PLC-FARM
ソフトウェアマニュアル

～JUNKWare/JWT～

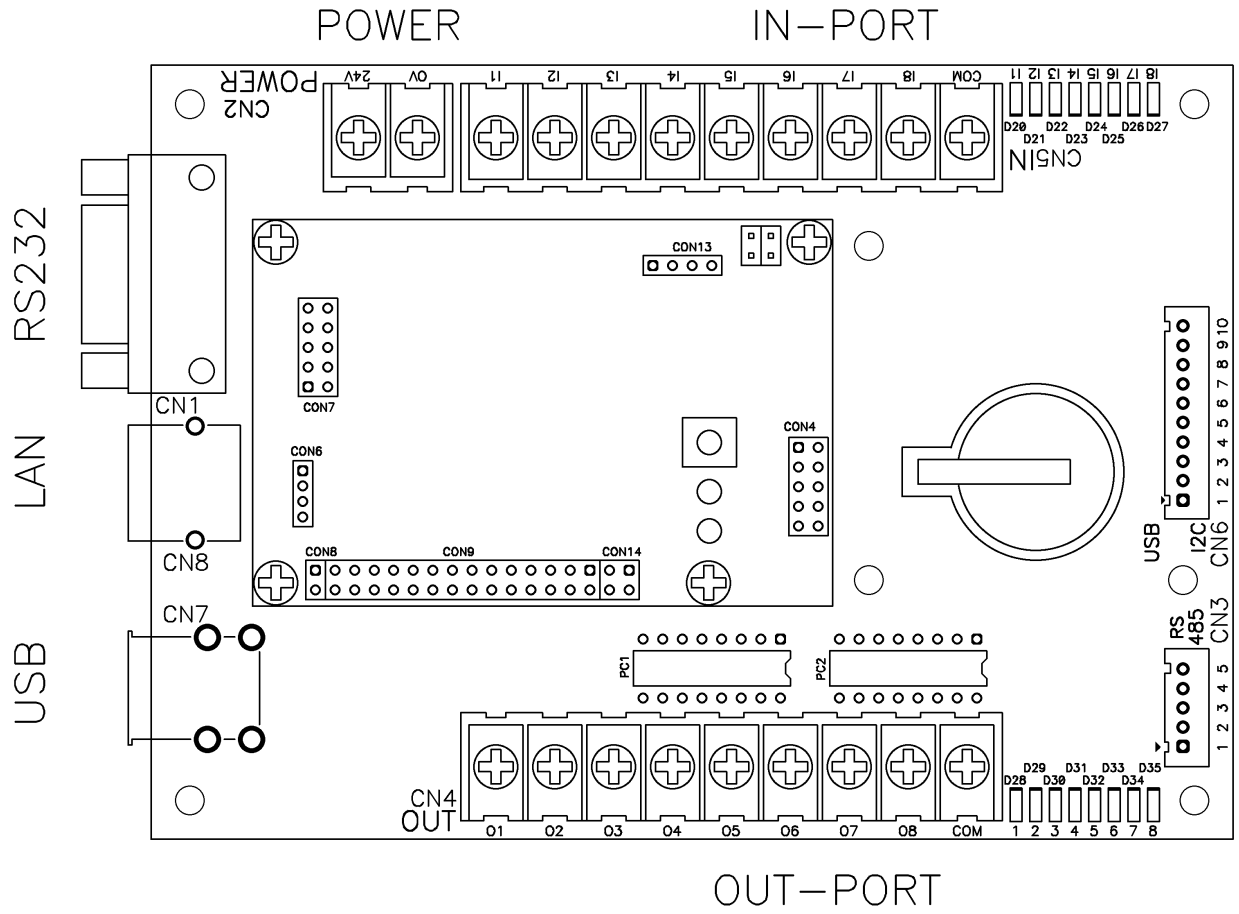
2012年8月3日

YD12039-00

株式会社ＹＯＯＤＳ

目　　次

I. PLC-FARMについて

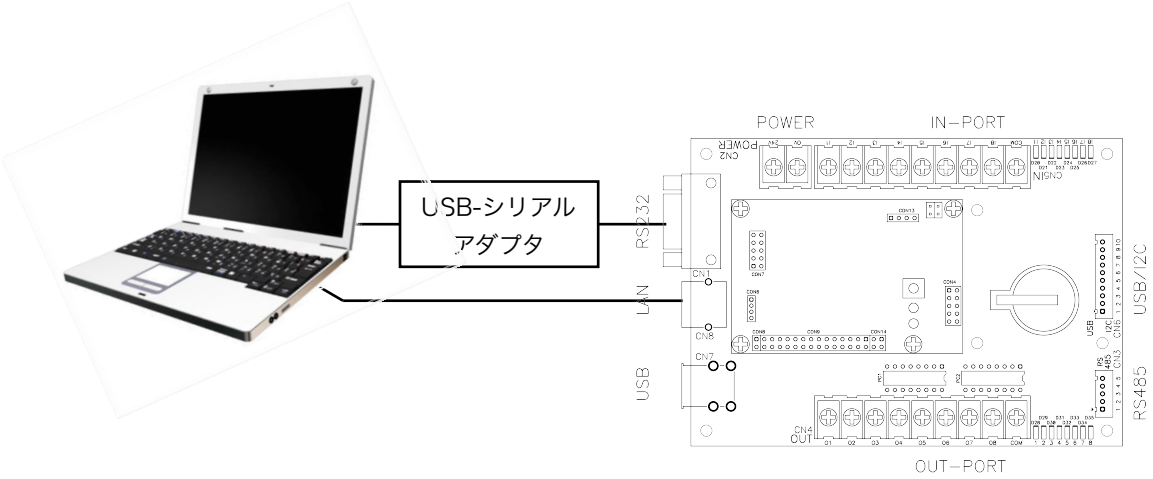


PLC-FARMはLinuxを搭載した組込用コンピュータです。このマニュアルでは、PLC-FARMを使ってシステムを構築する方法を説明します。PLC-FARMにはコンピュータのソフトウェアを効率的に開発するためのミドルウェアとして「JUNKWare」というソフトウェアPLCが搭載されています。「JUNKWare」を使うことにより、PLC-FARMのI/Oに簡単にアクセスしてバグの無いシステムを効率良く開発することができます。

また「JUNKWare」を使うユーザーインタフェースとして、Webブラウザベースで作業のできる「JWT(JUNKWare Web Tools)」を搭載しています。開発担当者は標準的なブラウザFirefoxさえ用意すれば、すぐに開発を開始でき、何等かのトラブルがあった場合もブラウザだけで調査/デバッグが可能です。

II. 開発環境のセットアップ

1. パソコンとの接続



PLC-FARMでシステムを開発するためには、1台のパソコンがあればOKです。通常、ソフトウェアを開発するには専用のシステム開発ツールが必要ですか、PLC-FARMでは特別なソフトウェアを用意する必要はありません。標準的なWEBブラウザであるFirefoxとJRE(Java Runtime Environment)をインストールしておいてください。

■Firefox

http://mozilla.jp/firefox/

■JRE

http://java.com/ja/download/

パソコンとPLC-FARMはEthernetで接続します。PLC-FARMには初期値として

IPアドレス　192.168.1.42

が設定してあるので、パソコン側のIPアドレスを192.168.1.xxxに設定してください。LANケーブルで直結する場合はクロスケーブルの方が確実ですが、最近のパソコンはケーブルを自動認識するので、特に気にする必要はないと思います。PLC-FARMの電源を入れて接続したら、DOS窓を開いて

ping 192.168.1.42

のようにpingでPLC-FARMからの返答があることを確認してください。

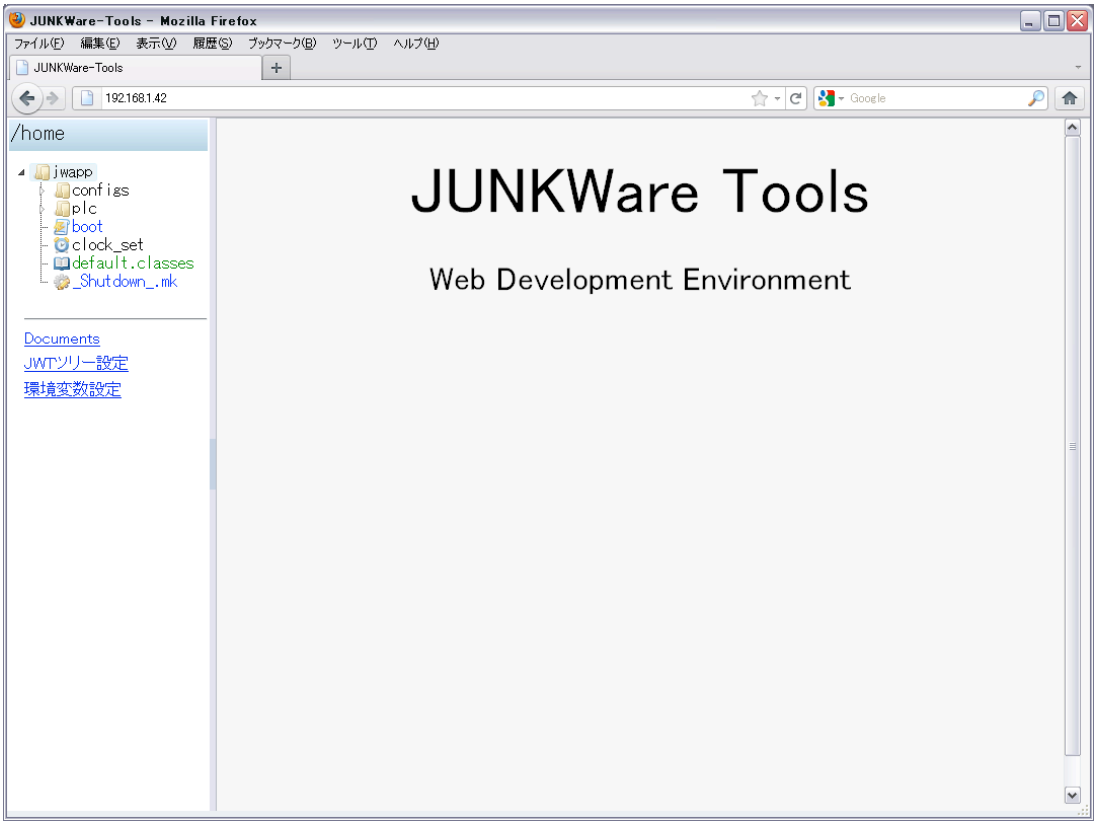
※シリアル接続はPLC-FARMの起動を確認することができるので便利ですが、必須ではありません。

2. Firefoxの起動とJWTへの接続

PLC-FARM～パソコンのネットワークの接続を確認したら、Firefoxを起動して

http://192.168.1.42/

にアクセスしてください。以下のような画面が出れば、PLC-FARMの開発環境への入り口まできたことになります。



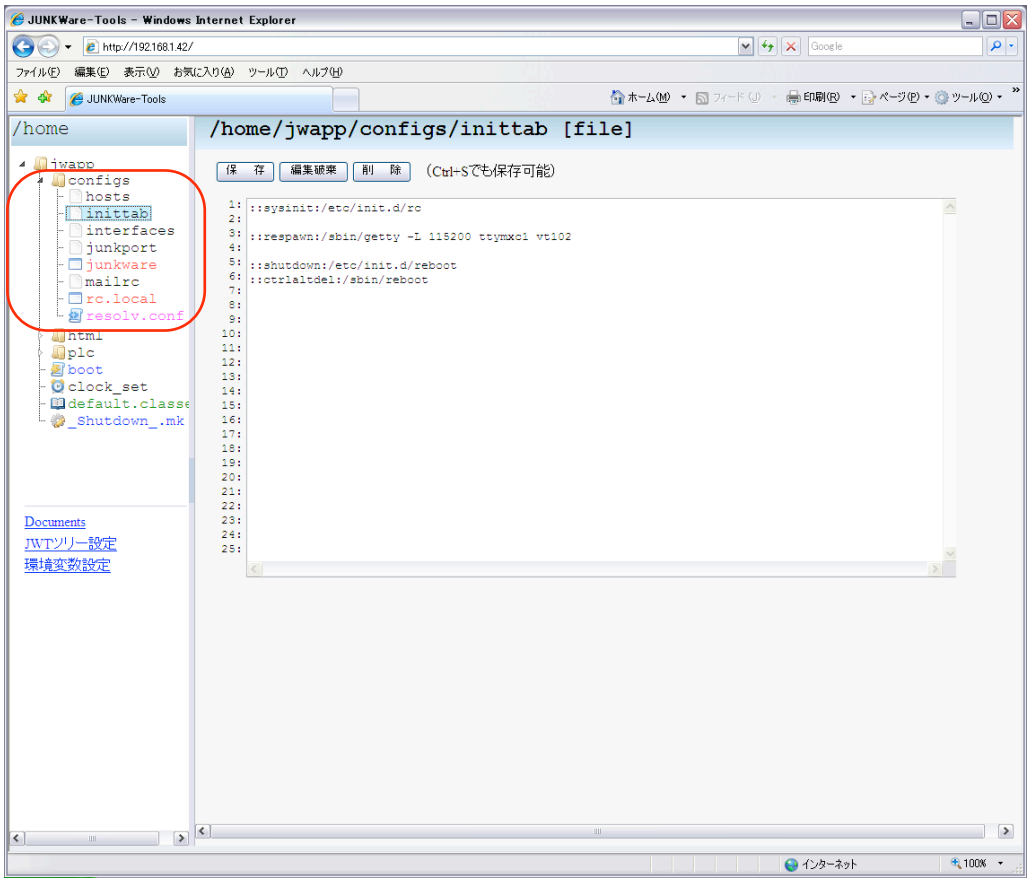
この画面がPLC-FARMに搭載されているソフトウェアPLC「JUNKWare」の開発環境「JWT」の画面になります。

※コンソールログインについて

PLC-FARMはLinuxで動作しているので、シリアルコンソールまたはsshクライアントでログインが可能です。ログインのアカウントはjwappユーザーを使ってください。root及びjwappユーザーアカウントの初期パスワードは以下の通りです。

root	yoods-ume
jwapp	jwapp

3. PLC-FARMの初期設定



JWTの左側ファイルツリーにある”configs”の左にある三角形クリックしてください。上図のようにconfigsフォルダに入っているPLC-FARM初期設定用のファイルhosts, inittab, interfaces, issue, junkport, junkware, mailrc, rc.local, resolv.confの9つのファイルが表示されます。それぞれファイルをクリックすると、右側のペインにその内容(テキストファイル)が表示されます。

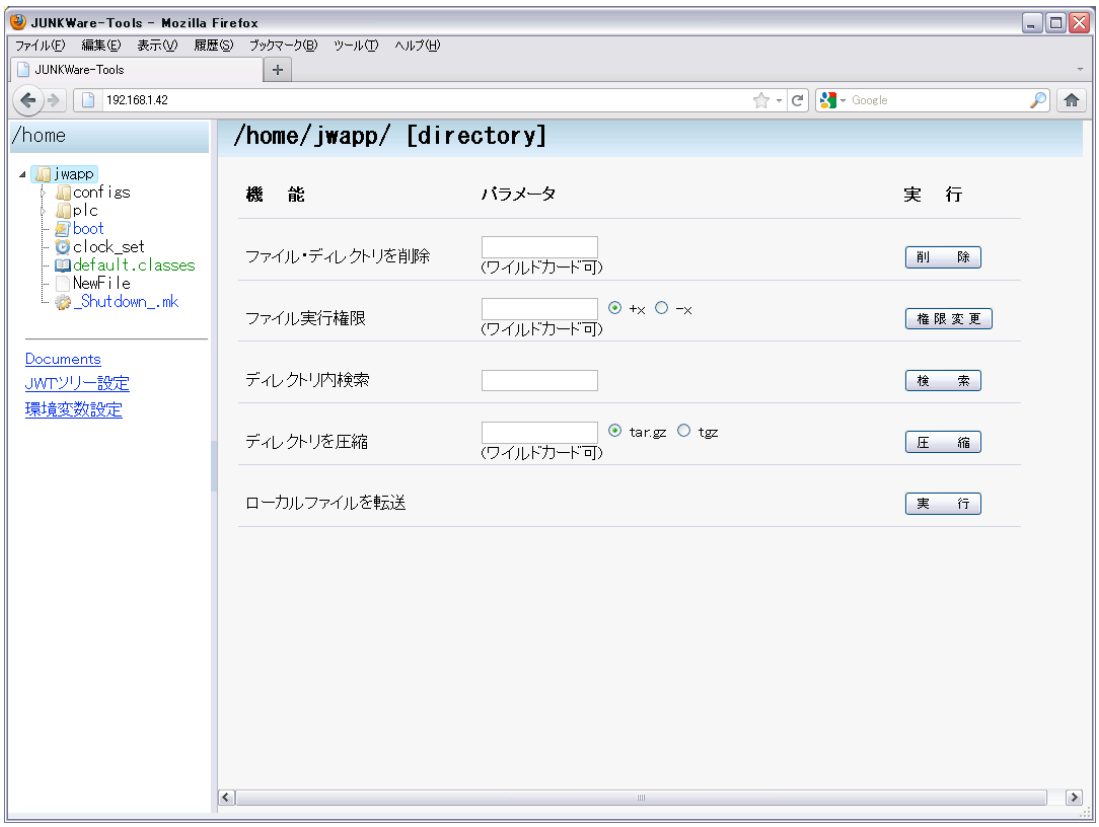
hosts	/etc/hostsにリンクされています。
inittab	/etc/inittabにリンクされています。シリアルコンソール等の設定ができます。
interfaces	/etc/network/interfacesにリンクされています。ネットワークの設定を行います。
issue	/etc/issueにリンクされています。コンソールにホスト情報やPLC-FARMのバージョン情報を出力します。read-onlyです
junkport	JUNKWareプロセスに外部からアクセスするためのTCPポート番号です。
junkware	/etc/init.d/junkwareの実体です。ソフトウェアPLC「JUNKWare」の起動を制御します。
mailrc	/home/jwapp/.mailrcにリンクされています。メール送信の設定ができます。
rc.local	/etc/init.d/rc.localにリンクされています。起動時の処理を追加できます。
resolv.conf	/etc/resolv.confにリンクされています。

これらのファイルはissueを除いて、自由に編集/保存できるので、ご使用の環境に併せて修正してください。

4. JWT操作の基本

JWTはファイルの修正が主な機能ですが、まず、基本的なファイルやディレクトリの追加/削除の機能について説明します。

・ディレクトリ(フォルダ)メニュー



ディレクトリアイコン(フォルダ)をクリックすると、右側ペインに上図のような画面が開きます(objs/を除く)。選択しているディレクトリ内のファイルやディレクトリに対して、以下の操作を提供しています。

- ファイル・ディレクトリを削除

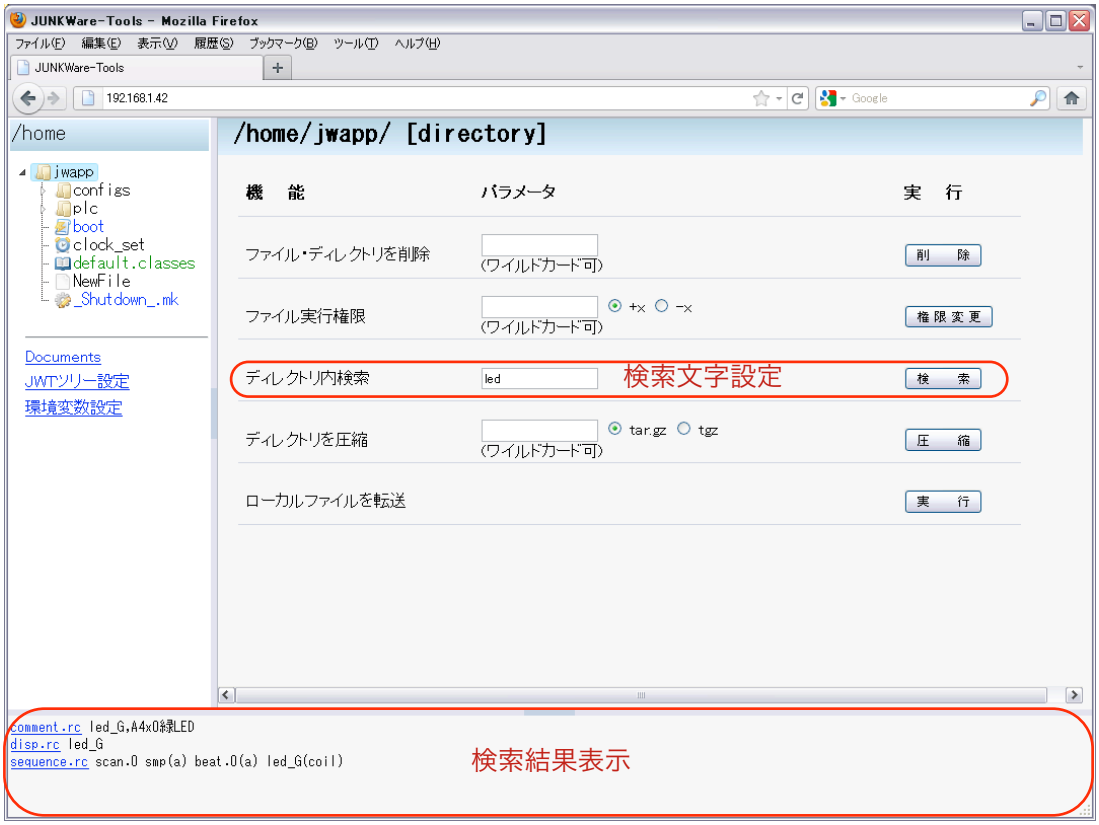
ファイル,ディレクトリを削除します。ひとつひとつ削除する場合は、左側のファイルツリー上で対象を右クリックしても削除できますが、ここではワイルドカードを使って一括削除ができます。

- ファイル実行権限

ファイルに対して実行権限の付与/削除を行います。シェルスクリプトを作る際に利用します。

- ディレクトリ内検索

ディレクトリ内のファイル中に含まれるキーワードを検索します。内部では”grep -r”コマンドを実行して、検索結果は画面下部に該当ファイルへのリンクとして表示します。



- ディレクトリを圧縮

選択中のディレクトリ内にあるサブディレクトリをtar.gz形式で圧縮します。ファイル名はtgzも選べます。実際には、作成したプロジェクトをローカルにバックアップをとる場合等に使用します。

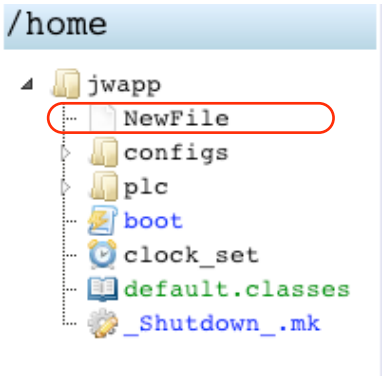
- ローカルファイルを転送

ローカルのパソコンにあるファイルをPLC-FARMに転送できます。ただし搭載しているフラッシュの容量は出荷時で1.5M程度しか余っていません。あまり大きなファイルを転送しないように注意してください。

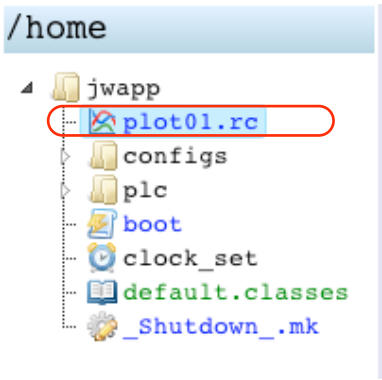
バックアップをとってあったプロジェクトを新しいPLC-FARMに展開するケース等で使用できます。

・**ファイル,ディレクトリの作成, 名称変更**

新しくファイルやディレクトリ(フォルダ)を作りたい、という場合は、作りたいディレクトリやファイルを含むディレクトリを右クリックして、コンテキストメニューを表示します。そのメニューに、「ファイル作成」,「ディレクトリ作成」という項目があるので、いずれかを選択してください。

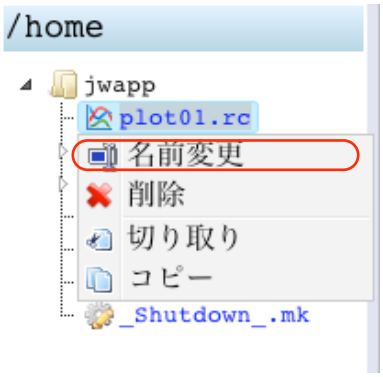


ファイルを選択すると「NewFile」という新しいファイルが作成されます。任意のファイル名を入力してください。



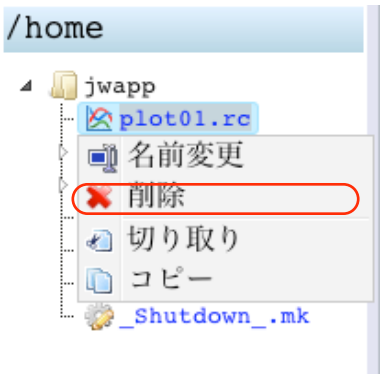
ここではplot01.rcという名前にしました。

ファイル名,ディレクトリ名を変更する場合は、ファイルツリーからファイルを選択して右クリックしてください。



「名前変更」を選択するとファイル名を変更することができます。

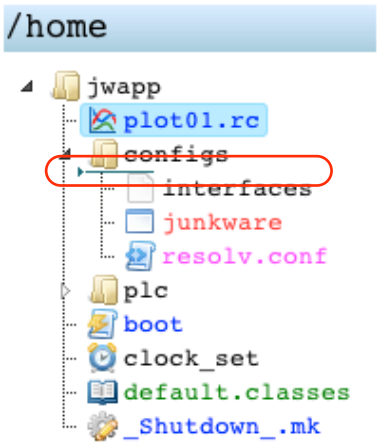
・ファイル,ディレクトリの削除



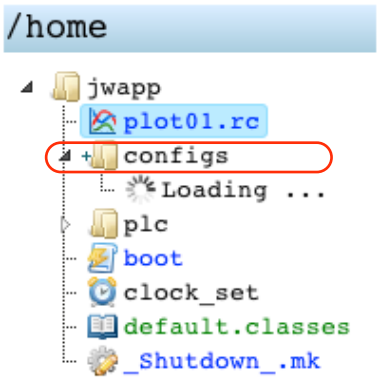
ファイル,ディレクトリを削除する場合には、対象となるファイル,ディレクトリを選択して「削除」を選択してください。これで対象のファイル,ディレクトリを削除することができます。ディレクトリ削除の場合、内部のファイルもすべて削除されます。

・**ファイル,ディレクトリの移動**

ファイル,ディレクトリを移動する場合は、直接マウスで対象をドラッグしてください。



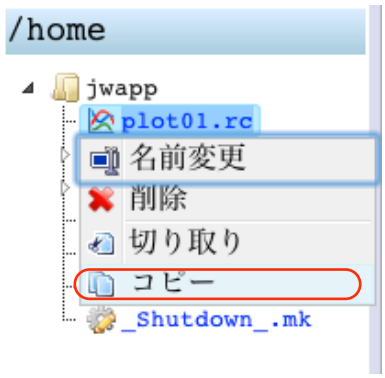
挿入点には上図のように目印が出ます。マウスをドラッグしながらこのポイントを任意の点に持っていき、リリースすることによりファイル,ディレクトリを移動することができます。



対象となるディレクリが開いていない場合は、そのディレクトリ上で停止すると、上図ように”+”マークが表示されてディレクトリが展開されます。

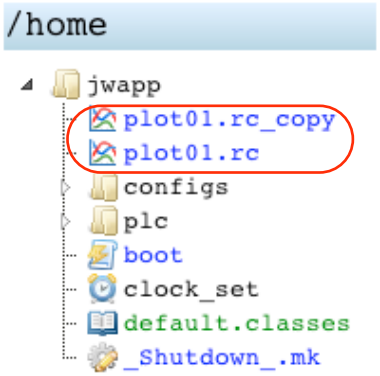
・**ファイル,ディレクトリのコピー**

ファイルやディレクトリをコピーする場合は、対象を右クリックして「コピー」を選択してください。

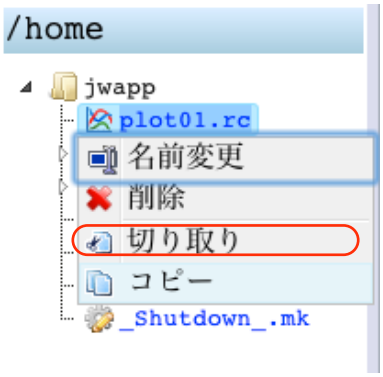


次にコピーを配置したいディレクトリを右クリックします。

表示されるメニューから貼付けを選択すると、先ほどコピーしたファイルが対象ディレクトリ内に複製されます。



同一ディレクトリ内にコピーした場合、上図のように”_copy”が付加されたファイルが作成されます。



同様に、メニューから「切り取り」を選択すると、ファイル移動の動作となります。

・objs/のタグファイルの追加

/home/jwapp/plcの下にあるobjs/ディレクトリはJUNKWareで用いるリレーやタイマ等のオブジェクト定義(タグファイル)が配置されています。ここに新しくファイルを追加することにより、リレー等の制御用パーツ(オブジェクト)を追加することができます。また削除することにより、不要なオブジェクトを削除することも可能です。

objs/ディレクトリをクリックすると右側ペインには下図のような画面が表示されます。

この右側ペインのメニューを使うことにより、新規タグを定義することができます。実際にオブジェクトを追加する方法については、「JUNKWareのクラスリファレンス」の項を参照してください。

また、前述のファイルコピーを使って既存オブジェクトをコピーして編集することにより、新規タグを追加することも可能です。

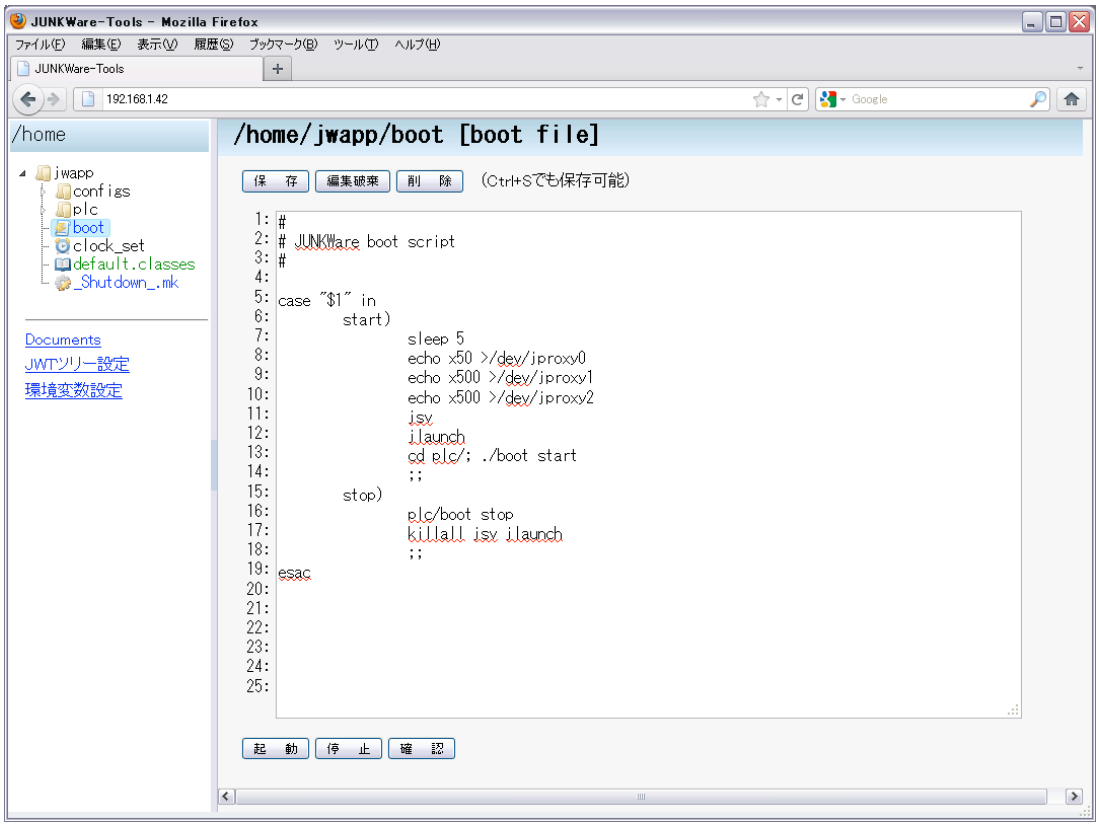
III. PLC-FARMを使ったシステム開発のチュートリアル

1. bootスクリプト

PLC-FARMは電源を投入するとすぐにLinuxが起動して、その後ソフトウェアPLCプロセスが起動します。ソフトウェアPLCの起動は、

/home/jwapp/boot

のシェルスクリプトが担当しています。JWTの左側ファイルツリーにある”boot”をクリックしてください。右側ペインに”boot”スクリプトの内容が表示されます。 このbootスクリプトの内容は、ここで直接、編集/保存することも可能です。ここでは簡単に実行している内容を説明します。



```
start)
      echo x50 >/dev/jproxy0
      echo x500 >/dev/jproxy1
      echo x500 >/dev/jproxy2
      jsv
      jlauhch
      cd plc/; ./boot start
```

JUNKWareでは/dev/jproxy[0〜7]というデバイスを使ってシーケンサに求められる一定周期のスキャン動作を実現しています。システムのリソースを最大限に利用できるように、JUNKWareでは複数のスレッドを起動して、それぞれに任意の動作周期を設定できます。デフォルトでは3つのスキャン動作を定義して、それぞれにスキャン周期を設定しています。x50とかx500という部分が周期時間で、単位は100μ秒です。従って、x50と設定された/dev/jproxy0は5ミリ秒周期のスキャン動作をすることになります。スキャン時間は任意に設定できるので、jproxy1,jproxy2のように、二つに同じ時間を設定しても構いません。

次にjsv,jlaunchというふたつのデーモンプロセスを起動しています。jsvはソフトウェアPLCのプロセスにネットワーク接続する機能を提供します。jlauhchはソフトウェアPLCから外部コマンドを実行するために利用します。

最後の行がplc/にあるソフトウェアPLCの実体を起動しています。ソフトウェアPLCの全ての機能は/home/jwapp/plc/以下に集約されています。これで判るように、JUNKWareのbootスクリプトは2段起動となっていて、

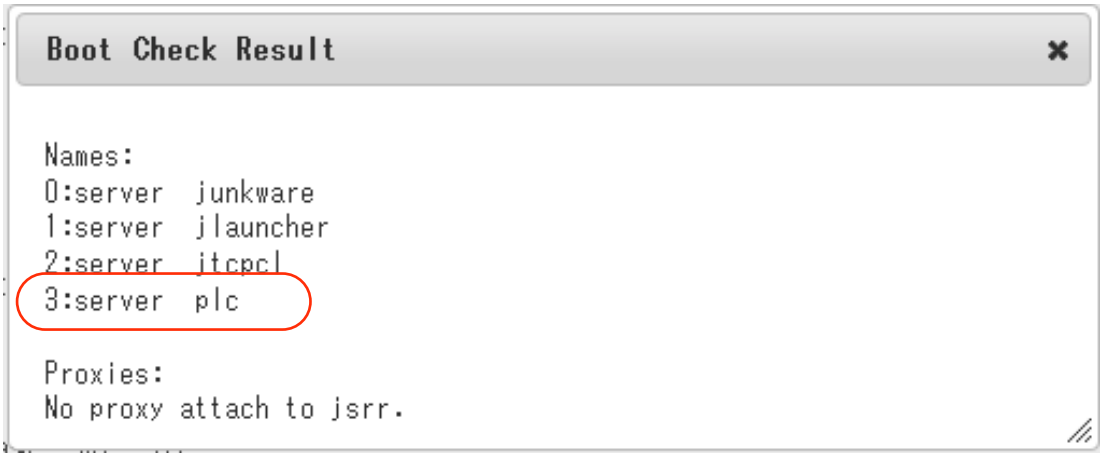
/home/jwapp/boot → /home/jwapp/plc/boot

という形で起動していて、それぞれの役割は以下のようになっています。

/home/jwapp/boot
: ソフトウェアPLCの実体を起動する前の環境設定等を行います。

/home/jwapp/plc/boot
: ソフトウェアPLCの実体を起動します。

bootスクリプトの下部に、起　動　停　止　確　認 というボタンが並んでいます。これらのボタンにより、ソフトウェアPLCの起動/停止/確認を行うことができます。まず、確認をクリックしてみてください。PLC-FARM起動時にソフトウェアPLCは起動しているので、



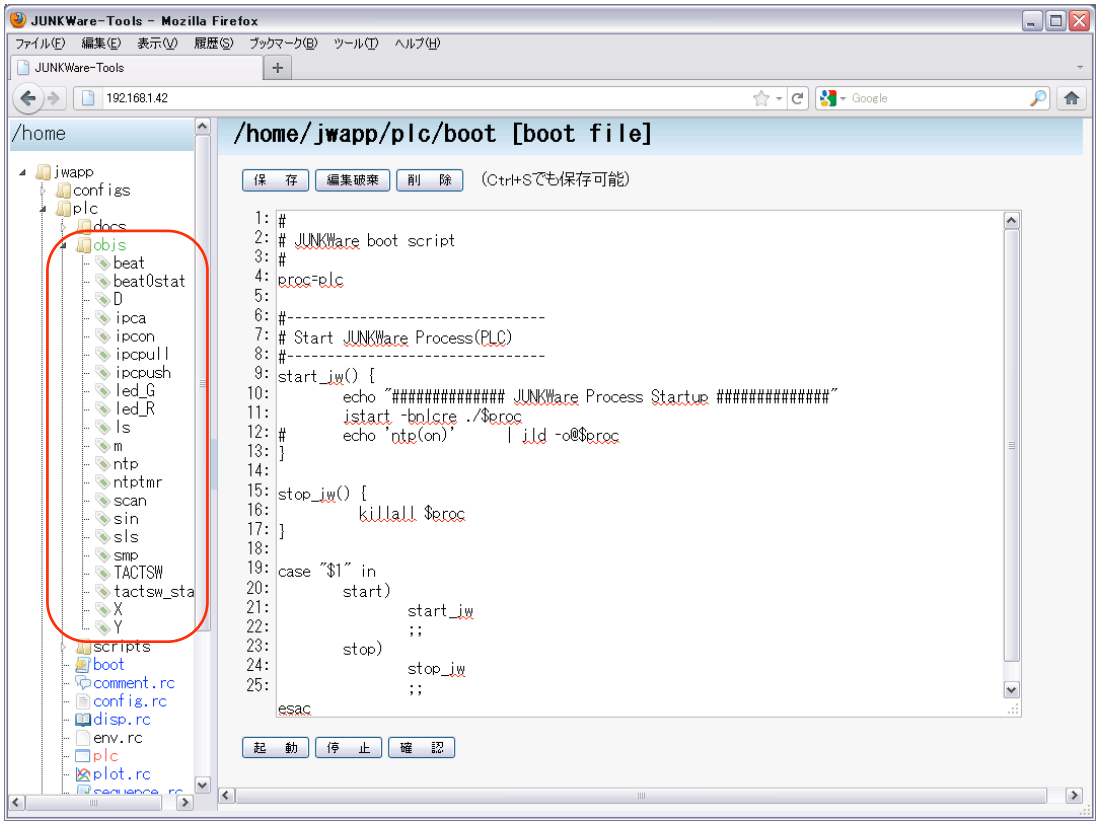
上記ダイアログが表示されて、その中にplcという名前のプロセスが存在していることが確認できます。ダイアログを閉じた後、次に停止ボタンをクリックしてください。実行完了ダイアログが表示されるので、それを消去した後、再度確認ボタンを押すと、上記plcプロセスが消えていることが確認できます。

次に起動ボタンを押してください。ちょっと待たされますがその後起動を完了のログを出力したダイアログが表示されます。これを閉じて再度確認ボタンを押してください。前の状態に戻ってplcプロセスが起動していることが確認できます。

※後に示すobjs/リソースやシーケンスリソースを変更して、その結果を反映するにはplcプロセスの再起動が必要です。その際には、このbootスクリプトの起動/停止/確認を使ってください。

2. JUNKWareオブジェクト

一般的にシーケンサには、様々なデバイスが用意されています。リレーやタイマがその基本ですが、JUNKWareではそれらのデバイスをオブジェクトと呼びます。JUNKWareではシーケンサと違って、オブジェクトは自由に追加/削除ができます。どのようなオブジェクトを使うことができるかは、後述の「JUNKWareのクラスリファレンス」を参照してください。



JWTの左側ファイルツリーにある”plc”の右側の三角形をクリックして、更にその中の”objs”を開いてください。ここに表示されるファイル群がPLC-FARMに事前に登録されているオブジェクト群です。以下、これらのオブジェクトについて説明します。例として示しているだけのオブジェクトもありますので必要ないと判断されたものは削除して、また、足らないものはリファレンスの例を参考に追加してください。

オブジェクトリソースはソフトウェアPLCのプロセスの再起動により、変更内容が反映されます。修正後はbootスクリプトの停止/起動ボタンによりプロセスを再起動してください。

PLC-FARMにプリセットされているオブジェクト

beat	自己発振するオブジェクトです。起動時onに設定されているのでそれぞれ設定された時間(0.1,0.2,0.3秒)毎にon/offを繰り返しています。
beat0stat	後で説明するグラフ表示用にbeat.0の発振データからCALCクラスを使って7/8の間でup/downするデータを作っています。
D	シーケンサのデータメモリと同じ働きをします。ここでは32個のデータメモリを用意しています。
ipca,ipcon,ipcpull,ipcpush	外部通信クラス群IPCの例を示すオブジェクトです。
led_G(R)	PLC-FARMのCPU部分であるArmadillo-420に搭載されているLED(緑,赤)を制御するためのオブジェクトです。
ls	外部コマンド実行クラスEXECの例を示すためのオブジェクトで、/bin/lsを起動します。
m	一般的なシーケンサでは内部リレーと呼ばれるオブジェクトです。ここでは16個の内部リレーを宣言しています。
ntp	外部コマンド実行クラスEXECの例を示すためのオブジェクトで、ntpサーバにアクセスするスクリプト/home/jwapp/plc/scripts/ntpを呼んでいます。
scan	JUNKWareのスキャン動作を担当するオブジェクトで、これは削除できません。前述の/dev/jproxy[0-7]を使って、配下のオブジェクトに一定周期で実行権を与えています。
sin	CALCクラスを使ったsin関数の例です。
sls	ソフトウェアリミットスイッチLIMITSWクラスのオブジェクト例です。
smp	データロガークラスSAMPLERのオブジェクト例です。後述のグラフ表示チュートリアルで使用します。
TACTSW	PLC-FARMのCPU部分であるArmadillo-420に搭載されているタクトスイッチの状態をモニタするためのオブジェクトです。
tactsw_stat	後で説明するグラフ表示用にTACTSWのon/off状態からCALCクラスを使って5/6の間でup/downするデータを作っています。
X,Y	PLC-FARMのDIOを使うためのオブジェクトです。削除はしないでください。

3. ラダー機能

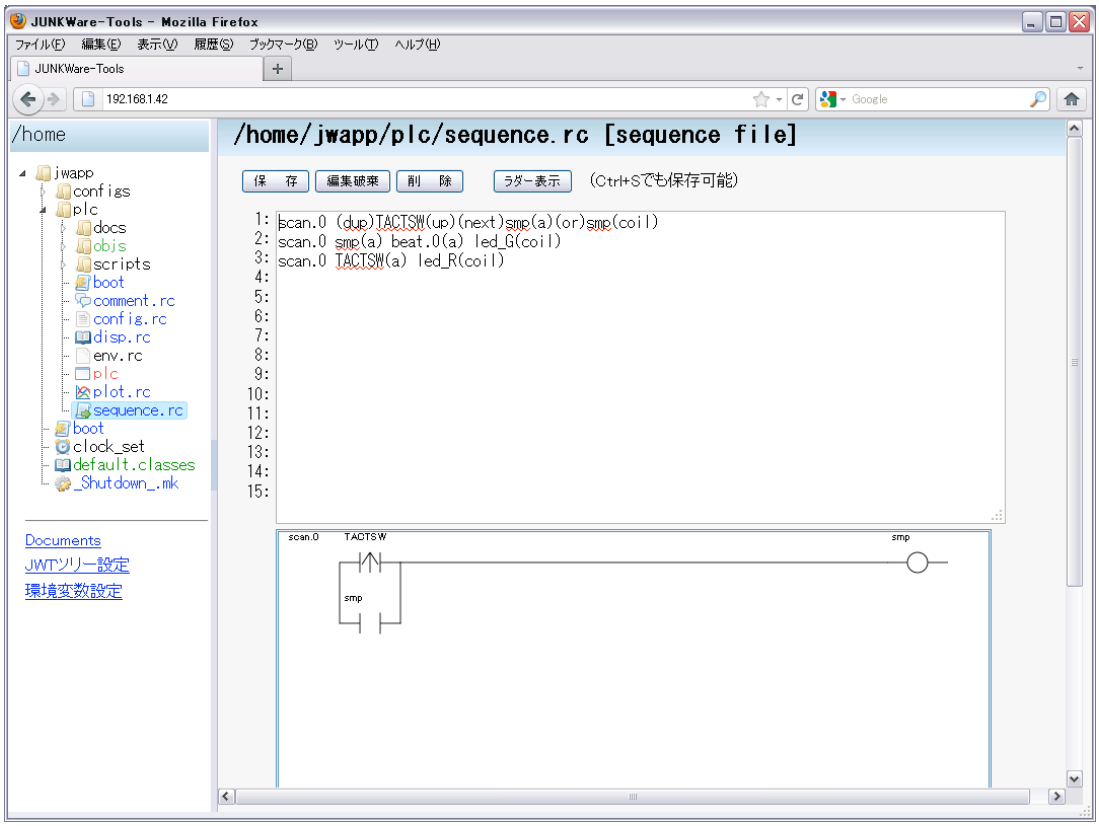
ここではPLC本来の最も表に出る部分であるラダー機能について説明します。JUNKWareにおけるラダーは以下の二つのファイルで設定します。

/home/jwapp/plc/sequence.rc
ソフトウェアPLCで使用するラダーを記述します。

/home/jwapp/plc/comment.rc
各オブジェクトのコメントを記述します。

いずれもテキストファイルとなっているため、直接エディタで編集/保存することができます。

- **sequence.rc**



JWTの左側ファイルツリーからplc/sequence.rcをクリックすると、右側ペインにその内容(テキスト)が表示されます。そのテキストの任意の行をクリックすると、下部の領域にその行に対応するラダーが表示されます。このテキストファイルは自由に編集/保存ができるので、ここでラダーの追加/削除/修正を行い、bootスクリプトでシステムを再起動すると変更内容を反映したシステムが起動します。

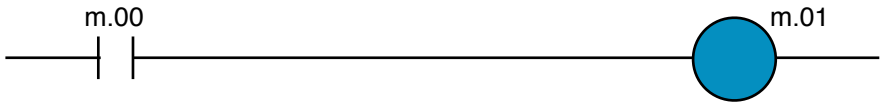
- ラダーに使うキーワード

次に、ラダーの記述方法について説明します。まず使用するキーワードを以下の表にまとめます。

タグ名	objs/に定義されたオブジェクトのファイル名です。rangeで複数のオブジェクトを宣言している場合は、ファイル名.xxという形でインデックスが付きます。
(a),(b)	タグ名の後につけて、タグを装飾する形で使用します。いわゆるa接点,b接点を表します。
(up)(down)	タグ名の後につけて、タグを装飾する形で使用します。タグのステータスがoff→onに変化した時、on→offに変化した際、1周期だけonになります。
(not)	(up)(down)の後につけて論理を反転する場合に使用します。1周期だけoffになる信号を生成します。
(dup)	OR回路を作る際に使用します。具体的にはOR分岐を開始するポイントに記述します。
(next)	最寄りの(dup)で分岐した地点にOR接続で折り返すポイントに記述します。
(or)	最寄りの(next)で折り返した地点にOR接続を復帰するポイントに記述します。
(coil)	タグ名の後に付けて回路の出力コイルに使用します。

次に、実際にラダーを書く方法について以下に例を挙げながら説明します。

- 例1　最も簡単なラダー



前述のように、JUNKWareでは複数のスキャンループを定義できます。まず、作成する回路がどの程度の応答性が必要かを検討して、適切なスキャン間隔のSCANGATEオブジェクトを選択します。今回は、scan.1を選択したとします。新しい行の最初にそのSCANGATEオブジェクトを書きます。

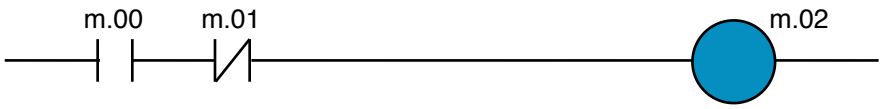
```
scan.1
```

その後に回路を記載します。この回路の場合、m.00のa接点をm.01コイルに接続するので、単純に以下のように書くことができます。コイルはタグ名の後に(coil)を記載します。

```
scan.1 m.00(a) m.01(coil)
```

もしm.00のb接点でm.01コイルを叩きたければ、m.00(b)と記述することになります。

- 例2　接点のand接続の例

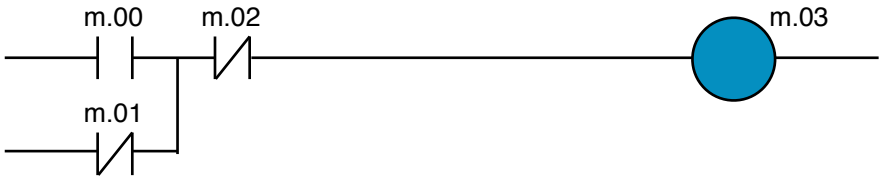


この回路はscan.0でスキャンするとします。

```
scan.0 m.00(a) m.01(b) m.02(coil)
```

and接続する場合は単純にタグを並べてください。この例ではand接続するm.00(a)とm.01(b)を並べています。

- 例3　接点のor接続の例



or接続した後でm.02(b)とのandをとっています。

```
scan.1 (dup) m.00(a)
```

まず分岐する点に(dup)を記述します。その後には分岐点以降の最初の行の回路を記述します。ここではm.00(a)を記述します。

次に(dup)した次の行に挿入点を移動するため(next)命令を記述します。これによって、次のオブジェクト挿入点が(dup)点から一段下の行に移動するので、ここにor接続する回路を記述します。

```
scan.1 (dup) m.00(a) (next) m.01(b)
```

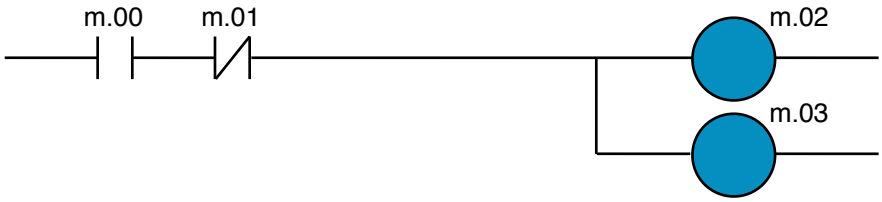
m.01(b)の後は先ほど(next)した点に再接続するため、ここに(or)を記述します。

```
scan.1 (dup) m.00(a) (next) m.01(b) (or)
```

これでor接続回路が完成です。このor回路にm.02(b)をand接続しますが、これは単に直列に書くだけです。その後にコイルを記述して回路が完成します。

```
scan.1 (dup) m.00(a) (next) m.01(b) (or) m.02(b) m.03(coil)
```

- 例4　複数のコイルに出力する例



m.00(a)とm.01(b)のandは並べるだけです。次に分岐するために(dup)を記述します。

```
scan.1 m.00(a) m.01(b) (dup)
```

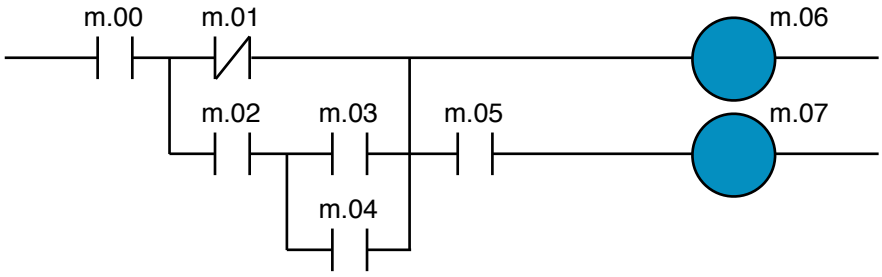
次に分岐した最初の行の回路を記述します。ここではコイルだけなのでコイルを記述してその行を終わります。(next)は不要です。

```
scan.1 m.00(a) m.01(b) (dup) m.02(coil)
```

最後に下段のコイルm.03を記述して終わりです。(or)を記載する必要はありません。

```
scan.1 m.00(a) m.01(b) (dup) m.02(coil) m.03(coil)
```

- 例5　or分岐の重複とコイル複数出力の例



or分岐してm.02(a)を記述するところまでは、これまでの説明通りです。そこで更にor分岐が入っているため(dup)を挿入してm.03(a)を記述します。

```
scan.1 m.00(a)(dup)m.01(b)(next)m.02(a)(dup)m.03(a)
```

ここで一旦最上段の(next)ポイントに(or)します。その後(next)して3段目に移動します。3段目のm.04(a)を記載して(or)するとor部が完成です。

```
scan.1 m.00(a)(dup)m.01(b)(next)m.02(a)(dup)m.03(a)(or)
  (next)m.04(a)(or)
```

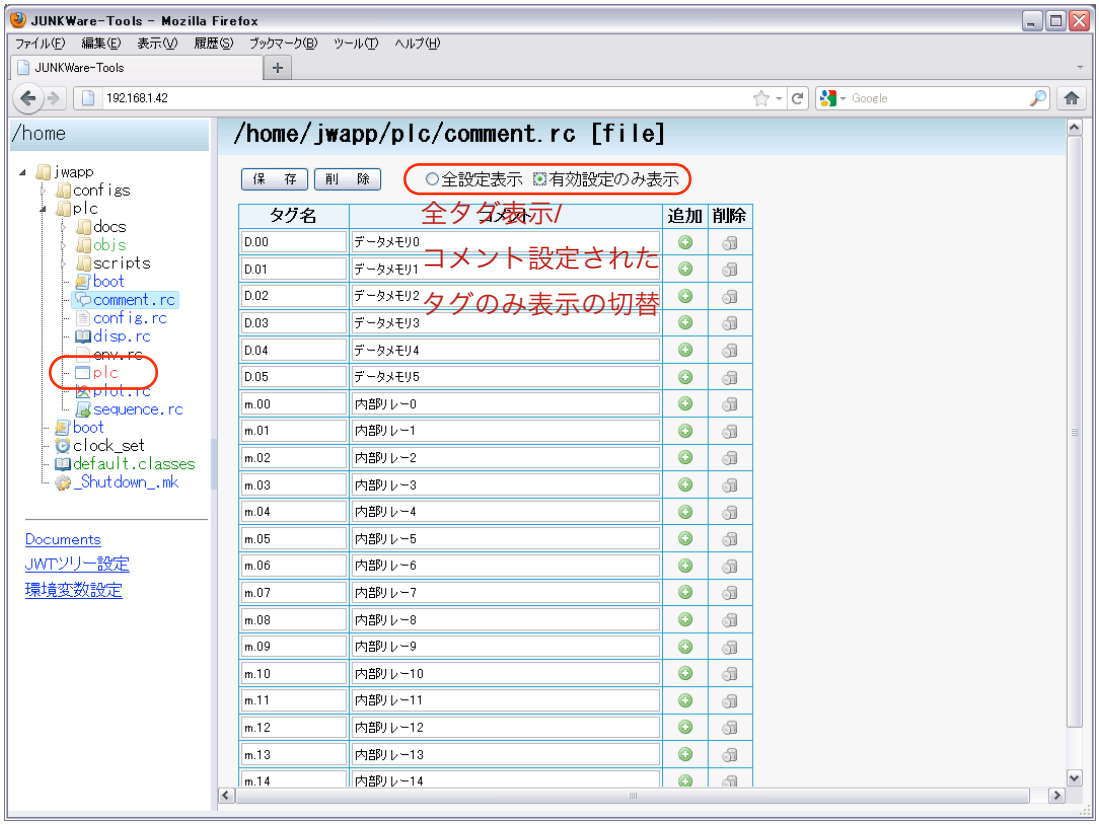
ここで出力コイル2系統へ分岐するため(dup)を入れて、m.06(coil)に出力します。

```
scan.1 m.00(a)(dup)m.01(b)(next)m.02(a)(dup)m.03(a)(or)
  (next)m.04(a)(or)(dup)m.06(coil)
```

そして下段の出力を記述しますが、コイルに出力する前に接点m.05(a)が入っていることに注意します。

```
scan.1 m.00(a)(dup)m.01(b)(next)m.02(a)(dup)m.03(a)(or)
  (next)m.04(a)(or)(dup)m.06(coil)m.05(a)m.07(coil)
```

- **comment.rc**



左側ファイルツリーからcomment.rcを選択すると上図のようなコメント設定一覧が右側ペインに表示されます。「全設定表示」とすると定義済みのすべてのタグが表示されます。「有効設定のみ表示」とすると、コメントの定義されているタグのみをを表示します。

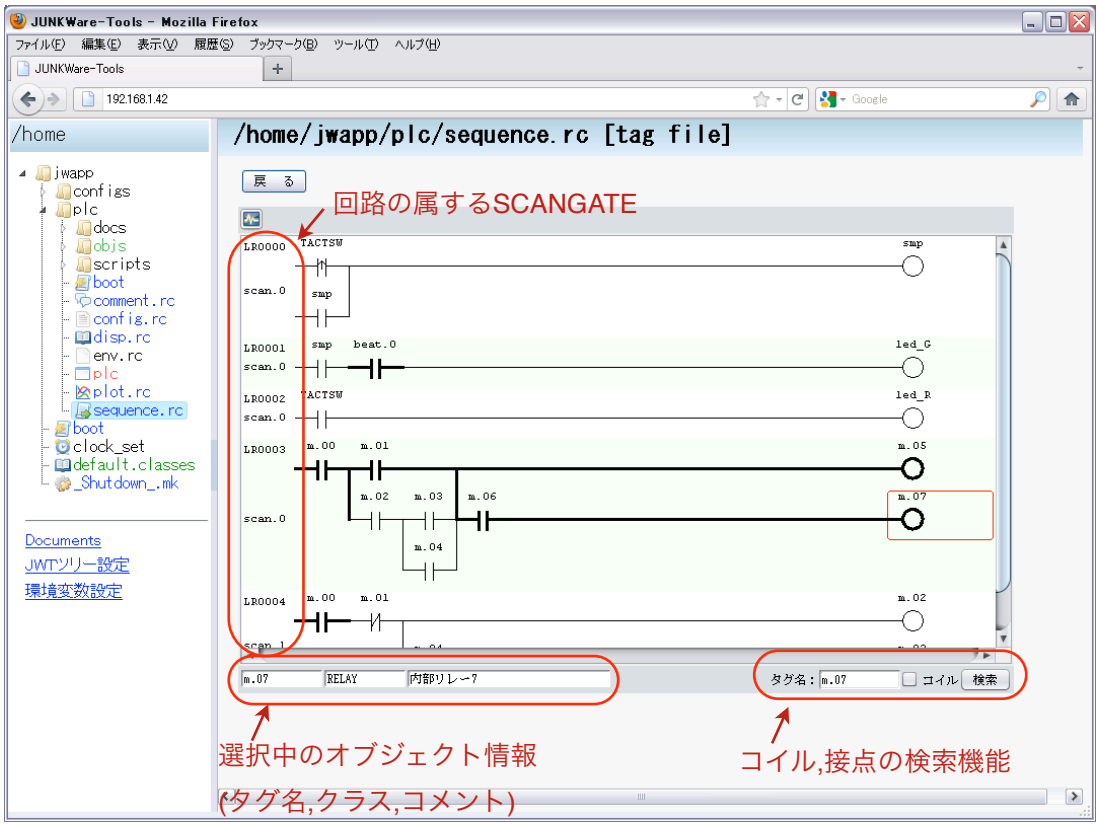
すべてのタグのコメントはここて設定できます。必要に応じてタグの説明を書いて保存することにより、ラダー図上でオブジェクトを選択することでコメントを確認することができます。

4. モニタ機能

・ラダーモニタ

sequence.rcは編集後、bootスクリプトを使ってソフトウェアPLCのプロセスを再起動することで、編集内容が反映された動きになります。

実際にラダーが動いていることをラダーモニタで確認してみましょう。sequence.rcを選択した時に上部に表示されるラダー表示ボタンをクリックしてください。JAVA Appletが起動するので少々時間がかかりますが、以下のような画面でラダーモニタを見ることができます。



・オブジェクトをクリックすることで、画面下部で選択中のオブジェクトのタグ名,クラス名,コメントを確認することができます。

・画面右下部の検索機能を使うと、タグ名でオブジェクトをインクリメンタルサーチできます。

・

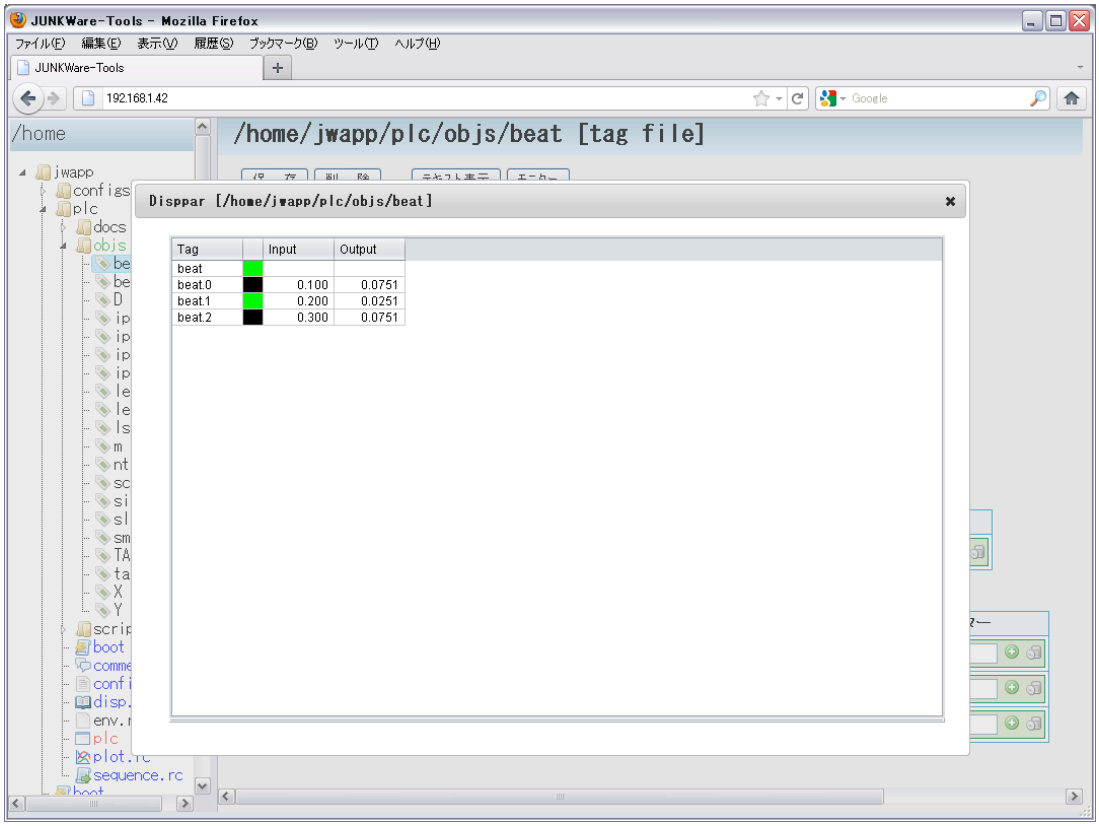
・タグモニタ機能(disppar)

JWTはタグの持つステータスやデータをリアルタイムにモニタする機能(disppar)があります。dispparを使う方法は二つあり、

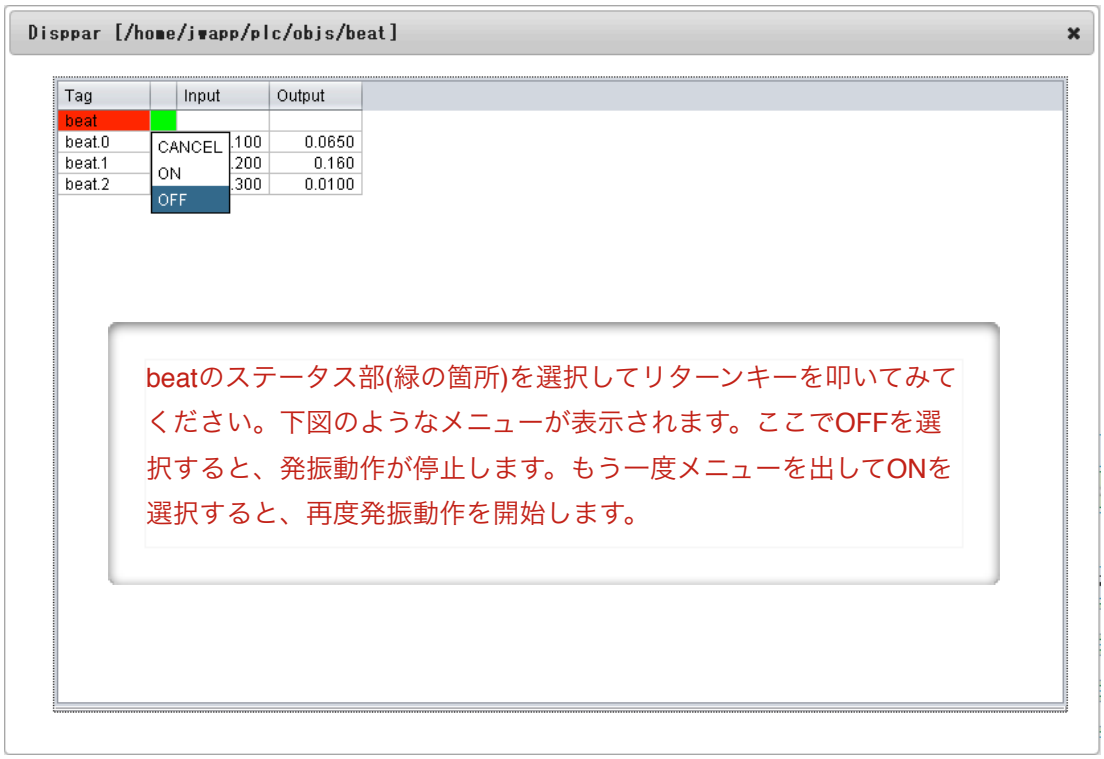
1. objs/の下にあるタグ定義ファイルからモニタボタンをクリック。
2. disp*.rc(*はワイルドカード)というファイルを用意して、ファイルに1行に1つのタグを書いて、モニタしたいタグを列記する。1と同様に、モニタボタンをクリック。

- タグファイルからのタグモニタ

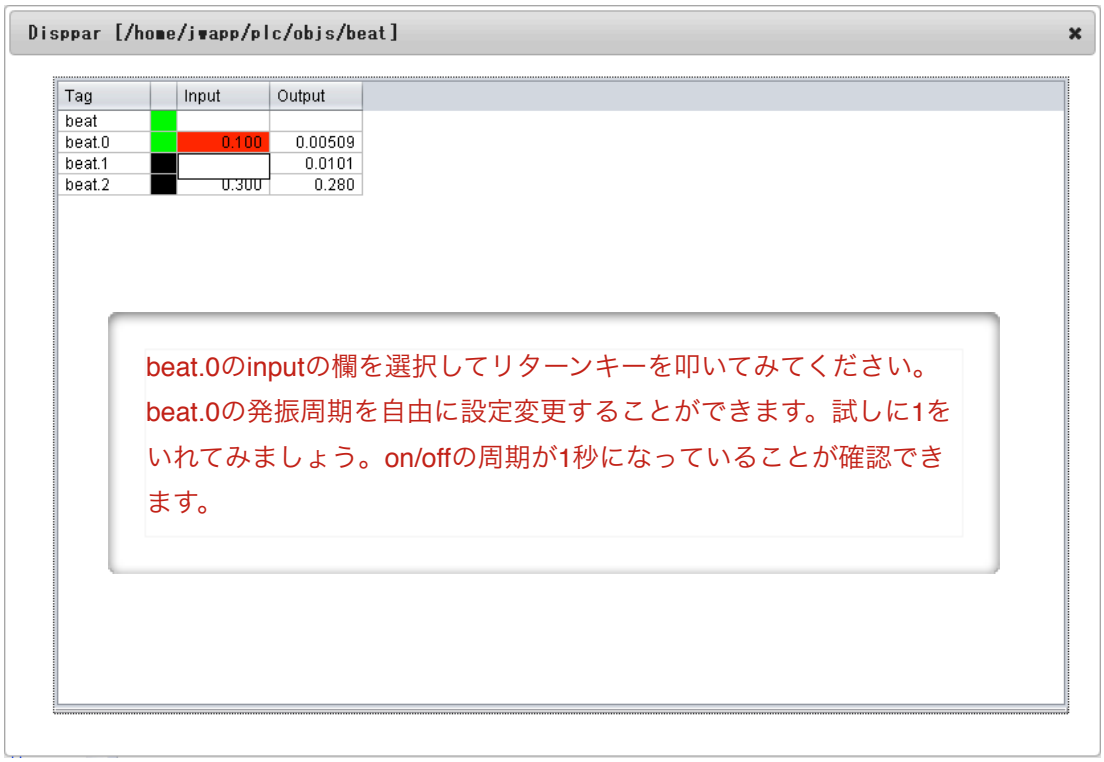
objs/以下にある任意のタグファイルを選択して、上部のモニタタグをクリックしてください。



上記はbeatタグをモニタした例です。beat.0～beat.2が設定通りに動いていることが確認できます。



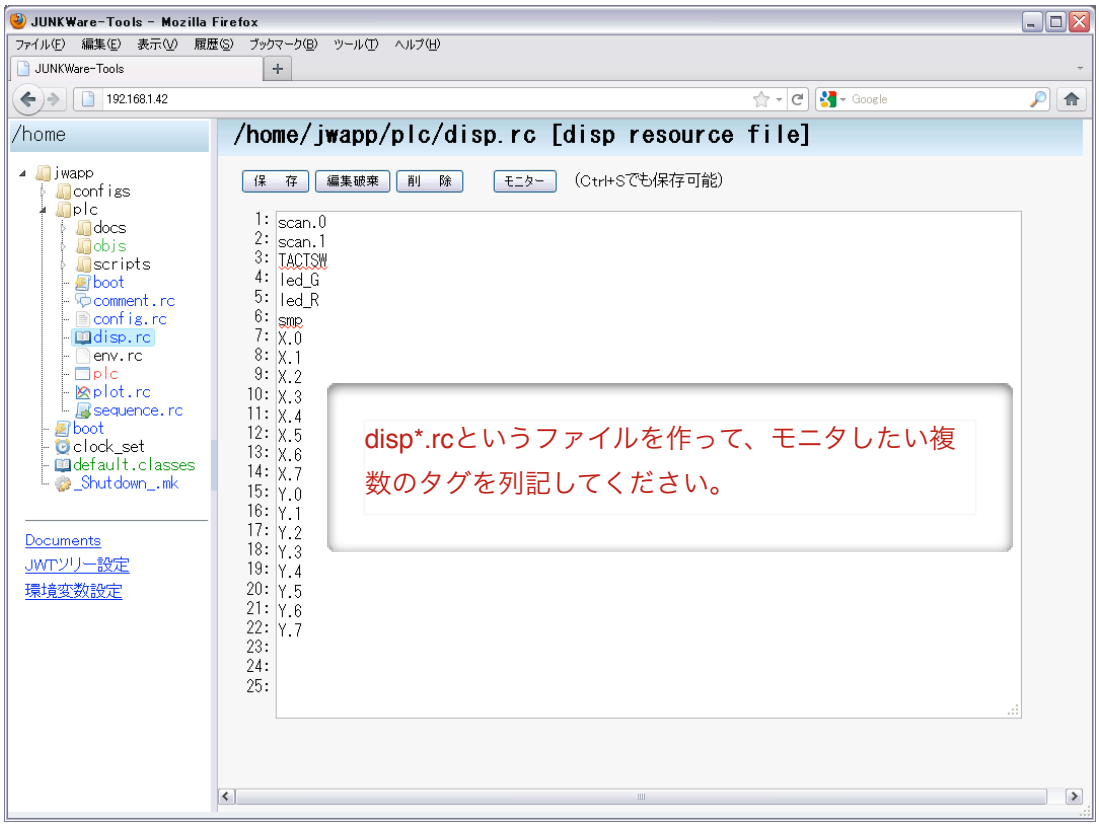
このようにdispparを使ってモニタするだけではなく、ソフトウェアPLC内部のデータを変更することも可能です。

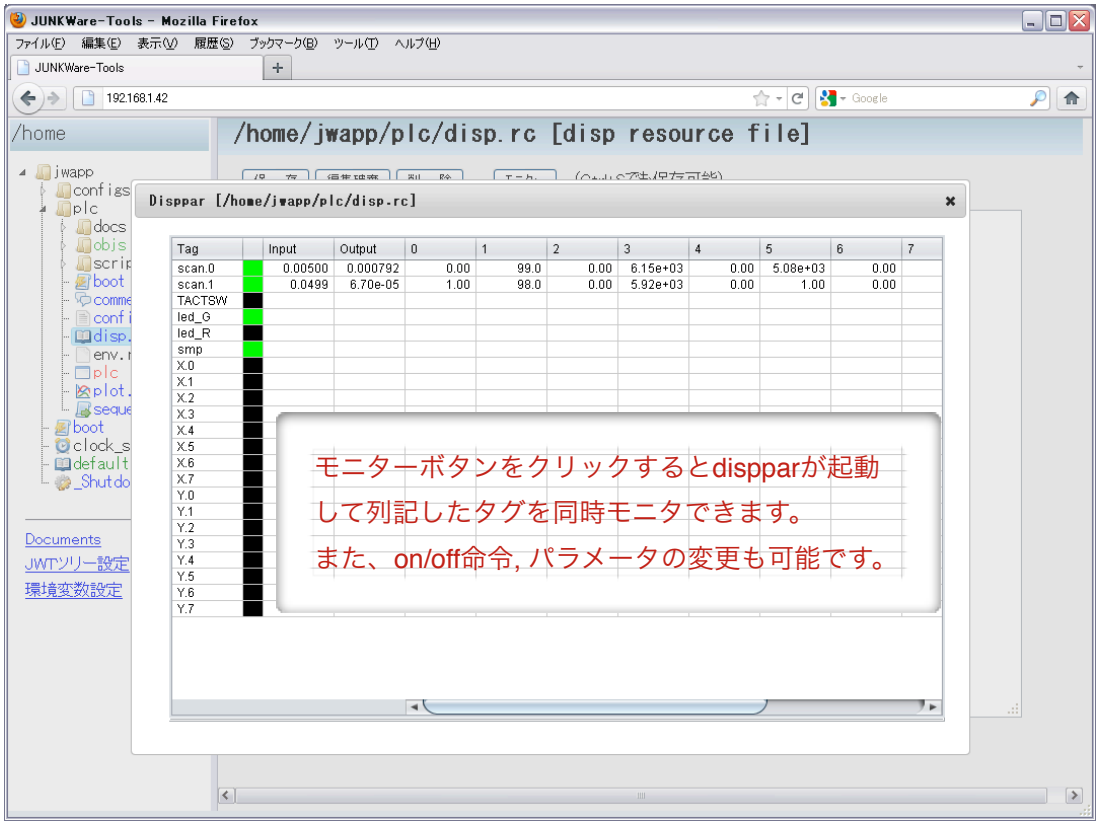


このように動いているソフトウェアの内部パラメータを自由に変更することができますが、この変更は一時的なもののため、恒久的にその内容を反映したい場合は、objs/のタグファイルにその内容を記述しておいてください。

- disp*rcによるタグモニタ

objs/下のタグファイルでは複数のタグを同時にモニタすることができません。そこでdisp*.rcに複数のタグ名をまとめてdispparを起動できます。

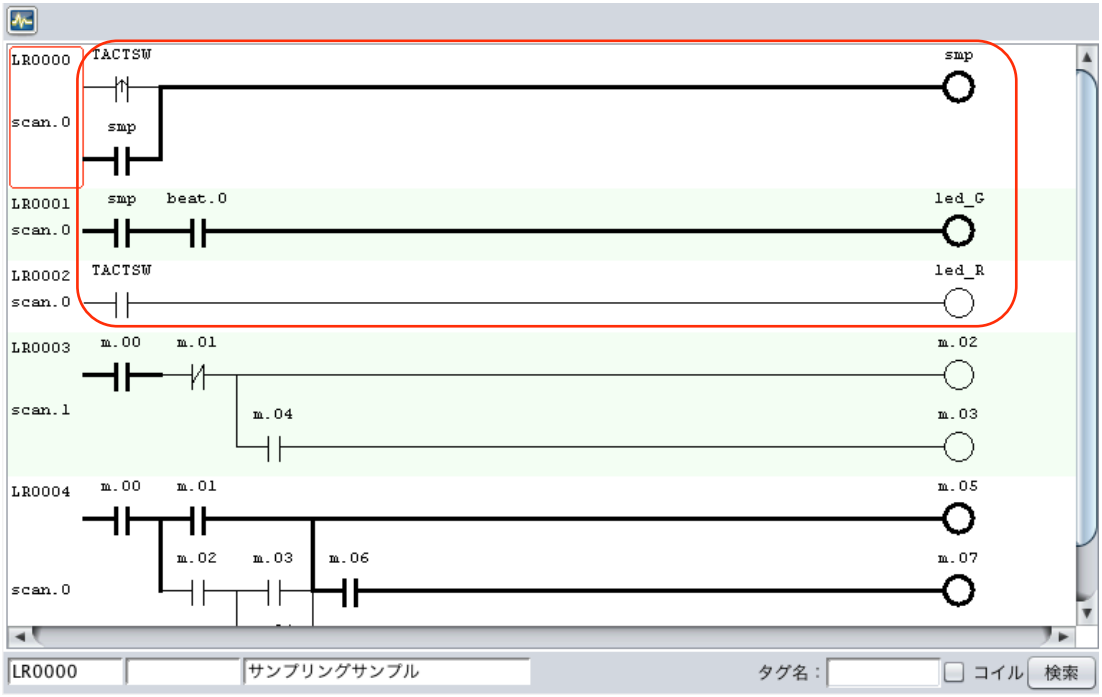




・データロガー機能とグラフ機能

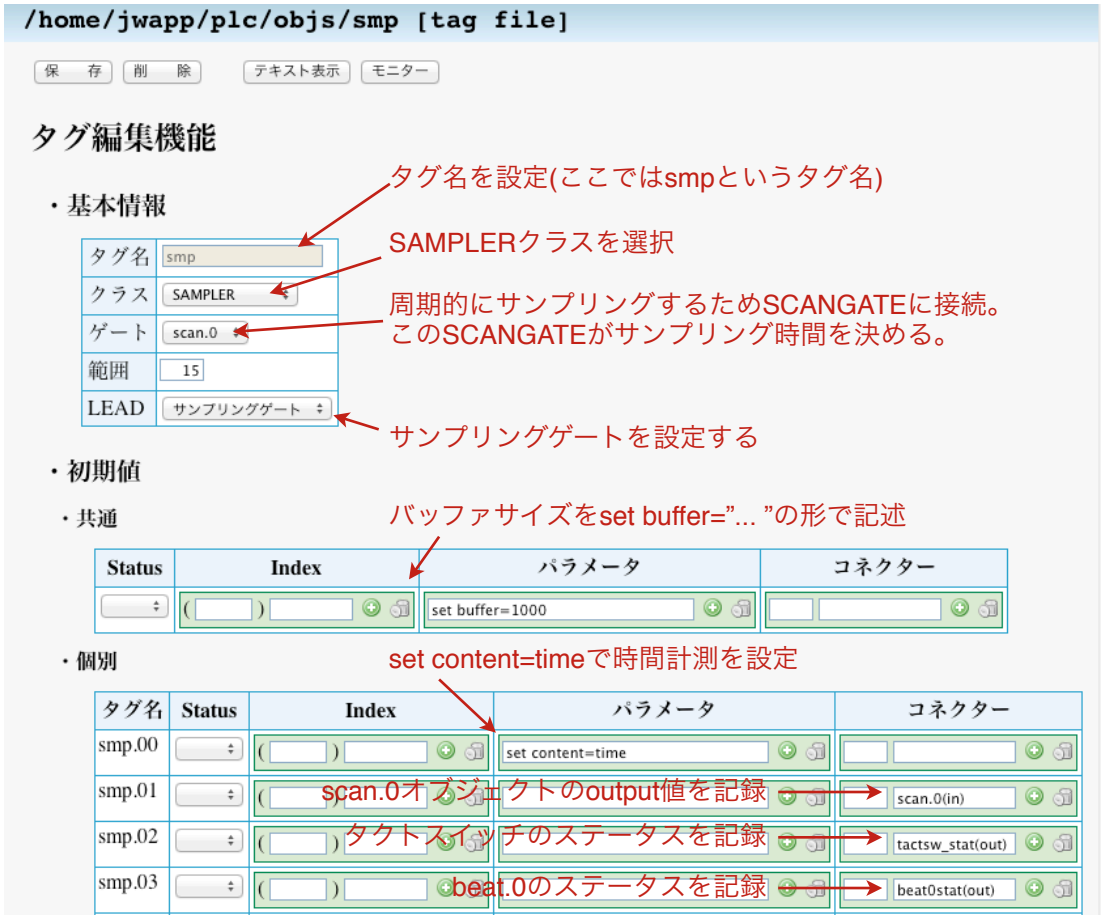
- データロガー機能

JUNJWareの特徴のひとつにデータロガー機能があります。その使い方を示した例がplc/sequence.rcにプリセットしてあり、下図に示すようにサンプル回路が記述されています。

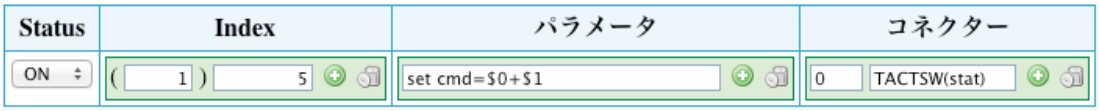


サンプリングはタクトスイッチをトリガとしてスタートして、0.005秒のサンプリングを1000個=5秒間サンプリングを実施します。サンプリング中、緑LEDがフリッカして、タクトスイッチが押下されると赤LEDが点灯します。サンプリング対象にはタクトスイッチのon/offも入っているので、サンプリング中にタクトスイッチも数回押してみてください。緑LEDのフリッカが止まったらサンプリング終了です。

サンプリングオブジェクトのsmpについては、リファレンスマニュアルのSAMPLERの項を参照してください。plc/objs/smpを見ると以下のようにサンプリング対象が定義されています。



ここでsmp.02測定対象になっているtactsw_statのタグ定義を見ると、計算用クラスCALCのオブジェクトで以下のように定義されています。



コネクタにTACTSWのステータスがあり、これをindex(0)に入力しています。常数としてindex(1)=5が設定されていて、計算式として

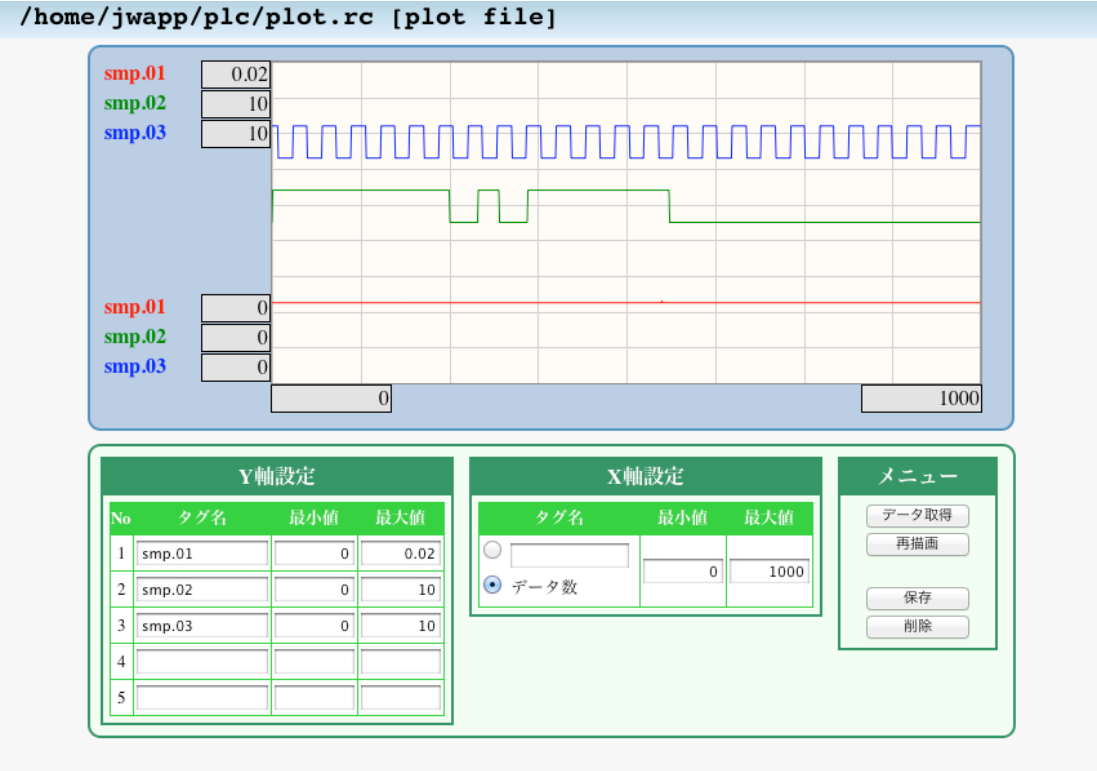
output=index(0)+index(1)

となっているので、TACTSWのステータスが0/1と変化すると、このオブジェクト(tactsw_stat)のoutputは5/6と変化します。smp.02はこの値の記録をとっていることになります。

同様に、smp.03にはbeat.0のステータスをoutput=7/8で記録しています。

- グラフ表示機能

サンプリングした結果をみてみましょう。plc/plot.rcをクリックしてください。ここにsmp.01～smp.03のデータバッファのデータをダウンロードしてきてグラフ表示するように設定してあります。「データ取得」ボタンをクリックすると以下のようなグラフが表示されます。



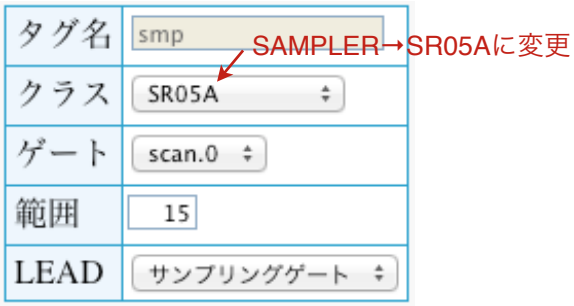
グラフの色は左のタグ名の色に一致しているので、赤-scan.0の周期，緑-TACTSWのステータス，青-beat.0のステータスを表しています。横軸は「データ数」になっているので1000個のデータを等間隔でプロットしています。smpではsmp.00に時間を記録しているので、X軸を時間にとることもできます。

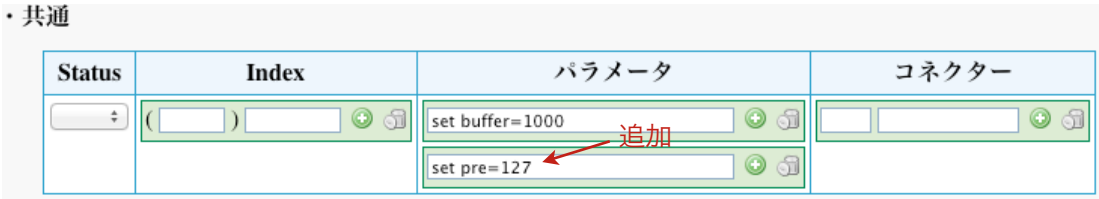
上図のように設定して、再度「データ取得」ボタンを押下してください。0～3秒のデータが表示されます。

※JWTはplot*.rc(*はワイルドカードで任意)を、このグラフ機能定義ファイルとして認識します。複数の定義ファイルを用意しておき、チューニング時に利用できるようにしておくと便利です。

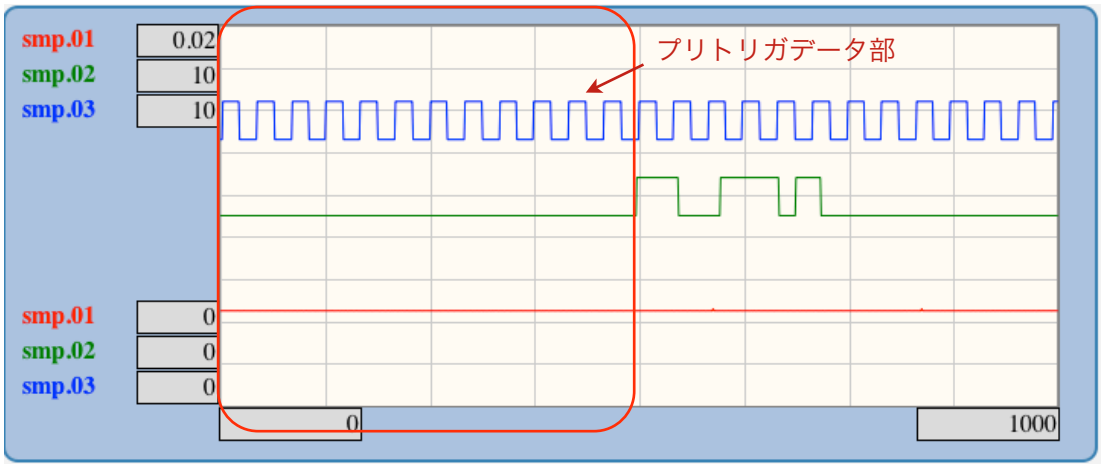
- プリトリガ機能

ここまでSAMPLERクラスを使ったデータロギング機能とグラフ表示機能を説明しましたが、JUNKWareにはSAMPLERクラスの派生としてプリトリガ機能を搭載したデータロギング用クラスも用意してあります。smpオブジェクトを表示して、以下のように変更してみてください。





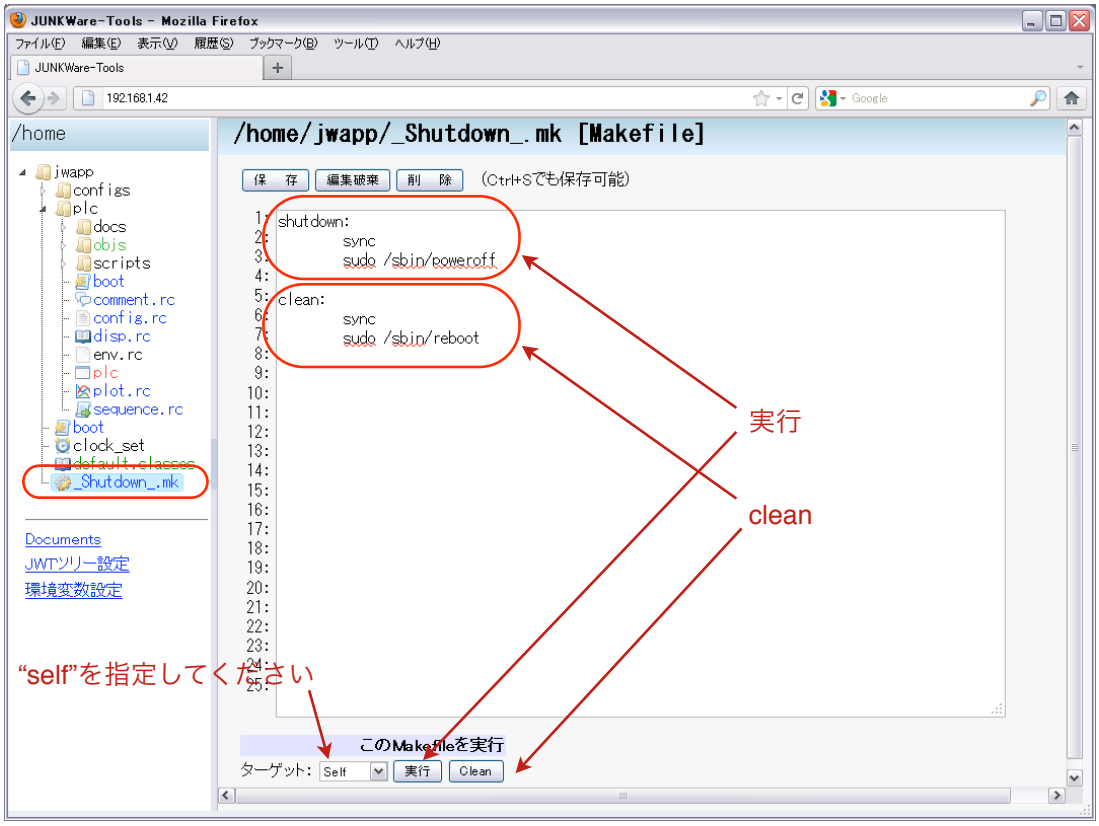
bootスクリプトでソフトウェアPLCプロセスを再起動の後、タクトスイッチでサンプリングを実行→plot.rcでグラフ表示すると以下のようなグラフになります。



プリトリガはロギング用バッファの[pre値]/256をトリガ発生前のデータ領域として使います。従って、タクトスイッチ押下の前のデータを取得することができます。pre値を適当に変えてみて、その動きを確認してみてください。

5. makeファイルの利用方法

JWTは*.mk(*は任意)ファイルをmakeファイルとして認識して実行することができます。makeファイルにはターゲットを設定しますが、定義されている最初のターゲットを画面下部の「実行」ボタンで、2番目のターゲットを「clean」ボタンで実行できます。makeファイルの中身については、JWTのエディタ機能で自由に修正/保存ができるので、ルーチンワークをmakeファイルにしておくと、作業効率をアップできます。



上記の例ではPLC-FARMにプリセットしてある_Shutdow_.mkを表示していますが、「実行」ボタンでPLC-FARMをシャットダウン, 「clean」ボタンでリブートする機能を定義しています。

IV. JUNKWareのクラスリファレンス

JUNKWareに標準で搭載されているクラスは以下の通りです。JUNKWareでは、これまでに説明してきた方法でこれらのクラスのオブジェクトリソースを定義することにより、すぐにこれらの機能を使うことできできます。

JUNKWare標準クラス一覧

種別	クラス名	機能
スキャンゲート	SCANGATE	ソフトウェアシーケンサのスキャン動作の周期を設定します。
汎用デバイス	RELAY	内部リレークラスです。
	THRU	内部データバッファです。シーケンサのデータメモリに相当します。
	FLAG	セットされた値によりステータスがon/offに変化します。0がセットされた場合offに、それ以外の場合onになります。
	ONDELAY	通常のタイマです。アサートされるとinputにセットされた時間(秒)経過後、自らのステータスがonになり、ディアサートされると即時offになります。
	OFFDELAY	offになる際に動作するタイマです。ディアサートされるとinputに設定された時間(秒)経過後offになります。
	COUNTER	入力のOFF→ONの立上り回数がプリセット値以上になるとONします。
	BEAT	自己発振するオブジェクトです。
ソフトウェアリミットスイッチ	LIMITSW	inputの値がしきい値を越すとonするリミットスイッチです。inputにはコンジットを接続することができます。
	PROXSW	inputの絶対値が設定範囲に入っている場合にonするスイッチです。

種別	クラス名	機能
	AREASW	inputの値がある範囲に入っている場合にonするスイッチです。
外部プロセス実行	EXEC	Linux上にあるコマンド(バイナリ, シェルスクリプト等)を実行します。
データロギング	SAMPGATE	SAMPLERを束ねるゲートです。
	SAMPLER	データ蓄積バッファです。
	SR05A	SAMPLERにプリトリガ機能を追加したバッファです。
計算	CALC	任意精度の汎用計算機です。
外部参照	IPCon	外部(プロセス)のタグをON/OFFします。
	IPCa	外部 (プロセス) のタグの状態を反映します。
	IPCpull	外部 (プロセス) のタグの値を反映します。
	IPCpush	外部 (プロセス) のタグの値を設定します。
デバイス	A4x0GPIO	Armadillo-4x0のGPIOをDIOとして使うためのクラスです。
	A4x0LED	Armadillo-4x0のLEDを制御するクラスです。
	A4x0TACTSW	Armadillo-4x0のタクトスイッチを制御するクラスです。
	DEVICE	デバイス(DIOカード,ADカード等)を定義するクラスです。
	ISVR	AD,カウンタ等のデバイスから値を取得するバッファです。
	ODVR	DA等のデバイスに値を出力するバッファです。
	INGATE	DIデバイスからのステータスを取得するバッファです。
	OUTGATE	DOデバイスに出力するバッファです。

※デバイスクラスのうち、A4x0GPIO以外はPLC-FARMで使用しないため、ここでは記述しません。

SCANGATE

動作概要　シーケンサの基本となるスキャン動作の周期を担当するクラスです。複数のSCANGATEを用いてリレー等のデバイスを振り分けることにより、CPUの処理を最適化することができます。

パラメータ

input　[Read]最新のスキャン時間がセットされます。

output　[Read]最新のスキャンで実質的な動作に必要だった時間がセットされます。

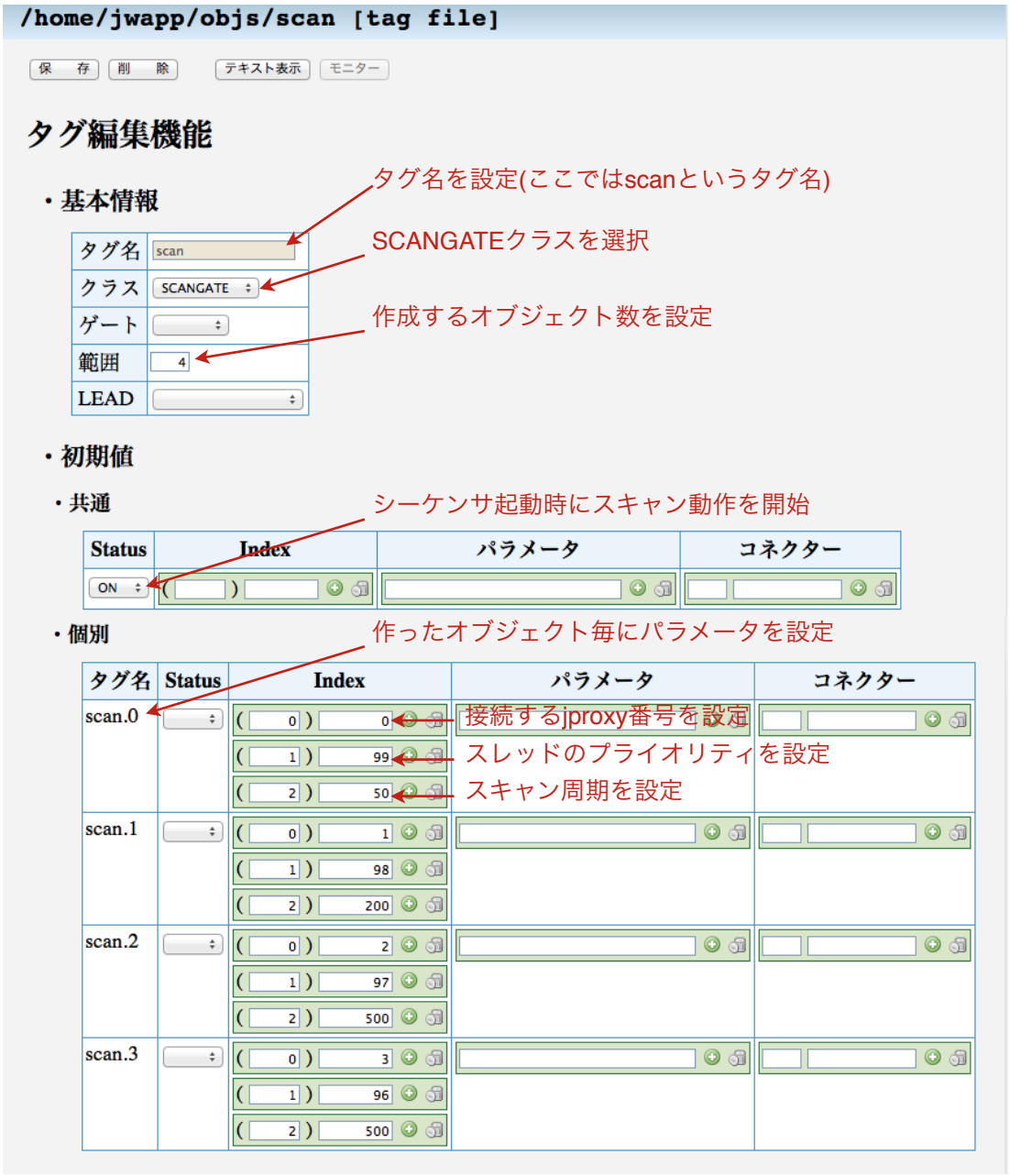
0　[Write]接続する/dev/jproxy[0〜7]のデバイスの番号0〜7をセットします。

1　[Write]スレッドのプライオリティ1〜99を設定します。

2　[Write]接続している/dev/jproxy[0〜7]の動作周期時間を設定します。単位は100μ秒(0.1m秒)です。

説　　明　SCANGATEはLinuxのスレッドで、カーネルから一定周期で実行権を与えられて動作します。一回のスキャン動作に設定された周期以上の時間がかかってしまった場合、一回分のスキャンが実行されなくなることに注意してください。また、実行時間に予測のつかないオブジェクトは他に影響を与えないようにSCANGATEを分離しておきましょう。

定義方法


/home/jwapp/objs/scan [tag file]
保 存
削 除
テキスト表示
モニター
タグ編集機能
・基本情報
タグ名
scan
クラス
SCANGATE
ゲート
範囲
4
LEAD
タグ名を設定(ここではscanというタグ名)
SCANGATEクラスを選択
作成するオブジェクト数を設定
・初期値
・共通
Status
Index
パラメータ
コネクター
ON
シーケンサ起動時にスキャン動作を開始
・個別
タグ名
Status
Index
パラメータ
コネクター
scan.0
scan.1
scan.2
scan.3
作ったオブジェクト毎にパラメータを設定
接続するjproxy番号を設定
スレッドのプライオリティを設定
スキャン周期を設定

RELAY

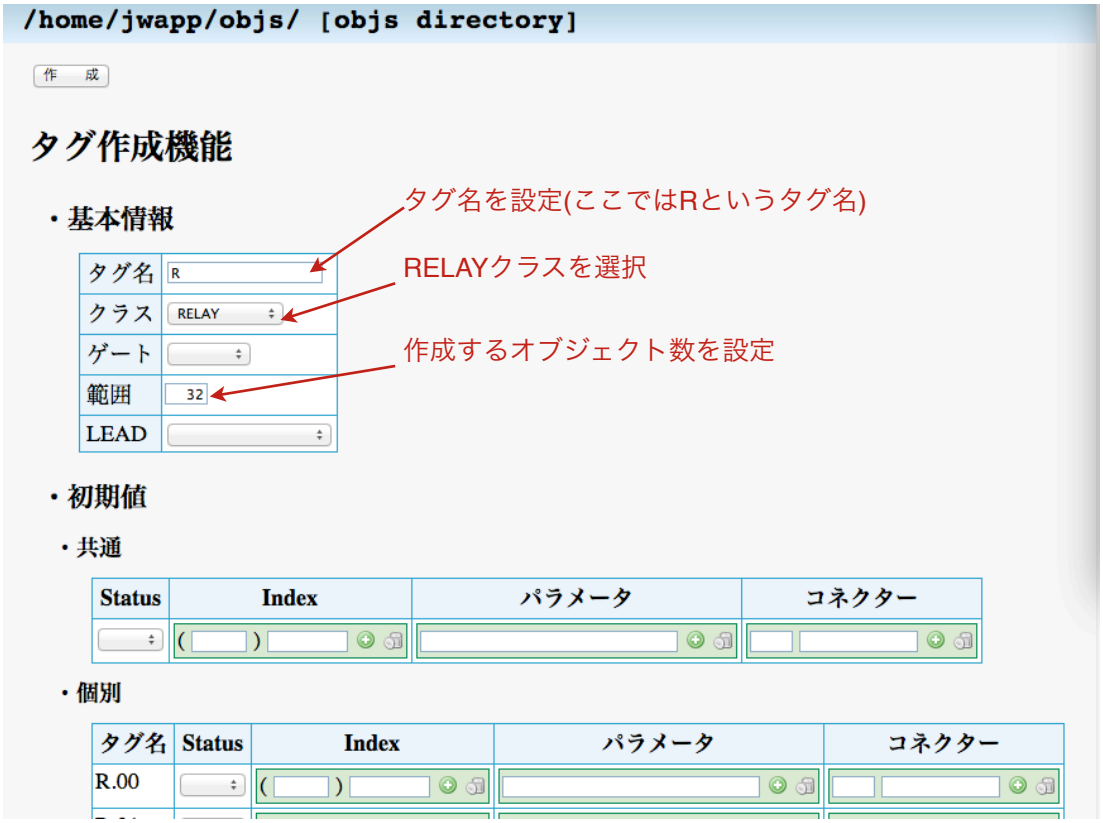
動作概要　内部リレー

パラメータ

Status　[Read]現在のステータスを取得できます。
[Wirte]ステータスをON/OFFできます。

説　　明　シーケンサで内部で使用する内部リレーです。

定義方法



THRU

動作概要　内部データバッファ

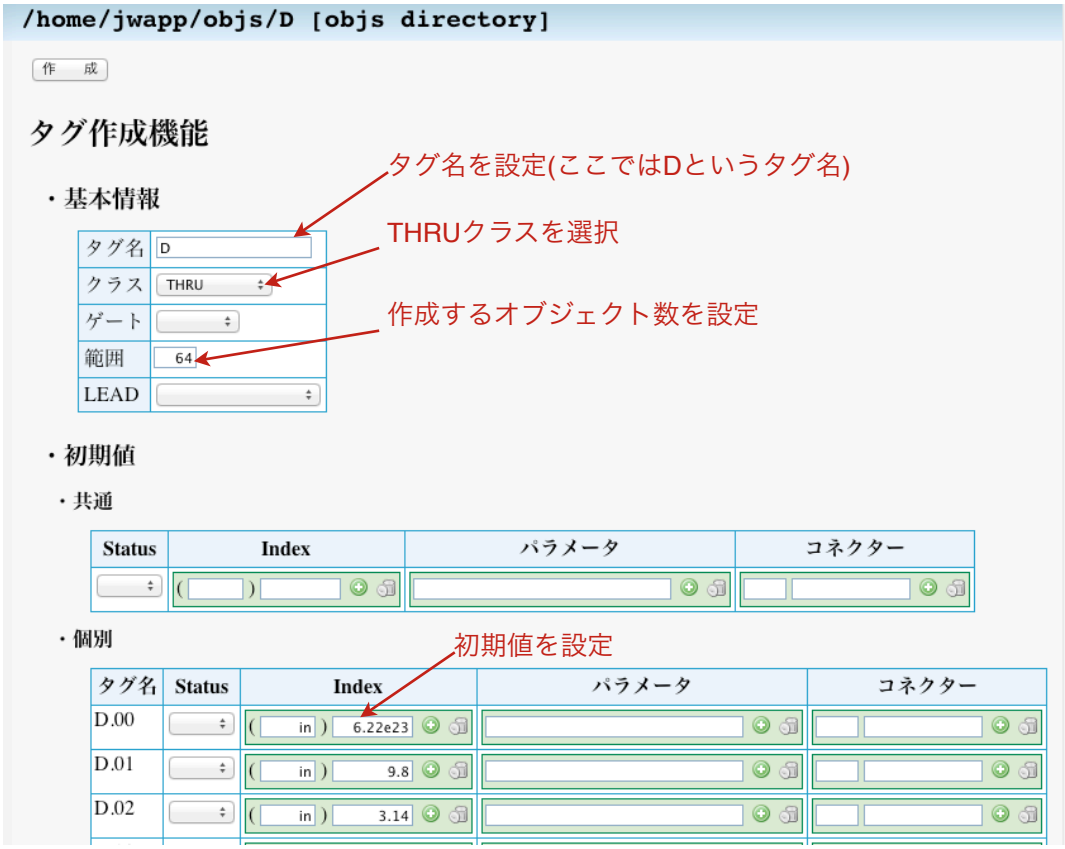
パラメータ

input　[Write]データを書込みます。

output　[Read]inputに書込んだデータをそのまま出力します。

説　　明　シーケンサのデータメモリと同じ使い方ができます。ただし、SRAMではありませんので、起動時にはゼロに初期化されています。

定義方法



FLAG

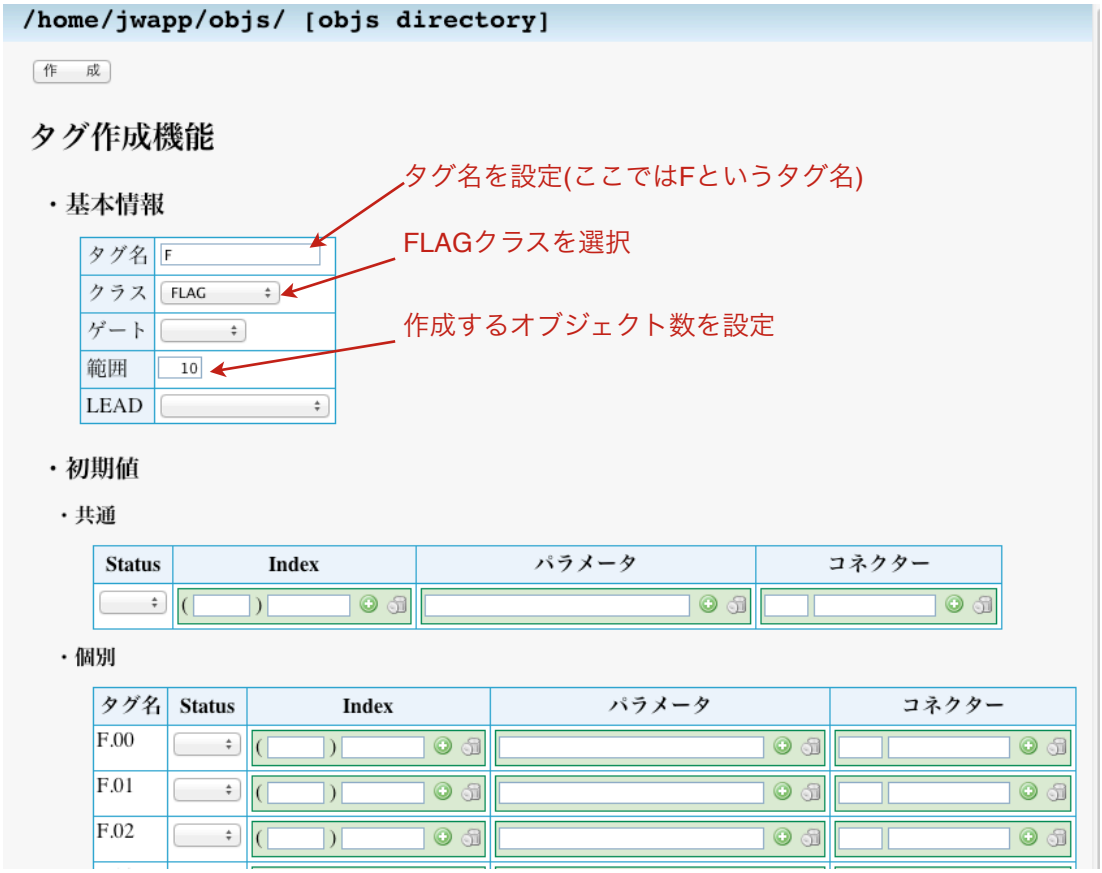
動作概要　ゼロ判定フラグ

パラメータ

status　[Read]input=0の場合OFF(0), それ以外の場合ON(1)となります。
input　[Write]データを書込みます。
output　[Read]status ON=1,OFF=0を出力します。

説　　明　入力された数値のゼロ判定に使います。-1<input<1の設定値でゼロ(OFF)と判定します。

定義方法



ONDELAY

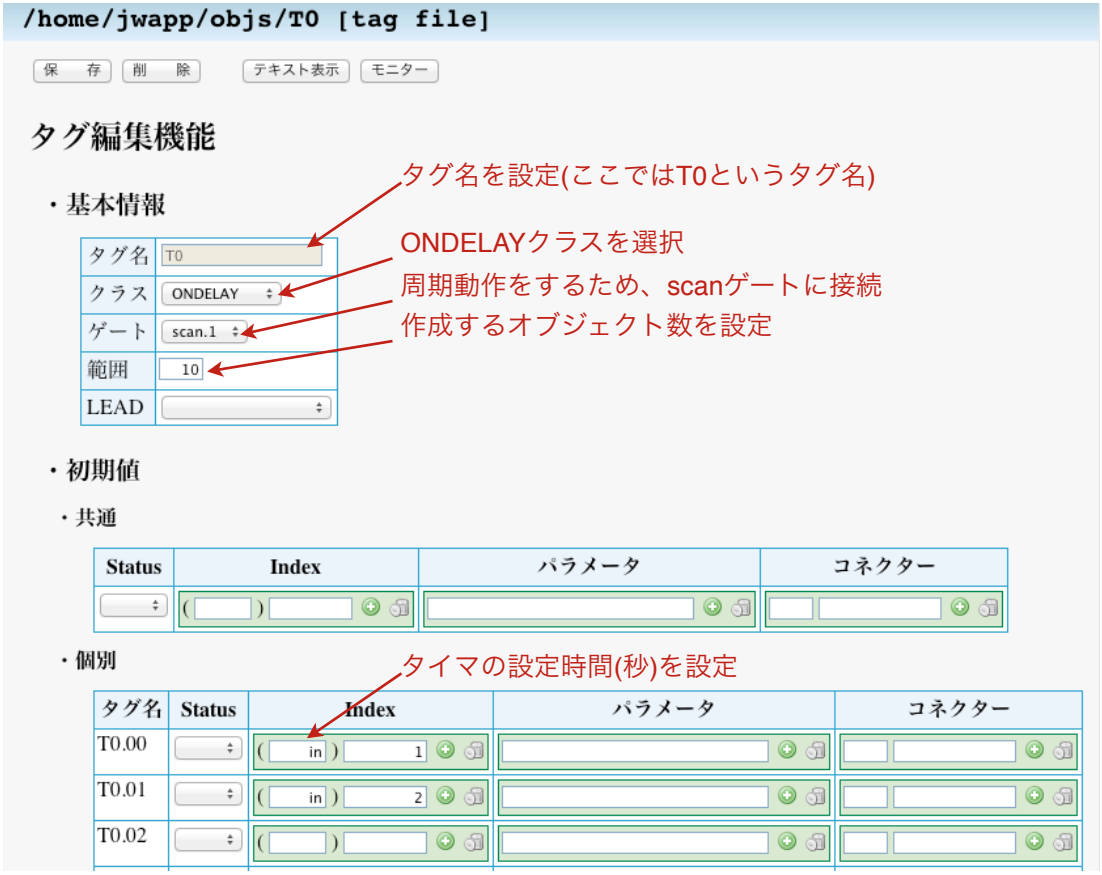
動作概要　タイマ(オンディレイ)

パラメータ

status　[Read]タイマ動作結果のステータスを出力します。
input　[Write]タイマ設定時間(秒)を設定します。
output　[Read]アサートされている間、設定時間のカウントダウン動作を行います。

説　　明　アサートされるとinputにセットされた時間(秒)をoutputにセットしてタイマ動作を開始します。outputの値が0になる時に自らのステータスをonにします。ディアサートされると即時offになります。

定義方法



OFFDELAY

動作概要　オフディレイタイマ

パラメータ

status　[Read]タイマ動作結果のステータスを出力します。
input　[Write]タイマ設定時間(秒)を設定します。
output　[Read]アサートされている間、設定時間のカウントダウン動作を行います。

説　　明　アサートされるとonになります。ディアサートされると、 inputにセットされた時間(秒)をoutputにセットしてタイマ動作を開始します。 outputの値が0になる時に自らのステータスをoffにします。

定義方法

COUNTER

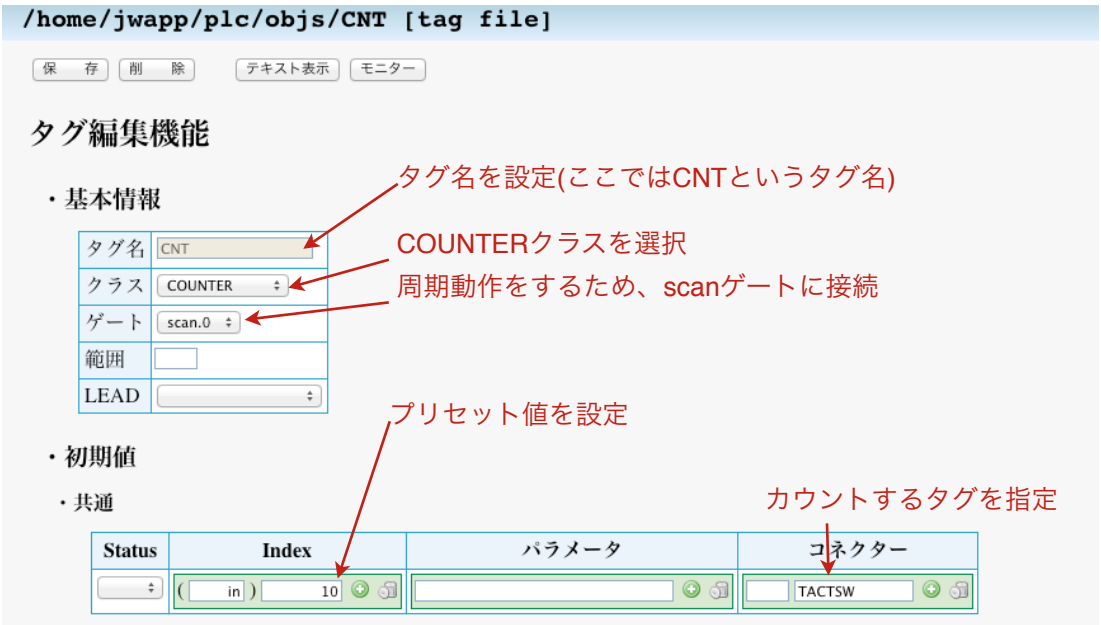
動作概要　カウンタ

パラメータ

status　[Read]カウンタ動作結果のステータスを出力します。
input　[Write]カウンタしきい (プリセット) 値を設定します。
output　[Read]現在のカウント数を出力します。

説　　明　アサートされるとカウント動作を開始します。ディアサートされると、カウント値をリセットして、ステータスはOFFになります。inputにはONするカウント値をプリセットしておきます。プリセット値を変更すると、カウント値はリセットされます。

定義方法



BEAT

動作概要　自己発振器

パラメータ

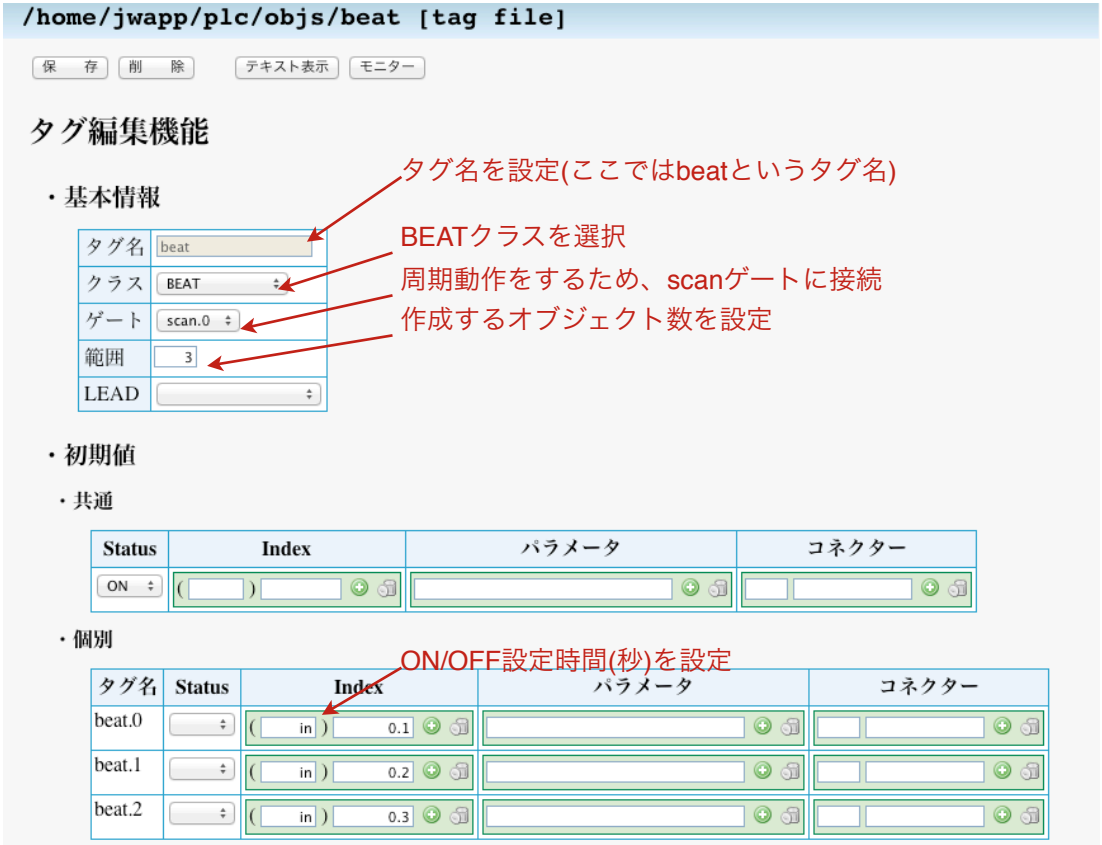
status　[Read]発信器としてのステータスを出力します。
[Write]レベルをACTIVE/INACTICEに設定します。

input　[Write]ON/OFF動作の変化する時間幅(秒)を設定します。

output　[Read]次にステータスが変化するまでの時間経過を出力します。

説　　明　アサートされると発振動作を開始して、ディアサートすると停止します。動作中は、inputに設定された時間毎にON/OFFにステータスが変化します。

定義方法



LIMITSW

動作概要　入力値と設定されたしきい値を比較してon/offするリミットスイッチです。

パラメータ

status　[Read]現在のステータス(on/off)を出力します。

input　[Read/Write]しきい値を設定します。

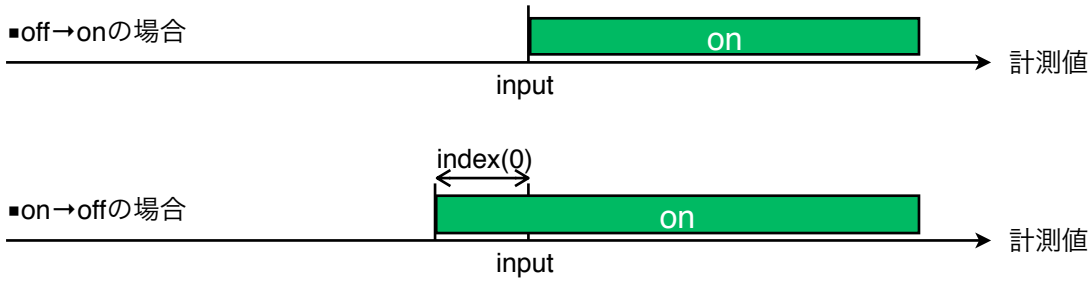
output　[Read] コネクタにより接続された入力値を出力します。

0　[Read/Write]ヒステリシスを設定します。

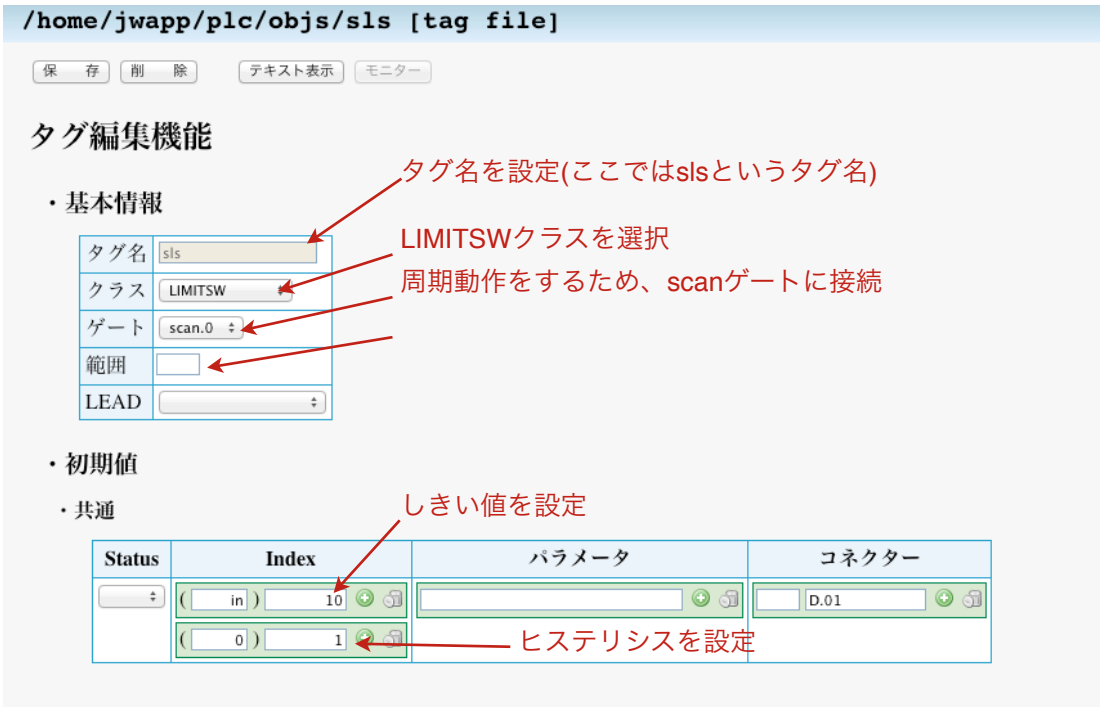
コネクタ　<%>タグ名(index)

上記形式で指定されたタグのデータを参照してリミットスイッチ動作をします。(index)でタグデータの場所を指定できます。(index)を省略した場合、タグのoutput値を用います。

説　　明　下図に示すように、outputのデータ(コネクタ接続)がinputに設定されたしきい値を越すとonになり、下回るとoffになります。index=0にしきい値を設定することができ、ここに値を設定するとoffになる値を変更することが可能です。



定義方法


/home/jwapp/plc/objs/sls [tag file]
保　存
削　除
テキスト表示
モニター
タグ編集機能
・基本情報
タグ名
sls
クラス
LIMITSW
ゲート
scan.0
範囲
LEAD
・初期値
・共通
Status
Index
パラメータ
コネクター
in
10
0
1
D.01
タグ名を設定(ここではslsというタグ名)
LIMITSWクラスを選択
周期動作をするため、scanゲートに接続
しきい値を設定
ヒステリシスを設定

PROXSW

動作概要　入力値と設定されたしきい値を比較してon/offするリミットスイッチです。

パラメータ

status	[Read]現在のステータス(on/off)を出力します。
input	[Read/Write]しきい値を設定します。
output	[Read] コネクタにより接続された入力値を出力します。
0	[Read/Write]領域シフト幅を設定します。
コネクタ	<%>タグ名(index) 上記形式で指定されたタグのデータを参照してリミットスイッチ動作をします。(index)でタグデータの場所を指定できます。(index)を省略した場合、タグのoutput値を用います。

説　　明　下図に示すように、outputのデータ(コネクタ接続)がinputに設定されたしきい値を越すとonになり、下回るとoffになります。index=0に領域判定幅を設定することができ、ここに値を設定するとにより、領域をシフトすることができます。

定義方法　下図にLIMITSWと同じなのでLIMITSWの項を参照してください。

AREASW

動作概要　入力値が指定された範囲内か外かでon/offするリミットスイッチです。

パラメータ

status　[Read]現在のステータス(on/off)を出力します。

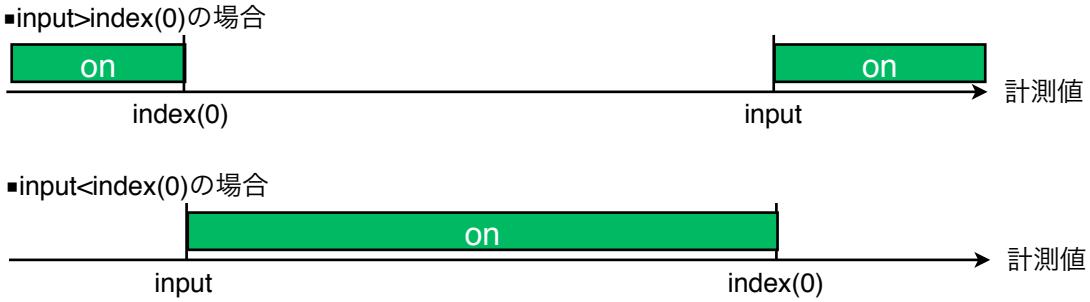
input　[Read/Write]onの下限値を設定します。

output　[Read] コネクタにより接続された入力値を出力します。

0　[Read/Write]onの上限値を設定します。

コネクタ　<%>タグ名(index)
上記形式で指定されたタグのデータを参照してリミットスイッチ動作をします。(index)でタグデータの場所を指定できます。(index)を省略した場合、タグのoutput値を用います。

説　　明　下図に示すように、outputのデータ(コネクタ接続)が下限(input)より大きく、index=0に設定された値より小さい場合にonします。(input)>(0)の場合は2つの領域でonします。



定義方法　下図にLIMITSWと同じなのでLIMITSWの項を参照してください。

EXEC

動作概要　シーケンサモジュールからLinuxの外部プロセス実行します。

パラメータ

status　[Read]外部コマンド実行中”on”になります。実行したプロセスが完了すると、”off”に戻ります。

[Write]外部コマンドの実行を開始します。

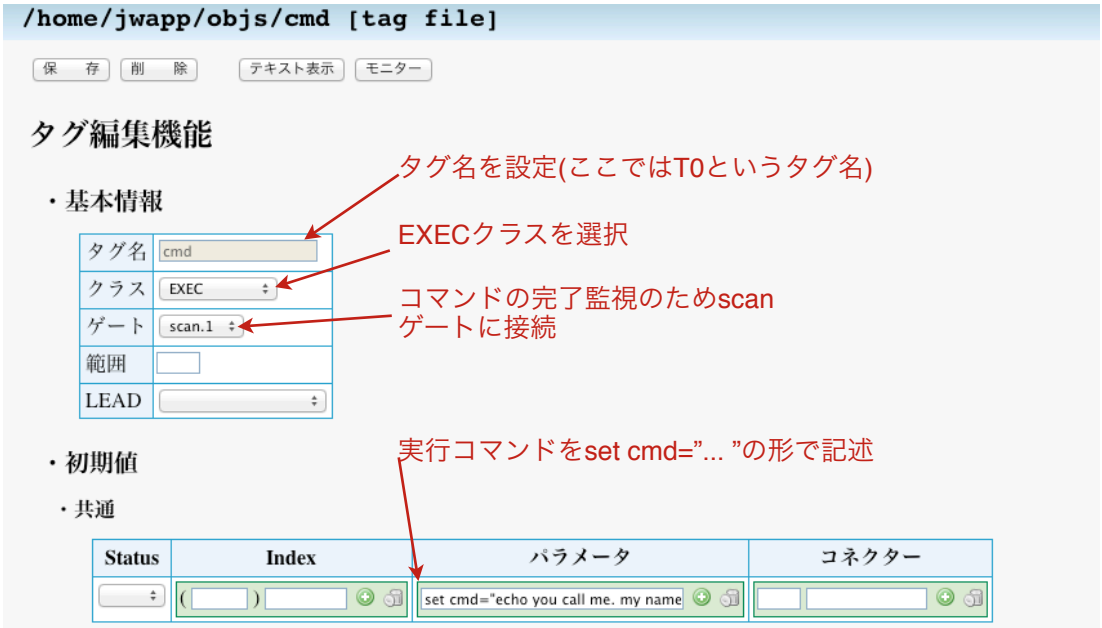
cmd　set　cmd=”...”の形で、...の部分に外部コマンドを格納します。jwappユーザーに設定されているパス内に無いコマンドを実行する場合はフルパスで指定してください。また、引数等もそのまま指定できます。

説　　明　外部コマンド(シェルスクリプトも含む)を実行することのできるクラスです。JUNKWareのPLCだけでは実現できない機能を外部コマンドで用意して、その実行のロジックをJUNKWareで記述するようなケースで使用してください。

※コマンド実行中は自身のstatusがonになるので、２重実行はできません。

※コンジットにより、パラメータの指定ができると良い

定義方法



SAMPLER

動作概要　データロギング用バッファクラスです。

パラメータ

status　[Read]データロギング実行中”on”になります。バッファがいっぱいになり、ロギングを完了すると、”off”に戻ります。
[Write]データロギングを開始/停止することができます。

ゲート　サンプリングしたい時間間隔のSCANGATEを指定してください。

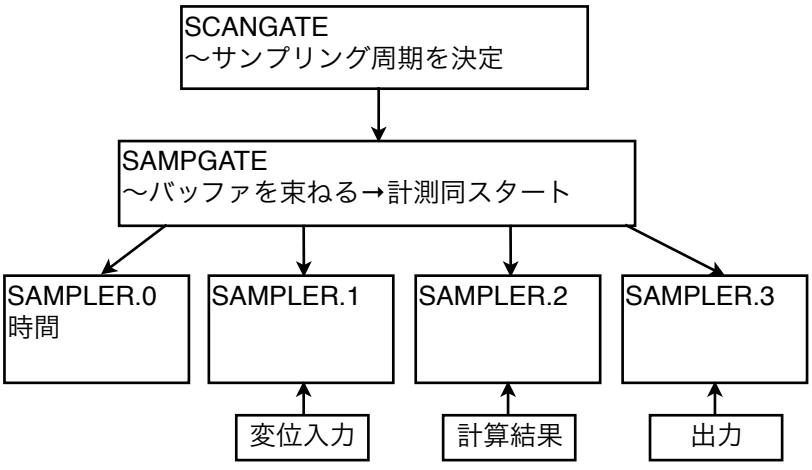
lead　SAMPLERのオブジェクトを束ねるSAMPGATEを指定します。

buffer　set　buffer=[バッファサイズ]の形で、使用するバッファサイズを指定します。接続するSCANGATEによりサンプリング周期が変わるので、
[バッファサイズ]×サンプリング周期=計測時間
を基準にバッファサイズを決めてください。

content　set　content=timeと指定することができます。この設定をされたバッファは計測時間[秒]をロギングします。

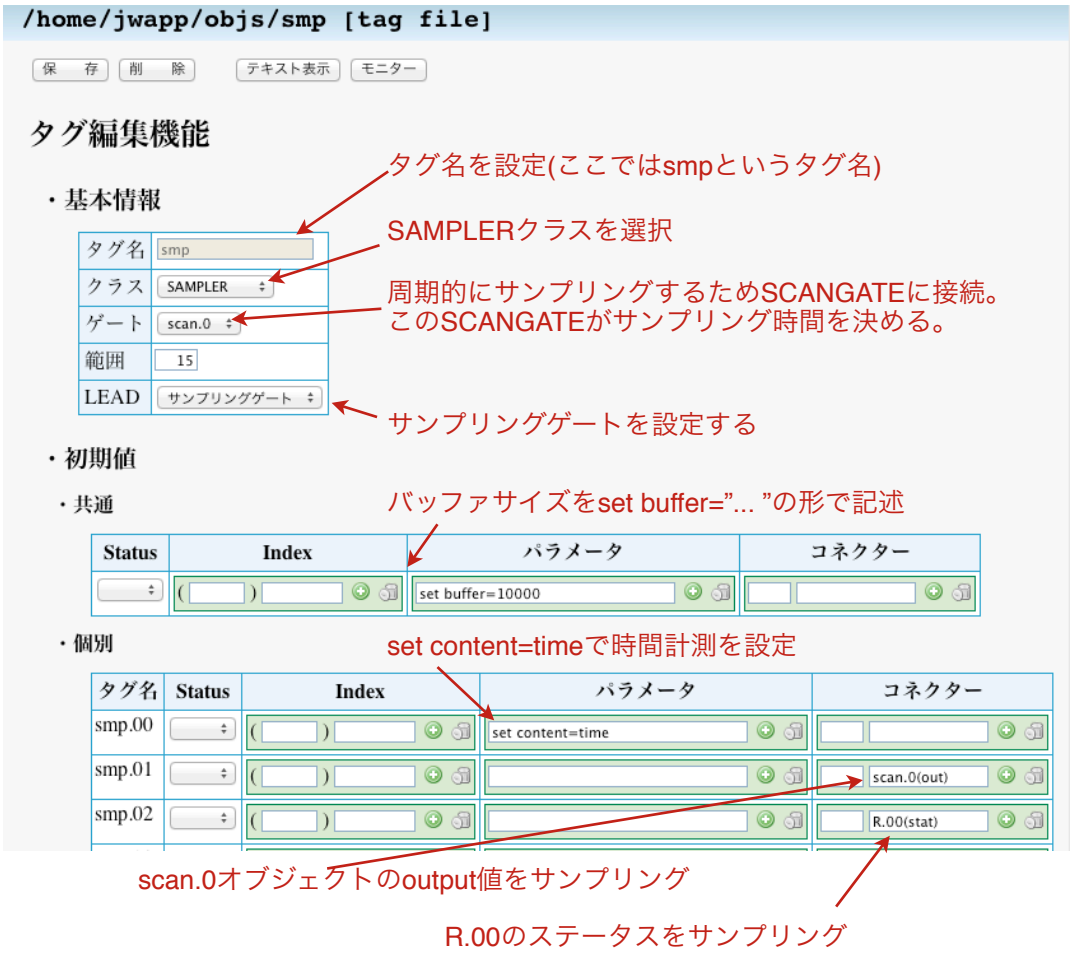
コネクタ　コネクタでロギングするデータのタグを指定します。タグ名のみを指定すると、outputのデータをロギングします。タグ名(index)と指定すれば、タグの指定されたインデックスのデータをロギングします。また、タグ名(stat)と指定することにより、ステータスを0,1でロギングすることができます。

説　　明　システムの動きを確認，調整するためにデータロギングするための機能を提供します。使い方が少々複雑ですが、以下のような構成になっていることに留意してください。



※ロギングしたデータはJWTのグラフ機能で表示することができます。

定義方法


/home/jwapp/objs/smp [tag file]
保　存
削　除
テキスト表示
モニター
タグ編集機能
・基本情報
タグ名
smp
クラス
SAMPLER
ゲート
scan.0
範囲
15
LEAD
サンプリングゲート
・初期値
・共通
Status
Index
パラメータ
コネクター
set buffer=10000
・個別
タグ名
smp.00
smp.01
smp.02
set content=time
scan.0(out)
R.00(stat)
タグ名を設定(ここではsmpというタグ名)
SAMPLERクラスを選択
周期的にサンプリングするためSCANGATEに接続。
このSCANGATEがサンプリング時間を決める。
サンプリングゲートを設定する
バッファサイズをset buffer=”... ”の形で記述
set content=timeで時間計測を設定
scan.0オブジェクトのoutput値をサンプリング
R.00のステータスをサンプリング

SR05A

動作概要　プリトリガ機能を持ったデータロガーです。

パラメータ

status　[Read]データロギング実行中”on”になります。バッファがいっぱいになり、ロギングを完了すると、”off”に戻ります。
[Write]データロギングを開始/停止することができます。

ゲート　サンプリングしたい時間間隔のSCANGATEを指定してください。

lead　SAMPLERのオブジェクトを束ねるSAMPGATEを指定します。

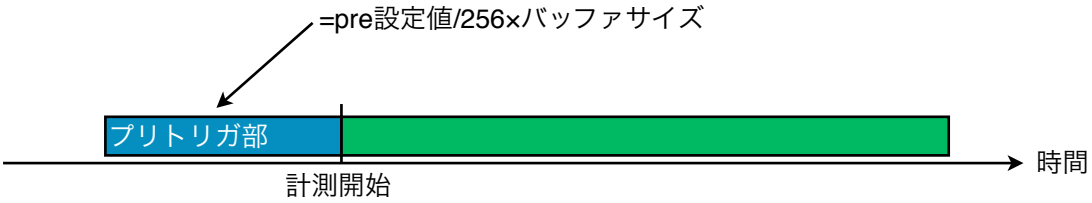
buffer　set　buffer=[バッファサイズ]の形で、使用するバッファサイズを指定します。接続するSCANGATEによりサンプリング周期が変わるので、
[バッファサイズ]×サンプリング周期=計測時間
を基準にバッファサイズを決めてください。

pre　set pre=0～255でプリトリガ機能に使うバッファサイズ]をで指定します。

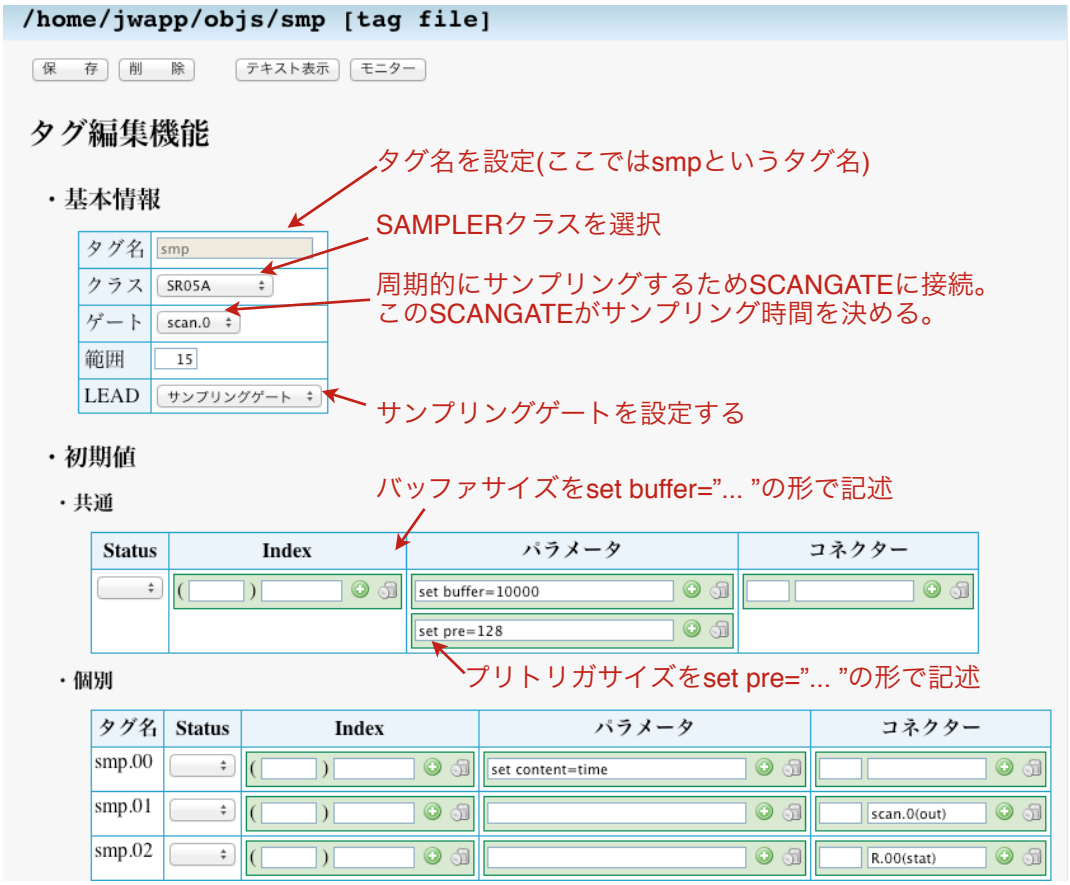
content　set　content=timeと指定することができます。この設定をされたバッファは計測時間[秒]をロギングします。

コネクタ　コネクタでロギングするデータのタグを指定します。タグ名のみを指定すると、outputのデータをロギングします。タグ名(index)と指定すれば、タグの指定されたインデックスのデータをロギングします。また、タグ名(stat)と指定することにより、ステータスを0,1でロギングすることができます。

説　　明　基本的な使い方はSAMPLERと同じです。preオプションでプリトリガに使うサイズを指定してください。255を指定するとすべてのバッファがプリトリガに割り当てられます。



定義方法


/home/jwapp/objs/smp [tag file]
保　存
削　除
テキスト表示
モニター
タグ編集機能
・基本情報
タグ名
smp
クラス
SR05A
ゲート
scan.0
範囲
15
LEAD
サンプリングゲート
タグ名を設定(ここではsmpというタグ名)
SAMPLERクラスを選択
周期的にサンプリングするためSCANGATEに接続。
このSCANGATEがサンプリング時間を決める。
サンプリングゲートを設定する
・初期値
・共通
バッファサイズをset buffer="... "の形で記述
Status
Index
パラメータ
コネクター
set buffer=10000
set pre=128
プリトリガサイズをset pre="... "の形で記述
・個別
タグ名
Status
Index
パラメータ
コネクター
smp.00
set content=time
smp.01
scan.0(out)
smp.02
R.00(stat)

CALC

動作概要　計算機能を提供します。

パラメータ

status　[Read]現在のステータスを出力します。
[Write]on-計算実行状態にします。off-計算を実行しません。

input　未使用

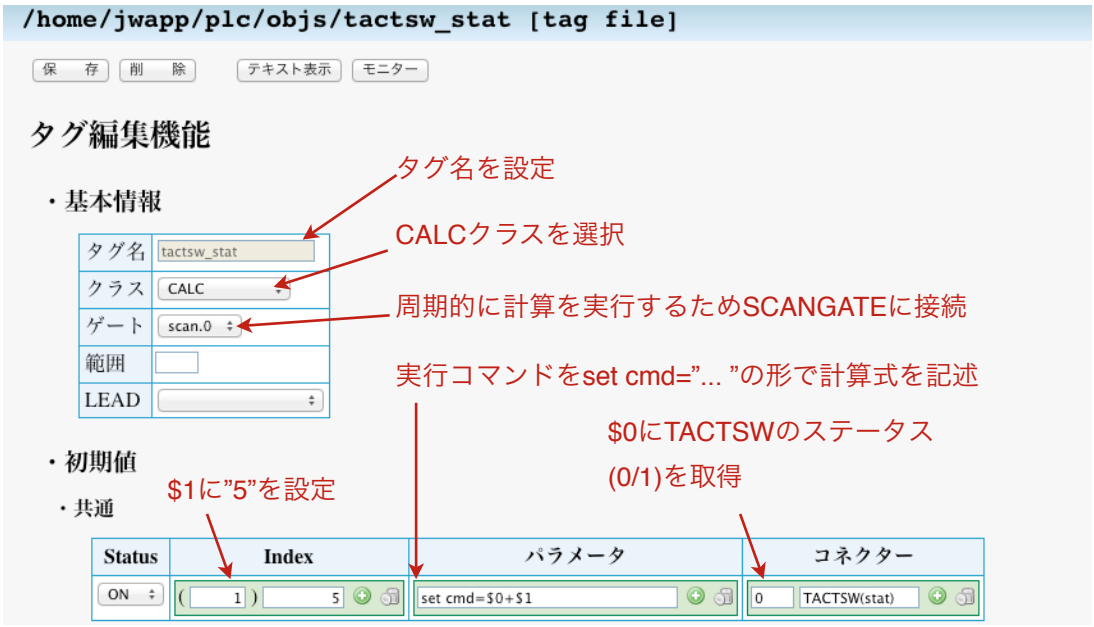
output　計算結果を出力します。

cmd　set　cmd="..."の形で、計算式を記述します。記述方法はUNIXのコマンド"bc"の形式に準じますが、基本的な記述方法は通常の加減乗除[+,-,*,/]を使うだけです。$0,$1,,,$9を使って自分の持っているデータを使うことができます。

説　　明　"bc"という外部コマンドを使って計算する機能を提供します。作ったオブジェクトのインデックスにコネクタを使って外部のデータを引用することができるため、内部のすべてのパラメータやデータを使用することが可能で、また小数点を含む演算もそのまま実行することが可能です。
bcは"-l"オプション付きで起動してあるので、以下の関数も使うことができます。

```
s (x)    sin (x の単位はラジアン)
c (x)    cos (x の単位はラジアン)
a (x)    atan (返り値の単位はラジアン)
l (x)    log (自然対数)
e (x)    exp (指数関数)
j (n,x)  整数 n 次のベッセル関数
```

定義方法



上記の例ではTACTSWのステータスをインデックス0に引っ張ってきて、インデックス1にプリセットしたデータ"5"を加算した結果をoutputに取得しています。

IPCon

動作概要　外部(プロセス)のタグをon/offします。

パラメータ

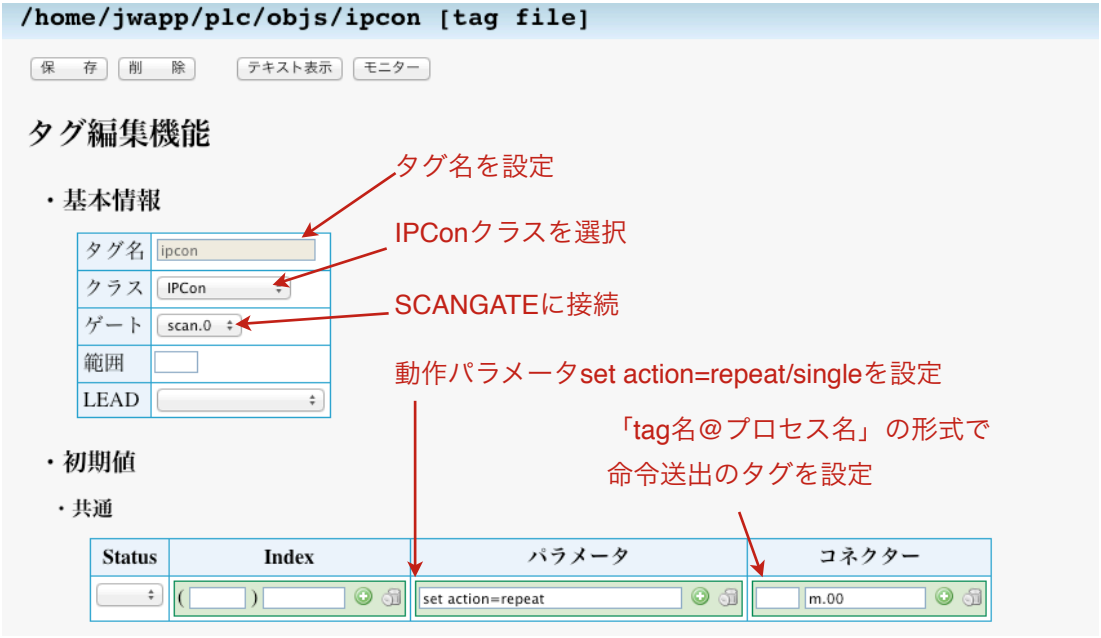
status　[Read]現在のステータスを出力します。
[Write]on書込み時、コンジットで接続されたオブジェクトにenableまたはdisableの命令を送出します。

action　set action=repeat/single(default)
“repeat”を設定することにより、ステータスの立上り時だけではなく、ステータス=onの間、毎スキャンenableまたはdisable命令を送出します。singleを設定するとoff→onの立上り時に一度だけ命令を送出します。
set action=off/on(default)
“off”を設定することにより、disable命令を送出します。デフォルトは”on”のため、enable命令を送出します。

コネクタ　<%>タグ名 [@プロセス]
enable,disable命令を送出するオブジェクトのタグを設定します。@プロセスを指定すると外部のプロセスのタグに命令を送ることができます。

説　　明　IPC=InterProcessCommunicationの意味です。
IPConは外部のオブジェクトにenable,disableの命令を送出します。基本的にonの場合に命令を送出し、action=repeatを設定すると、onの間命令を送出し続けます。
一度命令を送出した後再度送出するには、自身のステータスを一旦offにして、再度onにすることにより命令を送出します。

定義方法



上記の例ではipconのタグがonの間、scan.0の毎周期ごとにm.00にon命令を送出します。

IPCa

動作概要　外部(プロセス)のタグのステータスをコピーします。

パラメータ

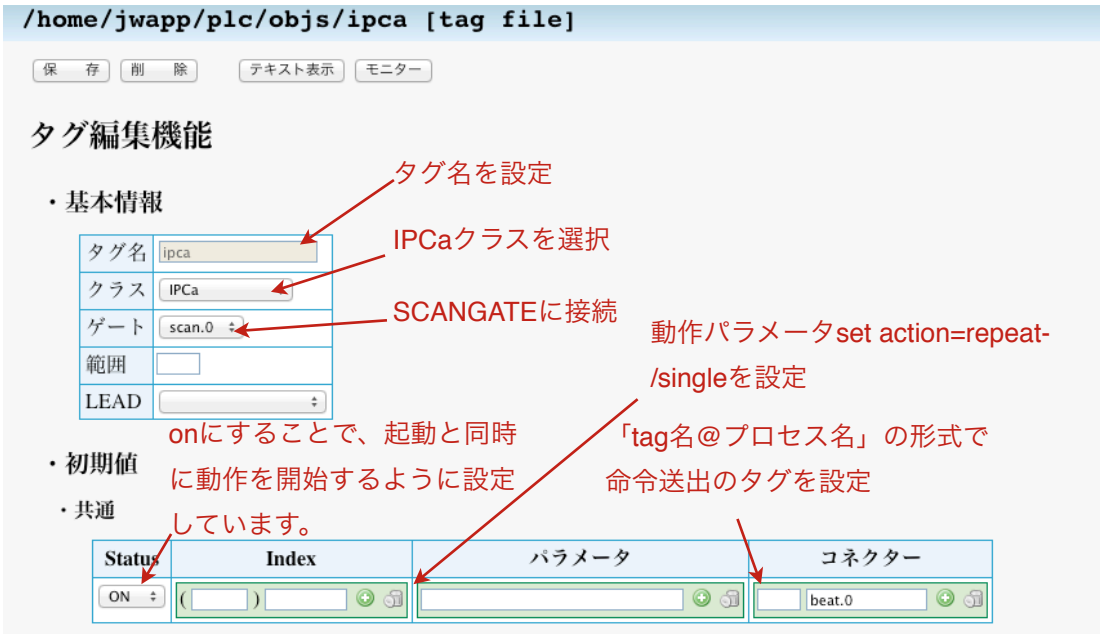
status　[Read]現在のステータスを出力します。
[Write]on命令受信後動作を開始します。off命令を受信すると動作を停止します。

action　set action=single/repeat(default)
“single”を指定することにより、enable時に一度だけステータスのコピーを実行します。

コネクタ　<%>タグ名 [@プロセス]
ステータスをコピーするオブジェクトのタグを設定します。@プロセスを指定すると外部のプロセスのタグを指定することができます。

説　　明　IPC=InterProcessCommunicationの意味です。
IPCaは外部のオブジェクトのステータスを自身のステータスにコピーします。action=singleを設定することにより、ラッチ動作を実現します。

定義方法



上記の例ではipcaのタグがシステム起動と同時にbeat.0のステータスを自身のステータスに反映する動作を行います。

IPCpull

動作概要　外部(プロセス)のタグの持つ値をコピーします。

パラメータ

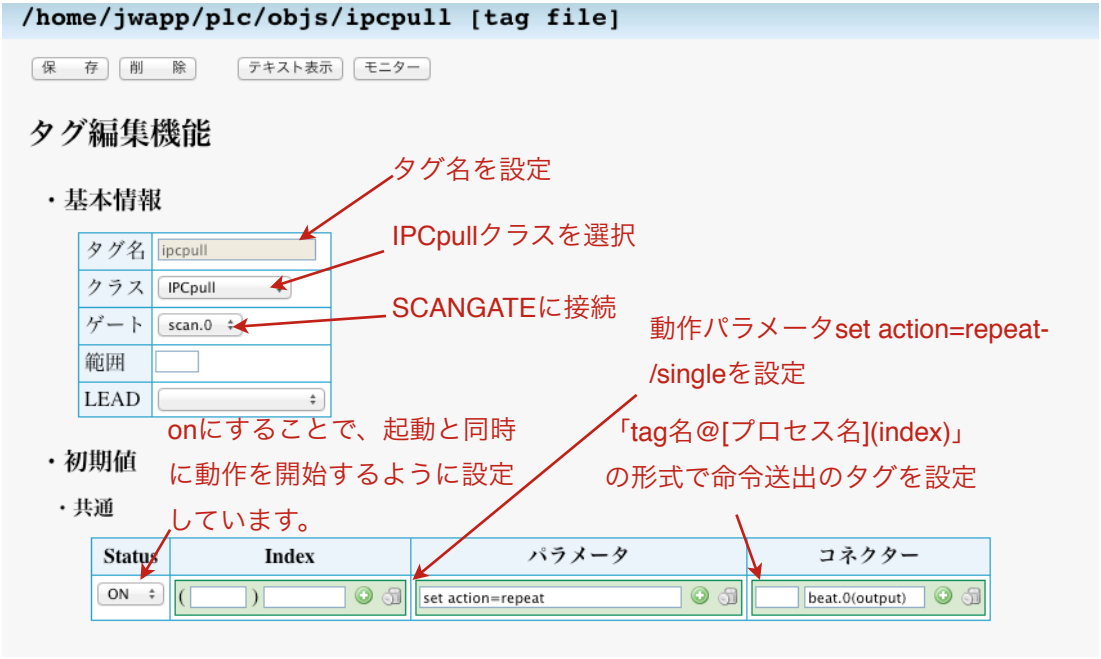
status　[Read]現在のステータスを出力します。
[Write]on命令受信後動作を開始します。off命令を受信すると動作を停止します。

action　set action=repeat/single(default)
“repeat”を指定することにより、on中継続的にデータのコピーを実行します。

コネクタ　<%>タグ名[@プロセス](index)
ステータスをコピーするオブジェクトのタグを設定します。@プロセスを指定すると外部のプロセスのタグを指定することができます。(index)にはinput,output,0,1,2,,,,のインデックス番号を指定します。(index)を省略した場合はoutputを対象とします。

説　　明　IPC=InterProcessCommunicationの意味です。
IPCpullは外部のオブジェクトのデータをコピーします。action=repeatを設定することにより、status=onの間継続的にデータのコピーを実行します。

定義方法



上記の例ではipcpullのタグがシステム起動と同時にbeat.0のoutputを自身のデータに反映する動作を行います。

IPCpush

動作概要　自身のデータを外部(プロセス)のタグにコピーします。

パラメータ

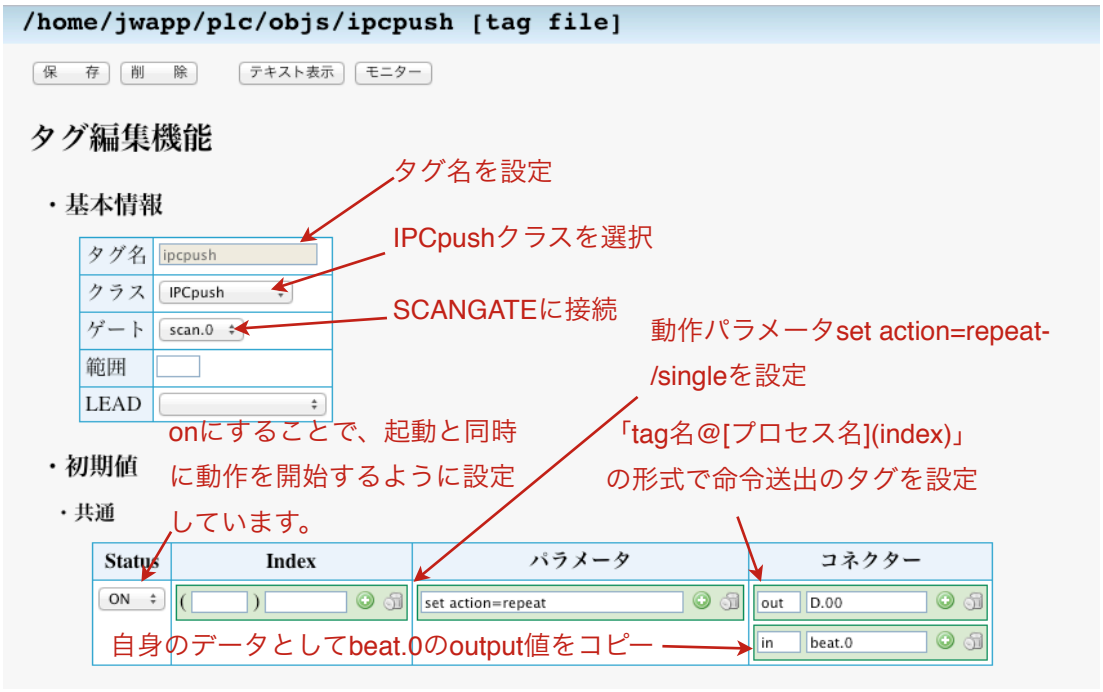
status　[Read]現在のステータスを出力します。
[Write]on命令受信後動作を開始します。off命令を受信すると動作を停止します。

action　set action=repeat/single(default)
“repeat”を指定することにより、on中継続的にデータのコピーを実行します。

コネクタ　<%out>タグ名[@プロセス](index)
ステータスをコピーするオブジェクトのタグを設定します。@プロセスを指定すると外部のプロセスのタグを指定することができます。(index)にはinput,output,0,1,2,,,,のインデックス番号を指定します。(index)を省略した場合はinputを対象とします。
<%in>タグ名(index)
自身のデータとして<%in>コネクタで指定したタグの値をコピーします。

説　　明　IPC=InterProcessCommunicationの意味です。
IPCpushは外部のオブジェクトに自身のデータをコピーします。
action=repeatを設定することにより、status=onの間継続的にデータのコピーを実行します。

定義方法



上記の例ではipcpullのタグがonになると、自身のinputとしてbeat.0のoutputコピーして、をD.00のinputに反映する動作を行います。

A4x0GPIO

動作概要　Armadillo-4x0のGPIOをDI,DOとして使う機能を提供します。

パラメータ

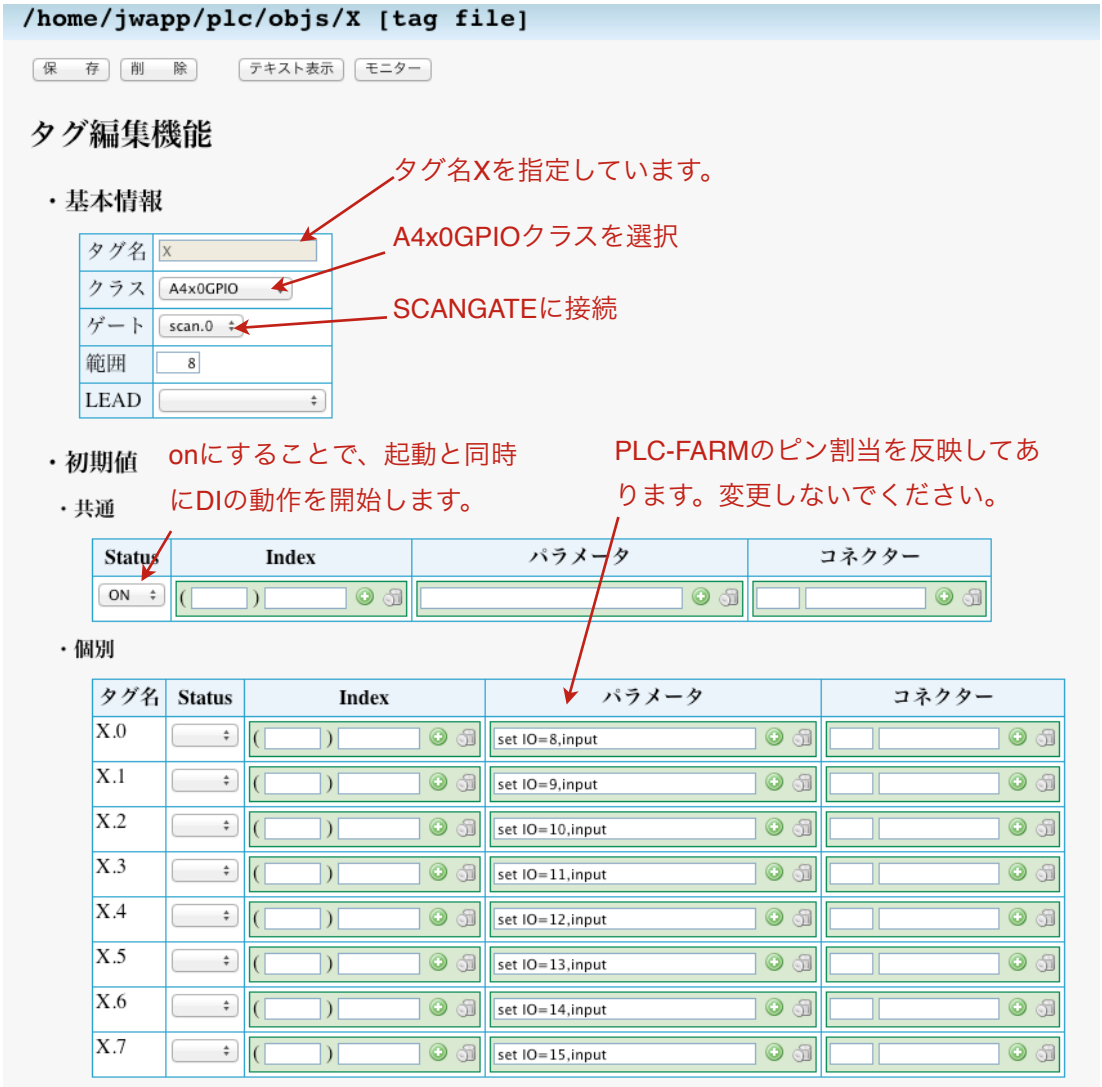
status　[Read]現在のステータスを出力します。
[Write]DO(output)として設定した場合、on/off命令を書込むことができます。

IO　set IO=[GPIOピン番号],input/output
上記フォーマットで、GPIOピン毎の機能(input,output)を設定します。

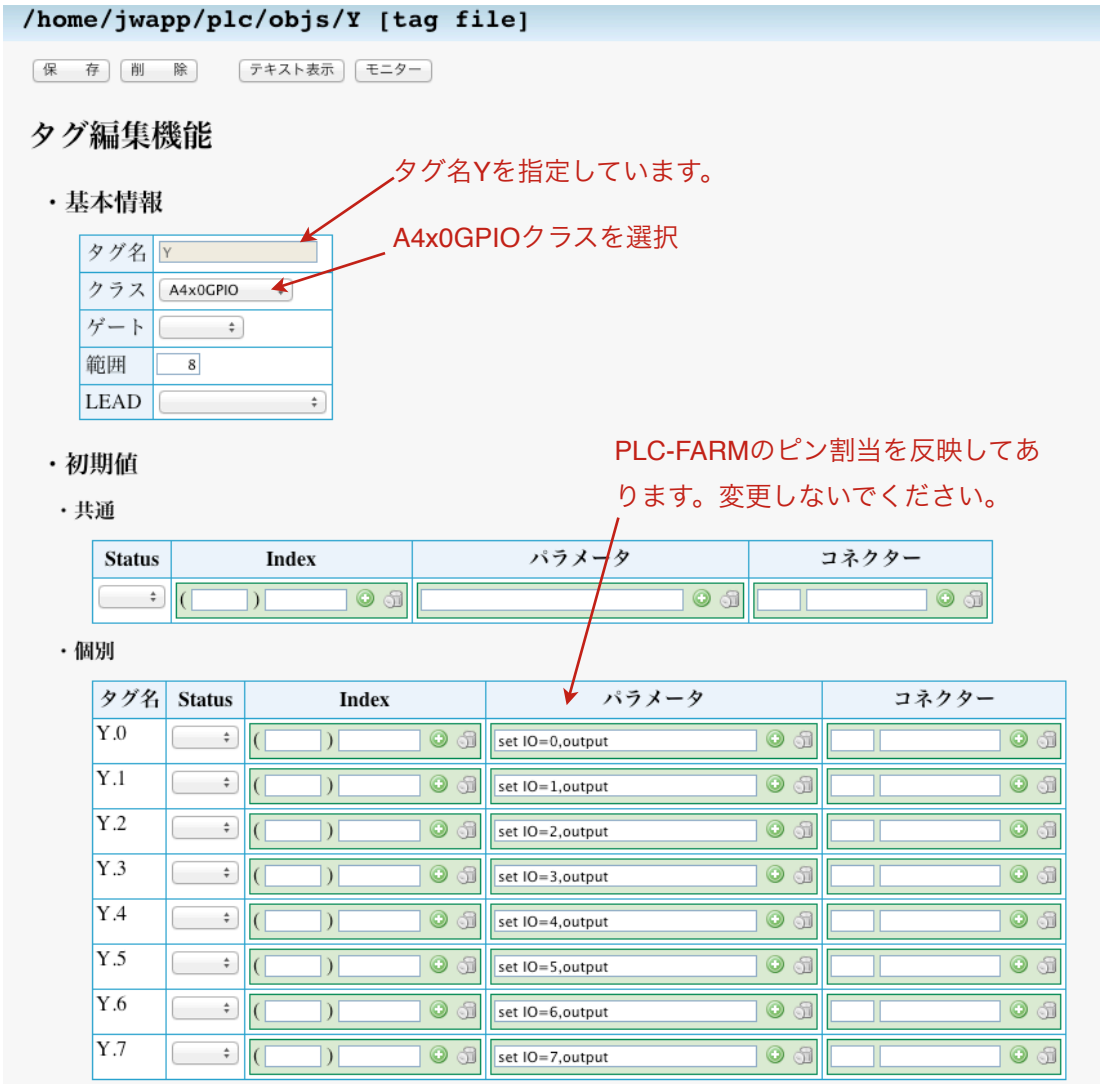
説　　明　set　IO=[GPIOピン番号],input/outputで指定するGPIOのピン番号は、Armadillo-4x0のソフトウェアマニュアルの「Armadillo-200シリーズ互換GPIOドライバー」の項に記載されている「GPIO名」の番号になります。

定義方法　プリセットされているXの定義です。PLC-FARMのGPIOの割付けを反映してあるので、GPIOピンの番号は変更しないでください。

入力　X



出力　Y


/home/jwapp/plc/objs/Y [tag file]
保　存
削　除
テキスト表示
モニター
タグ編集機能
・基本情報
タグ名
Y
クラス
A4x0GPIO
ゲート
範囲
8
LEAD
タグ名Yを指定しています。
A4x0GPIOクラスを選択
・初期値
・共通
Status
Index
パラメータ
コネクター
PLC-FARMのピン割当を反映してあります。変更しないでください。
・個別
タグ名
Status
Index
パラメータ
コネクター
Y.0
set IO=0,output
Y.1
set IO=1,output
Y.2
set IO=2,output
Y.3
set IO=3,output
Y.4
set IO=4,output
Y.5
set IO=5,output
Y.6
set IO=6,output
Y.7
set IO=7,output

A4x0LED

動作概要　Armadillo-4x0のLED(赤,緑)を使う機能を提供します。

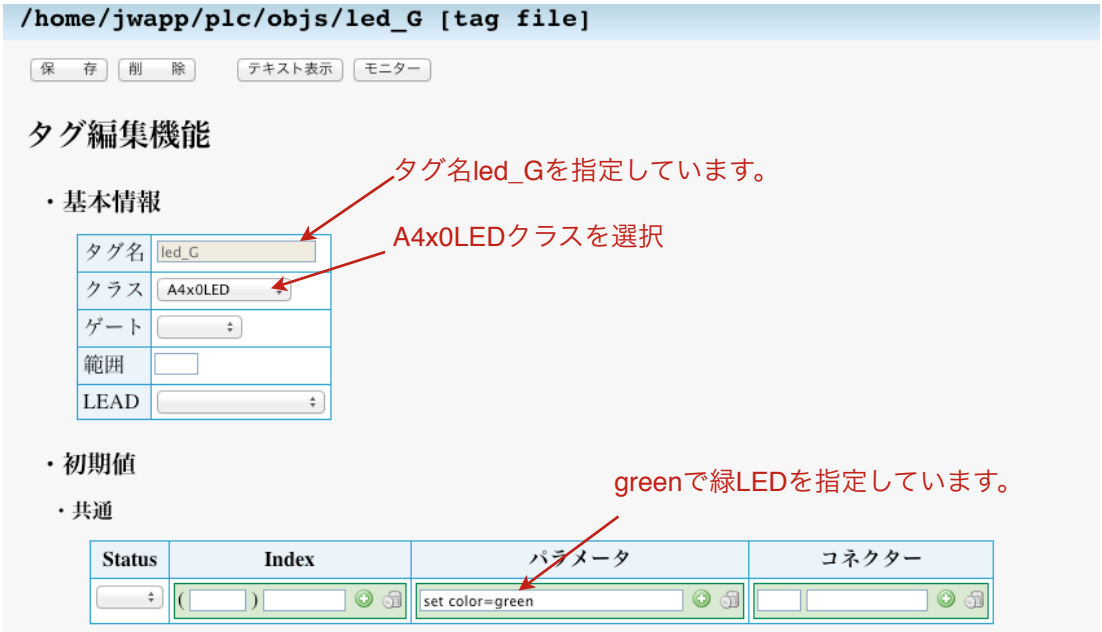
パラメータ

status　[Read]現在のステータスを出力します。
[Write]on/off命令により、LEDを点灯/消灯します。

color　set color=red/green

説　　明　PLC-FARMにはled_G(緑)，led_R(赤)の2つのオブジェクトが登録されていますので、そのまま使うことができます。

定義方法　プリセットされているled_Gの定義です。led_Rも同様に定義されています。



A4x0TACTSW

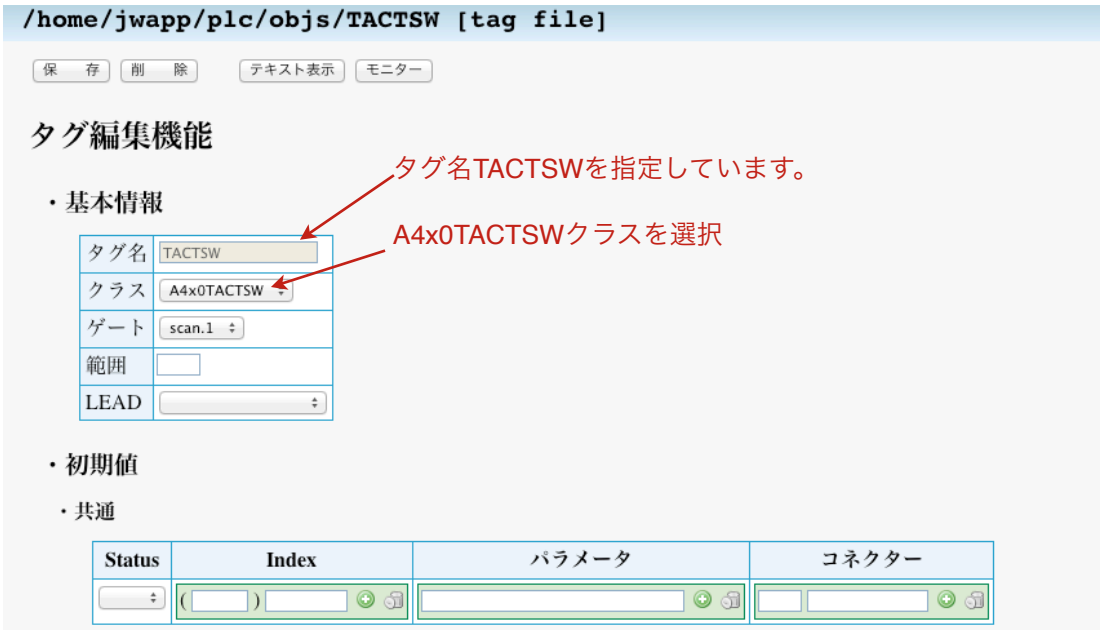
動作概要　Armadillo-4x0のタクトスイッチを使う機能を提供します。

パラメータ

status　[Read]現在のステータスを出力します。

説　　明　PLC-FARMにはTACTSWタグを用意していますので、そのまま使うことができます。

定義方法　プリセットされているTACTSWの定義です。



V. JUNKWare TOOLS

JUNKWareにはいくつかのLinux上で動作するコマンドラインツールが用意されています。JWTを使っている限り、これらのツールを意識的に使うことはありませんが、ソケットを使ってモニタする場合等、ソケット経由でこれらのコマンドラインツールを起動してデータ設定,現在値取得などの処理を行うことができます。基本的なコマンド以外ら、このようなソケット接続を受け付けるためのサーバ(jsv)も用意してあります。

ここでは、これらのJUNKWareのコマンドラインツールとサーバ用ツールについて説明します。

jsv

動作概要　ソケット接続により、JUNKWareのソフトウェアPLC内部にアクセスする機能を提供します。

起動パラメータ

jsv -p [port] -t [timeout] -4 -debug -help

-p　TCP接続のためのポート番号を設定します。指定されない場合は、/etc/junkportに記述されている数値を使用します。いずれの指定もない場合は、6666を使用します。

-t　接続タイムアウトの時間(秒)を設定します。

-4　IPv4のみで接続を受け付けます。指定のない場合、IPv4/IPv6のいずれでも接続を受け付けます。

-debug　デバッグログを出力します。通常は使用しません。

-help　ヘルプを表示します。

起動方法　jsv

デーモン起動するので、バックグラウンド起動の必要はありません。

使用(例)方法

jsh,jld,jdump,jppp,jfile,jtcpsvを起動できます。具体的なプロトコルについては株式会社YOODSまでお問合せください。

jlaunch

動作概要　JUNKWareのソフトウェアPLCからシェルスクリプト等の外部コマンドを起動する機能を提供します。

起動パラメータ

jlaunch -v

-v　デバッグログを出力します。通常は使用しません。

起動方法　jlaunch
デーモン起動するので、バックグラウンド起動の必要はありません。"**cat /dev/jsrr**"の出力で
　　0:server jlauncher
と出力される場合、jlaunchは既に起動しているので、二重起動しないようにしてください。

使用(例)方法

JUNKWareのソフトウェアPLCからEXECクラスを用いることにより利用できます。他の使い方はありません。逆に、jlaunchが起動してない場合、EXECクラスを使うことはできません。

jwait

動作概要　JUNKWareのソフトウェアPLCの起動を確認する機能を提供します。

起動パラメータ

jwait -o@[process_name] -t [timeout]

-o　@の後に確認したいプロセス名を指定します。PLC-FARMの場合、デフォルトのプロセス名は”plc”です。

-v　デバッグログを出力します。通常は使用しません。

起動方法　jwait -o@plc

シェルスクリプト中でこのように記載すると、JUNKWareプロセスの起動が完了するまで、スクリプトはこの場所でブロックされます。

使用(例)方法

plcプロセスの起動を確認する。

jwait -o@plc

この場合、jwaitはplcプロセスを確認できるまで待ち続ける。 起動確認できた場合は、"0"を返す。

タイムアウトを設定する場合

jwait -o@plc -t5

jwaitはplcプロセスの起動を5秒間監視する。起動確認できずに5秒経過すると エラーとして"1"を返す。

※JUNKWareプロセスの起動が完了する前に、jldやjshでアクセスすることはできないので、jwaitで確認をとってからアクセスするようにしてください。一度確認すれば、2回目のアクセス以降jwaitを用いる必要はありません。

jsh

動作概要　タグのステータス,データを取得します。

起動パラメータ

jsh -o@[process_name] -S

-o　@の後に確認したいプロセス名を指定します。PLC-FARMの場合、デフォルトのプロセス名は”plc”です。

-S　jshの起動,要求された処理の完了を確認したい場合に使用します。このオプション付きで起動された場合、jldは標準出力に起動直後に"OK"を出力します。通常は使用しません。

使用例

- **コイルscan.0の全内部情報を読出す**

  echo 'scan.0' | jsh -o@plc

  出力: (on) (input)0.00199795 (output)0.000198001 (0)0 (1)99 (2)0 (3)397 (4)0 (5)0 (6)0 (7)0 (8)0 (9)0 (10)100

- **コイルscan.0のステータス情報を読出す**

  echo 'scan.0(status)' | jsh -o@plc

  出力: (on)

- **コイルscan.0のinputの値を取得する**

  echo 'scan.0(in)' | jsh -o@plc

  出力: (input)0.00199055

- **コイルscan.0のoutputの値を取得する**

  echo 'scan.0(out)' | jsh -o@plc

  出力: (output)0.000236184

- **コイルscan.0のインデックス(1)の値を取得する**

  echo 'scan.0(1)' | jsh -o@plc

  出力: (1)99

jld

動作概要　タグのステータス,データを設定します。

起動パラメータ

jld -o@[process_name] -S

-o　@の後に確認したいプロセス名を指定します。PLC-FARMの場合、デフォルトのプロセス名は”plc”です。

-S　jldの起動,要求された処理の完了を確認したい場合に使用します。このオプション付きで起動された場合、jldは標準出力に起動直後に"OK"を出力します。通常は使用しません。

使用例

- **リレー R に(on)命令を送る**

  echo 'R(on)' | jld -o@plc

- **カウンタ CNT のしきい値を10に設定する**

  echo 'CNT(in)10' | jld -o@plc

- **タイマ T.00 の設定値を1.5秒に設定する**

  echo 'TMR(in)1.5' | jld -o@plc

jdump

動作概要　タグの持っているデータを読み出します。jshと違って、出力フォーマットにバリエーションがあり、一度に複数のタグデータをまとめて読出すことができます。

起動パラメータ

jdump -o[tag1]@[process_name],[tag2],... -v -f -d -S

-o[tag1]@[process_name],[tag2],...

@の後に確認したいプロセス名を指定します。PLC-FARMの場合、デフォルトのプロセス名は”plc”です。tag1には最初のモニタしたいタグ名を指定します。tag2以降は同一プロセス中の別のタグを複数並べて指定します。

-v　データの出力を一個毎に改行することにより縦に出力します。デフォルトはCSV形式となっています。

-f　各データにインデックスを付けて、XML形式ではない形で出力します。-fオプションを付けた場合、 強制的に-vオプションも付加されます。

-d　ステータス情報,input,outputの値も出力します。

-S　jdumpの起動,要求された処理の完了を確認したい場合に使用します。このオプション付きで起動された場合、jldは標準出力に起動直後に"OK"を出力します。通常は使用しません。

使用例

コイルscan.0の全内部情報を読出す-方法1

echo 'scan.0@plc' | jdump

出力:

<jdump>
<title>scan.0@plc</title>
<data>0,99,0,7565,0,0,0,0,0,0,100</data>
</jdump>

コイルscan.0の全内部情報を読出す-方法2

jdump -oscan.0@plc

出力:

```
<jdump>
<title>scan.0@plc</title>
<data>
0
99
0
1199
0
0
0
0
0
0
100
</data>
</jdump>
```

コイルscan.0の全内部情報を読出す-ステータス情報,input,outputの値を含む

jdump -oscan.0@plc -d

出力:

```
<jdump>
<title>scan.0@plc</title>
<data>(on)(input)0.00198916,
(output)0.00020515,0,99,0,475,0,0,0,0,0,0,100</data>
</jdump>
```

コイルscan.0の全内部情報を読出す-説明付き出力

```
echo 'scan.0@plc' | jdump -f
```

出力:

```
#data scan.0@plc
(on)
(input)0.0019979
(output)0.000205372
(0)0
(1)99
(2)0
(3)9140
(4)0
(5)0
(6)0
(7)0
(8)0
(9)0
(10)100
#end
```

標準入力からタグ(コイル)名を入力するインタラクティブな使用方法

jdump -S (-Sは無くても構わない)

出力:

```
OK     #jdumpの起動確認
scan.0@plc  #scan.0@plcを入力(最初はプロセス名付き)
<jdump>
<title>scan.0@plc</title>
<data>0,99,0,6197,0,0,0,0,0,0,100</data>
</jdump>
scan.1          #scan.1を入力(プロセス名省略)
<jdump>
<title>scan.1</title>
<data>1,98,0,1052,0,0,0,0,0,0,100</data>
</jdump>
scan.2          #scan.2を入力(プロセス名省略)
<jdump>
<title>scan.2</title>
<data>2,97,0,2103,0,0,0,0,0,0,100</data>
</jdump>
scan.0,scan.1        #scan,0scan.1をまとめて読込み
<jdump>
<title>scan.0,scan.1</title>
<data>0,99,0,2830,0,0,0,0,0,0,100</data>
<data>1,98,0,3199,0,0,0,0,0,0,100</data>
</jdump>
#最後はctrl-Dで終了
```

※JUNKWareには他にも様々なコマンドラインツールがありますが、内部的に使用されるコマンドも多いため、本マニュアルでは以上のツールの使用方法のみを公開します。

VI. 付録

1. NORフラッシュの容量について

Armadillo-420に搭載のフラッシュメモリは16MByteで、PLC-FARMにではベースになるLinuxシステム,JUNKWare等で93%の領域を使っています。そのため、ユーザーの使える領域は1.0MByteしかないことに注意してください。JUNKWareを使う限りでは、ユーザープログラムはほとんどテキストリソースになるため、1MByteの領域で足りる筈です。

2. マイクロSDカードの使い方

Armadillo-4x0にはマイクロSDカードが搭載されていて、ここを外部記憶領域や作業領域として使うことができます。常時使用する場合は、/etc/init.d/rc.localで

```
mount /dev/mmcblk0p1 /mnt
```

として起動時にマウントするようにしてください。

```
ln -s /mnt /home/jwapp/mmc
```

上記のように/home/jwappにリンクを作っておけば、JWTからも操作が可能になります。

3. hermitのモード

PLC-FARMに搭載したArmadillo-420ではブートローダーとしてhermitを使っています。このu-bootはJP2を使ってブートモードを切り替えることができます。

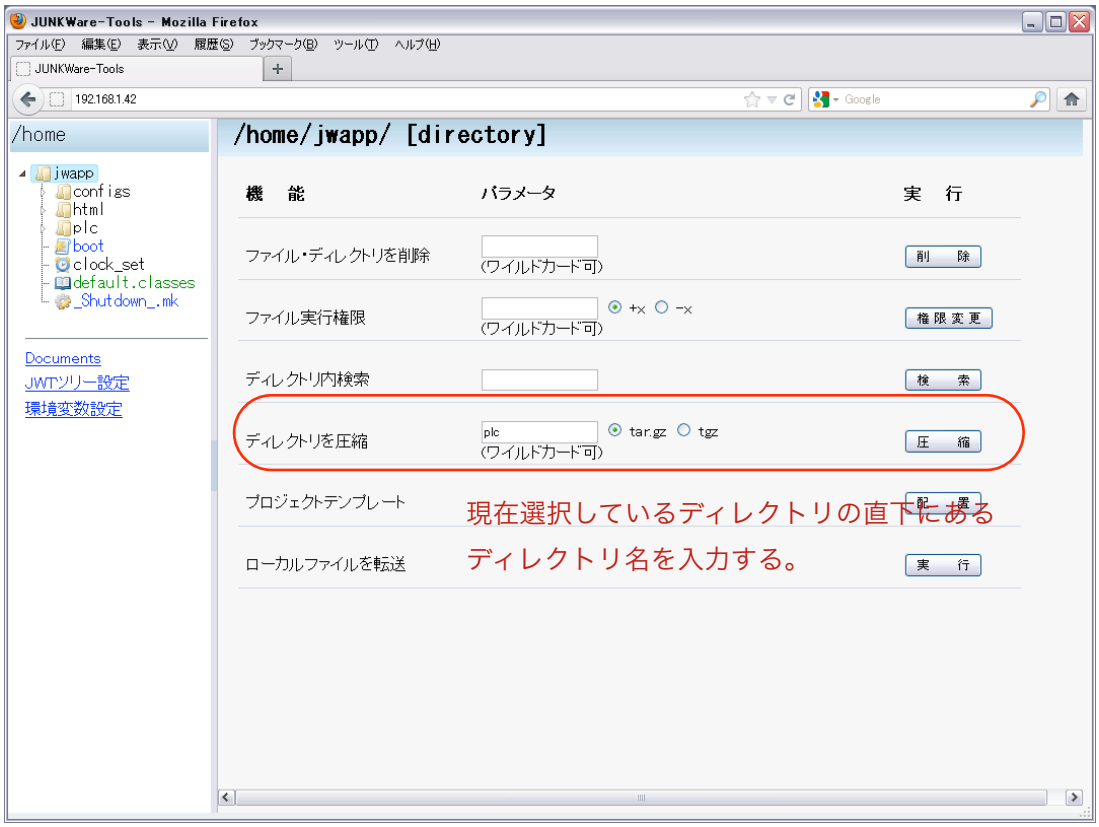
JP2	u-bootの動作
開	通常モードです。オンボードのNORフラッシュメモリから起動します。
閉	microSDから起動します。microSDには Armadillo-420を起動できるLinuxを入れておく必要があります。 また、電源投入時にタクトスイッチをONしておくことで、hermitのコマンドモードで起動することができます。

※PLC-FARM起動用microSDについては、お問合せください。

4. バックアップの取り方

PLC-FARMで開発したシステムをバックアップしたり、複数のPLC-FARMに横展開する場合、 **ディレクトリ(フォルダ)**メニューにあるディレクトリの圧縮機能と転送機能を使うと便利です。plcフォルダのバックアップをとる場合を例に示します。

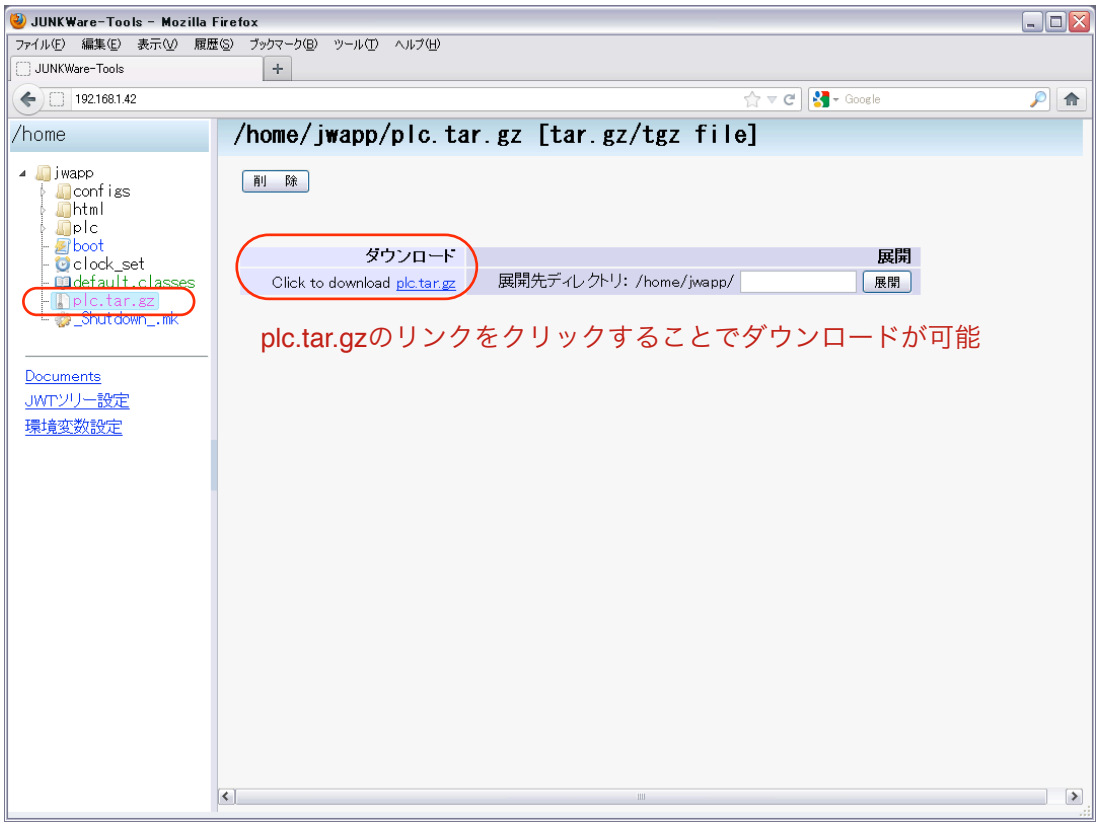
(1) バックアップの作成



上記のように、/home/jwappを選択するとその配下のディレクトリを圧縮することができます。/home/jwapp/plcをバックアップ用に圧縮することにします。その場合、「ディレクトリを圧縮」のテキストエリアに「plc」と入力して、「圧縮」ボタンを押下してください。

(2) バックアップファイルの取得

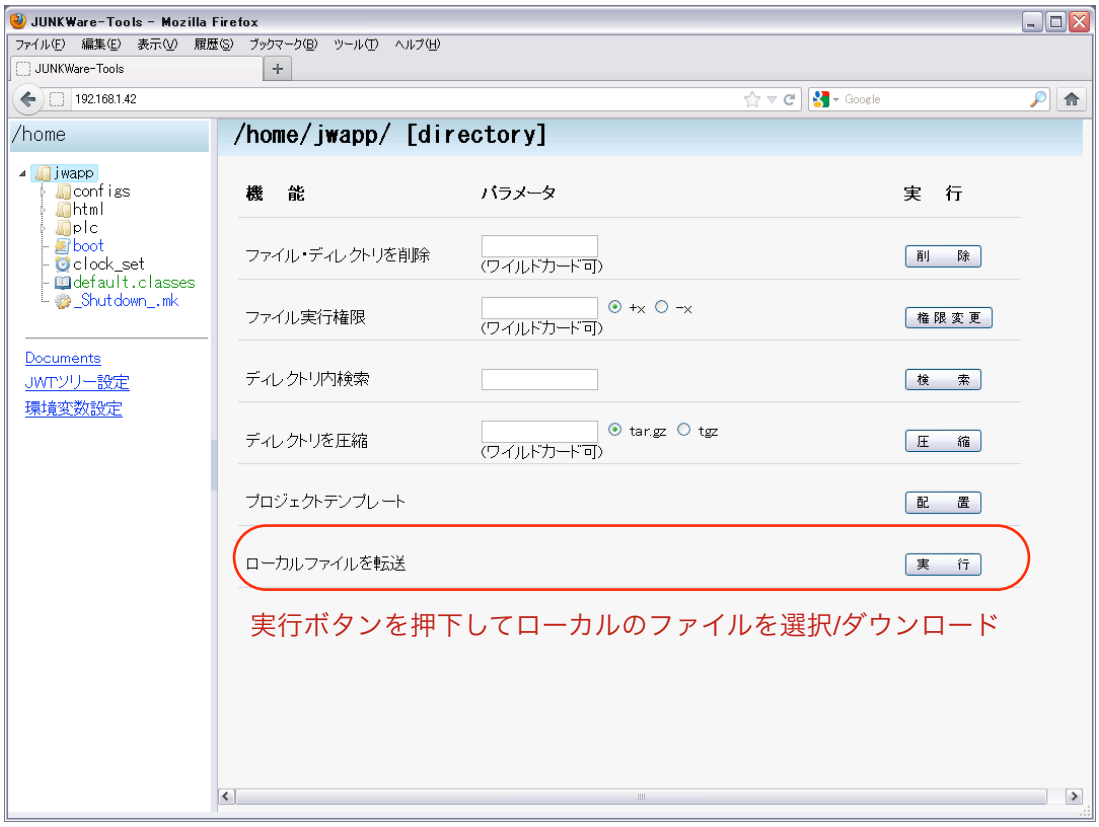
(1)の操作でplc.tar.gzが圧縮元のplcディレクトリと同じ位置に作成されます。これを選択すると下図のようなペインが表示されます。

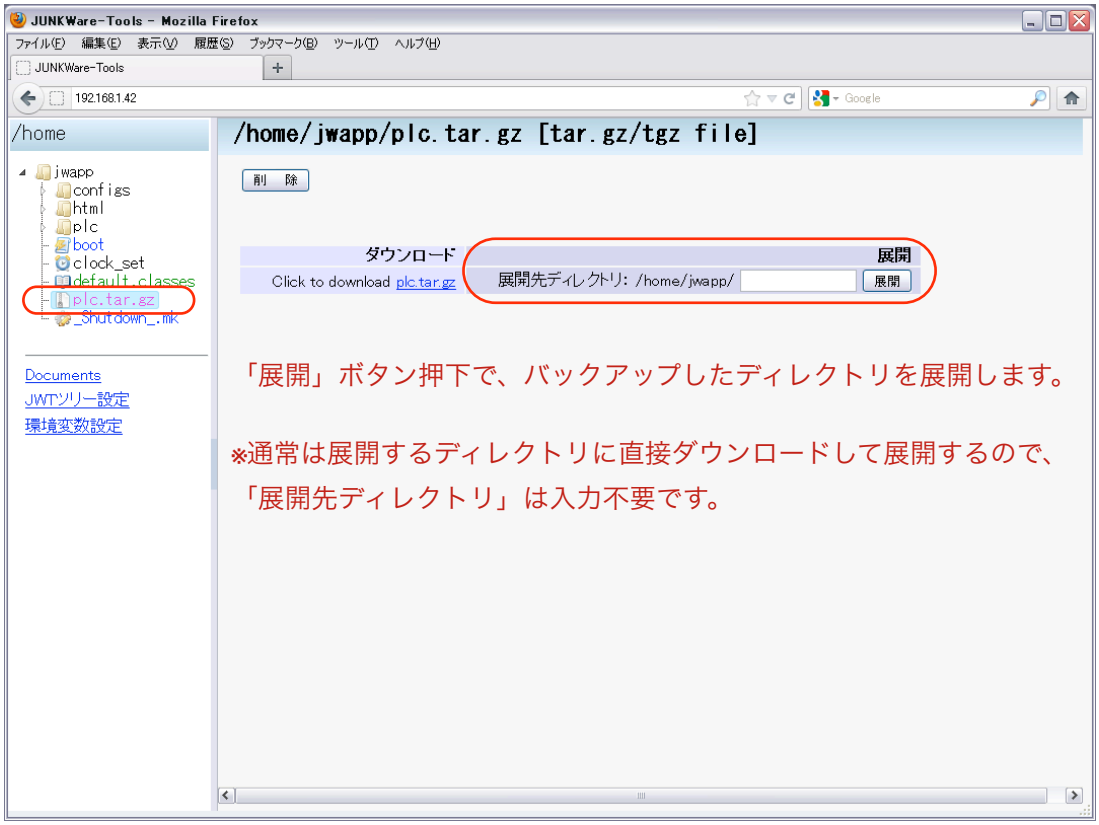


このplc.tar.gzのリンクをクリックすることにより、圧縮したバックアップファイルをローカルのパソコンに取得することができます。

(3) バックアップファイルのダウンロード と バックアップファイルの復元

ローカルのパソコンにあるバックアップをPLC-FARMにダウンロードして展開する場合は以下のようにしてください。





(4) configsディレクトリを復元する場合の注意

configsディレクトリには不可視のライセンスファイル(.jw.lic)が保存されています。configsディレクリのバックアップ(configs.tar.gz)を作成して、他のPLC-FARMで展開すると、ライセンスファイルを上書きしてしまうためJUNKWareが起動できなくなります。従って、configsディレクリについては、

・テキストのカット&ペーストで編集する

・バックアップから復元後、ライセンスファイルを再インストール

のいずれかで対応してください。

5. 時刻の設定方法

PLC-FARMにはRTCが搭載されていて、バッテリーバックアップされています。カレンダタイマの時刻合わせをする時には、JWTの左側ツリーのconfig/にあるclock_setをクリックしてください。右側ペインに上図のような時刻設定画面が表示されるので、”更新”をクリックすると、PLC-FARMに時刻がセットされます。なお、clock_setをクリックした時に表示される時刻はローカルパソコンの時刻になります。

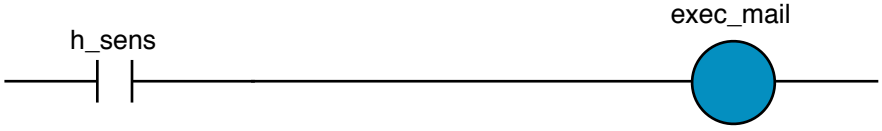
6. PLC-FARMからのメール送信について

sendmail等のMTAを入れることは容量的にできないので、「Heirloom mailx」コマンドを/usr/bin/mailへのシンボリックリンクとしてインストールしてあります。/home/jwapp/configs/mailrcを適切に編集することにより、メールを送信することが可能になります。この仕組みを使うことにより、 例えば、以下のようなメール発報の仕組みを作ることが可能です。

メール送信スクリプト作成

```
#!/bin/sh

SUBJECT="`hostname`_`date`" #ホスト名_日付をサブジェクトに
#ディレクトリの使用状況をhoge@hogeさんにメールする
du -k | /usr/bin/mail -s "$SUBJECT" hoge@hoge
END
sleep 10  #チャタリングによる連続発報を防止!!
```



※exec_mailはEXECクラスで、メール送信スクリプトを呼び出す
※h_sensは人感センサ

7. シリアルポートについて

PLC-FARMにはArmadillo-420のシリアルポートがユーザーに解放されています。PLC-FARMの工場出荷時にはこのポートはシリアルコンソールとして設定されているので、ユーザーアプリケーションでシリアルポートを使う場合は、その設定を変更する必要があります。以下にその方法を示します。

(1) /home/jwapp/configs/にあるinittabを開いてください。

(2) ttymxc1の設定してある行をコメントアウトします。

```
::sysinit:/etc/init.d/rc

#::respawn:/sbin/getty -L 115200 ttymxc1 vt102

::shutdown:/etc/init.d/reboot
::ctrlaltdel:/sbin/reboot
```

(3) 保存して再起動してください。

※起動時のkernelログはhermitでシリアルポートへの出力が設定してあるため、変更できません。

8. オンボードNORフラッシュメモリの復旧方法

PLC-FARMに搭載されているCPUボード(Armadillo)にはNORフラッシュメモリが搭載されていて、その中に

(1) hermitブートローダー

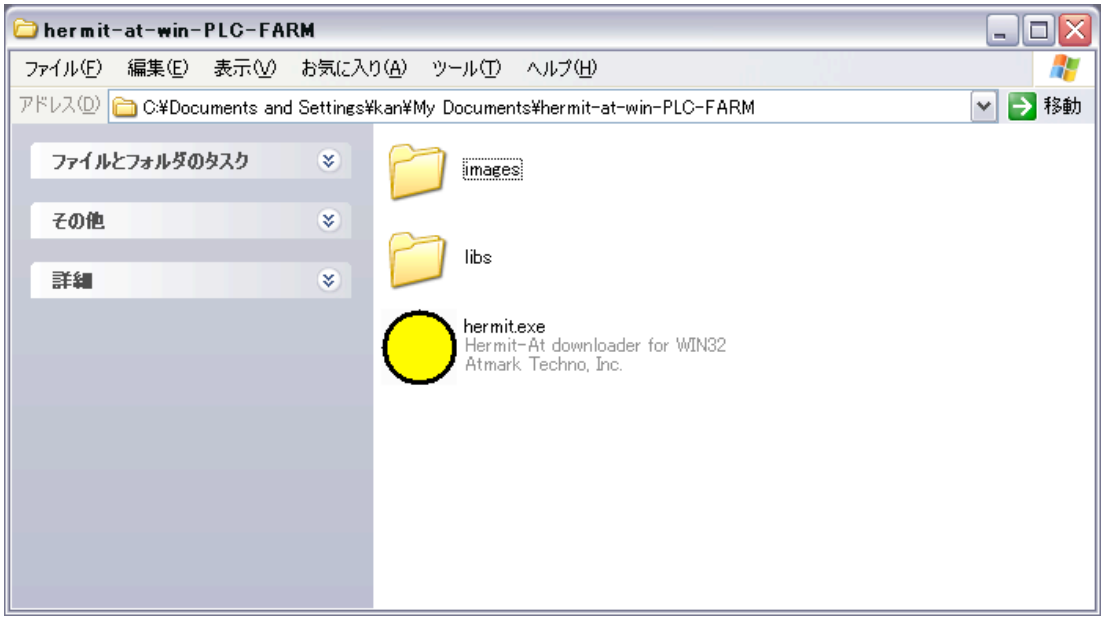
(2) Linuxカーネル

(3) Linuxユーザーランド

が格納されています。何等かのトラブルでこのフラッシュメモリの領域を壊してしまった場合や、出荷時の状態に初期化したい場合は付属CD-ROMに含まれるファイル　hermit-at-win-PLC-FARM.zip　を使って、以下の手順でフラッシュメモリを出荷時状態に戻してください。

(1) hermit-at-winと復元用イメージファイルのインストール

PLC-FARM付属のCD-ROM内にあるファイル　hermit-at-win-PLC-FARM.zip をWindowsパソコン内の任意の場所に解凍してください。

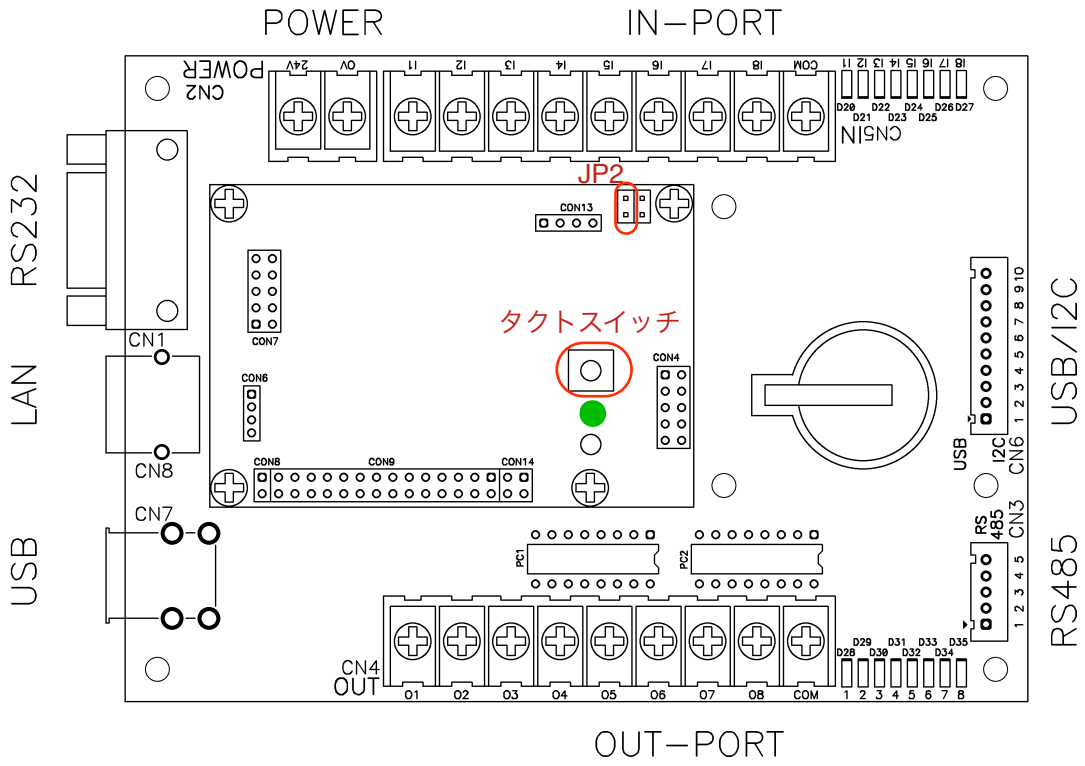


imagesフォルダ内に以下のファイルが含まれます。

loader-armadillo4x0-v2.1.1.bin	hermitブートローダー
zImage.bin	Linuxカーネル
rootfs-plc-farm-vX.XX.bin	Linuxユーザーランド

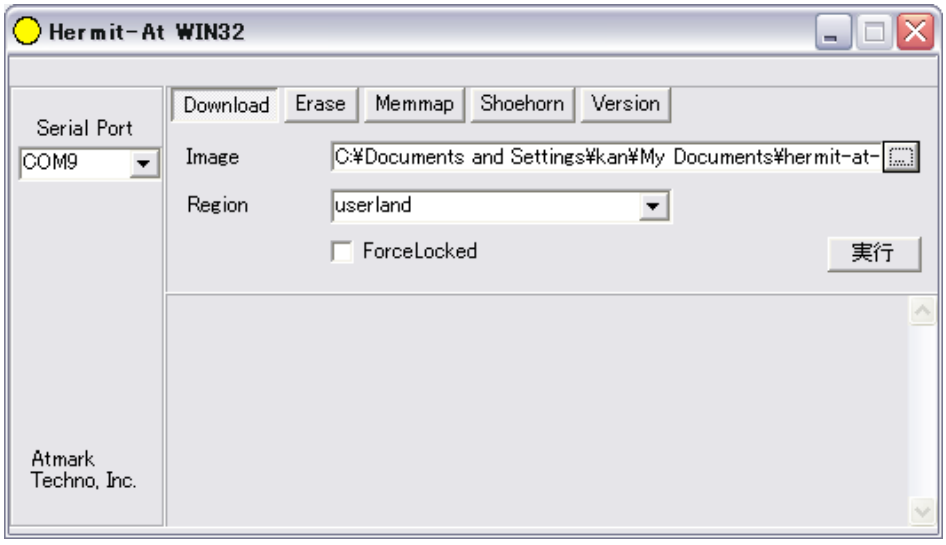
(2) PLC-FARMの書込み準備

PLC-FARMをhermitの書込みモードで起動します。ArmadilloのJP2をジャンパでショートした上で、タクトスイッチを押しながら電源を投入してください。緑色のランプが点灯してフラッシュメモリ書込モードになります。



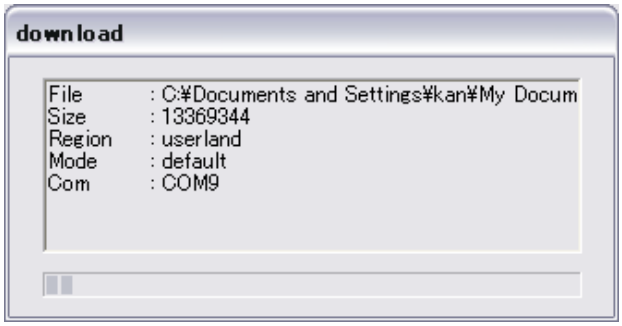
(3) ユーザーランドイメージの書込み

hermit-at-win.exeを実行します。以下のHermit-At：Downloadウィンドウが表示されます。



Armadilloと接続されているシリアルインターフェースを「Serial　Port」に指定してください。ドロップダウンリストに表示されない場合は、直接ポートを入力してください。

Imageには書き込むファイルを指定してください。Regionには書き込み対



象のリージョンを指定してください。allやbootloaderリージョンを指定する場合は、Force Lockedをチェックしてください。

すべて設定してから実行ボタンをクリックします。Hermit-At：downloadダイアログが表示されます。

書込みが終了したら、JP2のジャンパを外して再起動してください。

※通常、出荷時の状態に初期化するケースでhermit,　カーネルを書込む必要はありません。Linuxユーザーランドのみを書込んでください。

更新されたユーザーランドにはPLC-FARMの筐体毎に必要なライセンスファイルがインストールされていません。次項の「ライセンスファイルのインストール方法」に従ってライセンスをインストールしてください。

(4) ライセンスファイルのインストール

JUNKWareにはイーサーネットのMACアドレスによる起動時のライセンス認証が必要です。添付のCD-ROMにはライセンス認証用のファイルが入っています。”00110C-111A42.tar.gz”のように、ファイル名にMACアドレスが入ったtar.gzファイルです。ライセンスファイルを消してしまった場合や、Linuxユーザーランドを再インストールした場合は、このファイルを以下の手順でPLC-FARMに転送してインストールしてください。

(a)JWTで/home/jwappを選択

(b)右側の最下部にある「ローカルファイルを転送」の実行ボタンを押下し、ファイルダイアログからローカルのライセンスファイル(00110C-******.tar.gz)を選択

(c)/home/jwapp/にライセンスファイルが転送されているので、それを選択して右側の展開ボタンを押下。(ディレクトリ指定は不要です)

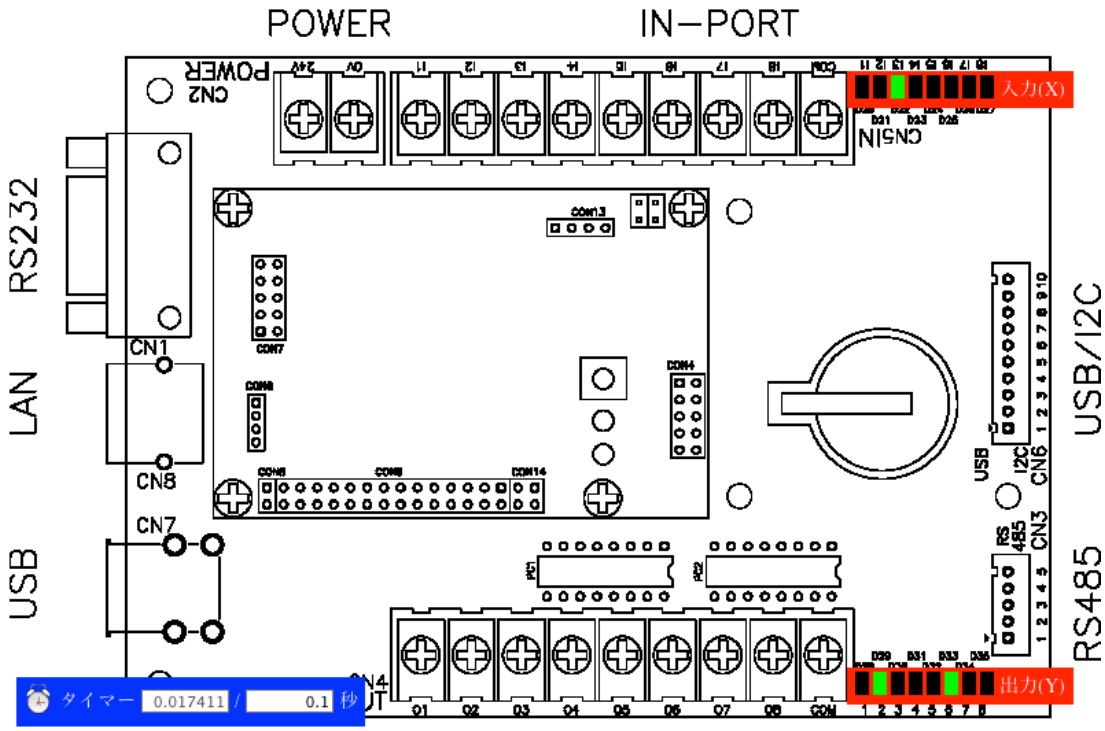
(d)configsに.jw.lic(不可視ファイル)展開されます。/home/jwapp/に転送したライセンスファイル(00110C-******.tar.gz)を右クリックして削除。

(e)PLC-FARMを再起動

9. WEBブラウザによるPLC-FARMのモニタ/設定画面の作り方(サンプル)

JUNKWareを使ってPLC-FARMのモニタをWEBブラウザ上に実現する方法をチュートリアル形式で示します。

・完成するモニタ画面のイメージ



完成したモニタ画面のイメージは上図のようになります。

- ボードのイメージをそのままビットマップとして使います。
- 入力/出力のLEDの部分にランプを配置し、入出力の状態を1秒おきに更新します。
- 内部で動いているbeatオブジェクトのモニタ値を表示します。
- beatオブジェクトの発振周期を画面上から変更できるようにします。

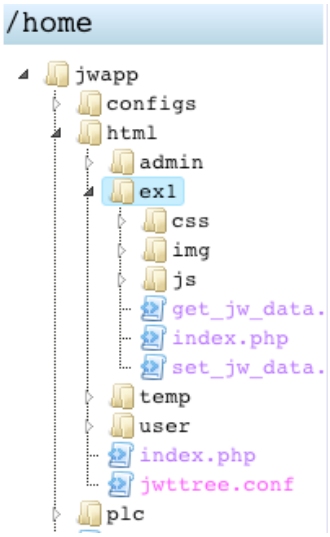
このサンプルのコードは一式/home/jwapp/html/ex1にあります。実際に

http://192.168.1.42/ex1/index.php

にアクセスすることで、動かすことができます。

・モニタ/設定画面作成のチュートリアル

手順1　　基本ファイルの配置



まず、ブラウザでアクセスするURLを決めて、ディレクトリ(フォルダ)を配置します。今回は”http://192.168.1.42/ex1/index.php”でこの機能を実現するように、上図のようなディレクトリ配置を作成しました。

※/home/jwapp/html/は/var/www/htmlへのシンボリックリンクになっています。

cssにはユーザー定義のスタイルシート、imgには使用するイメージファイル,jsにはJavaScriptを配置します。今回jQueryというJavaScriptを用いるため、jquery-1.7.1.min.jsというファイルをダウンロードしてきて配置しました。index.phpのヘッダ部では配置したファイルをインクルードするように記載します。

/ex1/index.php

```
<?php
        echo '<?xml version="1.0" encoding="utf-8"?>';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja" lang="ja">
<head>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   <meta http-equiv="content-script-type" content="text/javascript" />
   <meta http-equiv="content-style-type" content="text/css" />
   <title>サンプル</title>

   <noscript>
      <meta http-equiv="Refresh" content="0;URL=admin/noscript.html">
   </noscript>

   <script type="text/javascript" src="js/jquery-1.7.1.min.js"></script>
   <link rel="stylesheet" type="text/css" href="css/sample.css" media="screen" />

   <script type="text/javascript">
```

手順1　　ビットマップ画像の配置

まず、画面の中央に先ほど作成した画像を「div」タグの背景画像として指定します。 HTMLのコードは下記の通りです。注意するのは後から画像上にHTMLコンポーネントを配置する為、スタイルシートの「position」を「relative」としている事です。

/ex1/index.php

```
                                    ～～～　省略　～～～
    <body>
        <div id="container" style="position:relative;background-image:url(img/plc-farm.png);width:800px;height:565px;">
        </div>
    </body>
</html>
```

手順2　　画像上にHTMLコンポーネントを配置

(1) 出力LED表示

JUNKWareコントローラの出力用タグに対応しています。左から「Y.0」～「Y.7」の順に並んでいます。この度は、タグのステータスがONの場合は「緑」、OFFの場合は「黒」になるようにします。

(2) 入力LED表示

JUNKWareコントローラの入力用タグに対応しています。左から「X.0」～「X.7」の順に並んでいます。表示に関しては「(1)出力LED表示」と同様です。

(3) タイマー(beat)表示

JUNKWareコントローラのタイマータグに対応しています。左のテキストボックスが「現在値」で「TMR」タグの「output」値を参照し、右のテキストボックスが「最大値」で「TMR」タグの「input」値を参照しています。

/ex1/index.php

```
                                        ～～～　省略　～～～
<body>
        <div id="container" style="position:relative;background-image:url(img/plc-farm.png);width:800px;height:565px;">

                <!-- LEDパネル（下） -->
                <div class="led_pnl" id="led_y" style="position:absolute;top:480px;left:580px;">
                        <div class="led_lamp"> </div><!-- Y.0 -->
                        <div class="led_lamp"> </div><!-- Y.1 -->
                        <div class="led_lamp"> </div><!-- Y.2 -->
                        <div class="led_lamp"> </div><!-- Y.3 -->
                        <div class="led_lamp"> </div><!-- Y.4 -->
                        <div class="led_lamp"> </div><!-- Y.5 -->
                        <div class="led_lamp"> </div><!-- Y.6 -->
                        <div class="led_lamp"> </div><!-- Y.7 -->
                        <div class="title"> 出力(Y)</div>
                        <div class="led_lamp_end"></div>
                </div>

                <!-- LEDパネル（上） -->
                <div class="led_pnl" id="led_x" style="position:absolute;top:110px;left:580px;">
                        <div class="led_lamp"> </div><!-- X.0 -->
                        <div class="led_lamp"> </div><!-- X.1 -->
                        <div class="led_lamp"> </div><!-- X.2 -->
                        <div class="led_lamp"> </div><!-- X.3 -->
                        <div class="led_lamp"> </div><!-- X.4 -->
                        <div class="led_lamp"> </div><!-- X.5 -->
                        <div class="led_lamp"> </div><!-- X.6 -->
                        <div class="led_lamp"> </div><!-- X.7 -->
                        <div class="title"> 入力(X)</div>
                        <div class="led_lamp_end"></div>
                </div>

                <!-- タイマー -->
                <div class="timer_pnl" style="position:absolute;top:525px;left:30px;">
                        <table>
                                <tr>
                                        <td><img src="img/alarm.png" alt="alarm"></td>
                                        <td>タイマー</td>
                                        <td><input type="text" class="timer_txt" id="tmr_cur" value="" disabled ></td>
                                        <td>/</td>
                                        <td><input type="text" class="timer_txt" id="tmr_set" value=""></td>
                                        <td>秒</td>
                                </tr>
                        </table>
                </div>
        </div>
</body>
</html>
```

注意する必要がある箇所は、各項目は画像上に配置する必要があるので各項目の「div」タグの「position」を「absolute」と指定し、画像上の配置（「top（上から）」「left（左から）」何ピクセルか）を指定する必要があります。

以上で画面のレイアウトは終了です。

手順3　　データ要求POST用コードの記述

「jQuery」ライブラリを使用してソフトウェアPLCのデータの取得を行う記述をします。まずは、外部のPHPにモニタリングに必要な情報をPOST送信するHTMLのコードです。

/ex1/index.php

```
                                        ～～～　省略　～～～
<script type="text/javascript">
    $(function()  {
        updateValues();
    }
    function updateValues(){                          データ取得要求部
        var ledTags1="Y.0,Y.1,Y.2,Y.3,Y.4,Y.5,Y.6,Y.7";
        var ledTags2="X.0,X.1,X.2,X.3,X.4,X.5,X.6,X.7";
        var timerTags="beat.0";
        var params ={ "proc":"plc" ,"tags":ledTags1 + "," + ledTags2 + "," + timerTags};
        $.post("get_jw_data.php",params,function(data){
            /** POST送信の結果データを受信した際に呼ばれます。 **/

        },"json");
    }
```

HTMLコードがブラウザに読み込まれてすぐに「updateValues」関数が呼ばれます。これが　ソフトウェアPLCプロセス(@plc)から現在の値を取得し、各項目の値を更新するための関数になります。　ソフトウェアPLCプロセスからデータを取得するにはjQueryライブラリのpost関数（非同期にPOST送信を行う関数）を利用します。post関数の引数は下記の通りです。

```
$.post("<呼び出し用PHPのURL>",<パラメータ>,function(<受信データ>){
        <受信後の処理>
},"<受信データフォーマット>");
```

ソフトウェアPLCプロセス(@plc)からデータを取得するPHPのスクリプト名を「get_jw_data.php」とし、データ受信に必要なパラメータを下記のように定義します。

・「proc」：文字列。JUNKWareコントローラのプロセス名を指定する。

・「tags」：文字列。受信対象のタグをカンマ区切りで列挙する。

受信データフォーマットはjavascriptで取り扱い易い「"json"」形式を選択しています。

手順4　　データ取得実行部の記述

ソフトウェアPLC内のデータは、JUNKWare　Toolsのjshやjdumpを用いて行います。このサンプルではPHPを使ってjdumpを呼出して、その結果をPOST要求の結果として返します。ここでは、そのPHPのコードを記述します。

/ex1/get_jw_data.php

```
<?php
    $proc = $_REQUEST["proc"];
    $tags = $_REQUEST["tags"];                          パラメータ取得

    $tokens = explode(",",$tags);
    $jdumpCmd="";                                       データ取得コマンド作成
    $tags=array();
    foreach($tokens as $token){
        if(strlen($token)>0){
            array_push($tags,$token);
            $jdumpCmd.="$token@$proc ";
        }
    }
    $result=false;                                      コマンド実行
    $jdumpData=`echo "$jdumpCmd" | jdump -v -d`;
    $sidx=strpos($jdumpData,"<data>");
    $jwData=array();                                    データ解析
    if($sidx){
        $tokens = explode("</data>",substr($jdumpData,$sidx));
        for($i=0;$i<count($tokens);$i++){
            $valStr=trim($tokens[$i]);
            if($valStr==="</jdump>"){
                break;
            }
            $values=explode("\n",$valStr);
            $tagData=array();
            $idxValues=array();
            foreach($values as $value){
                if(strlen($value)==0 || $value[0]=='<'){
                    continue;
                }else if($value[0]=='('){
                    if(strpos($value,'on',1)==1){
                        $tagData["stat"]=1;
                    }else if(strpos($value,'off',1)==1){
                        $tagData["stat"]=0;
                    }else if(strpos($value,"input",1)==1){
                        $tagData["input"]=trim(substr($value,strlen("(input)")));
                    }else if(strpos($value,"output",1)==1){
                        $tagData["output"]=trim(substr($value,strlen("(output)")));
                    }
                }else{
                    array_push($idxValues,trim($value));
                }
            }
            if(count($idxValues)){
                $tagData['idxVal']=$idxValues;
            }
            $jwData[$tags[$i]]=$tagData;
        }
    }
    if(count($jwData)>0){
        $result=true;
    }
    $ret=array('result'=>$result,"jwData"=>$jwData);    実行結果出力
    echo json_encode($ret);
?>
```

(1)　パラメータ取得

POST送信されたパラメータを取得しています。今回は「proc(プロセス名)」と「tags(取得対象タグ)」です。

(2)　データ取得コマンド作成

jdumpを使用するデータ取得のコマンドを作成しています。jdumpの使い方についてはJUNKWare　Toolsのリファレンスマニュアルを参照してください。

(3)　コマンドの実行

作成したコマンドをLinuxのコマンドラインから実行しています。

(4)　データ解析

jdumpで出力されるデータは下記のような構成になっています。

```
<jdump>
<title><タグ１><．．．以降タグの数だけタグ名が続く></title>
<data>
<タグ１ステータス(on)または(off)>
(input)<タグ1　input値>
(output)< タグ1　output値>
<タグ1のindex<No>の値(．．．以降index値が存在するまで続く)>
</data>
*．．．以降、タグの数だけ<data>タグが続く。
</jdump>
```

この文字列をJava Scriptで扱えるように下記の構造へ置き換えています。

```
$jwData=array(
        <タグ名１>　=> array(
                stat => <0(off)/1(on)>,
                input =><input値>,
                output =><output 値>,
                idxVal => array(
                        <index値>,<．．．以降indx値の数だけ続く>
                )
        ),
        <．．．以降タグの数だけ続く。>
)
```

(5)　結果データ出力

作成したデータをPHPのjson_encodeメソッドを使用し出力しています。

手順5　**受信データを画面に反映**

POST要求に対して、ソフトウェアPLCの内部データを受信します。そのデータの内容を画面反映すれば、モニタ機能のが完成します。

/ex1/index.php

```
                                        ～～～　省略　～～～
var params ={ "proc":"plc" ,"tags":ledTags1 + "," + ledTags2 + "," + timerTags};
$.post("get_jw_data.php",params,function(data){
//alert(data);
//return;
        if(data.result){                                    データ更新部
                var ledY=$("#led_y .led_lamp");
                for(var i=0;i<8;i++){
                        updateLedLamp(ledY[i],data.jwData["Y." + i].stat);
                }
                var ledX=$("#led_x .led_lamp");
                for(var i=0;i<8;i++){
                        updateLedLamp(ledX[i],data.jwData["X." + i].stat);
                }

                $("#tmr_cur").val(data.jwData["beat.0"].output);
                if(first_){
                        $("#tmr_set").val(data.jwData["beat.0"].input);
                }

        }else{
                alert("データの取得に失敗しました。");

        }
        first_=false;
},"json");
                                        ～～～　省略　～～～

function updateLedLamp(obj,stat){      ランプ更新関数
        if(stat==0){
                $(obj).css('background-color','black');
        }else{
                $(obj).css('background-color','lime');
        }
}
```

POSTデータの解析部にデータ解析の内容を記述します。ランプの更新については数が多いので、データ更新関数を別に用意して呼び出すようにしています。

※タイマ(beat.0)のinputはタイマ設定値で、ここは最初の一回だけ更新するように記述しています。理由は後述します。

手順6　画面を定期更新する

ここまでで作った機能では画面を開いた時の結果しか取得できません。画面更新を定期的に呼び出すためのJavaScriptを追加します。

/ex1/index.php

```
～～～　省略　～～～
<script type="text/javascript">
        var INTERVAL = 1 * 1000;
        var first_=true;

        $(function()  {
                updateValues();

                setInterval(function(){
                        updateValues();
                },INTERVAL);

                $("#tmr_set").keypress(function (e){
```

updateValues()を定期呼出し

これでモニタ機能は完成です!!

手順7　　設定器を作成する

サンプル画面下部のタイマの右側テキストエリアは、beat.0のinputの値でbeat.0の周期動作間隔です。これを変更することにより、beat.0の動作周期を変更することができます。それでは、JavaScript(jQuery)を使って、このデータを更新する機能を追加しましょう。

/ex1/index.php

```
                                        ～～～　省略　～～～
$(function()  {
        updateValues();

                                        ～～～　省略　～～～

        $("#tmr_set").keypress(function (e){
                if(e.keyCode==13){                                  ENTERキー押下イベント
                        var val=Number($(this).val());
                        if(isNaN(val)){
                                alert("数値を入力して下さい");
                                return;
                        }

                        $(this).css("background-color","white");

                        var params={"proc":"plc","tag":"beat.0","idx":"input","val":val};
                        $.post("set_jw_data.php",params,function(data){
                                alert("値を更新しました。");
                                location.reload();
                        });
                }else{
                        $(this).css("background-color","yellow");
                }
                });                                                 それ以外キー押下イベント
});
```

idがtimer_setのテキストエリアについてキーイベントを設定します。

(1)　まず、エンターキー以外のキーイベントが発生した際は、このテキストエリアが選択されて何か入力されている状態のため、そのことを明示できるようにエリアを黄色にします。

(2)　何等かのデータが入力されてリターンキーが押下されたら、そのデータが数値であることを確認して、PLC-FARM側にその内容をポストします。今回は呼び出すPHPのスクリプト名を「set_jw_data.php」とし、必要なパラメータを下記のように定義します。

「proc」：文字列。JUNKWareコントローラのプロセス名を指定する。

「tag」：文字列。設定対象のタグを指定する。

「idx」：文字列。値を書き込むインデックスを指定する。

具体的には、proc=plc，tag=beat.0，idx=inputを指定して、set_jw_data.phpを呼び出しています。

次に呼び出されるPHPスクリプトを作りましょう。

/ex1/set_jw_data.php

```
<?php

        $proc = $_REQUEST["proc"];
        $tag = $_REQUEST["tag"];
        $idx = $_REQUEST["idx"];
        $val = $_REQUEST["val"];

        `echo "$tag ($idx) $val" | jld -o@$proc`;

?>
```

JUKWare　Toolsのjldというコマンドを呼び出しています。簡単なスクリプトなので、説明する必要もないと思います。jldの使い方の詳細については、JUNKWare Toolsのリファレンスマニュアルを参照してください。

なお、前述の定期更新の際、この”timer_set”は最初の一回しか更新しませんでした。それは、このキーイベント処理の際、キー入力中に更新されてしまうことを避けるためです。

以上で、設定器はできました!!

※このように、JUNKWareを使うとWEBベースのアプリケーション/画面を簡単につくることができます。出荷前検査の画面, メンテナンス用画面, 顧客用画面などいろいろな応用が考えられると思いますので、是非活用してみてください。

PLC-FARMソフトウェアマニュアル

株式会社 YOODS

URL: http://www.yoods.co.jp/

ソフトウェアについての技術的なお問合せは

☎ 083-976-0022

e-mail: info_yoods@yoods.co.jp