# Telemetria M2M

# Wykorzystanie protokołu http w GSM

*W poprzednich artykułach opisaliśmy demonstracyjne urządzenie z modułem typu WAVECOM Q2686. W pierwszym artykule pokazaliśmy, jak za pomocą prostej aplikacji w OpenAT sterować przekaźnikami. W kolejnym cyklu przedstawiono, jak wykorzystując potencjał dostępnych w OpenAT bibliotek by zbudować prosty, internetowy system telemetryczny. Wykorzystaliśmy protokół FTP do wysyłania zmierzonych danych. Teraz przedstawiamy bardziej zaawansowany projekt – będzie to ewolucja poprzedniego programu. Tym razem dane pomiarowe prześlemy bezpośrednio na stronę Web i zapiszemy w bazie danych SQL.*

## Aplikacja WEB – co to takiego?

Dynamiczny wzrost popularności Internetu w ostatnich czasach, stymulowany był przez serwisy WWW. W pierwotnym wydaniu, strony internetowe oferowałyby statyczną zawartość. Raz napisany kod strony nie był modyfikowany, lecz bezdusznie przesyłany do przeglądarki. Powstał więc język opisu strony, czyli HTML, oraz protokół wymiany danych, pomiędzy serwerem a klientem (przeglądarką) , czyli HTTP. Standardowe zapytanie, oprócz ścieżki do pliku oraz nazwy żądanego pliku, podczas rozwoju protokołu HTTP rozszerzono o możliwość przekazywania zmiennych. Równolegle rozwijano technikę dynamicznego generowania stron., Powstały pierwsze języki skryptowe, jak PHP czy Perl. Połączenie tych dwóch technologii pozwoliło na stworzenie nowej jakości. Strony nie były już statyczne, można było przekazywać dane i wpływać (w ten sposób) na generowaną stronę. Świat IT szybko zauważył nowe zastosowanie dla tego duetu. W wielkim skrócie: skrypt po stronie serwera, przeglądarka po stronie Klienta, pozwoliły na tworzenie graficznych aplikacji a wymianę danych oprzeć na protokole HTTP. Rezerwacja biletu lotniczego, kupowanie odkurzacza w sklepie internetowych czy pisanie wiadomości na forum internetowym to nic innego jak aplikacja Web. Uzupełnieniem tego duetu jest baza danych SQL po stronie serwera WWW, przechowująca zawartość strony jak i dane wprowadzane przez użytkownika.

## Protokół HTTP dla opornych

Zasada działania protokołu HTTP polega na prostym zapytaniu (przeglądarka) i odpowiedzi (serwer). Przejrzysty opis działania protokołu, można znaleźć na stronie http://www.jmarshall.com/easy/http/. W skrócie, wygląda to tak:

– Klient łączy się protokołem TCP/IP z serwerem (pod zadanym adresem), standardowo przez port 80.
– Gdy połączenie zostanie zestawione, klient wysyła zapytanie. Na zapytanie składa się linijka tekstu zakończona znakiem 0x0D (Enter). Koniec zapytania sygnalizuje pusta linia (dwa znaki 0x0D). Oto przykładowe zapytanie metodą GET, w którym żądamy podania pliku *index.html* z głównego katalogu danej strony. Nazwa Klienta (przeglądarki) to HTTPTool. Pusta linia w punkcie 3 oznacza koniec zapytania.

```
GET /index.html HTTP/1.0
User-Agent: HTTPTool/1.0
[pusta linia]
```

– Serwer interpretuje zapytanie i generuje odpowiedź. Ważną informacją w odpowiedzi serwera jest kod odpowiedzi. Jeśli serwer znajdzie żądany plik, zwraca kod 200. Dodatkowo informuje o typie odpowiedzi (tu jest to plik tekstowy w formacie HTML) oraz zwraca wielkość (w bajtach) przekazywanego pliku. Standardowo będzie to plik tekstowy w języku HTML kodu strony.

```
HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 1354
<html>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

Koniec transakcji wiąże się z rozłączeniem połączenia. Powyższy opis jest prawdziwy dla protokołu HTTP w wersji 1.0. Większość obecnie działających serwisów używa wersji 1.1. Nie wdając się w szczegóły, nowsza wersja umożliwia między innymi utrzymanie połączenia i wykorzystanie go dla kolejnej transakcji. Uzyskuje się przez to znaczący wzrost wydajności i przyśpiesza wyświetlanie strony, zmniejszając obciążenie serwera.

Ktoś może zapytać – a jak przekazywane są pliki graficzny czy inne pliki binarne? Odpowiedź jest prosta – tak samo. Zmianie ulega w odpowiedzi serwera nagłówek Content-Type, wskazujący na typ pliku. Dla plików JPG będzie to image/jpeg.

## Aplikacja OpenAT

W niniejszym artykule, wykorzystano większość kodu aplikacji, zaprezentowanej w poprzednim odcinku cyklu. Omówione zostaną tylko nowe elementy. Zacznijmy od omówienia nowych zmiennych globalnych. Przedstawiono je na **list. 1**.

Zmienna SampleList jest identyfikatorem listy jednokierunkowej, za pomocą której zaimplementowano mechanizm buforowania. Struktura http_ClientCtx zawiera szereg zmiennych, używanych przez blok program, odpowiedzialnego za połączenie HTTP. Jest to po kolei:

– uchwyt to kanału sesji (analogicznie, jak to było dla FTP);
– uchwytu kanału połączenia TCP/IP;
– bufor na odpowiedź serwera;

**List. 1. Nowe typy zmiennych**

```
typedef struct {
  wip_channel_t CnxChannel;       // session channel
  wip_channel_t DataChannel;      // data channel
  u8            RxDataBuf[512];   // response data buffer
  u32           RxDataLength;     // response data length
  s32           HttpStatus;       // last received page status code
  u8            State;            // WIP http client state
} http_ClientCtx_t;

http_ClientCtx_t http_ClientCtx;
wm_lst_t         SampleList;
```

**List. 2. Funkcja odczytująca dane pomiarowe i formująca zapytanie**

```
//------------------------------------------------ odpowiedź na polecenie: adc
s16 evh_ADC_Response(adl_atResponse_t *paras){  // paras - komunikat
   ascii *s, tmp[50];
   u16 idx;
   u16 vbat,tbat,temp,i;
   s=wm_strtok(paras->StrData,"\r\n");   // odbrany pomiar z komendy ADC
   if(wm_strcmp(s,"OK")) {
       s=wm_strtok(s, „+ADC: „);
       vbat=wm_atoi(s);
       s=strchr(s+1,',');                // pomiary oddzielone przecinkami
       tbat=wm_atoi(s+1);
       s=strchr(s+1,',');
       temp=wm_atoi(s+1);
       wm_sprintf(tmp, „?lt=%lu&vb=%u&tb=%u&t=%u",
                                  (u32)(TimeStamp.TimeStamp),vbat,tbat,temp);
       wip_debug( „Sample UBN: \"%s\"\r\n", tmp);
       s = adl_memGet( (u16)(wm_strlen(tmp)+1) );
       wm_strcpy( s, tmp );
       wm_lstAddItem ( SampleList, s );         //Add string to list
       idx = wm_lstGetCount( SampleList );
       wip_debug( "Samples list items: %u\r\n", idx);
       if (idx > MAX_LIST_ITEMS) {              // overflow protection
          wm_lstDeleteItem( SampleList, 0 );    // by deleting oldest one
          wip_debug( "Samples list overflow protection.\r\n");
       }
   }
   return TRUE;
}
```

**List. 3. Główna funkcja sterująca**

```
//---------------------------------------- 5 sek. timer aplikacji (cykliczny)
void evh_App_Timer ( u8 ID )  {                  // ID - timer nr ID

  if ( wm_lstGetCount( SampleList ) > 0 )
  {
      if ((http_ClientCtx.State == WIP_CEV_PEER_CLOSE) ||
          (http_ClientCtx.State == WIP_CEV_ERROR))
      {
          http_ClientCtx.State = WIP_CEV_DONE;
          if ( http_ClientCtx.HttpStatus == 200) {
               wip_debug("HTTP Transaction completed.\r\n");
               wm_lstDeleteItem( SampleList, 0);  // usuwamy wysłane dane
               http_ClientCtx.HttpStatus = 0;
          }
      } else
      if ( http_ClientCtx.State == WIP_CEV_DONE ) {
          if ( wm_lstGetCount( SampleList ) > 0 )
              http_ClientConnect();              // wyslij dane via http
      }
  }
}
```

**List. 4. Funkcja inicjująca wymianę danych protokołem HTTP**

```
//------------------------------------------------------------------ http_Client
static s32 http_ClientConnect(void) {
  s32 ret = 0;
  static ascii uri[200], *tmp;

  if (GPRS_Status != WIP_BEV_IP_CONNECTED) return -2;

  http_ClientCtx.State = WIP_CEV_LAST + 1;      // a little trick
  http_ClientCtx.HttpStatus = 0;
  // HTTP Session creation
  http_ClientCtx.CnxChannel = wip_HTTPClientCreateOpts(
                   NULL,                        // no handler
                   NULL,                        // no context
                   // default headers
                   WIP_COPT_HTTP_HEADER, "User-Agent", "WIPHTTP/1.0",
                   WIP_COPT_END );
  if (http_ClientCtx.CnxChannel == NULL) {
        wip_debug ( "Cannot create http session channel. Fatal error!\r\n" );
        ret = -1;
  } else {
    //
    tmp = wm_lstGetItem(SampleList, 0);
    wm_sprintf(uri, "%s%s", SERVER_URL, tmp);

    // HTTP GET command
    http_ClientCtx.DataChannel = wip_getFileOpts(
                   http_ClientCtx.CnxChannel,   // kanał sesji HTTP
                   uri,                         // żądany URL
                   evh_http_DataHandler,        // data handler
                   &http_ClientCtx,             // context
                   // request headers
                   WIP_COPT_HTTP_HEADER, "Accept","text/html",
                   WIP_COPT_HTTP_HEADER, "Accept-Language","en",
                   WIP_COPT_END );
    if (http_ClientCtx.DataChannel == NULL) {
            wip_debug ( "Cannot create http data channel\r\n" );
            wip_close( http_ClientCtx.CnxChannel);
            ret =-1;
    }
  }
  return(ret);
}
```

– zmienna określająca wielkość odpowiedzi w buforze;
– kod odpowiedzi http (patrz wyżej, opis protokołu), oraz
– stan aplikacji – prosty mechanizm automatu stanu.

Pierwszym nowym elementem, którym wyróżnia się w omawianej aplikacja OpenAT, jest bufor FIFO. Zbudowany jest on za pomocą dostępnego w funkcjach bibliotecznych mechanizmu listy jednokierunkowej. W funkcji adl_main znajduje się linijka inicjująca listę: SampleList = wm_lstCreate(WM_LIST_NONE, NULL);. Od tego momentu, do listy możemy dodawać elementy (funkcja: wm_lstAddItem), zliczać liczbę elementów listy (wm_lstGetCount) czy kasować dowolny element (wm_lstDeleteItem). Do funkcji dodającej element do listy przekazujemy wskaźnik na element. Wymaga to dynamicznej zaalokowania pamięci (heap). Usuwanie elementu polega na usunięciu dowiązania listy (patrz: zasada działania listy jednokierunkowej), a następnie na zwolnieniu pamięci. Stąd w poniższym kodzie nie ma jawnego wywołania funkcji free (wywołuje ją niejawnie funkcja usuwająca element z listy). Jako, że za pomocą listy implementujemy FIFO, pod pozycją 0 jest zawsze interesujący nas element (najstarszy). Zmodyfikowana wersja funkcji evh_ADC_Response, w stosunku do przykładu wykorzystującego technikę FTP, zamieszczono poniżej. Wykonany pomiar (AT+ADC) wywołuje tę funkcję. „Parsowane" są interesujące nas dane i formatowany jest składnik adresu URL (tu umieszczane są zmienne dla serwera) – linia 40. Przygotowane dane alokowane są na liście. Jeśli z jakiś powodów liczba elementów na liście przekroczy liczbę maksymalną, usuwane są najstarsze elementy. Zapobiega to przepełnieniu pamięci.

Sterowanie aplikacji zrealizowano za pomocą dodatkowego timera, wywołującego okresowo poniższą funkcję (inicjacja timera w funkcji adl_main). Jest to główna funkcja programu. **List. 2** przedstawiał, w jaki sposób dane pomiarowe są odczytywane i buforowane. **List. 3** pokazuje, jak i kiedy dane z bufora są przekazywane poprzez zapytanie http do serwera. Jeśli bufor zawiera jakieś elementy, a poprzednia transakcja zakończyła się – inicjowana jest nowa transakcja. Sukces wymiany danych owocuje usunięciem wysłanej porcji danych z listy. Jeśli wystąpi błąd, transmisja jest ponawiana, aż do skutku.

Na **list. 4** przedstawiono funkcje inicjującą transakcję HTTP. Dla uproszczenia, za każdym wywołaniem tworzona jest nowa sesja – funkcja wip_HTTPClientCreateOpts. Tę operację można wykonać raz, np. w funkcji adl_main. Z sesją nie jest związane żadne fizyczne połączenie TCP, jak to miało miejsce w przypadku protokołu FTP. Zapytanie metodą GET tworzone jest poprze wywołanie

**List. 5. Funkcja obsługująca połączenie TCP**

```
//-------------------------------------- Handler for the HTTP data channel
static void evh_http_DataHandler( wip_event_t *ev, void *ctx) {
                                  // ev - zdarzenie (event); ctx - nieużywane
  ascii tmpbuf[256];
  s32 status, len;
  http_ClientCtx_t *pHttpClientCtx = (http_ClientCtx_t*)ctx;

  pHttpClientCtx->State = ev->kind;
  switch(ev->kind)
  {
    case WIP_CEV_OPEN:
      wip_debug( „evh_http_DataHandler: Start\r\n” );
      pHttpClientCtx->RxDataLength  = 0;    // ready for getting response data
      wm_memset(pHttpClientCtx->RxDataBuf,0,sizeof(pHttpClientCtx-
>RxDataBuf));
      break;
    case WIP_CEV_READ:
      wip_debug( „evh_http_DataHandler: WIP_CEV_READ\r\n” );
      // we must read all available data to trigger WIP_CEV_READ again
      while( (len = wip_read( ev->channel, tmpbuf, sizeof(tmpbuf)-1)) > 0) {
           tmpbuf[len] = 0;
           wm_strcat(pHttpClientCtx->RxDataBuf, tmpbuf);
           pHttpClientCtx->RxDataLength += len; // compute length of response
      }
      break;
    case WIP_CEV_PEER_CLOSE:
      wip_debug (   „evh_http_DataHandler:WIP_CEV_PEER_CLOSE\r\n” );
      //  end of data: get some response information
      if( wip_getOpts( ev->channel, WIP_COPT_HTTP_STATUS_CODE, &status,
          WIP_COPT_END) == OK) {
               pHttpClientCtx->HttpStatus = status;
      }
      wip_close( ev->channel);                  // data channel must be closed
      wip_close( http_ClientCtx.CnxChannel);
      wip_debug( „HTTP StatusCode: %d\r\n”, status);
      wip_debug( „HTTP Response: %s\r\n”,pHttpClientCtx->RxDataBuf);
      break;
    case WIP_CEV_ERROR:
      wip_debug ( „evh_http_DataHandler: WIP_CEV_ERROR %d\r\n”,
                                          ev->content.error.errnum );
      wip_close( ev->channel);                  // connection to server broken
      wip_close( http_ClientCtx.CnxChannel);
      break;
    default:
      wip_debug (  „evh_http_DataHandler: unexpected event %d\r\n”, ev->kind
);
      break;
  }
}
```

funkcji wip_getFileOpts. Na liście parametrów do w/w funkcji znajdą się: uchwyt do sesji, adres URL ze zmiennymi (tworzony na podstawie adresu strony oraz porcji danych, pobranych z listy). Ponadto przekazywany jest adres funkcji kanału TCP (analogicznie, jak to było w przypadku FTP) oraz wskaźnik na tzw. kontekst. Jest to mechanizm umożliwiający identyfikację danej transakcji wewnątrz funkcji obsługi kanału TCP (tu jest to funkcja evh_http_DataHandler). Umożliwia to łatwe sterowanie połączeniem, jak również daje możliwość zestawienia kilku równoległych transakcji z serwerem.

Po nawiązaniu połączenia TCP/IP, wywołana zostanie funkcja evh_http_DataHandler z kodem zdarzenia WIP_CEV_OPEN oraz przekazany zostanie wskaźnik na właściwy kontekst. Jednocześnie klient wysyła stosowne nagłówki i zapytanie w stronę serwera. Gdy serwer odpowie na przesłane dane, funkcja prześle dane w stronę klienta, a funkcja evh_http_DataHandler zostanie wywołana z kodem WIP_CEV_READ. Umożliwia to odebranie danych i przekopiowanie ich do bufora skojarzonego z danym połączeniem – to jest właśnie rola kontekstu. Jeśli połączenie zakończy się poprawnie, kod zdarzenia zostanie ustawiony na WIP_CEV_PEER_CLOSE. W tym momencie można dokonać odczytu stosownych nagłówków odpowiedzi serwera (np. kod, typ zwróconego pliku, wielkość pliku, itp). Po szczegóły zapraszamy do dokumentacji biblioteki WIP. W przypadku wystąpienia błędu (np. rozłączenie lub podano błędny adres serwera) zgłoszony jest stosowny kod zdarzenia WIP_CEV_ERROR. Niezależnie czy sesja zakończyła się błędem czy poprawnie, stosowne kanały sesji oraz połączenia TCP są zamykane. Umożliwia to prawidłowe ponowne rozpoczęcie wymiany danych poprzez wywołanie funkcji http_ClientConnect (**list. 5**).

## Akwizycja – prosta aplikacja serwerowa w PHP

Najpopularniejszym obecnie językiem do budowy „inteligentnych” stron WWW jest język PHP. Jest to nowoczesny język programowania czwartej generacji. Programowanie w PHP jest bardzo zbliżone do programowania w C i każdy programista jest w stanie napisać prosty program już po kilku godzinach pracy. Na potrzeby naszego systemu telemetrycznego, skrypt będzie odbierał dane przekazywane w zapytaniu metodą GET. Tak odebrane dane są następnie zapisywane w prostej tabeli bazy danych. Szczegóły budowy tabeli omówiono w kolejnym rozdziale. Jeśli wszystko przebiegnie poprawnie, wygenerowany zostanie napis OK. W innym przypadku, zasygnalizowany zostanie błąd. Pozwala to na interakcje pomiędzy klientem a serwerem i umożliwia zweryfikować poprawność akwizycji przekazanych danych w tabeli (**list. 6**).

## SQL – jak efektywnie agregować dane pomiarowe

Aplikacja OpenAT jest w stanie dokonać pomiaru, zbuforować dane, a następnie przesłać je w formie zapytanie na stronę internetową. Aby dane móc gromadzić (np. celem dalszej obróbki), musimy użyć bazy danych. Tu z pomocą przychodzi wspomniana już technologia SQL, a dokładnie serwer MySQL. Poniżej przedstawiono skrypt tworzy tabelę, składającą się z 6 kolumn:

– klucz główny ID (tu będą to kolejne liczby całkowite),

**List. 6. Aplikacja Web serwera w języku PHP**

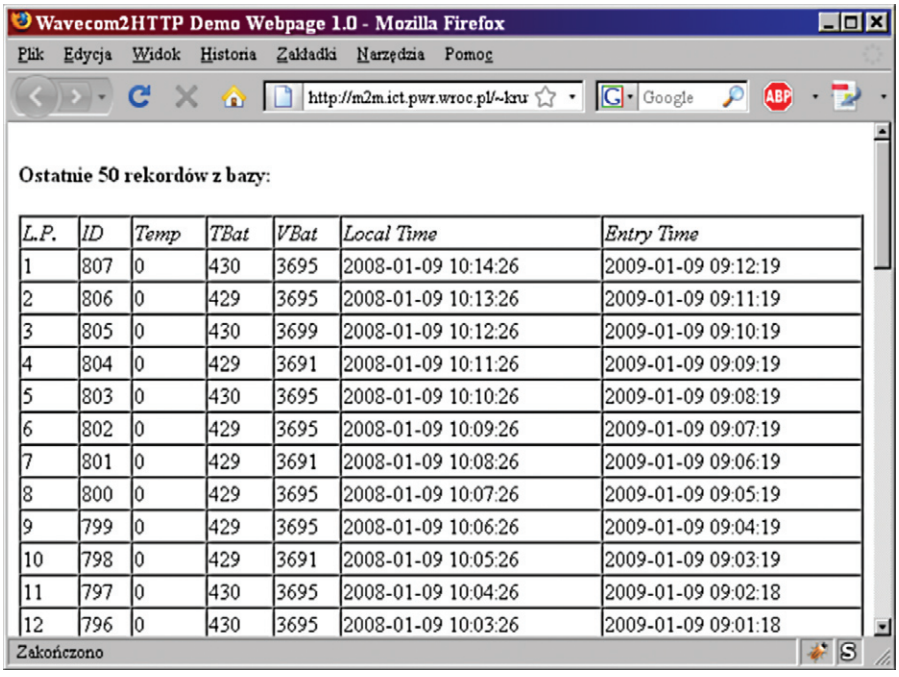
```
<?php
  if ( isset($_GET[,t’]) && isset($_GET[,tb’]) &&
                                  isset($_GET[,vb’]) &&    isset($_GET[,lt’]) )
  {
// mysql setup
$config[,db_server’]   = ,***.pwr.wroc.pl’;
$config[,db_username’] = ,***’;
$config[,db_password’] = ,***’;
$config[,db_schema’]   = ,***’;
// Connecting, selecting database
$mysqli = new mysqli($config[,db_server’], $config[,db_username’],
                                      $config[,db_password’],$config[,db_schema’] );
if (mysqli_connect_errno())
{
  echo ,Error #2’;
} else
{
  // Performing SQL query
  $query = „INSERT INTO logtab (temp, tbat, vbat, ltime)
                                              VALUES (?, ?, ?, FROM_
UNIXTIME(?));”;
    if ($stmt = $mysqli->prepare($query) )
    {
       $stmt->bind_param(‘dddd’, $_GET[‘t’], $_GET[‘tb’], $_GET[‘vb’],
$_GET[‘lt’]);
       $stmt->execute();
       $stmt->close();
     }
     // Closing connection
     mysqli_close($mysqli );
     echo ‘OK ‘;
    }
  } else
  {
    echo ‘Error #1’;
  }
?>
```

**List. 7. Kod SQL przykładowej tabeli bazy danych**

```
DROP TABLE IF EXISTS `logtab`;
CREATE TABLE  `logtab` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `temp` int(10) default '0',
  `tbat` int(10) default '0',
  `vbat` int(10) default '0',
  `ltime` timestamp NOT NULL default '0000-00-00 00:00:00',
  `etime` timestamp NOT NULL default
            CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,  PRIMARY KEY
(`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

**List. 8. Aplikacja w PHP prezentująca dane z bazy SQL na stronie WWW**

```
<head>
<title>Wavecom2HTTP Demo Webpage 1.0</title>
</head>
<body>
<br /><b>
  <div>Ostatnie 50 rekordów z bazy:</div></b>
<br />
<?php
  // mysql setup
  $config[,db_server']   = ,***.pwr.wroc.pl';
  $config[,db_username'] = ,***';
  $config[,db_password'] = ,***';
  $config[,db_schema']   = ,***';
  // Connecting, selecting database
  $mysqli = new mysqli($config[,db_server'], $config[,db_username'],
                       $config[,db_password'],$config[,db_schema'] );
  if (mysqli_connect_errno())
  {
    echo 'Problem z połączeniem z DBMS!';
  } else
  {
    // Performing SQL query
    $query = „SELECT * FROM logtab ORDER BY ID DESC LIMIT 0,50";
    if ($result = $mysqli->query( $query ))
        {
        echo ,<table width="50%" border="1" cellspacing="0">';
        echo
           „\r\n<tr><td><i>L.P.</i></td><td><i>ID</i></td><td><i>
               Temp</i></td><td><i>TBat</i></td><td><i>VBat</i></td><td><i>
                     Local Time</i></td><td><i>Entry Time</i></td></tr>\r\
n";
          $LP = 0;
          while ( $row = $result->fetch_array(MYSQLI_NUM) )
          {
         $LP++;
           echo
           „<tr><td>$LP</td><td>$row[0]</td><td>$row[1]</td><td>$row[2]
               </td><td>$row[3]</td><td>$row[4]</td><td>$row[5]</td></tr>\r\
n";
          }
          mysqli_free_result($result);
          echo „</table>\r\n";
        }
  }
?>
<p><i>Wavecom2HTTP demo webpage 1.0</i></p>
</body>
```




**Rys. 1. Efekt działania skryptu prezentującego dane pomiarowe**

- wartość temperatury TEMP,
- wartość temperatury baterii TBAT,
- napięcie baterii VBAT,
- kolumna z dagą i godziną pomiaru LTIME (czas lokalny modułu G2686),
- data i godzina wpisu do bazy danych ETIME (wstawiana automatycznie).

Klucza ID oraz czas wpisu ETIME dodawane są automatycznie przez serwer MySQL podczas zapytania. Czas wpisu pozwala zdiagnozować ewentualne opóźnienia w przekazywaniu danych. Mamy więc możliwość oceny działania mechanizmu buforowania, zastosowanego w aplikacji OpenAT. Pozostałe dane w tabeli pochodzą z aplikacji i są przekazywane w zapytaniu (**list. 7**).

## Prezentacja danych

W poprzednich punktach, przedstawiono trzon podstawowy systemu akwizycji danych telemetrycznych. Teraz czas na aplikację prezentującą zgromadzone dane. Działanie skryptu PHP polega na wykonaniu połączenia z serwerem bazy danych, wyszukaniu danych według pewnego kryterium (linia 240), a następnie wygenerowaniu kodu strony w języku HTML. W przykładzie wyszukujemy 50 najaktualniejszych danych z bazy, w kolejności: najświeższe dane pierwsze. Dla każdego rekordu, skrypt generuje odpowiedni kod nowego wiersza tabeli, zgodnie z wymaganiami języka HTML. Odświeżenie strony (ponowne wywołanie skryptu), spowoduje ponowne wykonanie kodu i aktualizacje generowanej strony (**list. 8**).

Efektem końcowym powinna być tabela z wynikami pomiarowymi. Jeśli wszystko odbędzie się bez przeszkód, zobaczymy widok, jak na **rys. 1**.

## Posumowanie

Przedstawiony projekt programu, a właściwie kompletny rozproszony system telemetryczny, stanowi praktyczny przykład wykorzystania technologii Web w telemetrii M2M. Konstrukcja aplikacji składowych ukazuje potencjał takie koncepcji architektury systemu akwizycji danych pomiarowych. Dla osób, które dopiero stawiają pierwsze kroki w OpenAT, a chwiałyby zbudować ciekawy i efektywny system telemetryczny (np. stację pogodową) – tu mają wszystko, czego potrzebują. Na potrzebny artykułu, oprogramowanie zostało uproszczone praktycznie do minimum, głównie celem zapewnienia odpowiedniej czytelności. Brakuje w nim choćby kontroli poprawności działania czy weryfikacji danych. Nie ma też żadnych mechanizmów autoryzacji, choć biblioteka WIP posiada takie rozszerzenie. Rozbudową o wspomniane, brakujące elementy pozostawiamy bardziej wnikliwym Czytelnikom.

**Jacek Majewski, Krzysztof Rutecki**
**Politechnika Wrocławska**
**(w ramach projektu kluczowego POIG.01.03.01-02-002)**