rf433_PUT.ino

```
#include "rf433.h"
void setup()
{
   Serial.begin(9600);
   rf(-1,10);
}
byte buf[RF_MAX_LEN]{'s','i','s','ä','l','t','ö'};
int milli=millis();
void loop()
{
  if(millis()>milli+1000){
    milli=millis();
    rfPUT((byte*)buf, 7);
  }
  if(Serial.available()){
    Serial.print(Serial.readString());
  }
}
```

rf433_GET.ino

```
#include "rf433.h"
void setup()
{
   Serial.begin(9600);
   rf(11,-1);
}
byte buf[RF_MAX_LEN];
void loop()
{
   if (byte ile=rfGET(buf)){
    for (byte i = 0; i < ile; i++) Serial.print((char)buf[i]);
    Serial.println();
   }
}
```

rf433.h

```
#ifndef rf433_h
#define rf433_h
#include <Arduino.h>
#include <stdlib.h>

#undef abs
#undef double
#undef round

#define RF_MAX_LEN 30
#define VW_MAX_PAYLOAD RF_MAX_LEN-3
#define VW_RX_RAMP_LEN 160
#define VW_RX_SAMPLES_PER_BIT 8
#define VW_RAMP_INC (VW_RX_RAMP_LEN/VW_RX_SAMPLES_PER_BIT)
#define VW_RAMP_TRANSITION VW_RX_RAMP_LEN/2
```

```
#define VW_RAMP_ADJUST 9
#define VW_RAMP_INC_RETARD (VW_RAMP_INC-VW_RAMP_ADJUST)
#define VW_RAMP_INC_ADVANCE (VW_RAMP_INC+VW_RAMP_ADJUST)
#define VW_HEADER_LEN 8
extern "C"
{
   extern void rf(int rxPin,int txPin);
   extern uint8_t rfPUT(byte* buf, byte len);//(>RF_MAX_LEN - 3)?0:1
   extern byte rfGET(byte* buf);
}
#endif
```

rf433.cpp

```
#include "rf433.h"
#include <util/crc16.h>

static uint8_t vw_rx_pin = 11;
static uint8_t vw_tx_pin = 12;
static uint8_t vw_tx_buf[(RF_MAX_LEN * 2) + VW_HEADER_LEN] = {0x2a, 0x2a, 0x2a, 0x2a, 0x2a, 0x2a, 0x38, 0x2c};
static uint8_t vw_tx_len = 0;
static uint8_t vw_tx_index = 0;
static uint8_t vw_tx_bit = 0;
static uint8_t vw_tx_sample = 0;
static volatile uint8_t vw_tx_enabled = 0;
static uint16_t vw_tx_msg_count = 0;
static uint8_t vw_rx_sample = 0;
static uint8_t vw_rx_last_sample = 0;
static uint8_t vw_rx_pll_ramp = 0;
static uint8_t vw_rx_integrator = 0;
static uint8_t vw_rx_active = 0;
static volatile uint8_t vw_rx_done = 0;
static uint8_t vw_rx_enabled = 0;
static uint16_t vw_rx_bits = 0;
static uint8_t vw_rx_bit_count = 0;
static uint8_t vw_rx_buf[RF_MAX_LEN];
static uint8_t vw_rx_count = 0;
static volatile uint8_t vw_rx_len = 0;
static uint8_t vw_rx_bad = 0;
static uint8_t vw_rx_good = 0;
static uint8_t symbols[] ={0xd,  0xe,  0x13, 0x15, 0x16, 0x19, 0x1a, 0x1c, 0x23, 0x25, 0x26, 0x29, 0x2a, 0x2c, 0x32, 0x34};
extern "C"
{
uint16_t vw_crc(uint8_t *ptr, uint8_t count)
{
   uint16_t crc = 0xffff;
   while (count-- > 0) crc = _crc_ccitt_update(crc, *ptr++);
   return crc;
}
uint8_t vw_symbol_6to4(uint8_t symbol)
{
    uint8_t i;
    for (i = 0; i < 16; i++)if (symbol == symbols[i]) return i;
    return 0;
}
void vw_rx_start()
{
    if (!vw_rx_enabled)
    {
            vw_rx_enabled = true;
            vw_rx_active = false;
    }
}
void vw_rx_stop()
{
    vw_rx_enabled = false;
}
static uint8_t _timer_calc(uint16_t speed, uint16_t max_ticks, uint16_t *nticks)
{
    uint16_t prescalers[] = {0, 1, 8, 64, 256, 1024, 3333};
    uint8_t prescaler=0;
    unsigned long ulticks;
    if (speed == 0)
    {
        *nticks = 0;
        return 0;
    }
    for (prescaler=1; prescaler < 7; prescaler += 1)
    {
        float clock_time = (1.0 / (float(F_CPU) / float(prescalers[prescaler])));
        float bit_time = ((1.0 / float(speed)) / 8.0);
        ulticks = long(bit_time / clock_time);
        if ((ulticks > 1) && (ulticks < max_ticks))
        {
            break;
        }
    }
    if ((prescaler == 6) || (ulticks < 2) || (ulticks > max_ticks))
    {
        *nticks = 0;
        return 0;
    }
    *nticks = ulticks;
    return prescaler;
}
void vw_setup(uint16_t speed)
{
    uint16_t nticks;
    uint8_t prescaler;
    prescaler = _timer_calc(speed, (uint16_t)-1, &nticks);
    if (!prescaler) return;
    TCCR1A = 0;
    TCCR1B = _BV(WGM12);
    TCCR1B |= prescaler;
    OCR1A = nticks;
    #ifdef TIMSK1
      TIMSK1 |= _BV(OCIE1A);
            #else
      TIMSK |= _BV(OCIE1A);
            #endif
    pinMode(vw_tx_pin, OUTPUT);
    pinMode(vw_rx_pin, INPUT);
}
void rf(int rxPin,int txPin){
  vw_tx_pin = (uint8_t)txPin;
  vw_rx_pin = (uint8_t)rxPin;
  vw_setup(2000);
  if(rxPin!=-1)vw_rx_start();
}
void vw_pll()
{
    if (vw_rx_sample) vw_rx_integrator++;
    if (vw_rx_sample != vw_rx_last_sample)
    {
            vw_rx_pll_ramp += ((vw_rx_pll_ramp < VW_RAMP_TRANSITION)
                                        ? VW_RAMP_INC_RETARD
                                        : VW_RAMP_INC_ADVANCE);
            vw_rx_last_sample = vw_rx_sample;
    }
    else
    {
            vw_rx_pll_ramp += VW_RAMP_INC;
    }
    if (vw_rx_pll_ramp >= VW_RX_RAMP_LEN)
    {
            vw_rx_bits >>= 1;
            if (vw_rx_integrator >= 5)
                vw_rx_bits |= 0x800;
            vw_rx_pll_ramp -= VW_RX_RAMP_LEN;
            vw_rx_integrator = 0;
            if (vw_rx_active)
            {
                if (++vw_rx_bit_count >= 12)
                {
                        uint8_t this_byte =
                            (vw_symbol_6to4(vw_rx_bits & 0x3f)) << 4
                            | vw_symbol_6to4(vw_rx_bits >> 6);
                        if (vw_rx_len == 0)
                        {
                            vw_rx_count = this_byte;
                            if (vw_rx_count < 4 || vw_rx_count > RF_MAX_LEN)
                            {
                                    vw_rx_active = false;
                                    vw_rx_bad++;
                        return;
```

```
                     }
                  }
                  vw_rx_buf[vw_rx_len++] = this_byte;
                  if (vw_rx_len >= vw_rx_count)
                  {
                     vw_rx_active = false;
                     vw_rx_good++;
                     vw_rx_done = true;
                  }
                  vw_rx_bit_count = 0;
            }
         }
         else if (vw_rx_bits == 0xb38)
         {
            vw_rx_active = true;
            vw_rx_bit_count = 0;
            vw_rx_len = 0;
            vw_rx_done = false;
         }
   }
}
void vw_tx_start()
{
   vw_tx_index = 0;
   vw_tx_bit = 0;
   vw_tx_sample = 0;
   vw_tx_enabled = true;
}
void vw_tx_stop()
{
   digitalWrite(vw_tx_pin, false);
   vw_tx_enabled = false;
}
uint8_t vx_tx_active()
{
   return vw_tx_enabled;
}
void vw_wait_tx()
{
   while (vw_tx_enabled) ;
}
void vw_wait_rx()
{
   while (!vw_rx_done) ;
}
uint8_t vw_wait_rx_max(unsigned long milliseconds)
{
   unsigned long start = millis();
   while (!vw_rx_done && ((millis() - start) < milliseconds)) ;
   return vw_rx_done;
}
uint8_t rfPUT(byte* buf, byte len)
```

```
{
   uint8_t i;
   uint8_t index = 0;
   uint16_t crc = 0xffff;
   uint8_t *p = vw_tx_buf + VW_HEADER_LEN;
   uint8_t count = (uint8_t)(len + 3);
   if (len > VW_MAX_PAYLOAD) return false;
   vw_wait_tx();
   crc = _crc_ccitt_update(crc, count);
   p[index++] = symbols[count >> 4];
   p[index++] = symbols[count & 0xf];
   for (i = 0; i < len; i++)
   {
         crc = _crc_ccitt_update(crc, (uint8_t)buf[i]);
         p[index++] = symbols[(uint8_t)buf[i] >> 4];
         p[index++] = symbols[(uint8_t)buf[i] & 0xf];
   }
   crc = ~crc;
   p[index++] = symbols[(crc >> 4)  & 0xf];
   p[index++] = symbols[crc & 0xf];
   p[index++] = symbols[(crc >> 12) & 0xf];
   p[index++] = symbols[(crc >> 8)  & 0xf];
   vw_tx_len = index + VW_HEADER_LEN;
   vw_tx_start();
   return true;
}
byte rfGET(byte* buf)
{
   uint8_t rxlen;
   if (!vw_rx_done) return 0;
   rxlen = vw_rx_len - 3;
   memcpy(buf, vw_rx_buf + 1, RF_MAX_LEN);
   vw_rx_done = false;
   if((vw_crc(vw_rx_buf, vw_rx_len) == 0xf0b8)==0)return 0;
   return rxlen;
}
SIGNAL(TIMER1_COMPA_vect){
   if (vw_rx_enabled && !vw_tx_enabled) vw_rx_sample = digitalRead(vw_rx_pin);
   if (vw_tx_enabled && vw_tx_sample++ == 0){
         if (vw_tx_index >= vw_tx_len)
         {
            vw_tx_stop();
            vw_tx_msg_count++;
         }
         else
         {
            digitalWrite(vw_tx_pin, vw_tx_buf[vw_tx_index] & (1 << vw_tx_bit++));
            if (vw_tx_bit >= 6)
            {
                  vw_tx_bit = 0;
                  vw_tx_index++;
            }
            }
    }
    if (vw_tx_sample > 7) vw_tx_sample = 0;
    if (vw_rx_enabled && !vw_tx_enabled) vw_pll();
}
}
```