Inhaltsverzeichnis

Spielereien mit einer LED Badge 3

Spielereien mit einer LED Badge

Günstig auf'm Flohmarkt geschossen, habe ich nun eine absolute NoName- LEDBadge als Ansteck-Objekt. Aus reinem Forscherdrang hab ich mal herausgefieselt, wie das Ding zu programmieren ist:

- Anschluß über USB-Seriell (Baudrate unbekannt, also tippe ich mal auf 9600 Baud)
- Datentransfer nur vom PC zum Badge, nix zurück

Das Datenpacket ist folgendermaßen

Ein kurzer Header, gefolgt von einem 0-Byte

```
Hello[0x00]
```

dann 8 Bytes, die die „Schriftart“ definieren. Die unteren 4 Bits geben den Effekt:

Wert	Effekt
0	Left
1	Right
2	up
3	down
4	freeze

Ist Bit 3 (=8) gesetzt, blinkt die Anzeige (Flash)

Die oberen 4 Bits geben die Scrollgeschwindigkeit (0..7)

dann kommt ein Byte 0x03

dann 6 leere Bytes (0x00)

dann werden 11 Blöcke geschrieben, die letzten 3 sind immer leer (gefüllt mit 0x00)

vor jedem Block wird 3 Sekunden Pause gemacht

Jeder Block ist 256 Bytes lang, gefüllt mit 0x00, ausser dem Text natürlich. Dieser hat normalen ISO-89xx- Dingsbums-Western-Europe- Zeichensatz, ausser den Zeichen 0x41- 0x60: Hier wird (wahrscheinlich zu Kompatibilität zu alten Geräten) immer 0x40 abgezogen, A (0x41) wird also zu 0x01

Und falls mal sowas mal selber programmieren will, ein Beisiel (zumindestens zum Öffnen des seriellen Ports) gabs unter http://home.ircnet.de/cru/ledticker/ von Veit Wahlich

```
#!/usr/bin/perl

# LED Ticker 1.1.1 (C)2005 Veit Wahlich <cru@zodia.de>
# Released under the terms of the GNU Public License, version 2 (GPLv2)

use strict;
use warnings;
```

```
use IO::File;
use POSIX;

# Default configuration
my $conf={
        version  => '1.1.1',
        port     => '/dev/ttyS0', # i.e. '/dev/ttyS0'
        speed    => 2,            # range is 0..4
        sign_id  => 0,            # range is 0..255, 0 is default
        page     => 'B',          # either page 'A' or 'B', defaults to 'B'
        mode     => 'text',       # operation mode defaults to text output
        noflash  => 0,            # noflash mode, defaults to 0
        nf_delay => 25000         # microseconds to wait on noflash
    };

sub main{
    my $fd;
    my $err=0;
    my $text;
    parseArgs()
        || do{
            showUsage();
            exit(1);
        };
    if($conf->{mode} eq 'text' && exists($ARGV[0]) && defined($ARGV[0])){
$text=constructText($ARGV[0],$conf->{speed},$conf->{sign_id},$conf->{page});
    }
    elsif($conf->{mode} eq 'timesync'){
        $text=constructTimesync($conf->{sign_id},localtime(time));
    }
    elsif($conf->{mode} eq 'setid' && exists($ARGV[0]) &&
defined($ARGV[0])){
        $text=constructSetID($ARGV[0]);
    }
    else{
        showUsage();
        exit(1);
    }
    if(-c $conf->{port}){
        if(defined($text)){
            if($conf->{noflash}){
                # require module here instead of using it above, so there is
no forced
                # dependency when not using this feature
                require Time::HiRes;
sendString(constructText("",$conf->{speed},$conf->{sign_id},$conf->{page}));
                Time::HiRes::usleep($conf->{nf_delay});
            }
            sendString($text);
        }
        else{
```

```
            showUsage();
            exit(1);
        }
    }
    else{
        die($conf->{port}.' is no character device!'."\n");
    }
}

# parseArgs()
# Parse the command line arguments and set config settings
#
sub parseArgs{
    my $arg;
    if(@ARGV == 0){
        return(0);
    }
    while(@ARGV > 1){
        $arg=shift(@ARGV);
        if($arg eq '-d'){
            $conf->{port}=shift(@ARGV) || return(0);
        }
        elsif($arg eq '-s'){
            $arg=shift(@ARGV);
            if(defined($arg) && $arg=~/^[0-4]$/){
                $conf->{speed}=$arg;
            }
            else{
                return(0);
            }
        }
        elsif($arg eq '-i'){
            $arg=shift(@ARGV);
            if(defined($arg) && $arg=~/^[0-9]+$/ && $arg >= 0 && $arg <=
255){
                $conf->{sign_id}=$arg;
            }
            else{
                return(0);
            }
        }
        elsif($arg eq '-p'){
            $arg=shift(@ARGV);
            if(defined($arg) && $arg=~/^(A|B)$/i){
                $conf->{page}=uc($arg);
            }
            else{
                return(0);
            }
        }
        elsif($arg eq '-m'){
```

```
            $arg=shift(@ARGV);
            if(defined($arg) && $arg=~/^(text|timesync|setid)$/){
                $conf->{mode}=$arg;
            }
            else{
                return(0);
            }
        }
        elsif($arg eq '-f'){
            $conf->{noflash}=1;
        }
        else{
            print(STDERR 'Unknown option: '.$arg."\n");
            return(0);
        }
    }
    return(1);
}

# sendString($string)
# Sends a control string to the LED ticker device
#
# Parameters:
#   $string     : string to send to the device
#
sub sendString{
    my($string)=@_;
    my $fd;
    return(undef) unless(defined($string));
    $fd=new IO::File($conf->{port},'w')
        || die('Error opening port '.$conf->{port}.': '.$!."\n");
    print($fd $string);
    $fd->close;
    return(1);
}

# $string=constructText($text,$speed,$sign_id,$page)
# Builds a text transfer control string for the LED ticker
#
# Parameters:
#   $text       : a text string that shall be displayed, max. 420 characters
#                 for page B, 80 chars for page A (reserved for logos)
#   $speed      : scroll speed, integer 0..4
#   $sign_id    : the LED ticker's ID, integer 0..255
#   $page       : page to programm, either 'A' or 'B'
#
# Returns:
#   $string     : the control string ready to be sent to device
#
sub constructText{
    my($text,$speed,$sign_id,$page)=@_;
```

```
    my $string;
    return(undef) unless(defined($text) && defined($speed) &&
defined($sign_id) && defined($page));
    $text=convertTextIso8859_15($text);
    # <E> is a reserved sequence - replacing it by <e>
    $text=~s/\<E\>/\<e\>/g;
    if($page eq 'A'){
        $text=substr($text,0,80);
    }
    elsif($page eq 'B'){
        $text=substr($text,0,420);
    }
    else{
        return(undef);
    }
    $string='<L1><P'.$page.'><FE><M'.chr(69-($speed % 5)).'><WC><FE>'.$text;
    $string=sprintf('<ID%02x>',($sign_id %
256)).$string.sprintf('%02x',checksum($string)).'<E>';
    return($string);
}

# $string=constructTimesync($sign_id,@localtime)
# Builds a time/date sync control string
#
# Parameters:
#   @localtime  : array in @array=localtime format
#
# Returns:
#   $string     : the control string ready to be sent to device
#
sub constructTimesync{
    my($sign_id,@time)=@_;
    my
$string='<SC>'.strftime('%y',@time).sprintf('%02d',strftime('%u',@time) % 7
+ 1).strftime('%m%d%H%M%S',@time);
    $string=sprintf('<ID%02x>',($sign_id %
256)).$string.sprintf('%02x',checksum($string)).'<E>';
    return($string);
}

# $string=constructSetID($new_id)
# Builds a string that (re)sets the device's ID to a new one
#
# Parameters:
#   $new_id     : ID to set device to, integer 0..255
#
# Returns:
#   $string     : the control string ready to be sent to device
#
sub constructSetID{
    my($new_id)=@_;
```

```
    return(sprintf('<ID><%02x><E>',$new_id));
}

# $value=checksum($string)
# Calculates a string's checksum
#
# Parameters:
#   $string     : string to checksum
#
# Returns:
#   $value      : checksum, integer 0..255
#
sub checksum{
    my($string)=@_;
    return(undef) unless(defined($string));
    my $value=0;
    my $i;
    for($i=0; $i < length($string); $i++){
        $value=$value^ord(substr($string,$i,1));
    }
    return($value);
}

# $out=convertTextIso8859_15($in)
# Converts a ISO8859-15 string to the device's equivalent charset.
# Also replaces newline/return by spaces.
#
# Parameters:
#   $in         : ISO8859-15 input string
#
# Returns:
# $out          : string in device's charset
#
sub convertTextIso8859_15{
    my($in)=@_;
    my $out='';
    my $i;
    my $char;
    my $table={
            # device does not support newlines:
            0x0A => 0x20, 0x0D => 0x20,
            # lots of umlauts and other special characters:
            0xC3 => 0x7F, 0xC2 => 0x80, 0xC1 => 0x81, 0xC0 => 0x82, 0xC4 =>
0x83,
            0xC5 => 0x84, 0xC6 => 0x85, 0xDF => 0x86, 0xC7 => 0x87, 0xD0 =>
0x88,
            0xC9 => 0x89, 0xCA => 0x8A, 0xC8 => 0x8B, 0xCB => 0x8C, 0xCD =>
0x8D,
            0xCC => 0x8E, 0xCE => 0x8F, 0xCF => 0x90, 0xD1 => 0x91, 0xD3 =>
0x92,
            0xD4 => 0x93, 0xD2 => 0x94, 0xD6 => 0x95, 0xD5 => 0x96, 0xD8 =>
```

```
0x97,
            0xDE => 0x98, 0xDA => 0x99, 0xD9 => 0x9A, 0xDB => 0x9B, 0xDC =>
0x9C,
            0xBE => 0x9D, 0xDD => 0x9E, 0xE3 => 0x9F, 0xE2 => 0xA0, 0xE1 =>
0xA1,
            0xE0 => 0xA2, 0xE4 => 0xA3, 0xE5 => 0xA4, 0xE6 => 0xA5, 0xE7 =>
0xA6,
            0xE9 => 0xA7, 0xEA => 0xA8, 0xE8 => 0xA9, 0xEB => 0xAA, 0xED =>
0xAB,
            0xEC => 0xAC, 0xEE => 0xAD, 0xEF => 0xAE, 0xF1 => 0xAF, 0xF3 =>
0xB0,
            0xF4 => 0xB1, 0xF2 => 0xB2, 0xF6 => 0xB3, 0xF5 => 0xB4, 0xF8 =>
0xB5,
            0xFE => 0xB6, 0xFA => 0xB7, 0xF9 => 0xB8, 0xFB => 0xB9, 0xFC =>
0xBA,
            0xFF => 0xBB, 0xFD => 0xBC, 0xA5 => 0xBD, 0xA3 => 0xBE, 0xA4 =>
0xBF
        };
    return(undef) unless(defined($in));
    for($i=0; $i < length($in); $i++){
        $char=substr($in,$i,1);
        if(exists($table->{ord($char)})){
            $out.=chr($table->{ord($char)});
        }
        else{
            $out.=$char;
        }
    }
    return($out);
}

# showUsage()
# Print usage info to STDOUT
#
sub showUsage{
    print(<<__END);

LED Ticker $conf->{version} (C)2005 Veit Wahlich <cru\@zodia.de>

Syntax: $0 [OPTIONS] { [-m text] <text> | -m timesync | -m setid <id> }

Commands:
  [-m text] <text>  Send new text to the LED ticker ("-m text" is optional)
  -m timesync       Synchronize the ticker with your system clock
  -m setid <id>     Set the device ID of the ticker to <id>. Use with
caution!

Options [defaults]:
  -d <device>       The serial port the ticker is connected to
[$conf->{port}]
  -s <speed>        Ticker scroll speed; 0..4 [$conf->{speed}]
```

```
  -i <id>           ID of the ticker to address; 0..255 [$conf->{sign_id}]
  -p <page>         Ticker page to programm; A (reserved) or B
[$conf->{page}]
  -f                Insert short delay to reduce flashing when programming
text

Note:
  Maximum text length is 420 characters for page B and 80 characters for
page A.
  Oversized text will be truncated.
  Special characters, German umlauts, etc. are converted from ISO 8859-15
input.

__END
}

main();
1;
```

Und das war's auch schon 🙂

From:
http://koehlers.de/wiki/ - **Steffen Köhlers Online- Bastelbuch**

Permanent link:
http://koehlers.de/wiki/doku.php?id=misc:ledbadge

Last update: **2010/07/24 14:13**