Sommario

MeterN 3
Concetti di base 3
Configurazione 3
Meter e Sensor 4
Formato dati 4
Le Comapps 5
Comapss standard 5
Comapss autoprodotte 5
Struttura delle comapps 6
Creare una comapps 6
Comapps pooler 6
Comapps daemon 8
Daemon_loop 9
daemon 10
daemon_com 12
prova 13

MeterN

Concetti di base

meterN ha lo scopo di monitorare contatori come quelli dell'energia elettrica.

Il suo lavoro è piuttosto semplice, recupera dal contattore la quantità totale con una frequenza di 5 min.

Può essere impostato come misuratore e registratore (logger) o come indicatore (visualizza un valore o uno stato).

Ci sono varie modalità di utilizzo dei misuratori:

- Meter = è usato per i “meters” strumenti che sono in grado di misurare delle entità.
- Sensor = servono per monitorare qualsiasi valore come temperatura o umidità.
- Indicator = servono semplicemente a mostrare un valore in tempo reale che non deve essere registrato, come la tensione. la frequenza, la percentuale di una batteria, ma possono essere usati anche per visualizzare lo stato di un oggetto come “on”, “off”, “aperto”, “chiuso”

L'interfaccia e suddivisa i 4 settori:



Zona 1: vengono visualizzati i dati del main pooling rilevati ogni 5 minuti, dati che verranno loggati, l'ultimo grafico in basso visualizza i totali giornalieri.

Zona 2: qui è presente un gauge che visualizza il dato live del meter Id 1 o, se impostato il meter “housenergy” il bilancio energetico tra prodotto - consumato - esportato.

Zona 3: in questa tabella sono visualizzati i dati live dei meter.

Zona 4: in questa tabella vengono visualizzati i dati degli “indicator”, dati che non verranno loggati.

Da qui si evince che pianificare in anticipo cosa si vuol far vedere, e dove, diviene di fondamentale importanza

Configurazione

Prima di iniziare la configurazione vera e propria di MeterN bisogna avere qualcosa da configurare, e a questo proposito entra in gioco la pianificazione.

Serve sapere quanti e quali meter o sensor intendiamo usare, quanti indicator e come visualizzarli (dargli un ordine che poi diventa la numerazione deli meter (MeterID).

Bisogna preparare le Comapps che si occupperanno di leggere i meter e assicurarsi che funzionino da terminale.

integrare le Comapps nel sistema.

Meter e Sensor

Quando configuri un contatore, ti verrà richiesto di impostare un 'Comando' principale “Main pooling” che restituirà la quantità totale (es: Litri Totali, Watt Totali all'ora, Totale m³, ..).

Per un sensore, potrebbe essere qualsiasi valore (es: Temperatura, Tensione, ..).

Poi puoi anche impostare un 'Live command', un valore non obbligatorio da mostrare sul cruscotto. meterN lo recupererà continuamente, il tempo di aggiornamento dipenderà principalmente dalla tua applicazione di comunicazione.

Mostrerà un valore istantaneo (es: Litri al minuto, Watt, ..) oppure potrebbe anche mostrare un testo, come lo stato di un dispositivo (es: Acceso/Spento, Caldo/Tiepido/Freddo, Carica/Scarica, ..).

Ora passiamo a decidere i Meter da utilizzare, facciamo alcuni esempi ma possono essercene altri secondo le proprie esigenze:

1. Produzione normalmente da fotovoltaico e se avete in funzione 123solar la comapps è già pronta con meter id 1
2. Energia consumata (consumi)
3. Energia importata (EIMP)
4. Energia esportata (EEXP)
5. Produzione PV_2 (fotovoltaico con accumulo)

1. Indicator (tensione, ampere, cosphi)

P.S.: Questa è la mia configurazione, ovviamente.

Bene ora abbiamo il primo dato per la configurazione, 5 meter , ma ci mancano ancora le comapps per leggere i nostri Meter.

Formato dati

Prima di partire con il codicg delle comapps dobbiamo aver chiaro il concetto di “formato dati”.

Il formato di output delle applicazioni di comunicazione deve rispettare il formato JSON o la struttura del set di dati IEC 62056.

Siccome tutti i dati dovranno seguire lo stesso standard, che è un dato di configurazione di MeterN, dobbiamo scegierlo ora e mantenerlo per sempre. Noi scegliamo lo standard IEC 62056 che è uno standard internazionale e comune per i contattori di ultima generazione, standard che sembra strano ma è abbastanza semplice, tutti i dati devono essere passati nel formato:

ID(VALUE*UNIT)

È lo stesso formato utilizzato nella, meglio nota, “interfaccia P1” un'interfaccia di comunicazione disponibile in alcuni contatori intelligenti per l'utilizzo dell'utilità di comunicazione locale.

Alcuni output validi:

1(10923*Wh)
water(6903*l)
1-0:1.8.1(00737044*Wh)
0-1:24.3.0(00643.165*m3)

temp(19.3*°)
humid(60.3*%)
tarif(cheap*Wh)
battery(charging*Wh)

Note importanti :

MeterN deve ricevere i valori senza alcun prefisso di unità. es: elect(39464*Wh) è valido. elect(39.464*kWh) non lo è.Per i contatori elettrici è richiesta una precisione di “W all'ora”.Per i valori decimali devono essere utilizzati solo i punti (non le virgole).I caratteri di delimitazione anteriori “(” e posteriori “)” sono mandatori, così come il carattere separatore “*” tra valore e unità.Il contatori ad impulsi o volumetrici non possono azzerarsi (es. dopo un surge), devono sempre aumentare fino al “pass-over”.

Le Comapps

Le comapps sono dei programmi, in qualsiasi linguaggio, che servono per interrogare un dispositivo di misura e leggerne i dati.

Per poterlo fare bisogna innanzitutto che il dispositivo abbia apposite API per poterlo interrogare, una volta che abbiamo i dati li dobbiamo formattare in un apposito formato leggibile da MeterN, vedi paragrafo precedente.

MeterN ha già delle comapss preconfezionate realizzate e messe a disposizione dall'autore, basta solo impostare i propri dati, altre dobbiamo realizzarle da noi, a questo scopo possiamo aiutarci studiando i vari esempi messi a disposizione dall'autore al seguente LINK

Comapss standard

- pool123s Se avete installato e funzionante 123solar questa comapss si intefaccia con l'altro applicativo e passa i dati necessari a MeterN
- remotepool123s come il precedente ma legge i dati su un sito remoto dove è installato 123solar
- houseenergy E' una comapps che legge vari meter, imposta vari Meter Id (i meter id sono fissi in quanto servono per calcolare i dati)

Comapss autoprodotte

Le comapps sono di due tipi, alla fine fanno le stesse cose, ma le fanno in modo diverso:

- pooler = leggono continuamente i dati direttamente dal meter (un dato alla volta)
- Daemon = sono dei programmi che leggono tutti i dati dal meter li parcheggiano su un file di supporto da dove vengono letti mediante un pooler

per capire la differenza di funzionamento facciamo un esempio comparativo, supponiamo di dover leggere 4 dati da un meter (W prodotti, Wh prodotti, W consumati, Wh consumati).
Con un pooler si devono fare 4 cicli di lettura sul meter (tipicamente un SDM o uno Shelly)
Con un Daemon si fa un solo ciclo di lettura sul meter e poi si legge 4 volte il file di supporto.

Il primo è da preferire se si leggono uno o due dati, il secondo è molto più performante se si devono leggere molti dati inoltre è molto utile nei casi in cui si debbano leggere dati da siti web dove ci sono limiti al numero di letture orarie e/o giornaliere

Struttura delle comapps

Pooler

Sono file singoli che vengono attivati direttamente da MeterN e si interfacciano direttamente col meter

Programma	Descrizione	Esempio
pooler.php	programma che legge i dati dal meter e li passa a MeterN	shelly1.php

Daemon

qui la cosa si fa più strutturata, dobbiamo sempre avere:

Programma	Descrizione	Esempio
com_daemon.php	Avvia il daemon_loop	shelly0_daemon.php
daemon_loop.php	legge i dati dal meter	shelly0_loop.php
daemon.php	pooler che legge i dati dal file di supporto	shelly0.php

Ogni pooler o daemon dovrà avere tanti parametri quanti sono i dati che gestisce per poter richiamare il dato necessario al momento, questo parametro verrà richiamato da MeterN e sarà inserito nella configurazione dei vari meter

Creare una comapps

Comapps pooler

Dobbiamo posizionare il programma nella directory che conterrà tutte le comapps, ma lo dobbiamo fare come amministratore, tutta la procedura andrebbe fatta seguendo il flusso senza interromperla per non perdere i diritti di amministratore senza accorgersene.
procedere nel seguente modo:

```
sudo -s
cd /var/www/comapps
```

Con il nostro edito preferito creiamo il file:

```
poolshelly.php
```

Al suo interno andremo a inserire il seguente codice:

```
#!/usr/bin/php
<?php

if (isset($_SERVER['REMOTE_ADDR'])) {
die('Direct access not permitted');
}

error_reporting(~E_WARNING);

// This script will output a ShellyEM counter into a meterN compatible format
```

```php
// Configure, then ln -s /var/www/comapps/poolshelly.php /usr/bin/poolshelly and chmod +x poolshelly.php
// Request Main command with 'poolshelly {energy | energy1 | power | power1}'

// Shelly IP
$IP = '192.168.xxx.yyy';
// Shelly request
$url = "http://"."$IP"."/status";
$remotedata = file_get_contents($url);

// meterN config

$METERID    = 1;

// No edit is needed below

if (isset($argv[1])) {

    $GP = null;
    if (!empty($remotedata)) {
        $memarray = json_decode($remotedata, true);
        if ($argv[1] == 'power') {
            $GP = $memarray['emeters'][0]['power'];
            $GP = round($GP, 1);
            echo "$METERID($GP*W)\n";
        }
        elseif ($argv[1] == 'energy') {
            if (isset($memarray['emeters'][0]['total'])) {
                $KWHT = round($memarray['emeters'][0]['total']); // Wh
                echo "$METERID($KWHT*Wh)\n";
            } else {
                die("Abording: KWHT[0] not defined\n");
            }
        }
        elseif ($argv[1] == 'power1') {
            $GP = $memarray['emeters'][1]['power'];
            $GP = round($GP, 1);
            echo "$METERID($GP*W)\n";
        }
        elseif ($argv[1] == 'energy1') {
            if (isset($memarray['emeters'][1]['total'])) {
                $KWHT = round($memarray['emeters'][1]['total']); // Wh
                echo "$METERID($KWHT*Wh)\n";
            } else {
                die("Abording: KWHT[1] not defined\n");
            }
        }
    } else {
        die("Abording: Nessun dato presente\n");
    }
} else {
    die("Usage: poolshelly { energy | energy1 | power | power1 }\n");
}
?>
```

dobbiamo inoltre rendere il file eseguibile dal sistema, per far questo eseguiamo i seguenti passaggi:

```
cd /var/www/comapps
chown www-data:www-data poolshelly.php
chmod a+x /var/www/comapps/poolshelly.php
```

a questo punto dobbiamo creare un link simbolico per poter richiamare il programma

```
ln -s /var/www/comapps/poolshelly.php /usr/bin/poolshelly
```

ora siamo pronti per l'operazione più importante, testare il software creato prima di darlo in pasto a MeterN.

Dalla shell, con riga di comando lanciare tutte le opzioni previste nel software per assicurarsi che diano il risultato voluto, se avete previsto la gestione degli errori, generateli, per assicurarvi che il software li gestisca correttamente, nell'esempio precedente:

```
poolshelly energy
poolshelly energy1
poolshelly power
poolshelly power1
poolshelly
```

l'ultimo genera l'errore di nessuna opzione richiesta e visualizza le opzioni possibili.

Quando tutto funziona possiamo abilitare il programma in MeterN, per far ciò dobbiamo modificare un file di sistema (sistema di MeterN) in questo modo:

```
cd /var/www/metern/config
nano allowed_comapps.php
```

alle voci esistenti aggiungere altre voci all'array dopo l'ultima esistente

```
$ALLWDCMD[n] = 'poolshelly energy';
$ALLWDCMD[n] = 'poolshelly power';
```

la struttura è la seguente:

1. [*n*] = il numero tra parentesi quadre deve essere un numero progressivo che segue quelle già presenti
2. poolshelly = link simbolico che richiama il nostro programma
3. energy = opzione da richiamare

inserire tante voci quante sono le opzioni che vi serve richiamare

Comapps daemon

Come detto in precedenza dobbiamo creare 3 programmi, per tenere in ordine i programmi, se non già fatto, creeremo una directory dove posizionare i com_daemon che in effetti non servono a manipolare i dati ma ad avviare i daemon_loop.

```
sudo -s
cd /var/www/comapps
mkdir -v daemon
```

Daemon_loop

Iniziamo ora a creare il programma di loop che andrà a leggere il nostro meter nello specifico si tratta di uno Shelly EM con una sola pinza che legge immissioni e prelievi distinguendoli con i valori positivi per i prelievi e negativi per le immissioni, il file lo chiameremo shelly_3_loop.php

shelly_3_loop.php

```
#!/usr/bin/php
<?php
if (isset($_SERVER['REMOTE_ADDR'])) {
    die('Direct access not permitted');
}
// Beware, only use a tmpfs as /dev/shm (ramfs) !

// This script will perform a loop to continuosly  request data to ShellyEM meter and put data into a Shelly_n_log.txt file
// Request the follow data {energy | power | volt | ampere | cosphi | energy1 | power1 }'
// **** Measurements performed with a single clamp that measures import + ed export - ******
// by Walter62  rev 00   on 02/07/23

$IP = '192.168.xxx.yyy'; // Shelly IP for import export energy
$CH = 0;                    // Channel "0 = import" or "1 = export" which you want use for volt-ampere-cosphi
$MID = 3;                   // meterN config

// No edit is needed below

$url = "http://"."$IP"."/status"; // Shelly request
$filelog = "shelly_" . $MID . "_log.txt"; // Temporary file for dataset exaple "selly_1_log.txt"

while (true) {
    $dataarray = array();
    $remotedata = "";

    $remotedata = file_get_contents($url);
    //print_r($remotedata);

    if (!empty($remotedata)) {
        $memarray = json_decode($remotedata, true);

        $GP   = round($memarray['emeters'][0]['power'],1); //W

        if ($GP <0){
            $GP_0 = 0;
            $GP_1 = abs($GP);
        } else {
            $GP_0 = $GP;
            $GP_1 = 0;
        }

        //channel 0 ==> Import
```

```
43.         //$GP_0   = round($memarray['emeters'][0]['power'],1); //W
44.         $KWHT_0 = round($memarray['emeters'][0]['total'],1); // Wh
45.
46.         //Channel 1 ==> Export
47.         //$GP_1   = round($memarray['emeters'][1]['power'],1); //W
48.         //$KWHT_1 = round($memarray['emeters'][1]['total_returned'],1);
    // Wh
49.         $KWHT_1 = round($memarray['emeters'][0]['total_returned'],1); //
    Wh
50.
51.         //channel $CH
52.         $VOLT   = round($memarray['emeters'][$CH]['voltage'],1); // V
53.         $AMP    = abs(round (${'GP_'.$CH}/$VOLT,1)); //A
54.         $PHI    = abs(round($memarray['emeters'][$CH]['pf'],2)); //
    cosphi
55.
56.         $dataarray[0] = $MID . '_0(' . $GP_0 . '*W)';    // n_0(nnn*W)
57.         $dataarray[1] = $MID . '_0(' . $KWHT_0 . '*Wh)';
58.         $dataarray[2] = $MID . '_1(' . $GP_1 . '*W)';
59.         $dataarray[3] = $MID . '_1(' . $KWHT_1 . '*Wh)';
60.         $dataarray[4] = $MID . '_V(' . $VOLT . '*V)';
61.         $dataarray[5] = $MID . '_A(' . $AMP . '*A)';
62.         $dataarray[6] = $MID . '_PHI(' . $PHI . '*phi)';
63.         $str          = implode(PHP_EOL, $dataarray);
64.         file_put_contents('/dev/shm/' . $filelog, $str);
65.
66.     } else {
67.         if (file_exists($filelog)) {
68.             $now = time();
69.             if ($now - filemtime($filelog)> 5) { // 5 sec
70.                 unlink('/dev/shm/' . $filelog);
71.             }
72.         }
73.     }
74.     usleep(500000);
75. }
76. ?>
```

Dato che con una pinza leggo sia il valore di energia importata che esportata ma con valore negativo, ho disabilitato la lettura del canale e dalla riga 34 alla 40 vado a leggere il valore di energia letto dalla pinza e in funzione che sia positivo o negativo assegno al canale appropriato il valore assoluto quindi positivo per entrambi i canali.

Ora che abbiamo il file compilato lo salviamo in:

```
/var/www/comapps
```

daemon

Ora passiamo alla stesura del programma che si occuperà di leggere i dati e renderli disponibili a MeterN, lo chiameremo shelly_3.php, il contenuto dovrà essere il seguente:

shelly_3.php

```
#!/usr/bin/php
<?php
if (isset($_SERVER['REMOTE_ADDR'])) {
    die('Direct access not permitted');
}
// This script will read output file of the ShellyEM counter and will expos the data to MeterN
// Configure, then ln -s /var/www/comapps/shelly_n.php /usr/bin/shelly_n and chmod +x shelly_n.php
// Request Main command with 'shelly_1 {energy | power | volt | ampere | cosphi | energy1 | power1 }'
// by Walter62  rev 00   on 02/07/23

$MID = 3;  // meter to call - shelly_n = shelly_3
$filelog = "shelly_" . $MID . "_log.txt"; // Temporary file for dataset exaple "selly_2_log.txt"

// No edit is needed

$help = "Usage: shelly_1 { energy | power | volt | ampere | cosphi | energy1 | power1 }\n";

if (!isset($argv[1])) {
    die($help);
}

if ($argv[1] == 'power'){
    $outstr =  exec('cat /dev/shm/'.$filelog .' | egrep "^'. $MID .'_0\(" | grep "*W)"');
}elseif ($argv[1] == 'energy'){
    $outstr =  exec('cat /dev/shm/'.$filelog .' | egrep "^'. $MID .'_0\(" | grep "*Wh)"');
}elseif ($argv[1] == 'power1'){
    $outstr =  exec('cat /dev/shm/'.$filelog .' | egrep "^'. $MID .'_1\(" | grep "*W)"');
}elseif ($argv[1] == 'energy1'){
    $outstr =  exec('cat /dev/shm/'.$filelog .' | egrep "^'. $MID .'_1\(" | grep "*Wh)"');
}elseif ($argv[1] == 'volt'){
    $outstr =  exec('cat /dev/shm/'.$filelog .' | egrep "^'. $MID .'_V\(" | grep "*V)"');
}elseif ($argv[1] == 'ampere'){
    $outstr =  exec('cat /dev/shm/'.$filelog .' | egrep "^'. $MID .'_A\(" | grep "*A)"');
}elseif ($argv[1] == 'cosphi'){
    $outstr =  exec('cat /dev/shm/'.$filelog .' | egrep "^'. $MID .'_PHI\(" | grep "*phi)"');
} else{
    die($help);
}

echo "$outstr";
?>
```

salviamo il file in:

```
/var/www/comapps
```

A questo punto abbiamo:

- Il daemon che legge i dati dal file di supporto
- Il daemon_loop che legge i file dal meter e li salva nel file di supporto

Il primo lo fa partire MeterN a chiamata, ma il secondo deve partire ad ogni avvio di sistema, o meglio di MeterN e rimanere in esecuzione.
Potremmo inserire l'avvio in “crontab” ma si avvierebbe anche se MeterN non è in funzione, per questo ci viene in auto MeterN con la sia feature interna di avvio dei daemon, per far ciò serve un programma che si occupi dell'avvio e di alcune altre attività per pulire le varie directory. Questo programma è il com_daemon.

daemon_com

Nel nostro caso, chiameremo il programma shelly_3_daemon.php e dovrà avere il seguente contenuto:

shelly_3_daemon.php

```
#!/usr/bin/php
<?php
/*
Check shelly_3_loop.php path bellow
Then ln -s  /path to/shelly_3_daemon.php /usr/bin/shelly_3_daemon
Start and stop the daemon via metern/config/config_daemon.php
And request values with shelly_3 command
*/

if (isset($_SERVER['REMOTE_ADDR'])) {
    die('Direct access not permitted');
}

if (file_exists('/dev/shm/shelly_3_daemon.pid')) {
    $cdpid = (int) file_get_contents('/dev/shm/shelly_3_daemon.pid');
    exec("ps -ef | grep $cdpid | grep shelly_3_daemon", $ret);
    if (!isset($ret[1])) {
        $cdpid = null;
        unlink('/dev/shm/shelly_3_daemon.pid');
    }
} else {
    $cdpid = null;
}

if (isset($argv[1])) {
    if (($argv[1] == 'start' || $argv[1] == 'stop') &&
file_exists('/dev/shm/shelly_3_log.txt')) {
        unlink('/dev/shm/shelly_3_log.txt');
    }
    if ($argv[1] == 'start') {
        if (is_null($cdpid)) {
            $command = 'php /var/www/comapps/shelly_3_loop.php' . ' >
/dev/null 2>&1 & echo $!;';
```

```
            $cdpid     = exec($command);
            file_put_contents('/dev/shm/shelly_3_daemon.pid', $cdpid);
        } else {
            echo "shelly_3_daemon seem to be running as $cdpid";
        }
    } else if ($argv[1] == 'stop') {
        if (!is_null($cdpid)) {
            $command = exec("kill $cdpid > /dev/null 2>&1 &");
        unlink('/dev/shm/shelly_3_daemon.pid');
        }
    } else {
        echo "Usage : shelly_3_daemon {start | stop}\n";
    }
} else {
    echo "Usage : shelly_3_daemon {start | stop}\n";
}
?>
```

prova

```
sfbsfb
FBFB
FBFDB
Bdbdfd
$a=25
```

Fonte:
Wiki - Progetti

Autore:
Walter62

Ultimo aggiornamento: **2023/07/25 09:42**