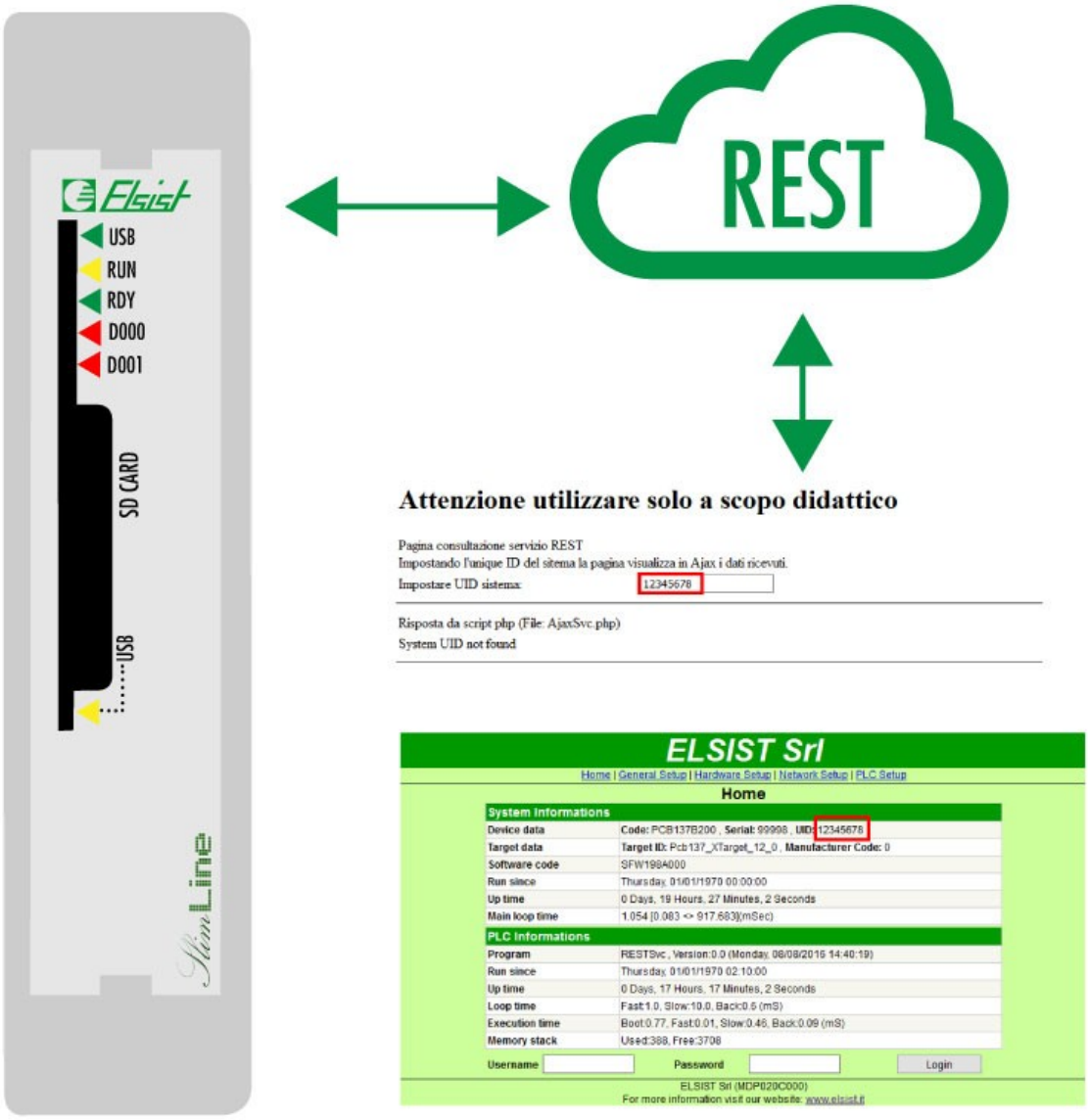
1.1 Libreria gestione servizio REST (eLLabRESTSvcLib)

Attenzione! Per utilizzare la libreria occorre importarla nel proprio progetto. Vedere capitolo relativo all'import delle librerie.

REST non è un'architettura nè uno standard, ma un insieme di linee guida per la realizzazione di un'architettura di sistema. Tutta la comunicazione tra il client ed il server avviene in HTTP quindi il server REST può essere facilmente ospitato in una server farm. Proprio per testare il servizio abbiamo pubblicato un server su un servizio di hosting gratuito http://www.slimline.altervista.org.

Utilizzando il FB ***RESTWSvcClient*** il sistema invia gli eventi generati nel sistema, ed in assenza di eventi, ciclicamente un frame di heartbeat al server nel cloud e riceve come risposta eventuali comandi dal server. Utilizzando una architettura client si superano tutti i problemi di IP pubblici il sistema funziona anche su reti NATtate in quanto è lui ad aprire la connessione verso l'IP pubblico del server nel cloud.

Appoggiandosi su di un FIFO gli eventi da inviare sono bufferizzati in locale con il timestamp relativo, se c'è connessione l'evento viene inviato al server nel cloud, ma se manca connessione il sistema tenta la connessione ed invia l'evento a connessione ripristinata. Insieme all'evento viene inviato anche il relativo timestamp e questo permette al server di collocare correttamente l'evento nel tempo.



1.1.1 RESTClient, connects to a REST web service

Type	Library
FB	eLLabRESTSvcLib_A610

Questo blocco funzione gestisce la connessione verso un server REST con protocollo HTTP.

Per garantire il buffering dei dati su eventi molto rapidi, il servizio utilizza il FB ***FIFOFile_v1*** il cui indirizzo di allocazione và passato in ***FIFOFile*** per gestire uno stack FIFO su di un file. In ***FIFOFilename*** occorre definire quale file utilizzare per appoggiare lo stack FIFO ed in ***FIFOSize*** che dimensione dare al file.

In ***FIFOIDx*** occorre fornire l'indirizzo di un array di 2 UDINT che devono essere allocati dal programma. Se si desidera che la situazione dei dati nel FIFO sia mantenuta allo spegnimento del sistema occorre allocare questo array in una memoria tampone.

In ***HTTPClient*** occorre indicare l'indirizzo di allocazione di un FB ***HTTPClient*** che gestisce la connessione al server REST. In ***HostAddress*** definire l'indirizzo IP o l'URL del server REST, è possibile definire anche l'***HostName*** che sarà utilizzato nella richiesta (Di solito è uguale a ***HostAddress***). In HostPort definire la porta del servizio REST (Di solito è la standard HTTP 80). In ***Page*** definire lo script di gestione del servizio REST.

Attivando enable ogni tempo definito in ***HBitTime*** viene inviato un messaggio di heartbeat al server REST, questo permette di controllare il corretto funzionamento del servizio anche in assenza di messaggi da inviare al server. L'uscita ***RSvcOn*** se attiva indica che la comunicazione con il server REST è attiva.

La comunicazione con il server REST è in formato JSON e quindi anche i dati che si desidera scambiare con il server devono essere codiificati in JSON. Per inviare messaggi al server REST occorre eseguire il FB ***FIFOFile_v1*** che trasferirà il messaggio nel buffer FIFO e successivamente al server. L'utilizzo di un buffer FIFO garantisce che tutti i messaggi siano inviati al server bufferizzandoli nel caso di mancata connessione.

La risposta ricevuta dal server REST è memorizzata in un buffer interno al FB (Di dimensione ***BLength*** il cui indirizzo è fornito in ***PBuffer***), alla ricezione si attiva ***SvcOk*** ed il programma utente potrà utilizare i dati presenti nel buffer.

In caso di errore ricezione dal server si attiva ***SvcError*** ed in ***PBuffer*** vi è la pagina ricevuta dal server, sarà possibile trasferirla in un file per visualizzarla da browser per indagare sull'errore.

Enable (BOOL)	Comando attivazione connessione al servizio REST.
SpyOn (BOOL)	Se attivo permette di spiare il funzionamento della FB.
FIFOFile (@FIFOFile_v1)	Indirizzo allocazione FB ***FIFOFile_v1***.
FIFOFilename (@STRING)	Percorso e nome del file in cui fare log (es.: 'SDCard/FIFO.bin').
FIFOSize (UDINT)	Dimensione in bytes dello stack FIFO.
FIFOIDx (@UDINT)	Pointer alla variabile indice di gestione FIFO (Deve essere un array di 2 UDINT).
HTTPClient (@HTTPClient)	Indirizzo allocazione FB HTTPClient di supporto.
HostAddress (@BYTE)	Indirizzo IP o URL del server a cui connettersi.
HostName (@BYTE)	Nome del server utilizzato nella richiesta (Di solito è uguale a ***HostAddress***).
HostPort (UINT)	Numero porta TCP a cui connettersi (Default 80).
Page (@BYTE))	Stringa di definizione script gestione REST.
HBitTime (UINT)	Tempo invio heartbeat al server (S). Ogni tempo definito viene inviato un messaggio di heartbeat al server.
BLength (UDINT)	Dimensione buffers Rx/Tx allocati da FB (SysRMalloc). Minimo 256 massimo 1500.
Enabled (BOOL)	Blocco funzione abilitato.
Fault (BOOL)	Attivo per un loop di programma se errore gestione.
RSvcOn (BOOL)	Servizio REST attivo. Si attiva su comunicazione con il server REST attiva, viene disattivato se si interrompe la comunicazione, errore su invio di messaggi o heartbeat.
SvcOk (BOOL)	Ok ricezione da server REST, si attiva per un loop su ricezione risposta.

SvcError (BOOL) Errore ricezione da server REST, si attiva per un loop su ricezione risposta. Il messaggio viene reinviato al server. Utilizzare uscita ***Fault*** per il controllo errore comunicazione con server.

PBuffer (@BYTE) Pointer buffers pagina ricevuta da server REST. Il buffer è allocato nel FB tramite funzione ***SysRMAlloc***, è valido per un solo loop su attivazione di ***SvcOk*** o ***SvcError***.

PktsOk (UDINT) Counter pacchetti REST correttamente scambiati con il server. Raggiunto il valore massimo il conteggio riprende da 0.

Resyncs (UDINT) Counter resincronizzazioni con il server REST. Raggiunto il valore massimo il conteggio riprende da 0. I messaggi scambiati tra FB e server hanno un identificativo che ne permette il controllo. In caso di disallineamento (Perdita di un messaggio) viene eseguita una resincronizzazione.

Errors (UDINT) Counter errori comunicazione con il server REST. Raggiunto il valore massimo il conteggio riprende da 0.

Trigger di spy

Se ***SpyOn*** attivo viene eseguita la funzione ***SysSpyData*** che permette di spiare il funzionamento della FB. Sono previsti vari livelli di triggers.

TFlags	Descrizione
16#00000001	'Td' Invio dati verso server REST
16#00000002	'Rd' Ricezione dati da server REST
16#40000000	'Er' Errori di esecuzione

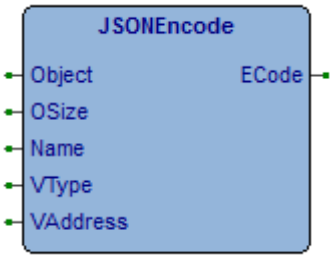
Codici di errore

In caso di errore si attiva l'uscita ***Fault***, con ***SysGetLastError*** è possibile rilevare il codice di errore. E' possibile avere la comparsa anche degli errori del FB HTTPGetPage che è utilizzato internamente.

Codice	Descrizione
10057020	FB eseguita in una task diversa dalla task di background.
10057030~8	Non sono stati definiti gli indirizzi nei parametri di ingresso.
10057050	Timeout esecuzione.
10057100~8	Errore lettura da file FIFO. Il file FIFO viene reinizializzato.
10057200	Messaggio in FIFO da inviare al server più lungo di ***BLength*** (Il messaggio è cancellato).
10057210	Messaggio heartbeat da inviare al server più lungo di ***BLength*** (Il messaggio non viene inviato).
10057300	Errore ricezione risposta HTTP da server REST, la risposta è più lunga di ***BLength***.
10057310	Errore ricezione risposta HTTP da server REST.
10057400	Troppi errori invio richiesta REST (Esauriti i retries).

1.1.2 JSONEncode, encodes a JSON message

Type	Library
FB	eLLabRESTSvcLib_A610



Questo blocco funzione può essere eseguito come una funzione, esegue la codifica di una variabile in un oggetto JSON.

In ***Object*** occorre definire l'indirizzo del buffer che contiene l'oggetto JSON, in ***OSize*** occorre definire la dimensione del buffer dell'oggetto JSON. Definendo in ***Name*** il nome, in ***VType*** il tipo, ed in ***VAddress*** l'indirizzo della variabile eseguendo il FB nell'oggetto JSON definito in ***Object*** verrà aggiunta la variabile.

In ***ECode*** viene ritornato l'eventuale codice di errore di esecuzione.

Object (@STRING) Indirizzo stringa definizione oggetto JSON.

OSize (UDINT) Definizione dimensione massima oggetto JSON.

Name (@STRING) Definizione nome variabile da inserire in oggetto.

VType (USINT) Tipo variabile da inserire in oggetto (Vedi tabella).

VAddress (@BYTE) Indirizzo variabile da inserire in oggetto.

ECode (USNT) Codice errore esecuzione
0: Nessun errore.
1: Oggetto JSON non inizia con “{“.
2: Oggetto JSON non finisce con “{“.
10: Tipo variabile non gestito.

Esempi

Nell'esempio viene creato un oggetto JSON con la definizione della variabile Wvariable.

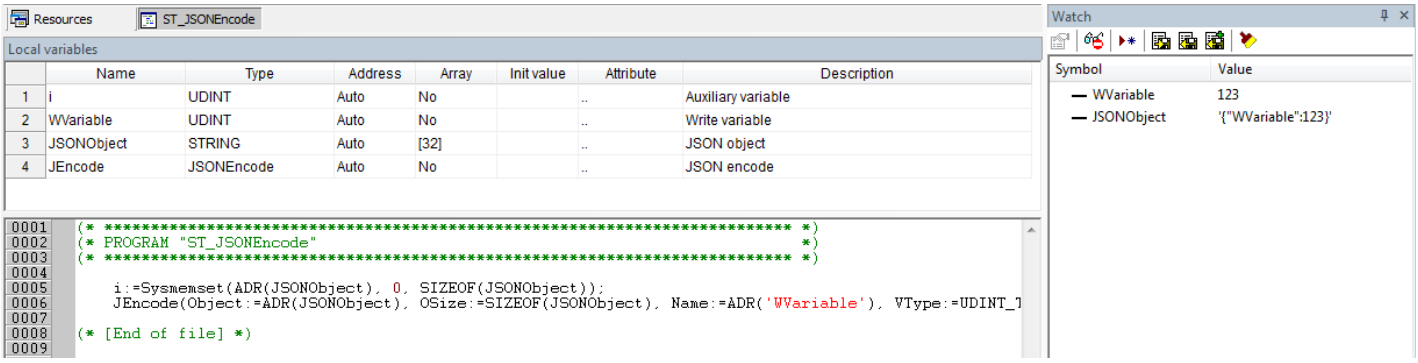
Definizione variabili

```
VAR
  JEncode : JSONEncode; (* JSON encode *)
  WVariable : UDINT; (* Write variable *)
  JSONObject : STRING[ 32 ]; (* JSON object *)
  i : UDINT; (* Auxiliary variable *)
END_VAR
```

Esempio ST

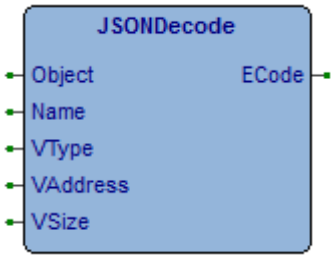
```
  i:=Sysmemset(ADR(JSONObject), 0, SIZEOF(JSONObject));
  JEncode(Object:=ADR(JSONObject), OSize:=SIZEOF(JSONObject), Name:=ADR('WVariable'), VType:=UDINT_TYPE, VAddress:=ADR(WVariable));

(* [End of file] *)
```

Screenshot programma in esecuzione



1.1.3 JSONDecode, decodes a JSON message

Type	Library
FB	eLLabRESTSvcLib_A610



Questo blocco funzione può essere eseguito come una funzione, esegue la decodifica di una variabile in un oggetto JSON.

Eseguendo il FB se nell'oggetto JSON definito in ***Object*** è presente la variabile definita in ***Name*** il suo valore verrà trasferito nel buffer definito in ***VAddress*** secondo il tipo definito in ***Vtype***. In ***VSize*** nel caso di variabili stringa occorre definire la massima lunghezza della stringa accettata.

In ***ECode*** viene ritornato l'eventuale codice di errore di esecuzione.

Object (@STRING) Indirizzo stringa definizione oggetto JSON.

Name (@STRING) Definizione nome variabile da inserire in oggetto.

VType (USINT) Tipo variabile da inserire in oggetto (Vedi tabella).

VAddress (@BYTE) Indirizzo variabile da inserire in oggetto.

VSize (UDINT) Definizione massima lunghezza stringa accettata. Solo se ***VType***=STRING_TYPE.

ECode (USNT) Codice errore esecuzione
0: Nessun errore.
1: Oggetto JSON non inizia con “{“.
2: Oggetto JSON non finisce con “{“.
3: Variabile non presente in oggetto JSON.
10, 11, 12: Errore in oggetto JSON.
13: La variabile JSON è di tipo stringa ma in ***VType*** è indicato un tipo numerico.
14: La variabile JSON è di tipo numerico ma in ***VType*** è indicato un tipo stringa.
21, 22, 23, 24, 25: Errore in decodifica valore variabile.
30, 31, 32, 33, 34, 35: Errore in memorizzazione valore variabile.

Esempi

Nell'esempio viene ricercata in un oggetto JSON la variabile Rvariable.

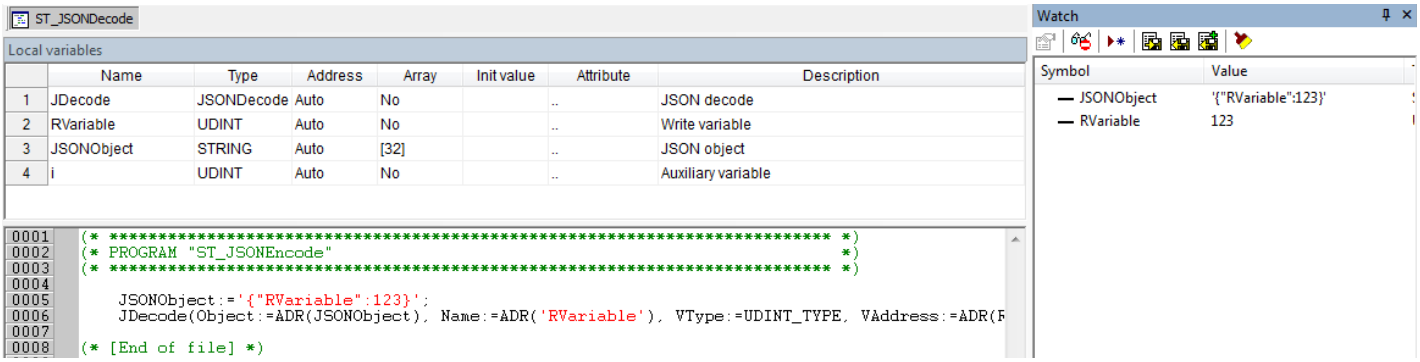
Definizione variabili

```
VAR
  JDecode : JSONDecode; (* JSON decode *)
  RVariable : UDINT; (* Read variable *)
  JSONObject : STRING[ 32 ]; (* JSON object *)
END_VAR
```

Esempio ST

```
  JSONObject:='{"RVariable":123}';
  JDecode(Object:=ADR(JSONObject), Name:=ADR('RVariable'), VType:=UDINT_TYPE, VAddress:=ADR(RVariable), VSize:=SIZEOF(RVariable));

(* [End of file] *)
```

Screenshot programma in esecuzione



Come testare il servizio REST

Il servizio REST gestito dal sistema provvede a dialogare tramite Internet con un server cloud, quindi per spiegarne il funzionamento abbiamo realizzato un server cloud su un servizio di hosting gratuito http://www.slimline.altervista.org.

L'architettura del server è basata su script php mentre la pagina di consultazione htm si appoggia a JQuery ed usa una tecnica Ajax per l'aggiornamento in tempo reale.

L'utilizzo del server cloud è libero a tutti in pratica basta caricare sul sistema SlimLine il programma dimostrativo PTP135A000 che è già configurato per inviare i dati al server cloud.

Definizione variabili

```
VAR
  REST : RESTWSvcClient_v1;  (* REST service FB *)
  LastError : UDINT;  (* Last execution error number *)
  RxPage : STRING[ 64 ];  (* Received page (Only for debug) *)
  i : INT;  (* Auxiliary counter *)
  DOut : USINT;  (* Digital output byte *)
  DInp : ARRAY[ 0..1 ] OF USINT;  (* Digital input byte *)
END_VAR
```

Esempio ST

```
  (* ---------------------------------------------------------------------- *)
  (* ESEGUO INIZIALIZZAZIONI                                                *)
  (* ---------------------------------------------------------------------- *)
  (* Eseguo inizializzazione variabili. *)

  IF (SysFirstLoop) THEN

    (* Configurazione HTTP client. *)

    HTTP.SpyOn:=TRUE; (* Spy On *)
    HTTP.Method:=TRUE; (* Request method, POST *)
    HTTP.HostAddress:=ADR('www.slimline.altervista.org'); (* Host address server REST *)
    HTTP.HostName:=HTTP.HostAddress; (* Host name server REST *)
    HTTP.Page:=ADR('/Mdp095a100/Ptp135a000/RESTSvc.php'); (* Pagina server REST *)
    HTTP.HostPort:=80; (* Porta server REST *)
    HTTP.DBSize:=512; (* Data buffer size *)
    HTTP.Timeout:=30000; (* Timeout (mS) *)

    (* Configurazione REST client. *)

    REST.SpyOn:=TRUE;
    REST.RESTSvBck:=ADR(RESTSvBck); (* REST service backup pointer*)
    REST.FIFOFile:='Storage/REST.bin'; (* Path and name of file where to log *)
    REST.FIFOSize:=1000; (* FIFO file size *)
    REST.HTTPClient:=ADR(HTTP); (* HTTP Client *)
    REST.HBitTime:=5; (* Heartbeat time (S) *)
    REST.BLength:=512; (* REST Request/Answer buffers length *)  END_IF;
  END_IF;

  (* ---------------------------------------------------------------------- *)
  (* GESTIONE SERVIZIO REST                                                 *)
  (* ---------------------------------------------------------------------- *)
  (* Eseguo gestione servizio REST. *)

  REST(Enable:=TRUE); (* Eseguo gestione servizio REST *)
  RESTSend(RSvID:=REST.RSvID, Add:=NULL);
  REST.SvcAck:=FALSE; (* REST service acknowledge *)
  IF (REST.Fault) THEN LastError:=SysGetLastError(TRUE); END_IF;

  (* ---------------------------------------------------------------------- *)
  (* INVIO INGRESSI DIGITALI AL SERVER REST                                 *)
  (* ---------------------------------------------------------------------- *)
  (* Eseguo compattazione ingressi su byte. *)

  DInp[0]:=16#00; (* Digital input byte *)
  IF (Di00CPU) THEN DInp[0]:=DInp[0] OR 16#01; END_IF;
  IF (Di01CPU) THEN DInp[0]:=DInp[0] OR 16#02; END_IF;
  IF (Di02CPU) THEN DInp[0]:=DInp[0] OR 16#04; END_IF;
  IF (Di03CPU) THEN DInp[0]:=DInp[0] OR 16#08; END_IF;
  IF (Di04CPU) THEN DInp[0]:=DInp[0] OR 16#10; END_IF;
  IF (Di05CPU) THEN DInp[0]:=DInp[0] OR 16#20; END_IF;

  (* Il FB "RESTWSvcClient" invia un messaggio di heartbeat al server ogni *)
  (* tempo impostato in "HBitTime". Per informare il server dello stato *)
```

```
  (* degli ingressi digitali lo invio su ogni variazione. *)

  IF (DInp[0] <> DInp[1]) THEN
    DInp[1]:=DInp[0]; (* Digital input byte *)
    i:=SysVarsnprintf(ADR(RESTSBf), SIZEOF(RESTSBf), 'DInp=%d', USINT_TYPE, ADR(DInp[0]));
    RESTSend(Add:=ADR(RESTSBf));
  END_IF;

  (* ---------------------------------------------------------------------- *)
  (* RICEZIONE USCITE DIGITALI DA SERVER REST                               *)
  (* ---------------------------------------------------------------------- *)
  (* Su ricezione Ok da servizio REST controllo l'Ok ricevuto. *)

  IF NOT(REST.SvcOk) THEN RETURN; END_IF;
  REST.SvcAck:=TRUE; (* REST service acknowledge *)

  (* Trasferisco pagina ricevuta in buffer. Solo per debug permette di *)
  (* visualizzare il contenuto della pagina ricevuta. *)

  i:=TO_INT(Sysmemmove(ADR(RxPage), REST.PBuffer, SIZEOF(RxPage)));

  (* Eseguo acquisizione valore uscite ricevuto da server. *)

  REST.RPAck:=0; (* REST parameters acknowledge *)
  IF (SysVarsscanf(SysStrFind(REST.PBuffer, ADR('DOut='), FIND_GET_END), '%d', USINT_TYPE, ADR(DOut))) THEN
REST.RPAck:=REST.RPAck+1; END_IF;

  (* Eseguo gestione uscite digitali. *)

  Do00CPU:=TO_BOOL(DOut AND 16#01); //Do00 CPU module
  Do01CPU:=TO_BOOL(DOut AND 16#02); //Do01 CPU module
  Do02CPU:=TO_BOOL(DOut AND 16#04); //Do02 CPU module
  Do03CPU:=TO_BOOL(DOut AND 16#08); //Do03 CPU module

(* [End of file] *)
```

Come si vede viene parametrizzato il FB ***REST*** con i parametri per la connessione al nostro server cloud di prova.

REST.Host:=ADR('www.slimline.altervista.org'); (Hostname servizio REST *)*
REST.Page:=ADR('/Mdp095a100/Ptp135a000/RESTSvc.php'); (Pagina servizio REST *)*
REST.Port:=80; (Porta servizio REST *)*

Il FB ***REST*** invia un messaggio al server cloud ogni tempo definito in ***HbitTime***. Su variazione stato ingressi digitali il FB ***RESTSend*** comanda l'invio al server cloud dello stato degli ingressi.

Ad ogni ricezione di un messaggio REST il server cloud registra i dati ricevuti in un file ini di appoggio ed invia al sistema il valore di comando delle uscite. Riporto di seguito listato del file ***RESTSvc.php*** a cui si connette il FB REST anche se il file sorgente si trova nel programma dimostrativo.

```
<?php
// ************************************************************************
// FUNZIONI CONVERSIONE DATI RICEVUTI
// ************************************************************************
// Funzioni per conversione dati.

function RxBYTE($Rx, $Ofs) {return(intval(substr($Rx, $Ofs, 2), 16));}
function RxWORD($Rx, $Ofs) {return(intval(substr($Rx, $Ofs, 4), 16));}
function RxDWORD($Rx, $Ofs) {return(intval(substr($Rx, $Ofs, 8), 16));}
function RxREAL($Rx, $Ofs) {$Pk=pack("L", intval(substr($Rx, $Ofs, 8), 16)); $Uk=unpack("f", $Pk);
return($Uk[1]);}

// ------------------------------------------------------------------------
// INCLUSIONE FILES
// ------------------------------------------------------------------------
// Inclusione files.

$HomeDir=substr($_SERVER['SCRIPT_FILENAME'],0,-strlen($_SERVER['SCRIPT_NAME'])); //Home directory
require_once($HomeDir."/Mdp095a100/Ptp135a000/Include.php"); //Inclusioni generali

// ------------------------------------------------------------------------
// CONTROLLO RICHIESTA IN ARRIVO
// ------------------------------------------------------------------------
// La richiesta deve contenere i campi, MID, UID, MV, RP. Se errore esco.

if (!CkReqPars(array("MID", "UID", "MV", "RP"))) exit("Wrong REST parameters");
if (!is_numeric($_REQUEST['UID'])) exit("Wrong system UID");
```

```
// Per ogni sistema (Riconoscibile dal suo "UID") esiste un file dati ne eseguo
// lettura e compilazione array globale.

ReadINIFile($_REQUEST['UID']); //File dati di sistema
$GLOBALS['Dt']['PollTime']=GetuTime()-$GLOBALS['Dt']['Heartbeat']; //Tempo poll sistema
$GLOBALS['Dt']['Heartbeat']=GetuTime(); //Data/Ora ultimo heartbeat (UTC)

// Salvo messaggio ricevuto.

$GLOBALS['Dt']['MV']=$_REQUEST['MV']; //Message version
$GLOBALS['Dt']['RP']=$_REQUEST['RP']; //Received parameters
$GLOBALS['Dt']['RxMessage']="MID={$GLOBALS['Dt']['MID']}, UID={$_REQUEST['UID']}, MV={$GLOBALS['Dt']['MV']},
RP={$GLOBALS['Dt']['RP']}"; //Last request
if (isset($_REQUEST['Data'])) $GLOBALS['Dt']['RxMessage'].=", Data={$_REQUEST['Data']}";

// ------------------------------------------------------------------------
// CONTROLLO ID MESSAGGIO
// ------------------------------------------------------------------------
// Controllo se ricevuto l'acknowledge dallo SlimLine del messaggio REST
// inviato precedentemente dal server. Controllo se il  MID ricevuto è
// corretto (Successivo al MID del messaggio precedente).

if ((($_REQUEST['MID']-$GLOBALS['Dt']['MID'])&0xFFFF) == 1)
{
  // Ricevuto MID successivo messaggio corretto (Nessun messaggio è
  // andato perso) utilizzo MID ricevuto.

  $GLOBALS['Dt']['MID']=$_REQUEST['MID']; //Message ID
}
else
{
  // Errore ricezione messaggi, occorre eseguire una resincronizzazione
  // sistema, viene inviato un numero random che sarà utilizzato dal
  // sistema come prossimo MID.

  $GLOBALS['Dt']['MID']=rand(0, 65535); //Message ID
  $GLOBALS['Dt']['Resyncs']++; //REST resyncronizations
}

// ------------------------------------------------------------------------
// ACQUISIZIONE INFORMAZIONI DAL MESSAGGIO DATI
// ------------------------------------------------------------------------
// Un messaggio dati contiene un campo "Data" composto da diversi campi, ogni
// byte occupa due caratteri ascii. I dati sono in Big endian, MSB ... LSB.
// +---+---+-+-+-+-+-+-+-+...+-+
// | Length|0|0| Epoch | Value |
// +---+---+-+-+-+-+-+-+-+...+-+
//
// Length: Lunghezza record (2 byte)
// Epoch: Epoch time (4 byte)
// Value: Stringa con valore (Lunghezza variabile)
// ------------------------------------------------------------------------
// Se messaggio ricevuto contiene campo "Data" eseguo acquisizione dati campo.

if (!isset($_REQUEST['Data'])) goto SENDDATA;
$GLOBALS['Dt']['Length']=RxWORD($_REQUEST['Data'], 0); //Lunghezza record dati
$GLOBALS['Dt']['Epoch']=RxDWORD($_REQUEST['Data'], 8); //Epoch time relativo al record dati
$GLOBALS['Dt']['Value']=substr($_REQUEST['Data'], 16, ($GLOBALS['Dt']['Length']-8)); //Stringa valore record
dati

// Nel campo "Value" il sistema SlimLIne invia le variabili indicandole nel modo
// "Var11=Valore|Var2=Valore|..." controllo che vi sia almeno un "=".
// Nel nostro esempio vi sarà una sola variabile "DInp=xx".

if (strpos($GLOBALS['Dt']['Value'], "=") === false) goto SENDDATA;

// Estraggo nome e valore delle variabili creando un array associativo.
// Mi troverò con $DtArray('DInp' => xx, ...)

$DtArray=array(); //Array associativo dati ricevuti
$Variables=explode("|", $GLOBALS['Dt']['Value']); //Variables array
foreach ($Variables as &$Variable)
{
  if (strpos($Variable, "=") !== false)
  {
    $VNameValue=explode("=", $Variable); //Array Nome/Valore variabile
    $DtArray=array_merge($DtArray, array($VNameValue[0] => $VNameValue[1]));
```

```
  }
}

// Appoggio stato ingressi digitali.

$GLOBALS['Dt']['DInp']=$DtArray['DInp']; //Stato ingressi digitali

// ------------------------------------------------------------------------
// INVIO DATI AL SISTEMA
// ------------------------------------------------------------------------
// Eseguo scrittura file per storicizzare i dati sul server.

SENDDATA:
WriteINIFile($_REQUEST['UID']); //Scrittura dile ini

// Inserisco la definizione dei campi da impostare, separo ogni campo con
// lo spazio per permettere nel sistema alla scanf di interrompersi sulla
// acquisizione di valori stringa. Nel nostro esempio vi è un solo campo.

$RetPars=sprintf("DOut=%d ", $GLOBALS['Dt']['DOut']);

// ------------------------------------------------------------------------
// RITORNO PAGINA AL CLIENT
// ------------------------------------------------------------------------
// Compilo messaggio di risposta che inizia con il MID. Il valore ritornato
// è calcolato sommando il valore di UID. In questo modo si garantisce che
// il sistema che riceve il messaggio possa verificalo utilizzando il suo
// unique ID.

$GLOBALS['Dt']['TxMessage']=sprintf("MID=%d", ($GLOBALS['Dt']['MID']+$_REQUEST['UID'])&0xFFFF); //Return page
$GLOBALS['Dt']['TxMessage'].=sprintf("&RP=%d", 0); //Return page
$GLOBALS['Dt']['TxMessage'].=sprintf("&Page=%s", $RetPars); //Return page
echo $GLOBALS['Dt']['TxMessage'];
?>
```

Come si vede dal listato ho volutamente realizzato una gestione molto semplice utilizzando per l'appoggio un file ini, ma in realtà chi ha dimestichezza con applicazioni web troverà molto più efficiente appoggiarsi ad un database.

Come funziona il servizio

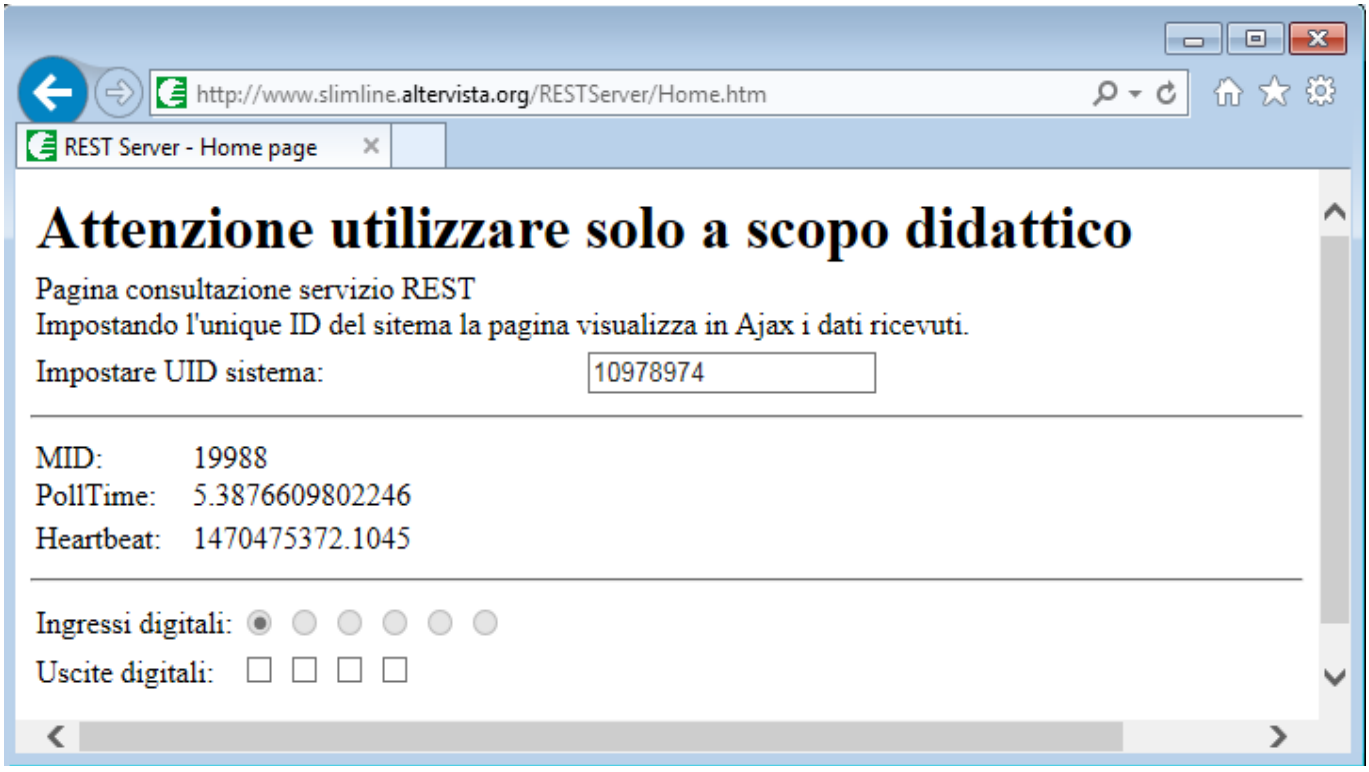
Come abbiamo visto sistema SlimLine invia i dati al server cloud (Esegue lo script RESTSvc.php inviando in GET i dati) che in base all'unique ID del sistema controlla se esiste già sul server un file ini dedicato. Se esiste ne esegue lettura in caso contrario viene creato un nuovo file, nel file sono contenuti tutti i dati necessari alla gestione del servizio. Ecco un listato di esempio del file ini.

```
MID="19867"
MV="1.0"
RP="1"
Length="14"
Epoch="1470474898"
Value="DInp=2"
DInp="2"
DOut="1"
RxMessage="MID=19866, UID=10978974, MV=1.0, RP=1"
TxMessage=
Resyncs="1"
PollTime="5.3649678230286"
Heartbeat="1470474724.041"
```

Come si vede è memorizzato il MID cioè l'ID progressivo del messaggio che permette di effettuare il controllo sui messaggi ricevuti. Sono poi memorizzati tutti gli altri campi al solo fine di debug, in questo modo è possibile capire i meccanismi di funzionamento del servizio.

I dati sono aggiornati ad ogni ricezione di messaggio dal sistema (Tempo di heartbeat o su variazione ingressi) e nel campo ***PollTime*** lo script php calcola il tempo intercorso tra gli aggiornamenti.

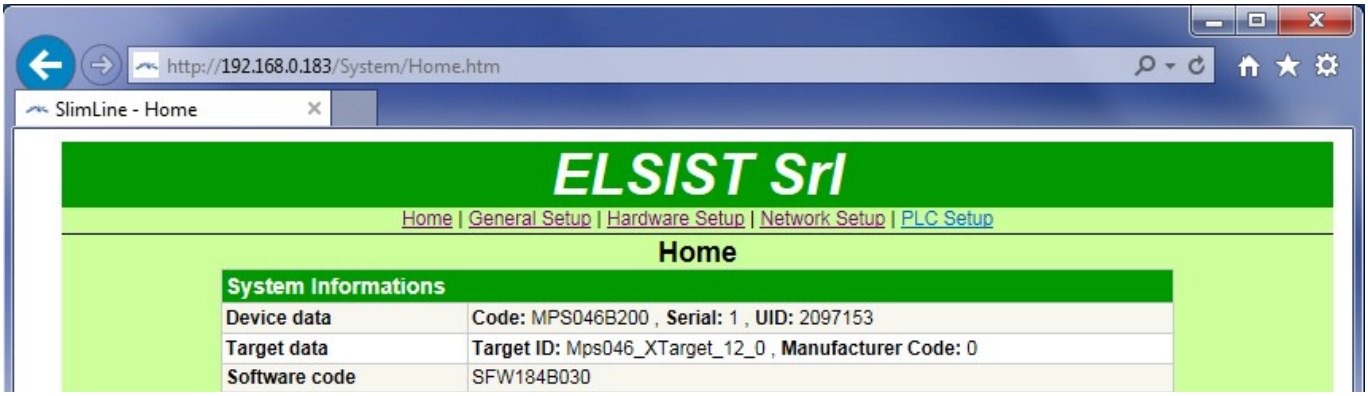
Per la visualizzazione dei dati vedere la pagina htm al link http://www.slimline.altervista.org. Come si vede accedendo alla pagina avremo qualcosa di simile.



Come si vede vi è un avvertimento di utilizzare il servizio a solo scopo didattico in quanto sono assenti password di accesso ed altre protezioni. Quindi risulta evidente che inserendo UID di sistema a caso è possibile trovare l'UID del vostro sistema e comandarne le uscite.

Tutto questo è stato fatto volutamente per limitare la complessità del dimostrativo, se si desidera utilizzarlo è certamente possibile partendo dai programmi sorgenti inserire più funzioni e password di accesso.

Ricordo che l'***UniqueID*** (UID) del sistema si può visualizzare in debug da LogicLab ma è anche visibile dalla pagina web.



Come dicevo la pagina htm utilizza JQuery ed è aggiornata in tempo reale con chiamate Ajax. Non mi dilungo sulla spiegazione dei TAGs html in quanto sono facilmente intuibili. Quasi interamente la pagina si basa su campi div che sono aggiornati da Javascript ecco Il sorgente.

```
// ------------------------------------------------------------------------
// FUNZIONE ESEGUITA SU LOAD PAGINA
// ------------------------------------------------------------------------
// Sul load della pagina attivo ajax.

$(document).ready(function()
{
  AjaxCall(); //Eseguo chiamata ajax su load pagina
  setInterval(AjaxCall, 5000); //Imposto chiamata ciclica ajax
});

// ------------------------------------------------------------------------
// RICHIESTA AJAX
// ------------------------------------------------------------------------
// Viene eseguita la richiesta ajax. Eseguo lo script "AjaxSvc.php" passando
// i parametri in POST.

function AjaxCall()
{
  // Compongo byte di gestione comando output.

  var DOut=0x00; //Digital output byte command
  if ($("#Do00CPU").is(':checked')) DOut+=0x01;
  if ($("#Do01CPU").is(':checked')) DOut+=0x02;
  if ($("#Do02CPU").is(':checked')) DOut+=0x04;
  if ($("#Do03CPU").is(':checked')) DOut+=0x08;

  // Eseguo invio richiesta ajax con parametri in POST.

  $.ajax(
  {
    type:"POST",
    url:"/Mdp095a100/Ptp135a000/AjaxSvc.php",
    data:"UID="+$("#UID").val()+"&DOut="+DOut,
    dataType:"html",

    // Funzione eseguita su successo della chiamata.

    success:function(Answer)
    {
      // Copio stringa ricevuta da script php per visualizzazione.

      $("div#Answer").html(urldecode(Answer));

      // Suddivido campi, sono separati dal carattere "|".

      var AArray=Answer.split('|'); //Answer array
      for(var i=0; i<AArray.length; i++)
      {
        // Suddivo i campi tra nome campo e valore. Valorizzo il
        // div di visualizzazione valore.

        var VArray=AArray[i].split('=');
```

```
          $("div#"+VArray[0]).html(urldecode(VArray[1]));
        }

        // Eseguo valorizzazione radio button stato ingressi digitali.

        $("#Di00CPU").prop("checked", ($("div#DInp").html()&0x01?true:false));
        $("#Di01CPU").prop("checked", ($("div#DInp").html()&0x02?true:false));
        $("#Di02CPU").prop("checked", ($("div#DInp").html()&0x04?true:false));
        $("#Di03CPU").prop("checked", ($("div#DInp").html()&0x08?true:false));
        $("#Di04CPU").prop("checked", ($("div#DInp").html()&0x10?true:false));
        $("#Di05CPU").prop("checked", ($("div#DInp").html()&0x20?true:false));
      },

      // Funzione eseguita su errore chiamata.

      error: function()
      {
        $("div#Answer").html("Call error");
      }
    });
  }

  // ------------------------------------------------------------------------
  // DECODIFICA CARATTERI URL
  // ------------------------------------------------------------------------
  // Per evitare che nel campo valore vi siano caratteri "=" che possono dare
  // fastidio nello split converto campo dati in entit€ URL e con questa
  // funzione ne eseguo la decodifica.

  function urldecode (str) {return decodeURIComponent((str + '').replace(/\+/g, '%20'));}
</script>
```

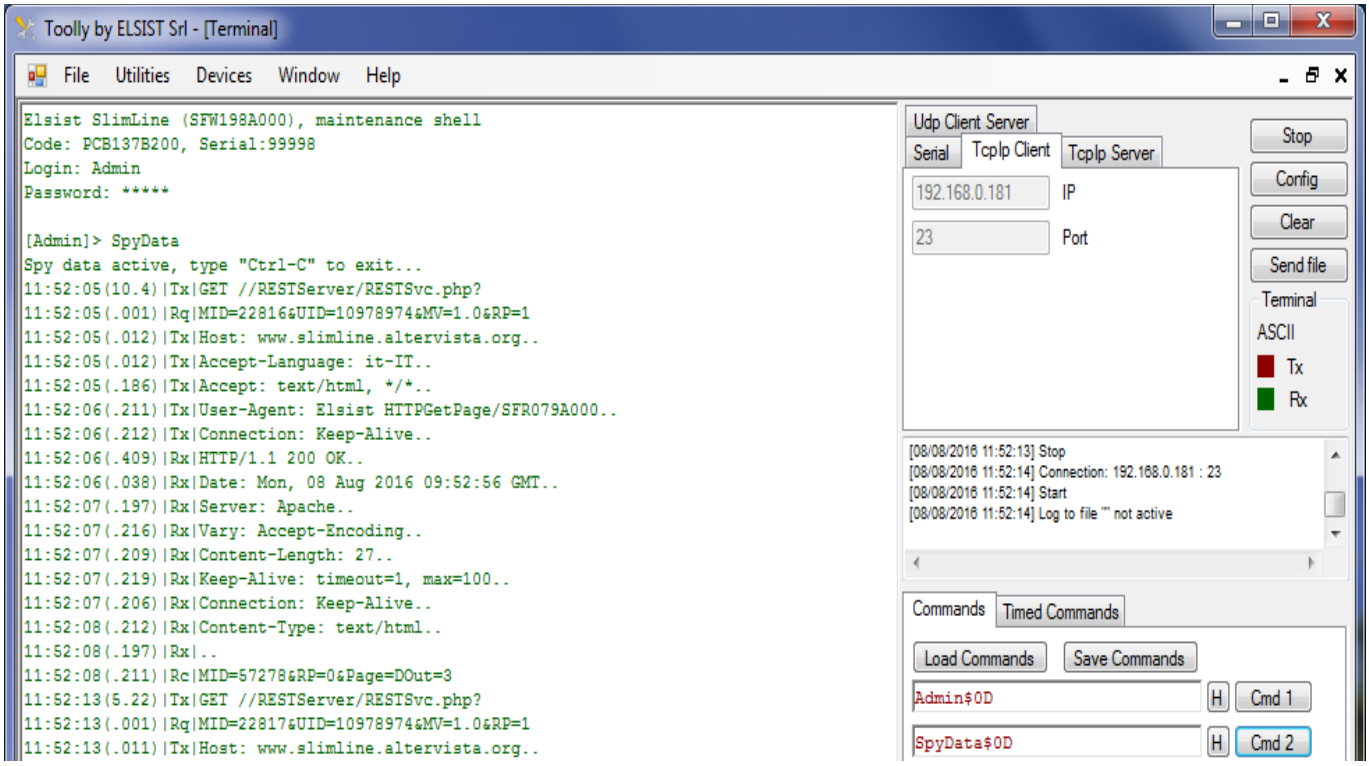
E' interessante notare che le variabili stringa per evitare il problema di trovarsi caratteri “=” all'interno sono state codificate come richieste URL dallo script php e quindi devono essere decodificate da Javascript.

La richiesta Ajax esegue sul server lo script ***AjaxSvc.php*** ecco il listato.

```
// ------------------------------------------------------------------------------
// INCLUSIONE FILES
// ------------------------------------------------------------------------------
// Inclusione files.

$HomeDir=substr($_SERVER['SCRIPT_FILENAME'],0,-strlen($_SERVER['SCRIPT_NAME'])); //Rilevo Home directory
require_once($HomeDir."/Mdp095a100/Ptp135a000/Include.php"); //Inclusioni generali

// ------------------------------------------------------------------------------
// CONTROLLO RICHIESTA IN ARRIVO
// ------------------------------------------------------------------------------
// La richiesta deve contenere i campi, UID, DOut. Se errore esco.

if (!CkReqPars(array("UID", "DOut"))) exit("Wrong REST parameters");
if (!is_numeric($_REQUEST['UID'])) exit("Wrong system UID");

// ------------------------------------------------------------------------------
// ESEGUO LETTURA FILE DATI DI SISTEMA
// ------------------------------------------------------------------------------
// Per ogni sistema (Riconoscibile dal suo "UID") esiste un file dati ne eseguo
// lettura e compilazione array globale.

if (!ReadINIFile($_REQUEST['UID'])) exit("System UID not found");

// Salvo byte comando uscite digitali.

$GLOBALS['Dt']['DOut']=$_REQUEST['DOut']; //Comando uscite digitali

// Eseguo scrittura file.

WriteINIFile($_REQUEST['UID']);

// ------------------------------------------------------------------------------
// RITORNO DATI A PAGINA WEB
// ------------------------------------------------------------------------------
// Ritorno dati a pagina web. Per evitare che nel campo valore vi siano
// caratteri "=" che danno fastidio nello split converto stringhe in entità URL.

$Return="MID={$GLOBALS['Dt']['MID']}"; //Message ID
$Return.="|MV={$GLOBALS['Dt']['MV']}"; //Message version
$Return.="|RP={$GLOBALS['Dt']['RP']}"; //Received parameters
$Return.="|Length={$GLOBALS['Dt']['Length']}"; //Lunghezza record dati
$Return.="|Epoch={$GLOBALS['Dt']['Epoch']}"; //Epoch time relativo al record dati
$Return.="|Value=".urlencode($GLOBALS['Dt']['Value']); //Stringa valore record dati
$Return.="|DInp={$GLOBALS['Dt']['DInp']}"; //Stato ingressi digitali
$Return.="|DOut={$GLOBALS['Dt']['DOut']}"; //Comando uscite digitali
$Return.="|RxMessage=".urlencode($GLOBALS['Dt']['RxMessage']); //Messaggio ricevuto
$Return.="|TxMessage=".urlencode($GLOBALS['Dt']['TxMessage']); //Messaggio trasmesso
$Return.="|Resyncs={$GLOBALS['Dt']['Resyncs']}"; //REST resyncronizations
$Return.="|PollTime={$GLOBALS['Dt']['PollTime']}"; //Tempo poll sistema
$Return.="|Heartbeat={$GLOBALS['Dt']['Heartbeat']}"; //Data/Ora ultimo heartbeat (UTC)
echo $Return;
```

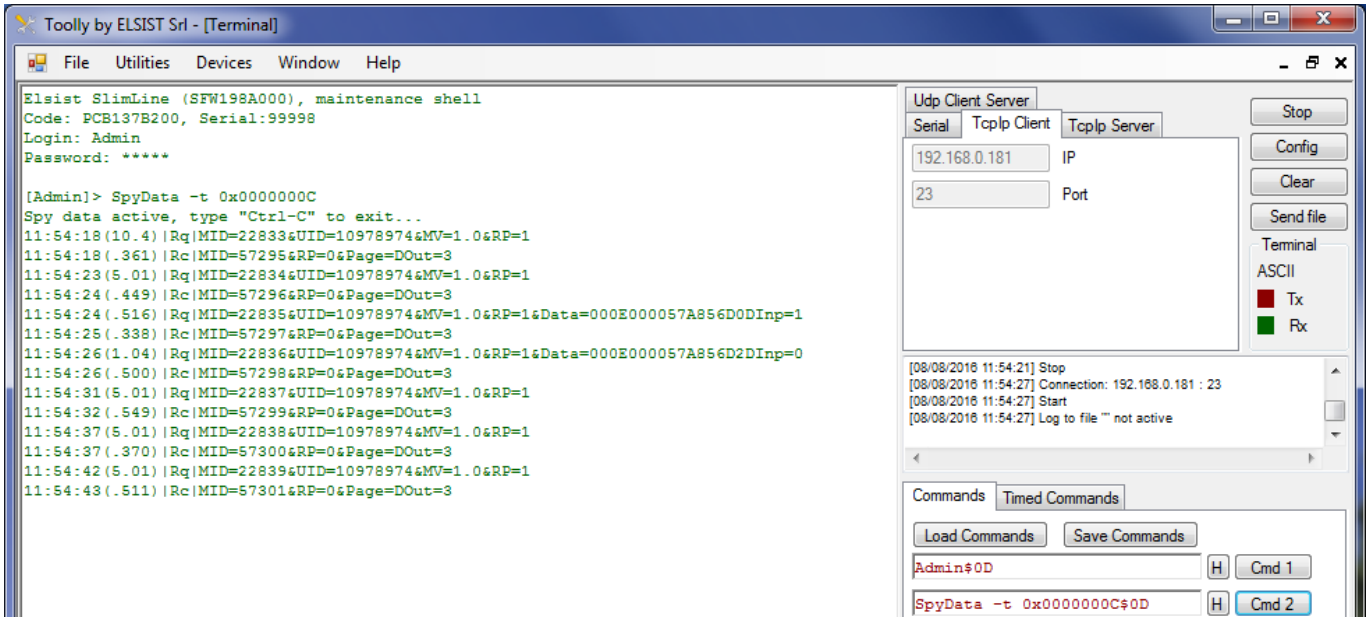
Come testare il dimostrativo

Per testare il dimostrativo basta caricare il programma su di un sistema SlimLine connesso alla rete aziendale ed opportunamente configurato per poter accedere ad Internet (Impostare Gateway e DNS server). Il programma è già configurato con l'URL del servizio su Altervista e con l'indirizzo dello script da eseguire. Come si vede dal sorgente è stato attivato lo Spy del FB ***RESTWSvcClient*** quindi dalla console di spionaggio è possibile verificare tutti i pacchetti in scambio tra il sistema ed il server cloud. Ecco la schermata di Toolly con l'attivazione della console di spionaggio totale ***SpyData<CR>*** (Tutti i pacchetti sono spiati).



In Tx le stringhe inviate al server ed in ***Rx*** quelle ricevute, a fianco di ogni riga vi è il tempo trascorso dalla esecuzione della riga precedente. I dati inviati sono etichettati con ***Rq*** in ***Rc*** quelli ricevuti.

Attivando il trigger sul comando di spionaggio ***SpyData -t 0x0000000C<CR>***, visualizzeremo solo i dati inviati e ricevuti senza visualizzare tutte le stringhe di gestione del protocollo HTTP.



Dalla analisi vederemo che normalmente ogni 5 secondi viene inviato un pacchetto di heartbeat, ma quando si ha la variazione di un ingresso digitale viene immediatamente inviato un messaggio con il campo ***Data*** che contiene lo stato degli ingressi.