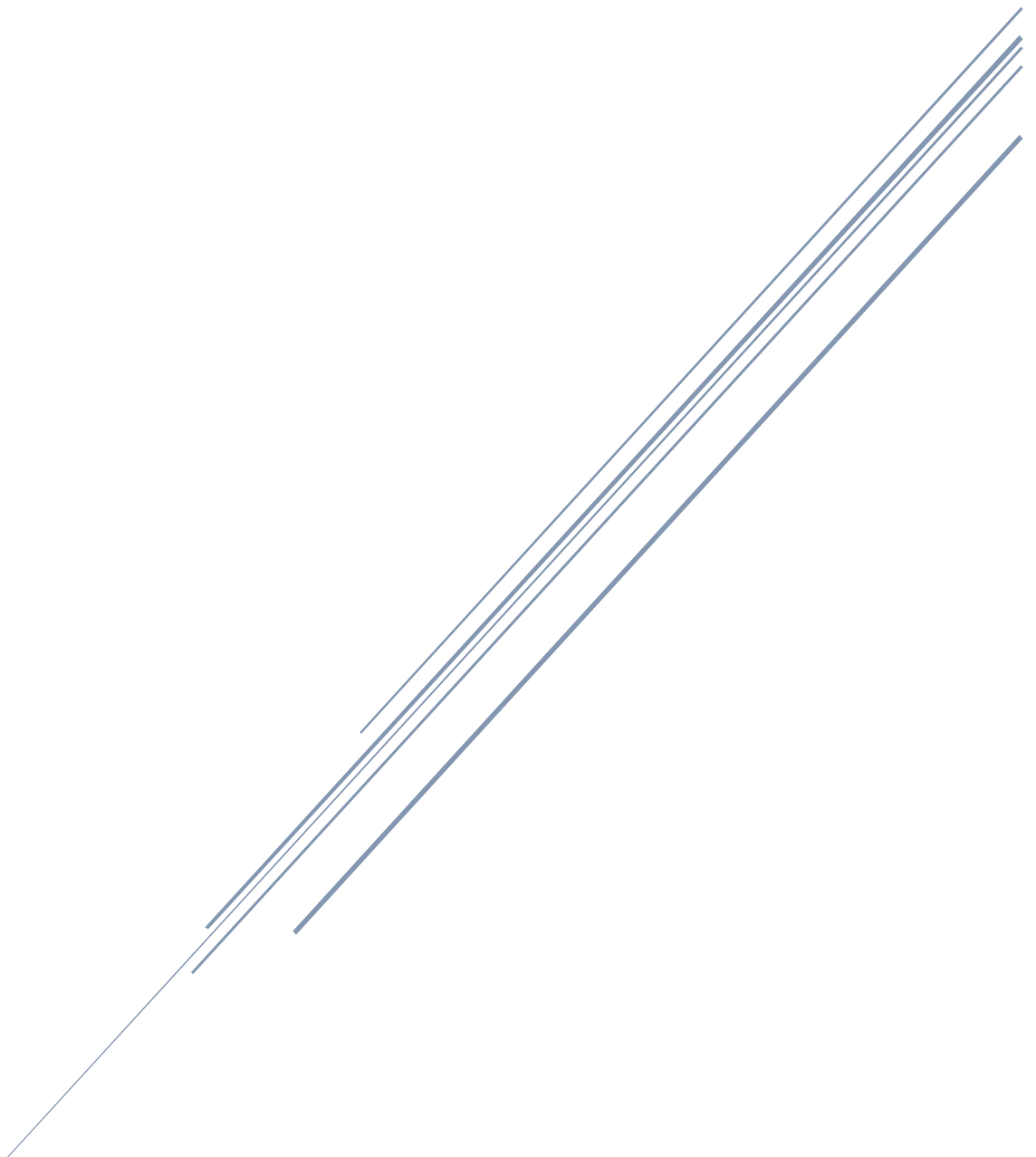


# 取扱説明書

データフローfor mruby/c の運用について



mruby/c コード生成器とその運用方法について簡単に説明します

## もくじ

---

.	
開発ツール「Node-Red」とマイコン用 Ruby コードの生成器について .....	2
「データ for mruby/c」の概要 .....	2
「Node-Red」について .....	2
Ruby コード生成器の仕組みについて .....	3
Ruby ソースコードの構成 .....	5
ノードデータベース .....	5
バッファ .....	6
データ制御部 .....	6
ノードプログラム .....	6
全てのノードに対し順々と動作信号を出すメソッド(ノードを呼び出す司令塔) .....	7
「データフロー for mruby/c」運用方法 .....	7
開発環境・使用機器 .....	7
「Node-Red」と Ruby コード生成器の導入 .....	7
「Node-Red」：マイコン用のノードライブラリの導入開発環境・使用機器 .....	8
「Node-Red」：プログラミングルール .....	9
「Node-Red」：マイコン用のノード仕様 .....	10
json 形式ファイルから Ruby コードの生成と運用の仕方 .....	15
デモンストレーション .....	18
オンボード LED の点滅 .....	18
光センサを用いた LED の点灯 .....	20
Grove 温湿度センサによる温度測定とターミナル表示 .....	22

## 開発ツール「Node-Red」とマイコン用 Ruby コードの生成器について

### 「データ for mruby/c」の概要

本ソフトウェアは「Node-Red」の一部の機能を利用し、Ruby コード生成器を通して mruby/c が搭載されているマイコン用の Ruby コードの生成を行います。「Node-Red」上では専用ノードをいくつか用意しており、これらを駆使しマイコンに様々な処理をさせることが可能です。また、外部ソフトウェアとして「mruby/c IDE」の開発環境を使用しているため、生成した Ruby コードを mruby コードにコンパイルでき、RBoard に転送することができます。これにより、ユーザーは最初からコーディングする必要はなく、ローコードでプログラミングが行えるようになります。

### 「Node-Red」について

「Node-Red」は、IoT アプリケーション開発における「ハードウェアデバイス」「API」「オンラインサービス」などを相互接続するために IBM により開発されたデータフローベースのビジュアルプログラミング開発ツールです。ブラウザベースの UI を持ち、ノードと呼ばれる各機能がまとめられたブロックを配置していき、これらを線でつなぎ合わせデータを送信・受信することで様々な処理を行わせることができます(Fig.1)。また、「Node-Red」は Node.js 上で実装されているため、Node.js が動かせる環境であれば実行することができ、さらには IBM Cloud や AWS といったクラウドサービス上でも動かすことができます。

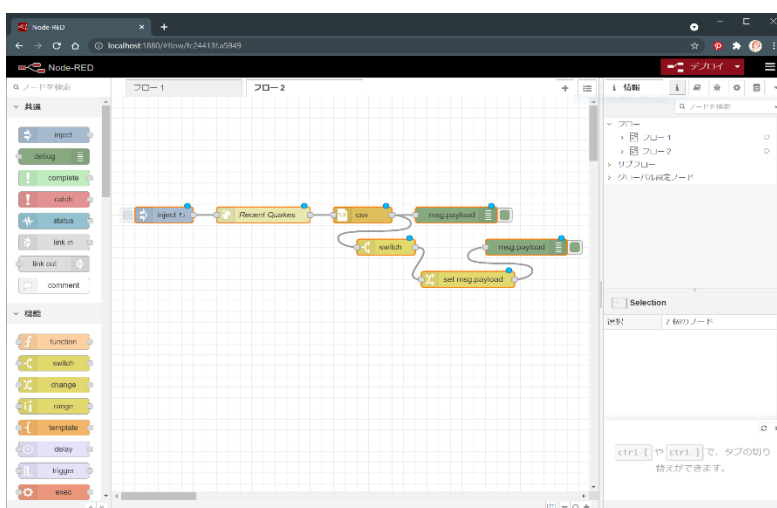


Fig. 1 Node-Red エディタ画面

「Node-Red」の特徴の1つとして、ノードを自作することができます。ノードの外観と設定項目のレイアウトなどを担う html とノード内の処理を担う Javascript で1つのノードが形成されています。

## Ruby コード生成器の仕組みについて

「Node-Red」で作成したプログラム(データフロー)は、json データとして保存されています。json データには、「Node-Red」のエディタ画面で配置されたノードの種類、接続関係、ノードの識別 ID やノードのプロパティなどが保存されています。Ruby コード生成器はこの json データに基づき、Ruby のソースコードを生成していきます。

Ruby コード生成器の動作は以下のフローチャートのような事を行っています。

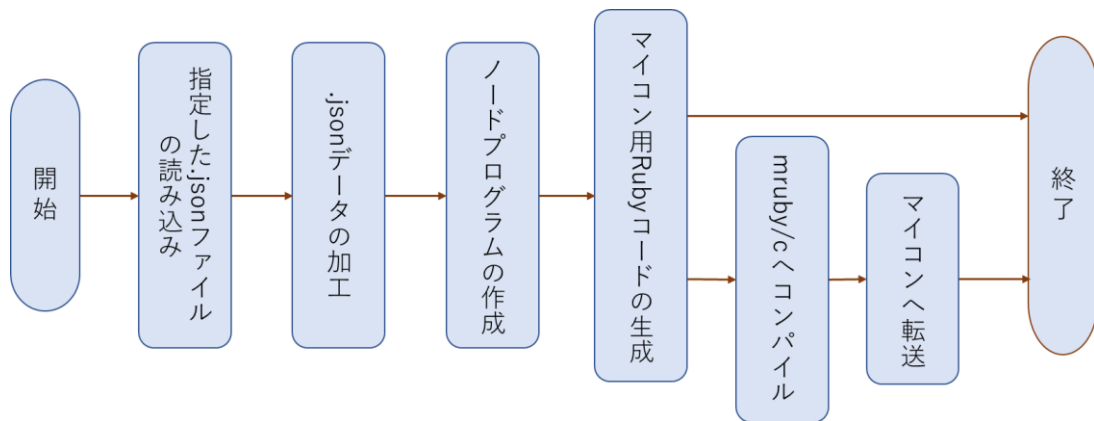


Fig. 2 Ruby コード生成器 フローチャート

まず、Ruby コード生成器は、json 形式ファイルを読み込み、Ruby のソースコード生成に不必要な情報の削除といった json データの加工を行います。その後、配置された全ノードの識別 ID ごとに、それぞれのノードのプロパティ情報をまとめていき、最終的には全てのノード情報をハッシュ形式のデータ(ノードデータベース)に加工します。以降は、ハッシュ形式のノードデータベースを用いて、Ruby コードを生成していくことになります。ノードデータベースから、「Node-Red」上で設計された動作を必要最小限のプログラムで実装するために、ノードプログラムを作成します。ノードプログラムとは、ノードごとに機能をまとめたプログラムになります。なお、一部のノードでは必要最低限の機能しかまとめられていません。例えば、「LED ノード」では LED を点灯させるために必要な初期設定、ピンの出力制御、オンボード LED の制御がまとめられています。しかし、例えばユーザーはオンボード LED の機能しか使わない場合、その他の機能(ピンの初期設

定、ピンの出力制御)は不必要なプログラムとなります。そのため、「Node-Red」上でユーザーが設定した機能だけを使用するために、必要最低限のプログラムを作成します。

Fig.3 はノードプログラムの作成・更新に関して、inject ノード、I2C ノード、LED ノード(オンボードの機能のみ使用)がノードデータベースにあった場合のフローチャートです。inject ノード、I2C ノードは部分的な機能の利用などはないため、各ノードで用意されたノードプログラムを抽出します。LED ノードは部分的な機能を有するので、「CreateLED.rb」を通して LED のノードプログラムを作成します。

これらにより、ノードごとの機能を実装しています。

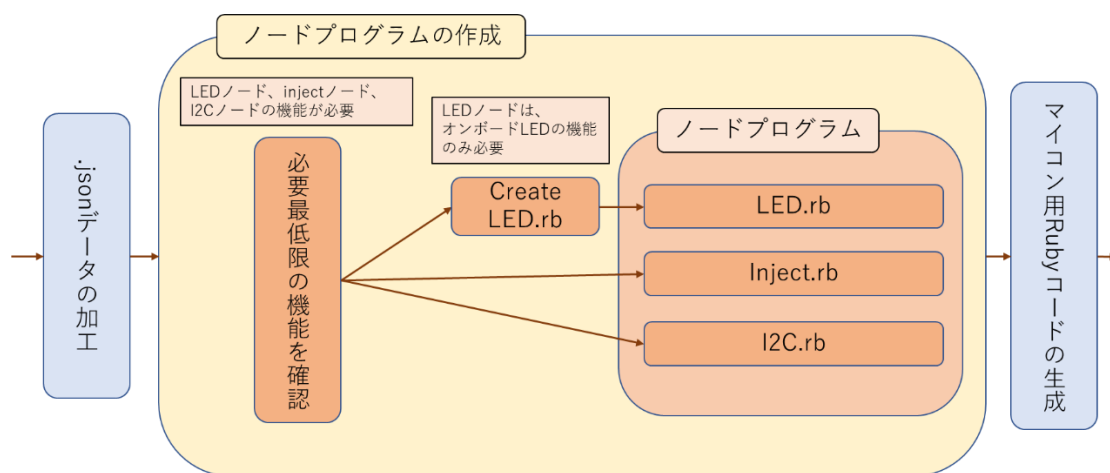


Fig. 3 ノードプログラムの作成

Ruby コード生成時、以下のような手順でプログラムを生成していきます。

- ①「Node-Red」で配置されたノードのデータベース(ハッシュ形式)の記述
- ②ノード間のデータやり取りなどを制御する情報処理部の記述
- ③各ノードに対応した設定や挙動を行うための処理部(ノードプログラム)の記述
- ④全てのノードに対し順々と指令を出す司令塔の記述

①では、json 形式のデータから変換されたノードデータベースを記述しています。②ではノード間のデータのやり取りを行うバッファとその制御に関するプログラムを記述します。③では、ノードデータベースに存在するノードに応じて用意したノードプログラムを記述していきます。最後に④では、全てのノードを順々に呼び出していくメインメソッドを記述します。

Ruby コード生成器は、実行時オプションにより mruby コードへのコンパイルや Rboard へのバイトコードの転送ができます(詳細は「データフロー for mruby/c」運用方法の章)。

## Ruby ソースコードの構成

json ファイルから生成されたソースコードは、Fig.4 のような構成をしています。

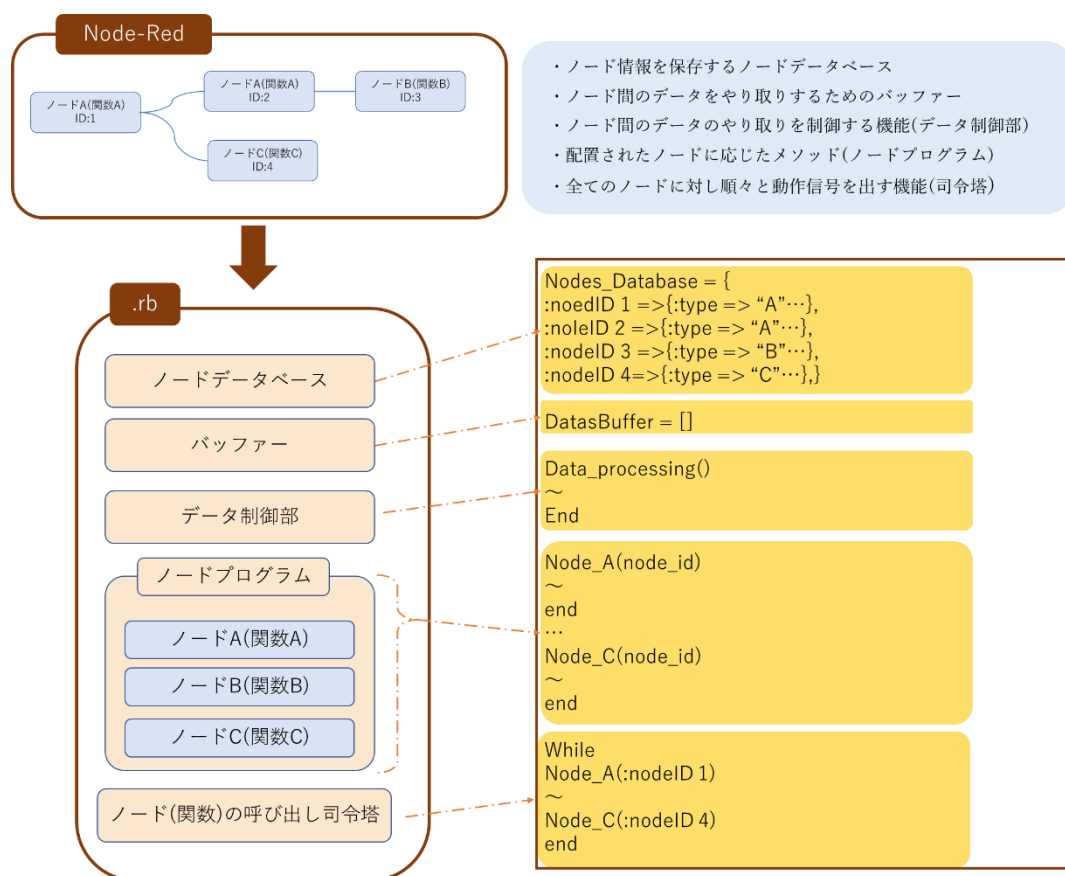


Fig. 4 生成されるソースコードの機能的な構成

### ノードデータベース

ノードデータベースは、配置された全てノードの識別 ID、種類、接続状況、設定項目を保存しています。ノードに関する処理は、このノードデータベースから参照されます。データベースはハッシュ形式で以下のように管理されています。

```
{ノード ID => {項目名 1 => 値, 項目名 2 => 値...}}
```

## バッファ

データ制御部で処理されたノードのデータが格納されます。1つのデータは配列形式で  
[ノード ID, 次に接続されたノード ID, 値] で管理されています。

## データ制御部

ノード間のデータのやり取りを制御するメソッドです。データ制御部は3つのモードがあり、「get”、“delete”、“create”」があります。それぞれのモードで呼び出された際、以下のような処理を行います。

”get”: バッファから自ノード宛のデータの有無を確認し、そのデータを取り出す

“delete”: バッファから自ノード宛のデータの有無を確認し、削除する

“create”: 自ノードの次のノード宛にデータを生成し、バッファに書き込む

この処理部により、ノード間のデータやり取りを実現しています。

## ノードプログラム

Fig.4にある「ノード A(関数 A)」などを示します。ノードの呼び出しを行う司令塔から呼びだされた場合、Node-Red で設定されたプログラムを動かします。また、プログラムを動かす前後でノード間のデータのやり取りを行うため、データ制御部が呼び出されます。全てのノードメソッドは共通して以下のようなフローチャートで処理されます。

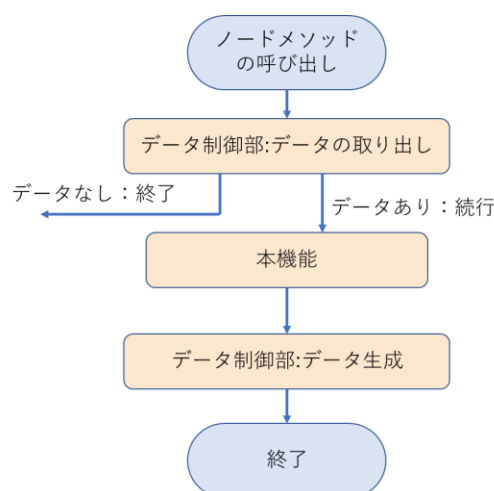


Fig. 5 ノードメソッドのフローチャート

全てのノードに対し順々と動作信号を出すメソッド(ノードを呼び出す司令塔)

各ノードの処理を行わせるようノード関数を呼び出すメソッドです。全てのノードに信号を出し終わった後も、繰り返しノード関数を呼び出し続けます。

## 「データフロー for mruby/c」運用方法

---

### 開発環境・使用機器

本ソフトウェアは、「Node.js」が動かせる環境であればご使用できます。

ソフトウェア：「Node.js v15.14.0」 「Node-red v1.3.5」 「mruby/c IDE v1.0.2」

プログラム言語：「Ruby 3.0.2」

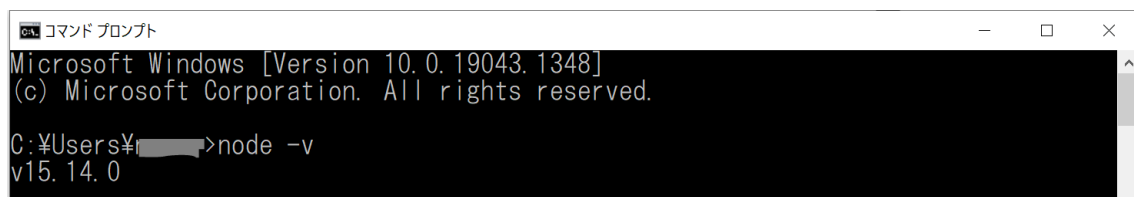
使用するマイコンボード：「Rboard」

### 「Node-Red」と Ruby コード生成器の導入

「Node-Red」を導入するために、「Node.js」を導入します。

ダウンロード URL：<https://nodejs.org/ja/download/>

コマンドプロンプトで「node -v」でバージョン確認が出来ましたら、インストールは無事完了しています。



```
コマンド プロンプト
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. All rights reserved.

C:\Users\¥[redacted]>node -v
v15.14.0
```

「Node-Red」は「node.js」に同封されている npm コマンドを利用します。下記コマンドを入力し「Node-Red」のインストールを行います。

```
npm install -g --unsafe-perm node-red
```



```
node-red
C:\Users\¥r[redacted]>npm install -g --unsafe-perm node-red
added 42 packages, removed 102 packages, changed 238 packages, and audited 267 packages in 21s
26 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

「node-red」で「Node-Red」が起動します。

```
node-red
C:\Users\¥r[redacted]>node-red
25 Nov 15:52:33 - [info]
Welcome to Node-RED
=====
25 Nov 15:52:33 - [info] Node-RED version: v2.1.3
25 Nov 15:52:33 - [info] Node.js version: v15.14.0
25 Nov 15:52:33 - [info] Windows_NT 10.0.19043 x64 LE
25 Nov 15:52:34 - [info] Loading palette nodes
25 Nov 15:52:36 - [warn] Missing node modules:
```

起動中、下記 URL にアクセスすることでブラウザが起動し、「Node-Red」のエディタ画面に移ります。

URL : <http://localhost:1880>

Ruby コード生成器は下記 URL からダウンロードしてください。

URL :

### 「Node-Red」：マイコン用のノードライブラリの導入開発環境・使用機器

「Node-Red」上でマイコン用のノードライブラリを入手します。下記 URL からダウンロードしてください。

URL :

コマンドプロンプトで、「npm install <node-red-contrib-mrubyc-rboard のパス>」を入力してください。その後、「Node-Red」を起動し、ノードパレットに Fig.6 のような mruby Rboard Nodes が追加されています。

```
C:\WINDOWS\system32\cmd.exe
C:\Users¥>npm install C:\Users¥\OneDrive¥デスクトップ¥node-red-contrib-mrbyc-rboard
up to date, audited 3 packages in 955ms
found 0 vulnerabilities
C:\Users¥nerur>
```

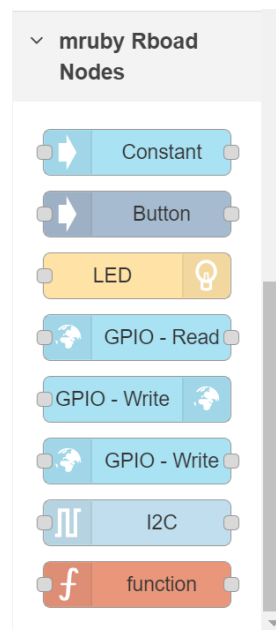





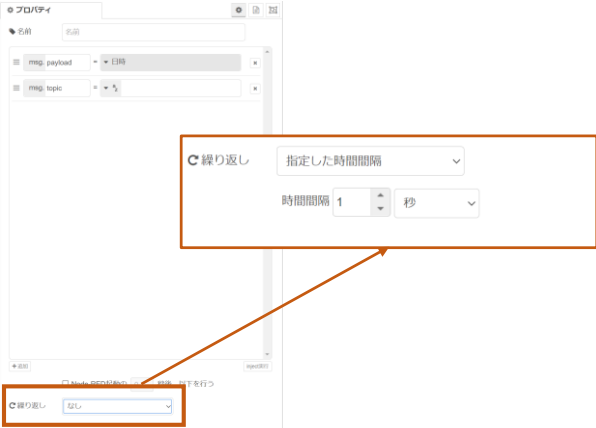
Fig. 6 マイコン用のノード

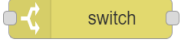
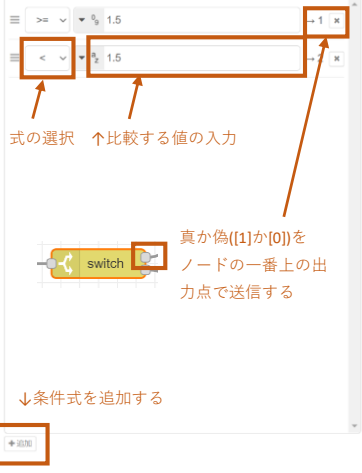
## 「Node-Red」：プログラミングルール






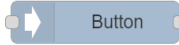

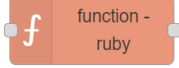
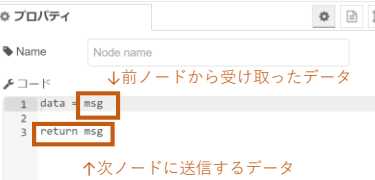
「Node-Red」上でのプログラミングは、ノードパレットからノードをドラッグ&ドロップし、ノードを配置します。配置後、ノードのプロパティを開き、様々な設定を行います。その後、ノードとノードを接続することを繰り返していくことにより、プログラムが完成します。その中、プログラミングにおいて以下のルールがあります。



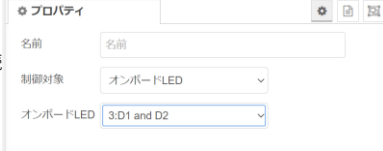

- ・「inject」ノードから必ずスタートする。
- ・各ノードの入力点への接続は基本的に 1 本まで
- ・各ノードの出力点からの接続は何本でも OK


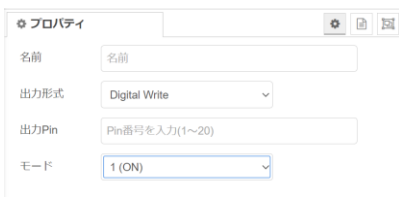

## 「Node-Red」：マイコン用のノード仕様

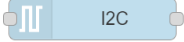

デザイン	ノード名	動作条件	送信データ	仕様
	inject	なし/一定時間経過	繰り返し設定 なし: 「1」 あり: 「1」 「0」	常に、または一定時間間隔でデータを生成する。
	<p>  <b>繰り返し</b> で「指定した時間間隔」を設定することでデータの生成を一定時間間隔で行うことが可能。また、 で時間調整もできるが、記述も可能。         </p> <p>           設定なしの場合は、常にデータ「1」を生成し続ける。            「指定した時間間隔」を設定すると、<b>inject</b>ノードがデータ「1」データ「0」交互に送信するようになる。         </p> 			

デザイン	ノード名	動作条件	送信データ	仕様
	switch	データを受け取った時	真: 「1」 偽: 「0」	受け取ったデータに対し判定を行う。
	プロパティ画面	<p>           ・左下の「追加」で条件式が追加。条件式が増え、<b>switch</b>ノードの出力点も増える            ・条件式の順番と、<b>switch</b>ノードの出力点の順番は対応している。            例えば、右図だと「&gt;= 1.5」が真である場合ノードの一番上の出力点がデータ「1」を出力する。            ・条件式のプルダウン欄で式を選択            ※「==」「!=」「&lt;」「&lt;=」「&gt;」「&gt;=」のみ対応            ・条件式のテキスト欄で比較したい値を入力(数値のみ)         </p>  <p>           式の選択 ↑ 比較する値の入力            ↓ 条件式を追加する            真か偽([1]か[0])をノードの一番上の出力点で送信する         </p>		

	ノード名	動作条件	送信データ	仕様
	delay	データを受け取った時	受け取ったデータ	設定した時間分、遅延させてデータを送信する。
	プロパティ画面	<p>・遅延時間を設定する。   で時間を設定することもできるが、記述することも可能  ※時間の単位が「秒」のみ対応。  0.1[s] 60[s] などは可能。</p>  <p>↑時間指定</p>		
	ノード名	動作条件	送信データ	仕様
	constant	データを受け取った時	設定した定数	設定された定数をデータとして送信する。
	プロパティ画面	<p>・送信したい定数を入力する。</p> 		
	ノード名	動作条件	送信データ	仕様
	Buttuon	データ「1」を受け取った時	ボタンの入力状態 オン:「1」 オフ:「0」	設定されたボタンの入力状態を読み取り、データとして送信する。
	プロパティ画面	<p>@ オンボードSW 使用する/使用しない  ・使用する：設定項目なし  ・使用しない：GPIOピンの設定  @対象Pin番号  ボタンモジュールと接続するポート番号の入力  @内部抵抗選択  「Pullなし」「Pull up」「Pull down」が選択可能</p> 		
	ノード名	動作条件	送信データ	仕様
	function-ruby	データを受け取った時	return で返される値	Rubyでコーディングが行える。
	プロパティ画面	<p>@コード  rubyコードを自由に記述できる。  受け取ったデータは、変数「msg」に格納されている。  また、次のノードに送るデータは、return で返された値を用いる。</p>  <p>↓前ノードから受け取ったデータ  ↑次ノードに送信するデータ</p>		


	ノード名	動作条件	送信データ	仕様
	LED	データ「1」または「0」を受け取った時	なし	設定されたLEDに対して、オン・オフ制御を行う。
	LED	<p>@制御対象</p> <ul style="list-style-type: none"> <li>・オンボードLED：ボードに搭載されてある4つのLEDを点灯させる。点灯の仕方は二進数の形式で点灯する。</li> <li>・Pinと接続しているLED：Pinと外部接続しているLEDを制御する。</li> </ul> <p>@Pin番号</p> <p>LEDと接続するポート番号の入力</p> <p>@LEDのON/OFFモード</p> <ul style="list-style-type: none"> <li>・0(OFF): データ「1」を受け取ると消灯</li> <li>・1(ON): データ「1」を受け取ると点灯</li> <li>・入力に従って(ONとOFFを切り替える): データ「0」を受け取ると消灯、データ「1」を受け取ると点灯と、1つのノード</li> </ul>	<p>↓オンボードLEDを使う場</p>  <p>↓外部のLEDを使う場合</p> 	
	プロパティ画面			
	ノード名	動作条件	送信データ	仕様
	GPIO-Read	データ「0」以外の受け取った時	Digital: 「1」 「0」 Analog: 電圧値	GPIOピンの入力に関する制御を行い、読み取った値をデータとして送信する。「Digital read」「Analog read」の2つのモードがある。
	GPIO-Read	<p>@入力形式</p> <p>Digital read: デジタル入力として設定する</p> <p>Analog read: アナログ入力として設定する</p> <p>@入力Pin</p> <p>設定したいポート番号を入力/選択する</p>		
	プロパティ画面			

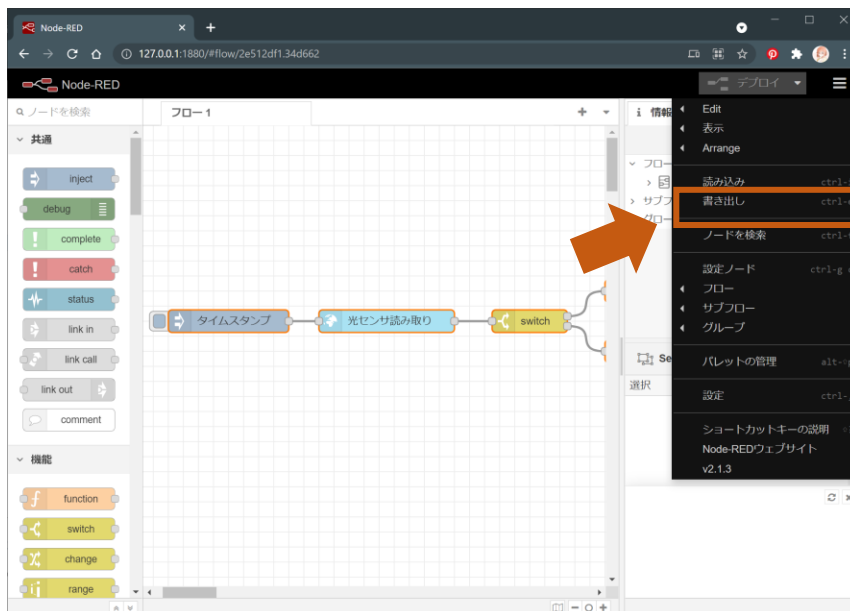
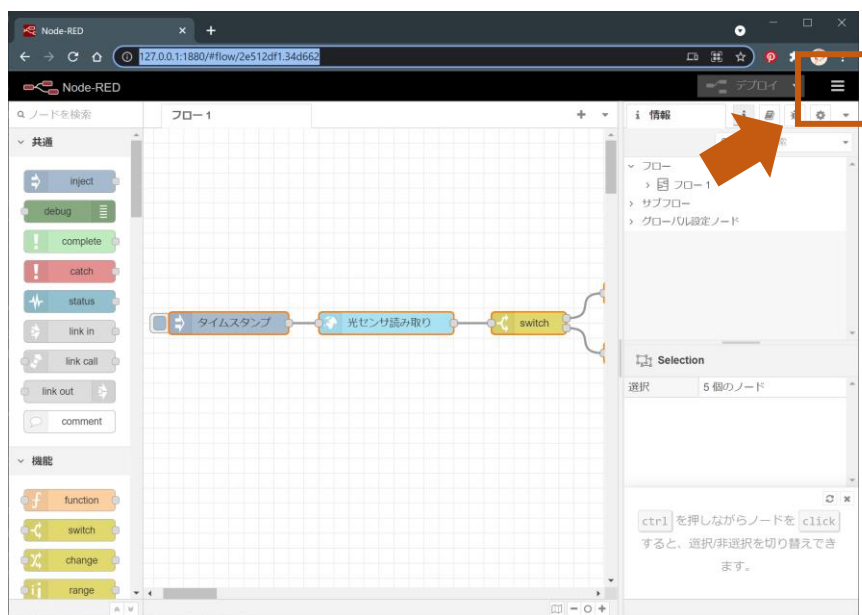
	ノード名	動作条件	送信データ	仕様
	GPIO-Write	データ「1」または「0」を受け取った時	なし	GPIOピンの出力に関する制御を行う。「Digital write」「PWM」の2つのモードがある。
	プロパティ画面	<div> <p><b>@出力形式</b>  <b>Digital write:</b> デジタル出力として設定する  <b>PWM:</b> PWM出力として設定する</p> <p><b>@出力Pin</b>            設定したいポート番号を入力/選択する</p> <p><b>@周期設定</b>  <b>time [ms]</b>と<b>double(倍率)</b>を入力し、周期設定を行う。  <b>time*double</b>で計算された数値が周期となる。</p> <p><b>@デューティー比[%]</b>            デューティー比を設定する</p> </div> <div> <p>↓デジタル出力モード時</p>  <p>↓PWM出力モード時</p>  </div>		

	ノード名	動作条件	送信データ	仕様
	I2C	データ「0」以外の受け取った時	モジュールから受け取ったバイトコード(配列)	I2C通信を行うモジュールの制御を行う。
	プロパティ画面	<p>             @slave address              16進数形式でスレーブアドレスを入力する              @コマンドバーの設定              ・左下の「追加」でコマンドバーを追加できる              ・コマンドバー左のプルダウン欄の「Write」「Read」でコマンドをモジュールに送るか受け取るかを設定する。              @[Write]              ・ register address :              アクセスするモジュールのレジスタを設定する(16進数)              ・ command :              コマンドを設定する(16進数)              ・ delay : コマンドを実行した後の遅延時間を設定する              @[Read]              ・ register address :              アクセスするモジュールのレジスタを設定する(16進数)              ・ byte:              受け取るバイトコードのバイト数を指定する              ・ delay : コマンドを実行した後の遅延時間を設定する                [Read]で受け取ったバイトコードのデータは配列型である。そのため、I2Cノードから出力されるデータは配列で渡される。              また、[Read]は2回以降仕様すると、最後に実行した[Read]のデータがノードから出力されるデータとなる。           </p>	<div> <div>             register address モジュールのアドレスを指定           </div> <div>             command コマンドを設定           </div> <div>             delay 遅延させる           </div>  <div>             「Write」「Read」の切り替えを行う           </div> <div>             byte 受け取るバイトコードの容量を設定           </div> <div>             ↓条件式を追加する           </div> </div>	

## json 形式ファイルから Ruby コードの生成と運用の仕方

「Node-Red」で作成したプログラムから json データを抽出します。「Node-Red」エディタ

画面右上のをクリックしメニューを開き、「書き出し」を選びます。





Node-RED

127.0.0.1:1880/#flow/2e512df1.34d662

Node-RED

フロー1

フローを書き出し

書き出し 選択したフロー 現在のタブ 全てのタブ

クリップボード フローを書き出し JSON

```
[
  {
    "id": "d6aad3f6.24377",
    "type": "inject",
    "z": "2e512df1.34d662",
    "name": "",
    "props": [
      {
        "p": "payload"
      },
      {
        "p": "topic",
        "vt": "str"
      }
    ],
    "repeat": "",
    "crontab": "",
    "once": false,
  }
]
```

インデントのないJSONフォーマット インデント付きのJSONフォーマット

中止 ダウンロード 書き出し

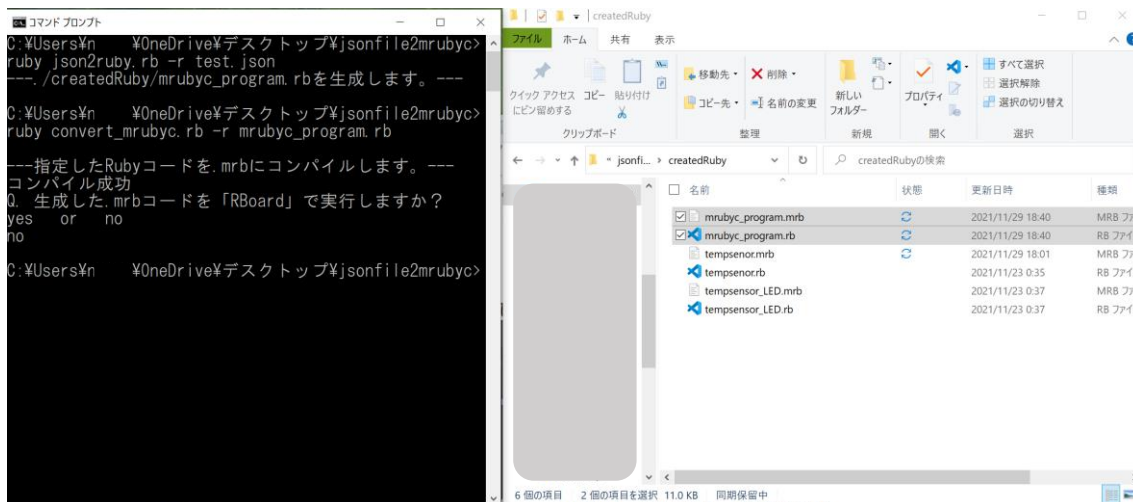
[16]

```
jsonfile2mrubyc> ruby json2ruby.rb -r test.json
```

「jsonfile2mrubyc > createdRuby」で mrubyc\_program.rb が確認できたら、

```
jsonfile2mrubyc> ruby convert_mrubyc.rb -r mrubyc_program.rb
```

で Ruby コードをコンパイルし、mruby/c コードを生成できます。



※「json2ruby.rb」「convert\_mrubyc.rb」は以下のオプションを持っています。

プログラム名	オプション一覧
json2ruby.rb	-r filename :入力した.json ファイルを読み込む -w filename :入力した.rb ファイルを作成する -d :コンパイル&マイコンへの転送を行う
convert_mrubyc.rb	-r filename :入力した.rb ファイルを読み込む

また、マイコンへの転送で使う USB ポートは、「convert\_mrubyc.rb」中の

```
< convert_mrubyc.rb >  
~  
$Poaat_Num = "COM6" #RBoard を USB 接続したときのポート番号  
~
```

で設定できます。

また、生成された Ruby コードはユーザーでも確認することができるため、mruby/c IDE といった統合開発環境への転用も可能です。

## デモンストレーション

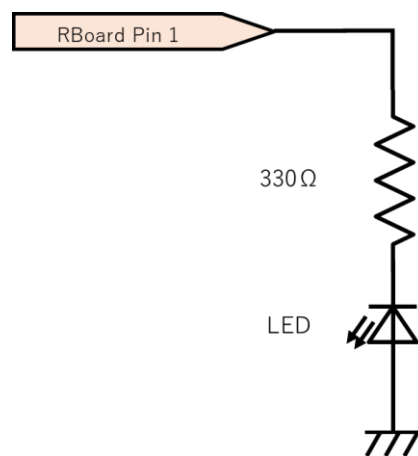
### オンボード LED の点滅

1 秒ごとに接続された LED が点灯・消灯を繰り返すプログラムです。

#### <使用機材>

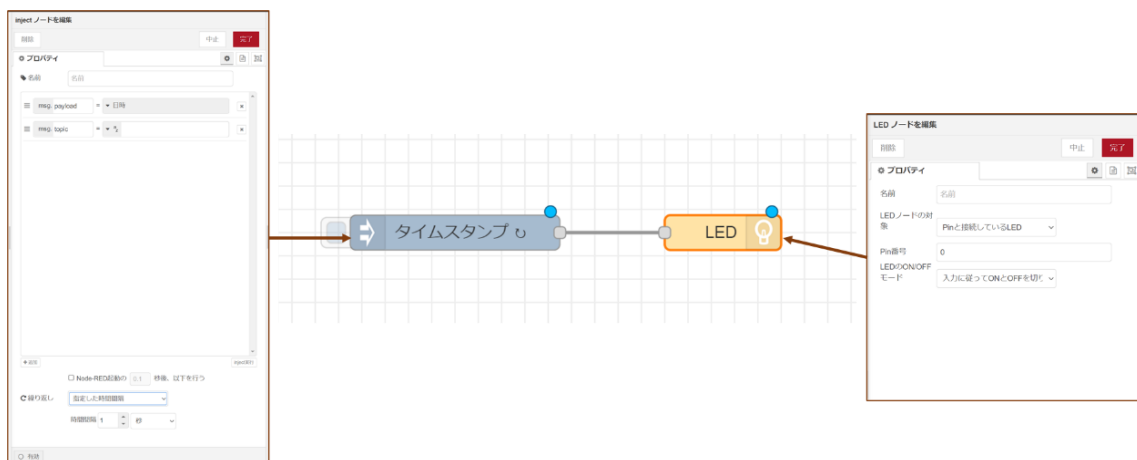
・ RBoard ・ MicroUSB ケーブル ・ LED ・ 抵抗  $330\Omega$

#### <回路>



#### <プログラム>

・ inject ノード×1      ・ LED ノード×1



inject ノードは 1 秒間隔でデータ「1」データ「0」を送信します。データを受け取った LED ノードはデータ「0」を受け取ると消灯、データ「1」を受け取ると点灯するようデジタル出力を行います。

<コマンド>

- ①作成したデータフローから.json ファイルを抽出します。
- ②「jsonfile2mrubyc > jsonFile」に「LEDtikatika.json」を作成します。
- ③プログラムを実行します。(今回は作成するプログラム名を指定します)

```
jsonfile2mrubyc> ruby json2ruby.rb -r LEDtikatika.json -w LEDtikatika.rb
```

- ④生成したプログラムをコンパイルしマイコンに転送します。

```
jsonfile2mrubyc> ruby convert_mrubyc.rb -r LEDtikatika.rb
```

※③と④をまとめて実行するなら、オプションとして「-d」を付けます。

```
jsonfile2mrubyc> ruby json2ruby.rb -r LEDtikatika.json -w LEDtikatika.rb -d
```

- ⑤書き込み完了すれば、接続された LED が 1 秒ごとに点滅します。

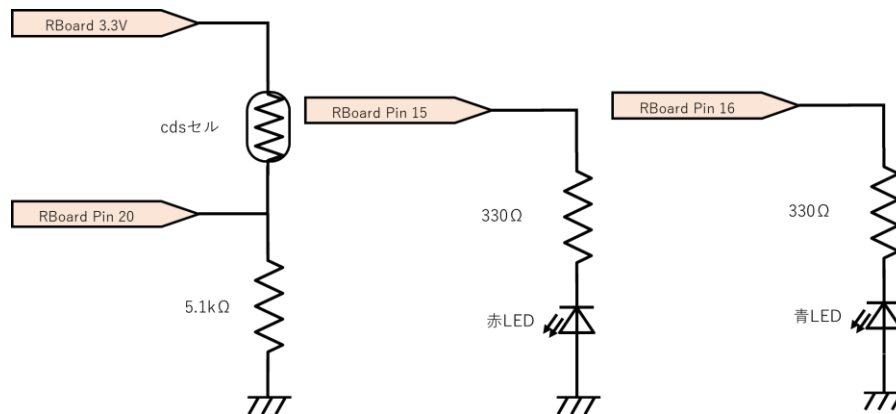
## 光センサを用いた LED の点灯

光センサ(CdS セル)を用いて LED 制御を行います。暗くなると、青色 LED が点灯し、明るいとき赤色 LED が点灯するプログラムです。

### <使用機材>

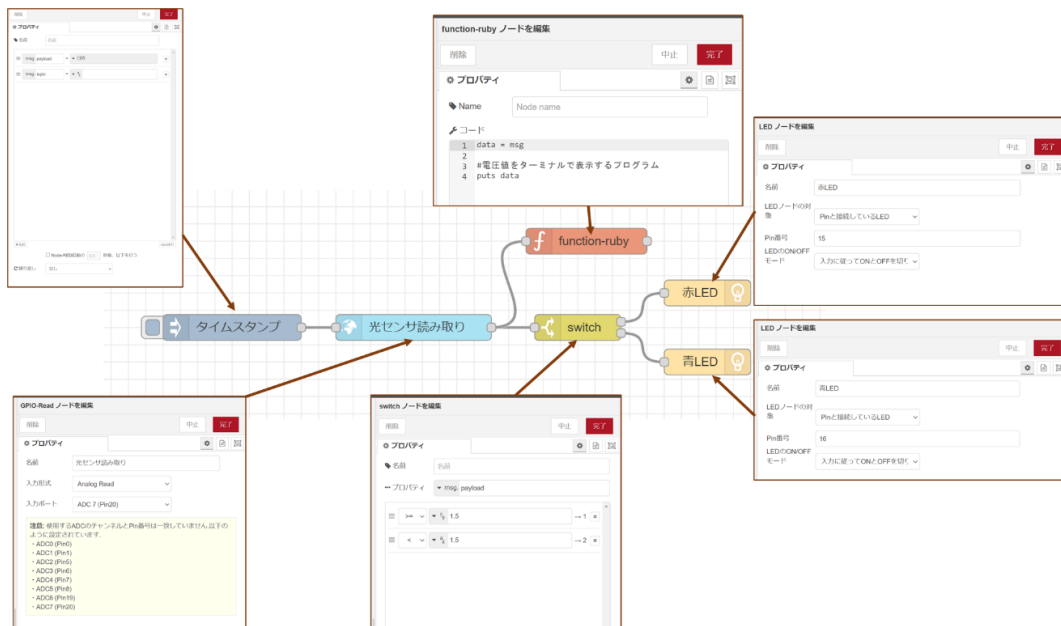
・ RBoard    ・ MicroUSB    ・ CdS セル    ・ LED×2    ・ 抵抗 330Ω×2

### <回路>



### <プログラム>

・ inject ノード×1    ・ GPIO-Read ノード×1    ・ function-ruby ノード×1  
・ switch ノード×1    ・ LED ノード×2



Inject ノードは「繰り返し」の設定を行っていないので、常時データ「1」が送信されます。データ「1」を受け取った GPIO-Read ノード(光センサ読み取り)はアナログ入力設定を行っているため電圧値を読み取り、送信します。送信された値は function-ruby ノードと switch ノードが受け取ります。function-ruby ノードはターミナル上に電圧値が表示され、switch ノードは電圧値を使って判定を行います。1.5v 以上であれば赤 LED が、1.5v 未満であれば青 LED が点灯します。

<コマンド>

- ①作成したデータフローから.json ファイルを抽出します。
- ②「jsonfile2mrubyc> jsonFile」に「cds\_LED.json」を作成します。
- ③プログラムを実行します。(今回は作成するプログラム名を指定します)

```
jsonfile2mrubyc> ruby json2ruby.rb -r cds_LED.json -w cds_LED.rb
```

- ④生成したプログラムをコンパイルしマイコンに転送します。

```
jsonfile2mrubyc> ruby convert_mrubyc.rb -r cds_LED.rb
```

※③と④をまとめて実行するなら、オプションとして「-d」を付けます。

```
jsonfile2mrubyc> ruby json2ruby.rb -r cds_LED.json -w cds_LED.rb -d
```

- ⑤書き込み完了すれば、光センサによる LED 制御が行われます。また、ターミナルソフトを利用すると、CdS セルによる電圧値(GPIO-Read ノードの出力)が表示されます。

ターミナルについて：<https://yoshihiroogura.github.io/RBoardDocument/console.html>

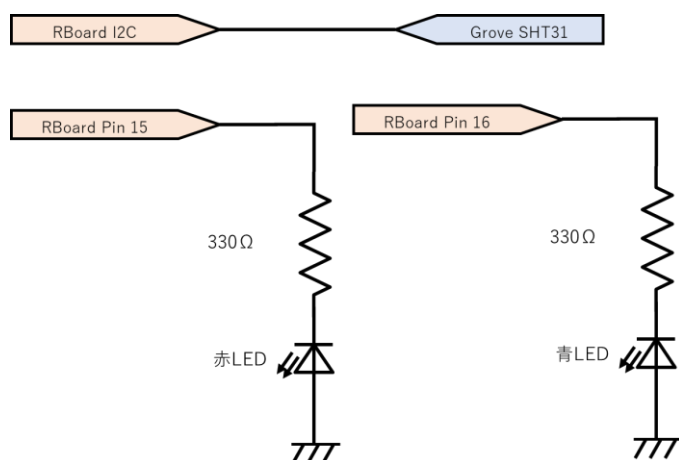
## Grove 温湿度センサによる温度測定とターミナル表示

Grove 温湿度センサ(SHT31)を使い、温度によるターミナル表示と LED 制御を行います。Grove センサの制御は I2C ノードを用いています。

<使用機材>

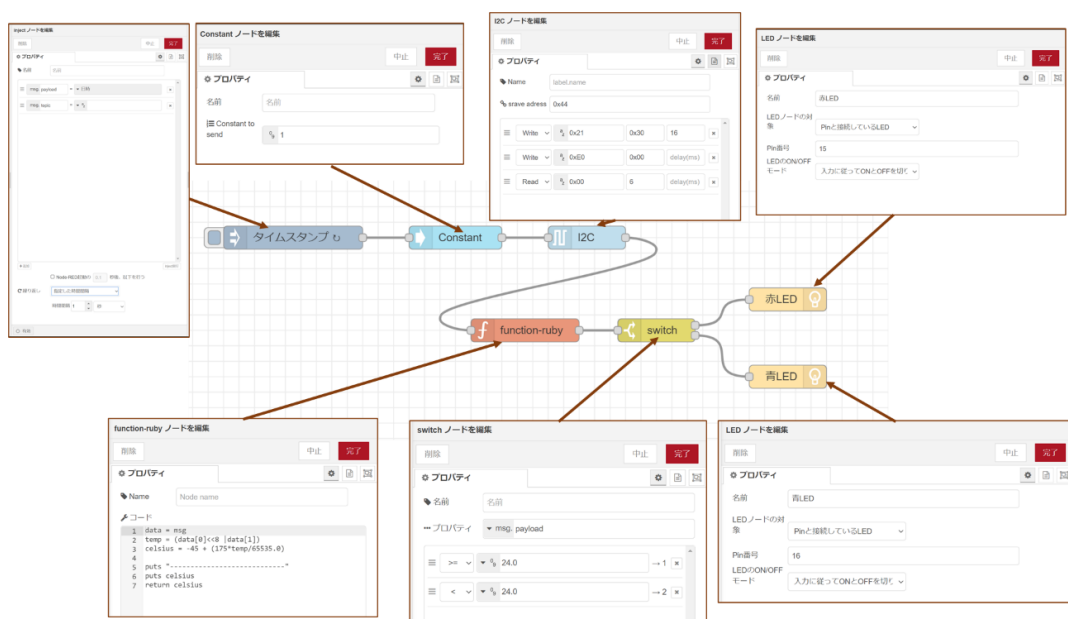
・ RBoard    ・ MicroUSB    ・ 温湿度センサ(SHT31)    ・ LED×2    ・ 抵抗 330Ω×2

<回路>



<プログラム>

・ inject ノード×1    ・ Constant ノード×1    ・ I2C ノード×1    ・ function-ruby ノード×1  
・ switch ノード×1    ・ LED ノード×2



Inject ノードは、1 秒ごとにデータ「0」データ「1」を交互に送信します。しかし、I2C ノードはデータ「0」を受信しても動作しないため、Constant ノードで受け取ったデータをデータ「1」にしています。データ「1」を受け取った I2C ノードは Grove 温湿度センサにコマンドを送信し、温度値を受け取ります。しかし、受け取った温度値はバイトコードであるため摂氏に変換する必要があります。そのため、変換処理は function-ruby ノードで行います。また、ターミナル上の温度値の表示も function-ruby ノードでプログラムしています。function-ruby ノードは次のノードに摂氏温度を送信し、switch ノードで判定を行います。24 度以上であれば赤色 LED が点灯し、24 度以下であれば青色 LED を点灯を行います。

①作成したデータフローから.json ファイルを抽出する。

②「jsonfile2mrubyc > jsonFile」に「tempsensor\_LED.json」を作成します。

③プログラムを実行します。(今回は作成するプログラム名を指定します)

```
jsonfile2mrubyc> ruby json2ruby.rb -r tempsensor_LED.json -w tempsensor_LED.rb
```

④生成したプログラムをコンパイルしマイコンに転送します。

```
jsonfile2mrubyc> ruby convert_mrubyc.rb -r tempsensor_LED.rb
```

※③と④をまとめて実行するなら、オプションとして「-d」を付けます。

```
jsonfile2mrubyc> ruby json2ruby.rb -r tempsensor_LED.json -w tempsensor_LED.rb -d
```

⑤書き込み完了すれば、Grove 温湿度センサによる LED 制御が行われます。また、ターミナルソフトを利用すると、function-ruby ノードから変換された摂氏温度が表示されます。

ターミナルについて：<https://yoshihiroogura.github.io/RBoardDocument/console.html>