Dr. Edgar Huckert
Stand: 15.11.2015

Eine alternative WEB Kamera mit Raspberry

Zwei Ansätze

Die für den Raspberry angebotene kleine Kamera zeigt eine überraschende Qualität und ist preiswert (ca. 30 Euro). Für die grundlegende Installation und erste Tests verweise ich auf [2]. Für ein ähnliches, aber weitaus aufwändigeres Projekt verweise ich auf [3] (verwendet das Paket *motion*, verwendet Yahoo Services).

Dieses kleine Projekt geht davon aus, dass die Kamera installiert ist und funktioniert. Hier wird angenommen, dass die *Raspian* Variante von Linux eingesetzt wird. Prinzipiell sollten jedoch auch andere Linux Distributionen einsetzbar sein, da keine exotischen Softwarekomponenten eingesetzt werden.

Prinzipiell gibt es **zwei denkbare Ansätze**, um die Bilder der Raspberry Kamera im WEB bereitzustellen:

- Installation eines WEB Servers (z.B. *Apache* oder *lighttpd*) auf dem Raspberry, Zugriff dann wie auf einen **WEB Server** (s. z.B. [3]).

- Verwendung eine bereits existierenden WEB Servers, Raspberry liefert nur die Daten an diesen Server – ist also **Client**.

Wir verwenden hier den zweiten Ansatz, weil wir annehmen, dass

- der Zugriff auf die Kamerabilder außerhalb des WLAN (z.B. aus einem Hotel) erfolgen soll

- weil wir den Raspberry nicht als eigenständigen Web Server im Internet registrieren wollen.

Natürlich setzt der zweite Ansatz voraus, dass ein WEB Server – wer eine „home page" hat, hat einen solchen WEB Server - existiert und dass Schreibrechte auf diesem Server vorliegen. Weiterhin sollte dieser Server PHP und eine kleine Datenbank (z.B. mySQL) anbieten.

Die Lösung

Das Projekt verwendet die folgenden Systemkomponenten auf dem Raspberry

PI:

- Kommando *raspistill* (legt Einzelbilder ab)
- Kommando *wput* (ein *ftp*-ähnlicher Client, nicht im Standard!)
- *cron* Prozess (startet Prozesse in minutengenauen Abständen, im Standard)
- GCC Compiler. Verwendet wurde Version gcc-4.6

Das Kommando *wput* dient dazu, die erzeugte Seite und das erfasste Bild auf den Server zu übertragen. Das Kommando *wput* ist normalerweise nicht in der *raspbian* Distribution enthalten. Es kann installiert werden über:

```
sudo apt-get install wput
```

Statt *wput* können auch andere Programme wie z.B. *curl* verwendet werden – dann ist natürlich die jeweilige Syntax anders.

Neben diesen Systemkomponenten ist ein kleines, in C++ geschriebenes Individualprogramm namens *makeMeta* nötig - der Quellcode ist in der ZIP Datei. Dieses Programm erzeugt rudimentäre **Metadaten** (Bildnummer, Datum, Uhrzeit) in Form einer einzeiligen Datei *meta1.txt*, die zusamen mit dem Bild auf den Webserver hochgeladen wird. Das Programm *makeMeta.cpp* ist einfach zu lesen – es verwendet keine speziellen Bibliotheken oder spezielle C++ Mechanismen. Es kann auch ersetzt werden durch Programme in anderen Programmiersprachen wie reines C, Python, Java, Basic oder gar durch ein (recht komplexes) Shell Skript. Für die Übersetzung reicht dieses Kommando auf dem Raspberry:

```
g++ -o makeMeta makeMeta.cpp
```

Apropos Datum und Uhrzeit: Raspberry kommt standardmäßig nicht mit einer Hardwareuhr (RTC). Deshalb muss u.U. entweder eine Hardwareuhr als Zusatzplatine hinzugefügt werden oder aber Datum und Uhrzeit müssen sofort nach dem Starten des Raspberry gesetzt werden, z.B. so:

```
sudo date --set "20150829 22:15:12"
```

Alternativ dazu kann die Uhrzeit auch aus dem Netz bezogen werden (in einigen WLAN Installationen geschieht dies automatisch) – das überschreitet allerdings den Rahmen dieses Beitrags.

Das Skript *camera.sh* wird vom *cron* Prozess in regelmäßigen Intervallen gestartet. Es muss mit einem geeigneten Programmiereditor (z.B. *vi*) geschrieben werden. Es sollte ausführbar sein. Das kann man mit dem Kommando

```
sudo chmod 555 ./camera.sh
```

erreichen.

Der Systemprozess *cron* muss nun konfiguriert werden, damit er das Skript *camera.sh* im Minutenabstand oder in größeren Abständen startet. *cron* kann nicht in kleineren Intervallen als eine Minute konfiguriert werden. Möglicherweise ist es auch technisch gar nicht möglich oder zumindest sehr aufwändig, mehr als ein Bild pro Sekunde auf den Webserver zu schieben, weil der Upload per *wput* je nach Internetverbindung sehr lange dauern kann. Bei mir dauert der Upload pro Bild je nach Tageszeit über 30 Sekunden. Downloads im Internet sind bekanntlich meist schneller als Uploads, weil Uploads seltener geschehen.

Wenn Bilder häufiger als im Minutentakt hochgeladen werden sollen, empfiehlt es sich, das *cron* Skript durch ein (z.B. in C oder C++ geschriebenes) Hintergrundprogramm zu ersetzen, das beim Hochfahren des Raspberry automatisch gestartet wird.

Da das *cron*-Skript vom Systemprogramm *cron* gestartet wird,dessen "current directory" nicht ohne Weiteres bekannt ist, ist es besser, absolute Pfade statt relative Pfade zu verwenden, also z.B.

```
/home/pi/camera/meta1.txt („absolut“)
```

statt

```
./meta1.txt                 („relativ“)
```

Es wird angenommen, dass alle projektrelevanten Dateien in einem eigenen Verzeichnis abgelegt werden, z.B. */home/pi/camera*. Dieses Verzeichnis sollte für alle beschreibbar sein. Das erreicht man z.B. über

```
sudo chmod 666 /home/pi/camera
```

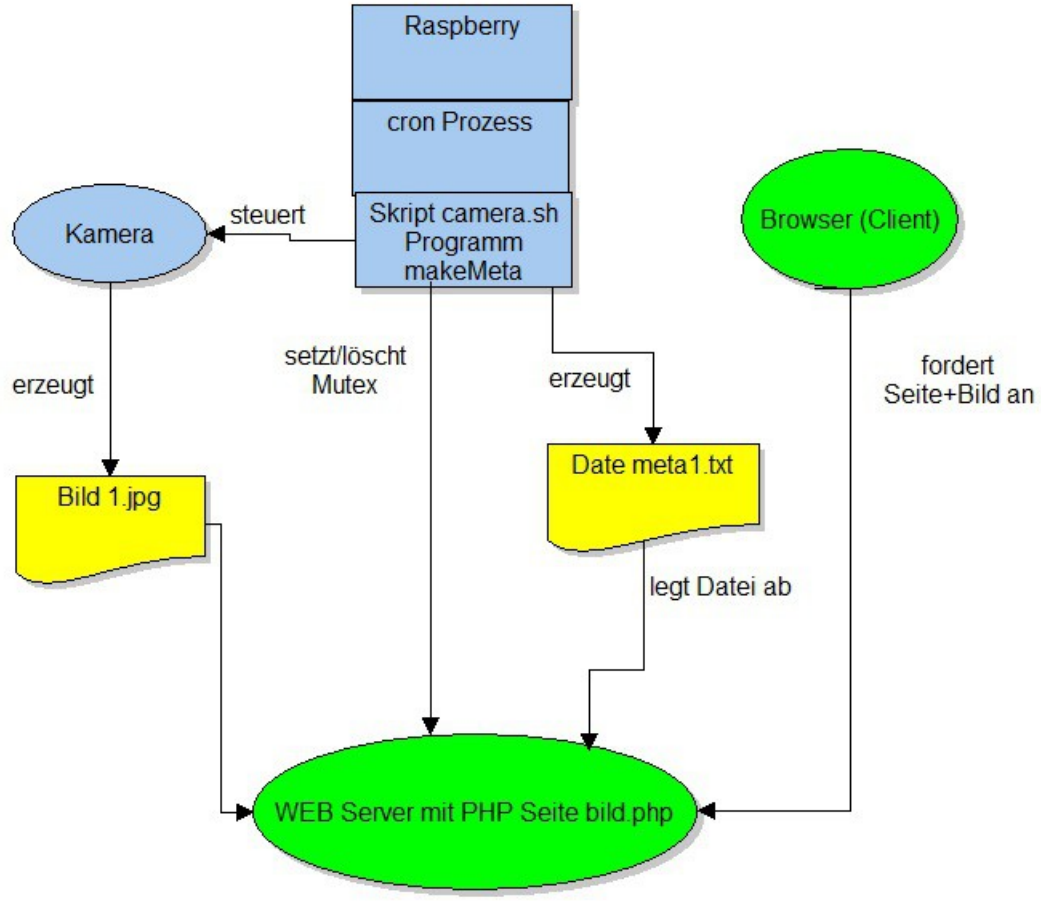
Zwei Dateien werden durch das Skript geschrieben: *1.jp*g und *meta1.tx*t. Diese Dateien sollten schreibbar für alle sein. Das erreicht man über die folgenden Kommandos (man stelle sich in das Verzeichnis */home/pi/camera*):

```
sudo chmod 777 meta1.txt
sudo chmod 777 1.jpg
```

Die Bilddatei *1.jpg* wird im Skript durch Aufruf des Kommandos *raspistill* erzeugt.

Die Datei *meta1.txt* dient nur dazu, die laufende Bildnummer sowie Datum und Uhrzeit auf dem Raspberry feszuhalten. Das Programm *makeMeta* (Aufruf s. Skript *camera.sh*) liest die Datei *meta1.txt* und schreibt sie verändert zurück. Die Datei *meta1.txt* wird zusammen mit dem Bild in *1.jpg* auf den Web Server hochgeladen, wo eine PHP Seite (diese Seite wird vom Browser aufgerufen) die Metadaten ausliest und in die „on the fly“ erzeugte Web Seite einbaut.

Prozesstruktur vereinfacht



Synchronisationsprobleme

Das Hauptproblem unseres Ansatzes liegt darin, eine Lösung zu finden für die Synchronisation zwischen schreibendem Zugriff (*wput*) und lesendem Zugriff über einen Browser (= Anforderung des Bilds). Wenn gleichzeitig geschrieben und gelesen wird, ist das Risiko sehr groß, dass ein unvollständiges Bild im WEB Browser angezeigt wird. Das ist nicht dramatisch, aber unschön.

Die Schreib- und Lesevorgänge sollten also synchronisiert werden, wobei „synchronisieren" hier heißt: sie sollen nicht gleichzeitig erfolgen.

Große Installationen lösen das Synchronisationsproblem meist dadurch, dass sie eine **Schattenkopie** des gesamten Seitenbaums vorhalten. Der Schreibvorgang erfolgt in diese Schattenkopie, die keinerlei externe lesende Zugriffe zulässt. Erst nach erfolgreichem Schreiben wird die Schattenkopie zur Arbeitskopie.

Bei Verwendung von Threads auf einem einzigen Rechner würde man explizite Synchronisationsmechanismen wie z.B. Mutexes verwenden. In Client/Server

Anwendungen werden meist Datenbanken zur Synchronisation verwendet. Größere Datenbanken (Oracle, DB2 etc.) synchronisieren lesende und schreibende Zugriffe automatisch. Hier jedoch haben wir es mit oft sehr einfachen WEB Servern zu tun, deren Innenleben unbekannt ist. Deshalb ist die Synchronisation nicht so einfach und vor allem **nicht perfekt** hinzukriegen.

Das Ganze funktioniert nur, wenn wir einen Teil der geforderten Logik auf dem Server implementieren können. Die Ausführung von CGI Programmen (das wäre uns die liebste Lösung) ist normalerweise nicht erlaubt. Also versuchen wir, diese Logik über PHP in den Griff zu kriegen. PHP wird oft selbst in kleinen WEB Hostingpaketen von den Providern unterstützt. In Zweifelsfällen gibt der Aufruf der PHP Funktion *phpinfo()* Klarheit.

Um einem WEB Browser zu signalisieren, dass gerade ein Upload geschieht, benutzen wir einen Eintrag in einem Feld einer *mySq*l Datenbank. Dazu brauchen wir eine simple Tabelle mit dem bekannten Aufbau „Schlüssel=Wert". Natürlich kann auch eine andere Datenbank als mySql verwendet werden. Wichtig ist nur, dass die Datenbankmaschine lesende und schreibende Zugriffe voneinander entkoppelt.

Die Tabelle (hier „ETC") kann z.B. so auf dem Web Server angelegt werden:

```
create table ETC (TXT varchar(20),KW varchar(20));
```

Natürlich kann auch eine existierende Tabelle verwendet werden. Wir brauchen nur eine einzige Zeile (row, record) in dieser Tabelle.

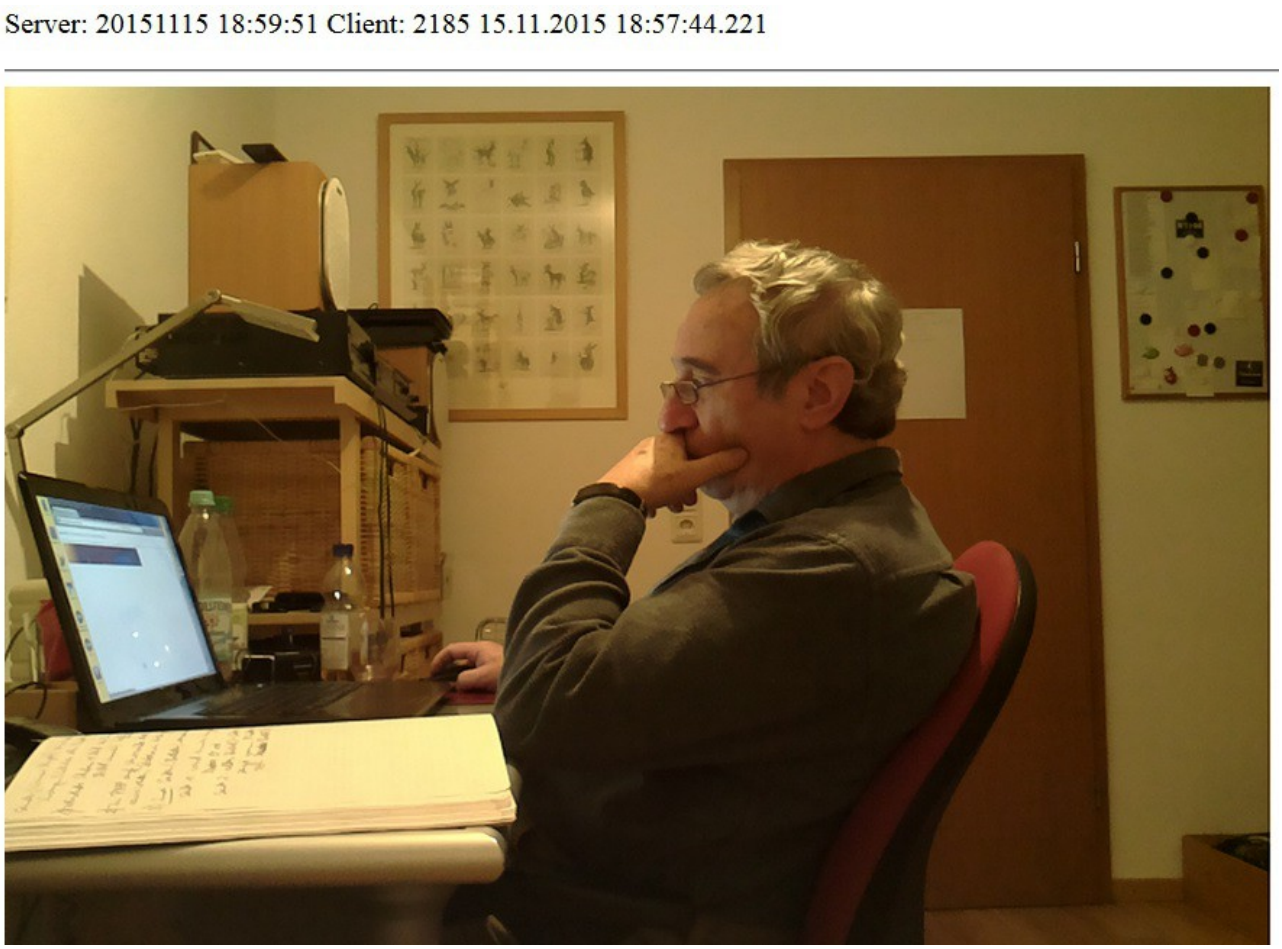
Der schreibende Prozess (Skript auf dem Raspberry) schreibt eine „1" vor dem Beginn der Bildübertragung in diese Tabelle. Wenn er mit der Übertragung der Dateien fertig ist, schreibt er eine „0" in die Tabelle, setzt also das Mutexflag zurück. Der lesende Prozess (die PHP Seite für die Anzeige) liest die gleiche Tabelle und prüft, ob eine „0" darin steht. Dann und nur dann schickt er das Bild zum Browser. Falls gerade geschrieben wird - also eine „1" in der Spalte steht – schickt er einen Hinweis an den Browser, z.B. den Text „Das Bild wird gerade geladen".

Das Setzen des Mutex Flags versteckt sich in der PHP Seite *createMutex.php*, deren Kernlogik per Kommando *wget* im *cron* Skript ausgelöst wird. Die ebenfalls per *wget* angeforderte PHP Seite *removeMutex.php* setzt das Mutex-ähnliche Flag in der Datenbank auf dem Server zurück. Beide Skripte sind übrigens normale HTML Seiten mit PHP Anteilen, die allerdings keinen Output erzeugen.

Es ist wichtig, dass die parallel zum Bild erzeugte Datei mit den Metadaten ebenfalls innerhalb der Mutexklammer geschrieben wird.

Beispielsbild mit Metadaten

Nach Aufruf der weiter unten abgebildeten PHP Seite wird ein Bild wie das folgende ausgegeben:



In der Zeile über dem Kamerabild werden die Metadaten angezeigt:

- Datum und Uhrzeit auf dem Server (Web Server)
- Bildnummer, Datum und Uhrzeit auf dem Client (Raspberry)

wobei hier „Server" und „Client" aus Sicht der Datenerzeugung gesehen werden.

Cron Skript

```
# this should be executed by cron (started via /etc/crontab)
#
# produce an image (file 1.jpg)
raspistill -md 4 -e jpg -q 70 -t 500 -o /home/pi/camera/1.jpg
# set the mutex in a DB on the server
wget http://abc.xyz.com/privat/createMutex.php
>/home/pi/camera/dummy1.txt
# upload the image (=file 1.jpg)
wput -u -B -a /home/pi/camera/wputlog.txt /home/pi/camera/1.jpg
ftp://user:password@abc.xyz.com/public_html/privat/1.jpg
# generate a short meta file (=meta1.txt)
/home/pi/camera/makeMeta
# upload the meta file
wput -u -B -a /home/pi/camera/wputlog.txt
/home/pi/camera/meta1.txt
ftp://user:password@abc.xyz.com/public_html/privat/meta1.txt
# reset the mutex in a DB on the server
wget http://abc.xyz.com/privat/removeMutex.php
>/home/pi/camera/dummy2.txt
```

In diesem Skript müssen natürlich einige Zugriffsparameter (URL Bestandteile) ersetzt werden.

Die serverseitigen Mutexskripte

In beiden HTML/PHP Skripten müssen einige Zugriffsparameter (s. vorhergehendes Skript) ersetzt werden. Im SELECT Statement müssen Spaltenname und Schlüssel ersetzt werden.

Skript createMutex.php

```
<HTML>
<TITLE>Proj. Web Cam EH: create mutex</TITLE>
<BODY>
  <?php
    // Loesung ueber MySQL DB
    $server     = "abc.xyz.com";
    $user       = "username";
    $passwort   = "password";
    $datenbank  = "databasename";
```

```
    $tabelle    = "tablename";

    MYSQL_CONNECT($server, $user, $passwort) or die
      ("Fehler Oeffnen DB ETC");
       MYSQL_SELECT_DB($datenbank) or die
       ("<H3>DB ETC nicht vorhanden?");
    $stmt = "UPDATE ETC SET TXT='1' WHERE KW='WEBCAMSTATE';";
    MYSQL_QUERY($stmt); // or die("Fehler UPDATE in Tabelle ETC");
    $errmsg = MYSQL_ERROR();
    echo "<H3>MySQL error=" . $errmsg . "</H3>";
    //
    MYSQL_CLOSE();
   ?>
</BODY>
</HTML>
```

Skript removeMutex.php

```
<HTML>
<TITLE>Proj. Web Cam EH: create mutex</TITLE>
<BODY>
  <?php
    // Loesung ueber MySQL DB
    $server     = "abc.xyz.com";
    $user       = "username";
    $passwort   = "password";
    $datenbank  = "databasename";
    $tabelle    = "tablename";

    MYSQL_CONNECT($server, $user, $passwort) or die
      ("Fehler Oeffnen DB ETC");
       MYSQL_SELECT_DB($datenbank) or die
       ("<H3>DB ETC nicht vorhanden?");
    $stmt = "UPDATE $tabelle SET TXT='1' WHERE KW='WEBCAMSTATE';";
    MYSQL_QUERY($stmt); // or die("Fehler UPDATE in Tabelle ETC");
    $errmsg = MYSQL_ERROR();
    echo "<H3>MySQL error=" . $errmsg . "</H3>";
    //
    MYSQL_CLOSE();
```

```
   ?>
</BODY>
</HTML>
```

Bildanzeige: die PHP Seite im Server

Eine reine HTML Datei im Server reicht nicht: sie kann das Synchronisationsproblem nicht lösen. Wir brauchen Logik auf der Serverseite. Deshalb wird die nachfolgende gezeigt PHP Seite *bild.php* eingesetzt (sie kann natürlich erweitert werden).

Die Logik des PHP Programms folgt den oben erläuterten Prinzipien:

- Prüfen, ob ein Mutex (Wert eines Feldes in der Datenbank) gesetzt ist
- Falls ja: Ausgabe einer Fehlermeldung bzw. eines Warnhinweises
- Falls nein: Ausgabe der Metainformationen (Bildnummer, Serverdatum, Clientdatum) und der Bilddatei (s. HTML Befehl *IMG SRC...*)

Zur Ermittlung der Metainformationen wird die Datei *meta1.txt* ausgewertet, die zuvor vom con-Skript übertragen wurde.

```
<?php
  function outputDates()
  {
    $ret = false;
    $dateTime   = date('Ymd H:i:s');
    echo "<P>Server: $dateTime ";
    $ifd = fopen('./meta1.txt', 'r');
    if ($ifd)
    {
      $s = fgets($ifd, 64);
      if ($s)
      {
        echo "Client: $s";
      }
      fclose($ifd);
      $ret = true;
    }
    echo "</P>\n<HR>\n";
    return $ret;
  }   // end function outputDates()

  //ini_set("log_errors", "On");
  //ini_set("error_log", "./php.log");

  echo "<HTML><BODY>\n";
  echo "<TITLE>Bildanzeige rein PHP mit DB</TITLE>\n";
  //
  $server     = "abc.xyz.com";           // change this!
```

```
  $user       = "username";              // change this!
  $passwort   = "password";              // change this!
  $datenbank  = "databasename";          // change this!
  $tabelle    = "tablename";             // change this!
  $name       = './1.jpg';
  //
  MYSQL_CONNECT($server, $user, $passwort) or die
      ("<H3>Datenbankserver $server nicht erreichbar</H3>");
  MYSQL_SELECT_DB($datenbank) or die("DB nicht erreichbar");
  // Suche starten
  // Note: the SELECT string must probably be changed!
  $result = MYSQL_QUERY
     ("SELECT TXT FROM ETC WHERE KW='WEBCAMSTATE'");
  if (! $result)
  {
    echo "<H2>Kein SELECT Ergebnis</H2>\n";
  }
  else
  {
    $noRows = 0;
    while($row  =  mysql_fetch_row($result))
    {
      for($i=0;  $i < mysql_num_fields($result);  $i++)
      {

        $field = $row[$i];
        if ($field == '0')
        {
           outputDates();
           echo "<IMG SRC=$name WIDTH=700 HEIGHT=500>\n";
        }
        else
        {
          outputDates();
          echo "<H3>Bild wird z.Z. geladen</H3>\n";
        }
      }  // end for ...
      $noRows = $noRows + 1;
    }  // end while ...
  }   // end else
  //
  MYSQL_CLOSE();
  echo "<BR>\n";
  echo "</BODY>\n</HTML>\n";
?>
```

C++ Programm makeMeta.cpp

Dieses C++ Programm muss auf dem Raspberry übersetzt werden. Es erzeugt die Datei *meta1.txt*, die zusammen mit dem Bild auf den Web Server hochge-

laden wird. Der Aufruf erfolgt aus dem cron Skript *camera.sh*.

```
// module makeMeta.cpp
// Erzeugt eine Datei mit Meta-Daten zum Hochladen
// auf deeb Server, Dateiname=meta1.txt
// E.Huckert 08-2015

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/timeb.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>

#include "hu_utils.h"

// ------------------------------
// Liefert die letzte Bildnummer aus Datei meta1.txt
unsigned getImgNr()
{
  FILE *meta = NULL;
  unsigned nr = 0;
  //
  meta = ::fopen("/home/pi/camera/meta1.txt", "r");
  if (meta != NULL)
  {
    char buf[32];
    ::fgets(buf, sizeof(buf), meta);
    nr = ::atoi(buf);
    ::fclose(meta);
  }
  return nr;
}   // end getImgNr()

// ------------------------------
int main(int argc, char *argv[])
{
  FILE *ofd = NULL;
  FILE *meta = NULL;
  unsigned imgNr = 0;
  int ret = 0;
  char buf1[16], buf2[16];
  //
  // Metadaten (z.Z. nur Bildnummer) ermitteln
  // Benutzt dazu die letzte Version von Datei meta1.txt
  imgNr = getImgNr();
  imgNr++;
  //
  // eine Datei namens meta1.txt erzeugen
  ofd = ::fopen("/home/pi/camera/meta1.txt", "w");
  if (ofd == NULL)
```

```
  {
    ret = -1;
    goto zurueck;
  }
  ::get_date(buf1);
  ::get_time(buf2);
  ::fprintf(ofd, "%u %s %s\n", imgNr, buf1, buf2);
  //
  zurueck:
  if (ofd != NULL)
    ::fclose(ofd);
  return ret;
}   // end main()
```

Der Eintrag in der cron Steuerdatei /etc/crontab

Der nachfolgend gezeigte Eintrag in der Steuerdatei für den *cron* Prozess geht davon aus, dass alle 3 Minuten ein Bild erzeugt und hochgeladen werden soll:

```
3,6,9,12,15,18,21,24,27,30,33,36,39,42,45,48,51,54,57 * * * *
     root /home/pi/camera/camera.sh
```

Name und Pfad des Skripts müssen ggf. angepasst werden.

Literatur

[1]: https://www.raspberrypi.org/documentation/usage/camera/raspicam/timelapse.md

[2]: http://raspberrypiguide.de/howtos/raspberry-pi-camera-how-to/

[3]: http://www.laub-home.de/wiki/Raspberry_Pi_als_Webcam_Server