

```
1 <?php
2 // ****
3 // Project : PHPToModbus
4 // Programmer : Sergio Bertana
5 // Date : 19/05/2015
6 // ****
7 // Eseguo acquisizione stutura modbus dallo SlimLine.
8 // -----
9
10 // -----
11 // INCLUSIONE FILES
12 // -----
13 // Inclusione funzioni standard.
14
15 $_SERVER[ 'DOCUMENT_ROOT' ]="/var/www/";
16 require_once($_SERVER[ 'DOCUMENT_ROOT' ]. "PHPScripts/ModbusMaster.php");
17
18 // Istanzia classe modbus ed eseguo comando "Read Multiple Registers".
19
20 $Modbus=new ModbusMaster( "192.168.0.180" , "TCP" , 5 , false );
21 $RData=$Modbus->ReadMultipleRegisters(1, 1000, 4);
22
23 // A solo scopo di test eseguo report dello stato comando. Se errore esecuzione
24 // salto a fine script.
25
26 echo "<br>Modbus status: ".$Modbus->Status."<br>";
27 if ($RData === false) {MMError($Modbus, $HmCDT); goto END_SCRIPT;}
28 echo "<br>";
29
30 // Eseguo report variabili lette dallo SlimLine.
31
32 echo "CmdCount: ".$Modbus->RxUINT($RData, 0)."<br>"; //UINT, CmdCount, Command counter
33 echo "Temperature: ".$Modbus->RxREAL($RData, 4)."<br>"; //REAL, Temperature,
Temperature value
34
35 END_SCRIPT:
36 unset($Modbus); //Distruttore di classe
37 ?>
```

```
<?php
// ****
// Project      : PHPToModbus
// Programmer   : Sergio Bertana
// Date         : 19/05/2015
// ****
// Questo file contiene le funzioni di gestione tabelle databases.
// ----

// ######
// CLASSE GESTIONE MODBUS MASTER
// #####

class ModbusMaster
{
    // -----
    // VARIABILI GLOBALI PUBBLICHE DI CLASSE
    // -----
    // Eseguo dichiarazione variabili globali pubbliche di classe.

    public $Status="" ; //Socket status

    // -----
    // VARIABILI GLOBALI PRIVATE DI CLASSE
    // -----
    // Eseguo dichiarazione variabili globali private di classe.

    private $Host="127.0.0.1"; //Indirizzo IP host
    private $Port="502"; //Porta host
    private $Client=""; //Indirizzo IP client
    private $CPort="502"; //Porta client
    private $Timeout=5; //Timeout (S)
    private $Protocol="UDP"; //Socket protocol (TCP, UDP)
    private $Sock; //Socket gestione protocollo
    private $Endianness=false; //Endianness(little=false, big=true)
    private $IsConnect=false; //Socket connesso

    // #####
    // FUNZIONI PUBBLICHE
    // #####
    // *****

    // ****
    // COSTRUTTORE DI CLASSE
    // ****
    // Costruzione oggetto di classe.

    //
    // Parametri funzione:
    // $Host: Indirizzo IP host
    // $Protocol: Definizione protocollo
    //
    // La funzione non prevede ritorni.
    // ----

    public function __construct($Host, $Protocol, $Timeout, $Endianness)
    {
        // Salvo parametri nelle variabili di classe.

        $this->Protocol=$Protocol; //Socket protocol (TCP, UDP)
```

```
$this->Host=$Host; //Indirizzo IP host
$this->Endianness=$Endianness; //Endianness(little=false, big=true)
$this->Timeout=$Timeout; //Timeout (S)

// Eseguo connessione al socket. La connessione sarà persistente
// per tutte le chiamate ai metodi dell'oggetto. Verrà automaticamente
// disconnessa alla distruzione dell'oggetto.

switch($this->Protocol)
{
    case "TCP": $this->Sock=socket_create(AF_INET, SOCK_STREAM, SOL_TCP); break;
    //Socket gestione protocollo
    case "UDP": $this->Sock=socket_create(AF_INET, SOCK_DGRAM, SOL_UDP); break;
    //Socket gestione protocollo
    default: $this->Status.="- Unknown socket protocol, should be 'TCP' or 'UDP' ";
    return;
}

// Bind the client socket to a specific local port.

if (strlen($this->Client) > 0)
{
    $Result=socket_bind($this->Sock, $this->Client, $this->CPort);
    if ($Result === false) {$this->Status.="- Bind failed: (".$Result"), ".
    socket_strerror(socket_last_error($this->Sock)); return;}
    $this->Status.="- Bound"; //Socket status
}

// Socket settings and connect.

socket_set_option($this->Sock, SOL_SOCKET, SO_SNDTIMEO, array('sec'=>$this->Timeout,
'usec'=>0));
$Result=@socket_connect($this->Sock, $this->Host, $this->Port);
if ($Result === false) {$this->Status.="- Connect failed: (".$Result"), ".
socket_strerror(socket_last_error($this->Sock)); return;}
$this->Status.="- Connected"; //Socket status
$this->IsConnect=true; //Socket connesso
}

// ****
// DISTRUTTORE DI CLASSE
// ****
// Distruzione oggetto di classe.
//
// La funzione non ha parametri.
// La funzione non prevede ritorni.
// ----

public function __destruct()
{
    // Se socket connesso lo disconnetto.

    if ($this->IsConnect)
    {
        $this->IsConnect=false; //Socket connesso
        socket_close($this->Sock);
    }
}
```

```

// *****
// FUNZIONI CONVERSIONE DATI IEC IN PACKET
// *****
// Conversione dati IEC in dati per funzione "WriteMultipleRegisters".
// Dato il valore, ritornano la stringa da inserire in packet.
// -----
// Funzioni per conversione dati per funzione "WriteMultipleRegisters".

function TxBYTE($Value) {return(chr($Value & 0xFF));}
function TxUINT($Value) {return(self::TxBYTE($Value >> 8).self::TxBYTE($Value));}
function TxINT($Value) {return(self::TxBYTE($Value >> 8).self::TxBYTE($Value));}
function TxUDINT($Value) {return(self::TxEndian($Value));}
function TxDINT($Value) {return(self::TxEndian($Value));}

// *****
// FUNZIONI CONVERSIONE DATI RICEVUTI IN DATI IEC
// *****
// Conversione dati ricevuti da funzione "ReadMultipleRegisters" in IEC.
// Dato l'array ricevuto, ritornano il valore.
// -----
// Funzioni alias per conversione dati da funzione "ReadCoilStatus".

function RxBOOL($Rx, $Ofs) {return(ord($Rx[$Ofs]));}

// Funzioni alias per conversione dati da funzione "ReadMultipleRegisters".

function RxUSINT($Rx, $Ofs) {return(self::RxBYTE($Rx, $Ofs));}
function RxUINT($Rx, $Ofs) {return(self::RxWORD($Rx, $Ofs));}
function RxUDINT($Rx, $Ofs) {return(self::RxDWORD($Rx, $Ofs));}

// Funzioni per conversione dati da funzione "ReadMultipleRegisters".

function RxBYTE($Rx, $Ofs) {if (!$this->Endianness) return(ord($Rx[$Ofs+1])); else return
(ord($Rx[$Ofs]));}
function RxWORD($Rx, $Ofs) {return((ord($Rx[$Ofs])*256)+ord($Rx[$Ofs+1]));}
function RxDWORD($Rx, $Ofs) {return(self::RxEndian(substr($Rx, $Ofs, 4)));}
function RxREAL($Rx, $Ofs) {$Pk=pack("L", self::RxEndian(substr($Rx, $Ofs, 4))); $Uk=
unpack("f", $Pk); return($Uk[1]);}

// Funzione conversione dato stringa.

function RxSTRING($Rx, $Ofs)
{
    for($Str=""; $Ofs < 256; $Ofs+=2)
    {
        $As=substr($Rx, $Ofs, 2); //Sono variabili WORD separati i 2 bytes
        if (!$this->Endianness)
        {
            if (ord($As[1])) $Str.=$As[1]; else return($Str); //Carico MSB
            if (ord($As[0])) $Str.=$As[0]; else return($Str); //Carico LSB
        }
        else
        {
            if (ord($As[0])) $Str.=$As[0]; else return($Str); //Carico MSB
            if (ord($As[1])) $Str.=$As[1]; else return($Str); //Carico LSB
        }
    }
}

```

```
    return( "RxSTRING error" );
}

// ****
// READ COIL STATUS
// ****
// Gestione comando Modbus "Read Coil Status".
//
// Parametri funzione:
// $UnitID: Indirizzo di nodo
// $Address: Indirizzo registri
// $Coils: Numero contatti da leggere
//
// La funzione ritorna il dato ricevuto. "false" se errore.
// ----

function ReadCoilStatus($UnitID, $Address, $Coils)
{
    // Controllo se socket connesso. Se non connesso non riporto status
    // per mantenere indicazioni riportate in apertura.

    if (!$this->IsConnect) return(false);
    $this->Status="ReadCoilStatus: START";

    // Invio pacchetto interrogazione.

    $TPacket='';
    $TPacket.=$this->TxINT(rand(0,65000)); //Transaction ID
    $TPacket.=$this->TxINT(0); //Protocol ID
    $TPacket.=$this->TxINT(5+1); //Lenght
    $TPacket.=$this->TxBYTE($UnitID); //Unit ID
    $TPacket.=$this->TxBYTE(1); //Function code
    $TPacket.=$this->TxINT($Address-1); //Register address
    $TPacket.=$this->TxINT($Coils); //Nr of coils
    $this->Send($TPacket); //Trasmissione pacchetto

    // Attesa pacchetto di risposta.

    $RPacket=$this->Receive(); //Received packet
    if ($RPacket === false) {$RData=false; goto FUNCTION_END;}

    // Controllo codice di risposta, controllo se numero bytes ritornati è
    // corretto e ritorno stringa dati.

    if (!$this->ResponseCode($TPacket, $RPacket)) {$RData=false; goto FUNCTION_END;}
    if (ord($RPacket[8]) != ceil($Coils/8)) {$RData=false; goto FUNCTION_END;}

    // Eseguo loop su tutti i bytes ricevuti.

    for ($RData="", $Byte=0; $Byte < ord($RPacket[8]); $Byte++)
    {
        // Eseguo loop sugli 8 bits del byte.

        for($i=0; $i<8; $i++)
            if ((ord($RPacket[$Byte+9]) >> $i)&0x01) $RData.=chr(1); else $RData.=chr(0);
    }

    // Termino comando ed esco.
```

```
FUNCTION_END:  
$this->Status.= " - ReadCoilStatus: DONE";  
return($RData);  
}  
  
// ****=  
// READ MULTIPLE REGISTERS  
// ****=  
// Gestione comando Modbus "Read Holding Registers".  
//  
// Parametri funzione:  
// $UnitID: Indirizzo di nodo  
// $Address: Indirizzo registri  
// $Registers: Numero registri da leggere  
//  
// La funzione ritorna il dato ricevuto. "false" se errore.  
// -----  
  
function ReadMultipleRegisters($UnitID, $Address, $Registers)  
{  
    // Controllo se socket connesso. Se non connesso non riporto status  
    // per mantenere indicazioni riportate in apertura.  
  
    if (!$this->IsConnect) return(false);  
    $this->Status="ReadMultipleRegisters: START";  
  
    // Invio pacchetto interrogazione.  
  
    $TPacket='';  
    $TPacket.=$this->TxINT(rand(0,65000)); //Transaction ID  
    $TPacket.=$this->TxINT(0); //Protocol ID  
    $TPacket.=$this->TxINT(5+1); //Length  
    $TPacket.=$this->TxBYTE($UnitID); //Unit ID  
    $TPacket.=$this->TxBYTE(3); //Function code  
    $TPacket.=$this->TxINT($Address-1); //Register address  
    $TPacket.=$this->TxINT($Registers); //Nr of registers  
    $this->Send($TPacket); //Transmission packet  
  
    // Attesa pacchetto di risposta.  
  
    $RPacket=$this->Receive(); //Received packet  
    if ($RPacket === false) { $RData=false; goto FUNCTION_END; }  
  
    // Controllo codice di risposta, controllo se numero bytes ritornati è  
    // corretto e ritorno stringa dati.  
  
    if (!$this->ResponseCode($TPacket, $RPacket)) { $RData=false; goto FUNCTION_END; }  
    if (ord($RPacket[8]) != ($Registers*2)) { $RData=false; goto FUNCTION_END; }  
    $RData=substr($RPacket, 9, ord($RPacket[8])); //Received data  
  
    // Termino comando ed esco.  
  
FUNCTION_END:  
$this->Status.= " - ReadMultipleRegisters: DONE";  
return($RData);  
}
```

```

// ****
// WRITE SINGLE COIL
// ****
// Gestione comando Modbus "Write Single Coil".
//
// Parametri funzione:
// $UnitID: Indirizzo di nodo
// $Address: Indirizzo coil
// $Value: Valore da scrivere
//
// La funzione ritorna il dato ricevuto. "false" se errore.
// ----

function WriteSingleCoil($UnitID, $Address, $Value)
{
    // Controllo se socket connesso. Se non connesso non riporto status
    // per mantenere indicazioni riportate in apertura.

    if (!$this->IsConnect) return(false);
    $this->Status="WriteSingleCoil: START";

    // Invio pacchetto interrogazione.

    $TPacket='';
    $TPacket.=$this->TxINT(rand(0,65000)); //Transaction ID
    $TPacket.=$this->TxINT(0); //Protocol ID
    $TPacket.=$this->TxINT(5+1); //Lenght
    $TPacket.=$this->TxBYTE($UnitID); //Unit ID
    $TPacket.=$this->TxBYTE(5); //Function code
    $TPacket.=$this->TxINT($Address-1); //Coil address

    // Trasmissione valore coil.

    if (!$Value) $TPacket.=$this->TxINT(0x000); else $TPacket.=$this->TxINT(0xFF00);
    $this->Send($TPacket); //Trasmissione pacchetto

    // Attesa pacchetto di risposta.

    $RPacket=$this->Receive(); //Received packet
    if ($RPacket === false) { $RData=false; goto FUNCTION_END; }

    // Controllo codice di risposta e converto pacchetto in array binario.

    if (!$this->ResponseCode($TPacket, $RPacket)) { $RData=false; goto FUNCTION_END; }
    $RData=true; //Ok esecuzione

    // Termino comando ed esco.

    FUNCTION_END:
    $this->Status.=- WriteMultipleRegisters: DONE";
    return($RData);
}

// ****
// WRITE MULTIPLE REGISTERS
// ****
// Gestione comando Modbus "Write Holding Registers".
//

```

```

// Parametri funzione:
// $UnitID: Indirizzo di nodo
// $Address: Indirizzo registri
// $Registers: Numero registri da leggere
//
// La funzione ritorna "false" se errore, "true" se Ok.
// -----
// Controllo se socket connesso. Se non connesso non riporto status
// per mantenere indicazioni riportate in apertura.

if (!$this->IsConnect) return(false);
$this->Status="WriteMultipleRegisters: START";

// Invio pacchetto interrogazione.

$TPacket='';
$TPacket.=$this->TxINT(rand(0,65000)); //Transaction ID
$TPacket.=$this->TxINT(0); //Protocol ID
$TPacket.=$this->TxINT(($Registers*2)+7); //Length
$TPacket.=$this->TxBYTE($UnitID); //Unit ID
$TPacket.=$this->TxBYTE(16); //Function code
$TPacket.=$this->TxINT($Address-1); //Register address
$TPacket.=$this->TxINT($Registers); //Nr of registers
$TPacket.=$this->TxBYTE(($Registers*2)); //Byte count
$TPacket.=$TData; //Dati pacchetto
$this->Send($TPacket); //Trasmissione pacchetto

// Attesa pacchetto di risposta.

$RPacket=$this->Receive(); //Received packet
if ($RPacket === false) {$RData=false; goto FUNCTION_END;}

// Controllo codice di risposta e converto pacchetto in array binario.

if (!$this->ResponseCode($TPacket, $RPacket)) {$RData=false; goto FUNCTION_END;}
$RData=true; //Ok esecuzione

// Termino comando ed esco.

FUNCTION_END:
$this->Status.=- WriteMultipleRegisters: DONE";
return($RData);
}

// #####
// FUNZIONI PRIVATE
// #####
// *****

// INVIO PACCHETTO DATI
// *****
// Gestisce l'invio del pacchetto dati.
//
// Parametri funzione:
// $Packet: Pacchetto dati da inviare

```

```
//  
// La funzione non ha ritorni.  
// -----  
  
private function Send($Packet)  
{  
    if (!$this->IsConnect) return;  
    $this->Status.= " - Tx Packet: ".$this->PacketPrint($Packet);  
    socket_write($this->Sock, $Packet, strlen($Packet));  
}  
  
// *****  
// RICEZIONE PACCHETTO DATI  
// *****  
// Gestisce la ricezione del pacchetto dati.  
//  
// La funzione non ha parametri.  
// La funzione ritorna il pacchetto ricevuto. "false" se errore.  
// -----  
  
private function Receive()  
{  
    if (!$this->IsConnect) return;  
    socket_set_nonblock($this->Sock);  
    $Readsocks[] = $this->Sock;  
    $Writesocks = NULL;  
    $Exceptsocks = NULL;  
    $Receive = "";  
    $LastAccess = time();  
    $this->Status.= " - Receive data";  
  
    // The sockets listed in the $Readsocks array will be watched to see if  
    // characters become available for reading.  
  
    while (socket_select($Readsocks, $Writesocks, $Exceptsocks, 0, 300000) != false)  
    {  
        if (in_array($this->Sock, $Readsocks))  
        {  
            while (@socket_recv($this->Sock, $Receive, 2000, 0)) {$this->Status.= " - Rx  
            Packet: ".$this->PacketPrint($Receive); return($Receive);}  
            $LastAccess = time();  
        }  
        else  
        {  
            if ((time() - $LastAccess) >= $this->Timeout)  
            {  
                $this->Status.= " - Watchdog [ ".$this->Timeout." sec] to ".$this->Host;  
                return(false);  
            }  
        }  
        $Readsocks[] = $this->Sock;  
    }  
}  
  
// *****  
// CONTROLLO CODICE DI RISPOSTA  
// *****  
// Controlla codice di risposta. Viene controllato l'header del pacchetto.
```

```
// Transaction ID, Protocol ID, Length, Unit ID, Function code
//
// Parametri funzione:
// $TPacket: Pacchetto dati inviato
// $RPacket: Pacchetto dati ricevuto
//
// La funzione ritorna, false: Pacchetto in errore, true: Pacchetto Ok.
// -----
//

private function ResponseCode($TPacket, $RPacket)
{
    // Eseguo controllo se header pacchetto corretto.

    if (self::RxUINT($RPacket, 0) != self::RxUINT($TPacket, 0)) {$this->Status.= " - Transaction ID error"; return(false);}
    if (self::RxUINT($RPacket, 2) != 0) {$this->Status.= " - Protocol ID"; return(false);}
    if (self::RxBYTE($RPacket, 6) != self::RxBYTE($RPacket, 6)) {$this->Status.= " - Unit ID"; return(false);}

    if ((ord($RPacket[7]) & 0x80) > 0)
    {
        // Failure code

        $FCode=ord($RPacket[8]); //Failure code

        // failure code strings.

        $Failures = array(
            0x01 => "ILLEGAL FUNCTION",
            0x02 => "ILLEGAL DATA ADDRESS",
            0x03 => "ILLEGAL DATA VALUE",
            0x04 => "SLAVE DEVICE FAILURE",
            0x05 => "ACKNOWLEDGE",
            0x06 => "SLAVE DEVICE BUSY",
            0x08 => "MEMORY PARITY ERROR",
            0x0A => "GATEWAY PATH UNAVAILABLE",
            0x0B => "GATEWAY TARGET DEVICE FAILED TO RESPOND");

        // Get failure string.

        if (key_exists($FCode, $Failures)) {$this->Status.= " - Modbus error code: $FCode ($Failures[$FCode])"; return(false);}
        $this->Status.= " - Modbus error code: UNDEFINED FAILURE CODE";
        return(false);
    }
    return(true);
}

// ****
// STAMPA PACCHETTO DATI
// ****
// Stampa un pacchetto dati in esadecimale.
//
// Parametri funzione:
// $Packet: Pacchetto dati da stampare
//
// La funzione ritorna stringa dati.
// -----
```

```
private function PacketPrint($Packet)
{
    $String = " [ ";
    for ($i=0; $i<strlen($Packet); $i++)
    {
        $String .= dechex((ord($Packet[$i]) >> 4)&0x0F);
        $String .= dechex(ord($Packet[$i])&0x0F);
        if ($i<strlen($Packet)-1) $String .= " ";
    }
    $String .= "]";
    return($String);
}

// *****
// GESTIONE ENDIANNESSE
// *****
// Funzioni per la gestione dell'endianness dei dati.
// TxEndian: Dato il valore, ritorna stringa da inserire in packet.
// ----

private function TxEndian($Value)
{
    if (!$this->Endianness) return(self::TxBYTE($Value >> 8).self::TxBYTE($Value).self::
    TxBYTE($Value >> 24).self::TxBYTE($Value >> 16));
    return(self::TxBYTE($Value >> 24).self::TxBYTE($Value >> 16).self::TxBYTE($Value >> 8)
    .self::TxBYTE($Value));
}

private function RxEndian($RData)
{
    if (!$this->Endianness) return(((ord($RData[2])*256)+ord($RData[3]))*65536)+((ord(
    $RData[0])*256)+ord($RData[1]));
    return(((ord($RData[0])*256)+ord($RData[1]))*65536)+((ord($RData[2])*256)+ord($RData[
    3]));
}
```