Einfach möglich

von Christopher Ernestus

Serielle Geräte ins Web bringen

Ob nun der örtliche Tourismus-Verband die aktuellen Wetterdaten weltweit präsentieren oder das Fertigungsunternehmen den Status einer Anlage per „Web-Based Management" überwachen will: Manchmal müssen vorhandene Geräte, die von Haus aus nicht einmal einen Netzwerkanschluss haben, mit einer Weboberfläche ausgestattet und im Internet/Intranet verfügbar gemacht werden. Das scheint auf den ersten Blick schwieriger, als es ist: Eine kostengünstige kleine Box, ein PHP-Skript und der Zugang zu einem PHP-fähigen Webserver genügen, wie hier am Beispiel einer Wetterstation gezeigt wird.

Für den Webentwickler tun sich ganz neue Möglichkeiten durch das Vordringen des „Web-Based Management" in allen Bereichen der Technik auf, aber auch mit der Anbindung externer Geräte etwa zu Werbezwecken, die über die klassische Homepage-Erstellung und die datenbankorientierte Webentwicklung hinausgehen.

Nicht zuletzt bei der Automatisierung in der Industrie wächst der Wunsch nach Weboberflächen, denn die Vorteile des Web-Based Management werden hier zunehmend erkannt: So kann etwa während der Nachtschicht ein Service-Techniker von irgendwo in der Welt mit dem Browser, ohne spezielle Software auf seinem PC, den Zustand einer Anlage diagnostizieren. Durch übersichtliche, grafisch aufbereitete Seiten kann er im Notfall schnell die Situation erfassen und die richtigen Maßnahmen treffen.

Oft werden mit solchen web-basierenden Lösungen alte proprietäre Programme abgelöst, die direkt über eine lokale serielle Schnittstelle mit dem Gerät oder der Maschine kommunizierten. Außerhalb der schnelllebigen Web- und IT-Welt trifft man dabei jedoch oft auf Geräte, die durch ihre hohen Anschaffungskosten noch eine Lebens- und Abschreibungsdauer von vielen Jahren, manchmal Jahrzehnten vor sich haben. Solche Geräte kennen häufig noch keinen Ethernet-Anschluss, sondern nur serielle Schnittstellen wie RS-232, RS-422, RS-485 oder 20mA.

Der Anschluss eines solchen seriellen Gerätes an das Ethernet ist mit einem speziellen Interface möglich, etwa dem „ComServer" von Wiesemann & Theis (W & T) für weniger als 300 Euro. Dieser ist kaum größer als eine Zigarettenschachtel und besitzt nach außen je einen Anschluss für Ethernet (RJ45) und RS-232 (9-polig, umschaltbar auch für RS-422 und RS-485). Schon nach Eingabe der seriellen Übertragungsparameter wie Baud-Rate usw. sowie der IP-Adresse, Subnet Mask etc. und dem Stecken der Verbindungskabel ist eine transparente Netzverbindung zwischen Webserver und seriellem Gerät möglich.

Um das Prinzip möglichst verständlich darzustellen, wurde als Beispiel eine Wetterstation mit serieller Schnittstelle gewählt. Da in der Webseite dynamische Inhalte dargestellt werden, die nur auf der Serverseite und nicht auf der Clientseite verfügbar sind, liegt ein typischer Anwendungsfall für eine serverseitige Skriptsprache wie PHP vor. Nur wird hier nicht (wie wohl sonst in den meisten Fällen) eine Datenbankanfrage gestartet, sondern ein externes Gerät angesprochen. Prinzipiell wäre das Gleiche auch mit anderen Skriptsprachen und -umgebungen möglich (ASP, JSP, ColdFusion oder CGI/PERL).

Das PHP-Skript, das hier als Beispiel dient und zum Download zur Verfügung steht (www.wut.de/ unter ComServer → Applikationen), ist sehr umfangreich kommentiert, sodass sich an dieser Stelle eine Erläuterung des Sourcecodes im Einzelnen erübrigt. Der grundsätzliche Ablauf ist in Abbildung 2 dargestellt: Der Anwender ruft mit dem Browser eine bestimmte, auf dem Server abgelegte Seite (hier: *wetter.php*) auf (1), die ihm die aktuellen Wetterdaten in einer vom Webentwickler vorbereiteten Form darstellen soll. Da es sich bei der Webseite um eine PHP-Datei handelt, übergibt der Webserver diese an den PHP-Interpreter (2), der die PHP-Anteile abarbeitet. Mit den im Sprachumfang von PHP enthaltenen Funktionen zur Socket-Behandlung baut der PHP-Interpreter nun eine Verbindung zum ComServer (3) auf und fragt darüber die aktuellen Wetterdaten ab.

Die Kommunikation zwischen ComServer und Wetterstation über die serielle Schnittstelle (4 und 5) ist für das PHP-Skript nicht sichtbar. Sie wird vom ComServer völlig autark behandelt. Das PHP-Skript „sieht" nur die IP-Adresse des ComServers, dessen Antwort (6) nun über die Socket-Verbindung eingelesen wird. Im PHP-Skript wird nun die notwendige Fehlerbehandlung durchgeführt. Bei erfolgreicher Verbindung wird der ankommende Byte-Strom in einem String abgelegt. In unserem Beispiel werden die einzelnen ASCII-Zeichen des Strings nacheinander als 8-Bit-Binärwerte ausgelesen, interpretiert und umgerechnet. In einigen Fällen werden aus

Info

Der Quellcode zum Artikel kann von www.wut.de heruntergeladen werden.

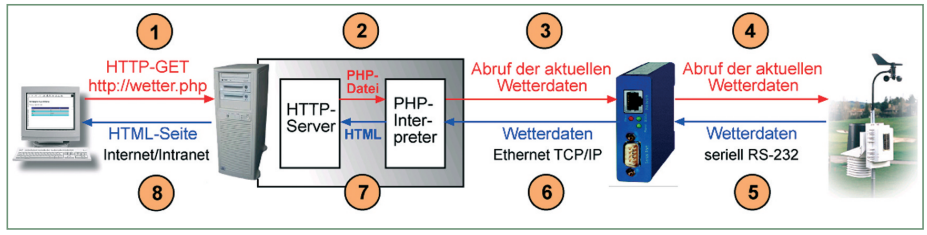

Abb. 1: Wetterstation logisch

Listing 1

```
<?php
 // Abfrage der Wetterstation "Davis Monitor II", angeschlossen an Com-Server
172.16.232.98.

 // Max. Lebensdauer im Browser-Cache in Sekunden festlegen. 5 Minuten sind sinnvoll, da
 // sich in dieser Zeit das Wetter nicht entscheidend ändert. Schlecht wäre hingegen nach
 // Aufruf des Lesezeichens im Browser als erstes Daten vom Vorabend zu bekommen.
 header( "Cache-Control: max-age=300" );

 // In diesem ersten PHP-Block werden die Wetterdaten im rohen Binärformat und mit
 // Zeitstempel über eine Socketverbindung und den Com-Server aus der Wetterstation
 // gelesen. Falls es dabei Probleme gibt, wird stattdessen auf Daten aus einem Binär-Cache
 // zurückgegriffen und ein entsprechender Fehlerstring ausgefüllt.

 // Der Pfadname der Binär-Cachedatei, muß in einem Verzeichnis liegen, für das der
 // HTTP-Server Schreibrechte hat.
 $strCacheFile = "/var/lib/wwwrun/weather.bin";

 // IP-Adresse des Com-Servers
 $IP = "172.16.232.98";

 // Initialisierung. Fehlermeldung leer" bedeutet "OK"
 $strStatus = "";
 $binData = "";

 // Öffnen der TCP-Verbindung mit 2 Sekunden Timeout zum Com-Server auf Port 8000
 $mySocket = @fsockopen( $IP, 8000, &$errno, &$errstr, 2 );

 // Zweck der folgenden Aktion ist, im Fall „Connection refused *und* sehr alte Cache-
 // Inhalt" unserem vermuteten Konkurrenten wenigstens noch eine Chance zu geben, den
 // Cache für  uns zu aktualisieren.
 if( $errno == 111 and (time() - @filemtime( $strCacheFile ) > 10) )
  sleep( 1 );

 if( $mySocket )
  {
  // Daten senden (1 Block Wetterdaten anfordern)
  @fputs( $mySocket, "LOOP\xff\xff\r" );
  // Es sollte mindestens ein Zeichen Antwort empfangen werden
  socket_set_timeout( $mySocket, 2 );
  $c = fgetc( $mySocket );
  if( !$c )
  {
  $strStatus = „Wetterstation sendet keine Daten.";
  // Den Com-Server resetten, da evtl. gar kein Endgerät angeschlossen war, und es
  // dann mehrere Minuten lang noch weiter versuchen würde, die Daten auszuliefern.
  // Reset des Com-Servers durch  Ansprechen von Port 9084.
  $dummySocket = @fsockopen( $IP, 9084 );
  @fclose( $dummySocket );
  }
  else if( ord( $c ) == 6 )
  {
  // Lesen der 18 Byte langen Binärantwort Wetterstation über die Socket-Vebindung
  $binData = fread( $mySocket, 18 );
  $timeStamp = time();
  // und diese in der Cache-Datei speichern
  $myFile = @fopen( $strCacheFile, "wb" );
  @fwrite( $myFile, $binData, 18 );
  @fclose( $myFile );
  }
  else
  $strStatus = sprintf( "Unerwartetes Quittungsbyte 0x%02x.", ord( $c ) );
  // Schließen der Socket-Verbindung - würde sonst auch am Ende des Scripts
  // automatisch erfolgen.
  fclose( $mySocket );
  }

 // Falls keine Daten empfangen wurden, diese der Cache-Datei entnehmen.
 if( empty( $binData ) )
  {
  $myFile = @fopen( $strCacheFile, "rb" );
  $binData = @fread( $myFile, 18 );
  @fclose( $myFile );
  $timeStamp = @filemtime( $strCacheFile );
  }

 if( !$mySocket )
  {
  // Verbindung fehlgeschlagen, typische Fehlercodes sind dabei 111 für RST (=Connection
  // refused) und 110 für Timeout. "Connection Refused" melden wir nur, wenn außerdem
  // die Daten der Cache-Datei ungewöhnlich alt sind, d. h. wenn der Com-Server
  // anscheinend dauerhaft von jemand anders belegt ist.
  if( $errno != 111 or time() - $timeStamp > 15 )
   $strStatus = "TCP-Verbindung zur Wetterstation fehlgeschlagen ($errstr).";
  }
 //Erster statischer HTML-Teil für die Seitendarstellung
?>
<HTML>
 <HEAD>
  <TITLE>Davis Weather Monitor II</TITLE>
  <META http-equiv="Refresh" content="60">
 </HEAD>

 <BODY bgcolor="maroon">
  <TABLE align="center" width="90%" cellspacing="10" cellpadding="10">
   <TR>
    <TD bgcolor="white">
     <H2>W&T Firmengebäude Wuppertal-Oberbarmen</H2>
     Wetterdaten vom <?php setlocale( LC_TIME, „de_DE" );
     echo strftime( "%d. %b %Y, %X (%Z)", $timeStamp ); ?>

     <BR>
     <BR>
     <TABLE width="90%" align="center">

      <TR bgcolor="navy">
       <TD width="50%"><FONT color="white">Messwert</FONT></TD>
       <TD width="25%"><FONT color="white">innen</FONT></TD>
       <TD width="25%"><FONT color="white">außen</FONT></TD>
      </TR>

      <TR bgcolor="silver">
       <TD>Temperatur</TD>
       <?php
        // In den folgenden drei PHP-Blöcken werden aus den Binärdaten der Wetter-
        // station die Temperatur- Feuchtigkeits-, Luftdruck und Windwerte extrahiert und
        // als HTML-Code eingefügt. Dabei werden die als ASCII-Zeichen in einem String
        // vorliegen den Wert  einzeln als 8-Bit-Binärwerte interpretiert und weiter
        // verarbeitet.Byte 13-18 der Binärdaten werden in dieser Anwendung ignoriert.
        // Diese Teile sind spezifisch für die verwendete Wetterstation und dienen als
        // Beispiel Byte 1+2 sowie Byte 3+4 auswerten, zu 16-Bit-Binärwerten zusammen-
        // setzen und in Celsius-Temperaturen umrechnen und auf HTML-Seite darstellen.
        for( $i = 1; $i <= 3; $i += 2 )
         {
         $val = ord($binData{$i}) + (ord($binData{$i+1}) << 8);
         $val = ($val - 320)/18.0;
         printf( " <TD>%.1f °C</TD>\n", $val      );
         }
       ?>
      </TR>

      <TR bgcolor="aqua">
       <TD>Luftfeuchtigkeit</TD>
```

Listing 1 – Fortsetzung

```
        <?php
        // Byte 10 und 11 auswerten und als 2 Luftfeuchtigkeiten auf HTML-Seite <
        // darstellen falls Sensoren angeschlossen
        for( $i = 10; $i <= 11; $i++ )
         {
          $val = ord($binData{$i});
          if( $val == 0x80 )
           $val = "-"; // kein Sensor angeschlossen
          else
           $val .= " %";
          echo " <TD>$val</TD>\n";
         }
        ?>
       </TR>

       <TR bgcolor="silver">
        <TD>Wind</TD>
        <TD colspan="2">
        <?php
         // Byte 5 auswerten, in km/h umrechnen und als Windgeschwindigkeit auf HTML-
         // Seite ausgeben Byte 6+7 auswerten, zu 16-Bit-Binärwerten zusammensetzen
         // und aus dieser Winkelangabe eine Windrichtung ermitteln und in HTML darstellen
         $val = ord($binData{5}) * 1.609;
         printf( "%.1f km/h", $val );
         $val = ord($binData{6}) + (ord($binData{7}) << 8);
         $val = intval($val / 22.5 + 0.5) % 16;
         $kompass = array( "N", "NNO", "NO", "ONO",
              "O", "OSO", "SO", "SSO",
              "S", "SSW", "SW", "WSW",
              "W", "WNW", "NW", "NNW" );
         echo „ aus ${kompass[ $val ]}</TD>\n";
        ?>
       </TR>

       <TR bgcolor="aqua">
        <TD>Luftdruck</TD>
        <TD colspan="2">
        <?php
         // Byte 8+9 auswerten, zu 16-Bit-Binärwerten zusammensetzen, in
         // mbar-Druck umrechnen und auf HTML-Seite darstellen.
         $val = ord($binData{8}) + (ord($binData{9}) << 8);
         $val = intval( $val / 29.5 );
         echo "$val mbar</TD>\n";
        ?>
       </TR>

      </TABLE>
      <BR>
     </TD>
    </TR>

   <?php
// Eventuelle Fehlermeldungen der Station werden auf der HTML-Seite dargestellt:
    if( !empty( $strStatus ) )
     {
     ?>
     <TR bgcolor="red">
     <TD><FONT color="white">
     <?php echo $strStatus; ?>
     </FONT></TD>
     </TR>
    <?php } ?>

  </TABLE>
 </BODY>
</HTML>
```

Abb. 2: Wetterstation physikalisch

je zwei Byte ein 16-Bit-Integer-Wert generiert. Außerdem sind hier angelsächsische Maßeinheiten wie „Meilen pro Stunde" in metrische Maßeinheiten umzurechnen. Dieser Teil ist natürlich völlig spezifisch für das jeweilige Gerät und muss an die eigenen Anforderungen angepasst werden. Mit dem Beispielskript dürfte jedoch das Prinzip so weit deutlich werden, dass die Anpassung an andere Geräte kein Problem darstellen sollte. Vorausgesetzt sind natürlich PHP-Grundkenntnisse, ein Handbuch des seriellen Gerätes und die Möglichkeit, die Kommunikation mit dem Gerät zu testen.

Der PHP-Interpreter generiert nun (7) aus den statischen HTML-Anteilen, die die PHP-Seite bereits enthält, und den dynamisch ermittelten Werten eine vollständige HTML-Seite, die (8) der Webserver an den Browser zurückliefert.

Selbstverständlich kann zusätzlich der PHP-Interpreter die Wetterdaten z.B. in eine Datenbank auf dem Server schreiben. Der Hersteller des ComServers hat auch deshalb bewusst darauf verzichtet, eine entsprechende Programmiermöglichkeit innerhalb der Box vorzusehen. Innerhalb der Box bestünde nämlich nicht nur das Problem begrenzter Ressourcen, sondern es müsste dem Anwender eine eigene Entwicklungsumgebung zur Verfügung gestellt werden. Bei der hier skizzierten Lösung kann dagegen der Anwender in seiner gewohnten Entwicklungsumgebung arbeiten und muss sich bei seiner Programmierung nicht mit den Eigenschaften der Box auseinander setzen.

Ein Webserver bzw. ein PHP-Skript kann außerdem gleich mehrere ComServer ansteuern und deren Daten bündeln und gemeinsam verarbeiten. Dies hat auch Sicherheitsvorteile: Nur der ohnehin erreichbare Webserver muss von außen aus dem Internet erreicht werden können. Es sind also keine zusätzlichen Freischaltungen in der äußeren Firewall notwendig.

Da die Socket-Verbindungen auch über WAN-Strecken funktionieren, könnten mit der hier beschriebenen Methode sogar weltweit verteilte, serielle Wetterstationen über ein einziges PHP-Skript auf einem Webserver abgefragt und ihre Messwerte visualisiert werden.