```c
/*
 *  Hamlib Rotator backend - GS-232
 *  Copyright (c) 2001-2010 by Stephane Fillod
 *  Copyright (c)      2009 by Jason Winningham
 *
 *   This library is free software; you can redistribute it and/or
 *   modify it under the terms of the GNU Lesser General Public
 *   License as published by the Free Software Foundation; either
.
 *
 *   This library is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 *   Lesser General Public License for more details.
 *
 *   You should have received a copy of the GNU Lesser General Public

0-1301  USA
 *
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>  /* String function definitions */
#include <unistd.h>  /* UNIX standard function definitions */
#include <math.h>

#include "hamlib/rotator.h"
#include "serial.h"
#include "misc.h"
#include "register.h"

#include "gs232a.h"

#define EOM "\r"
#define REPLY_EOM "\r"

#define BUFSZ 64

/**
 * gs232_transaction
 *
 * cmdstr - Command to be sent to the rig.
y is
 *        is needed, but answer will still be read.
vide
nd.
 *
 * returns:
 *   RIG_OK   -  if no error occured.
 *   RIG_EIO  -  if an I/O error occured while sending/receiving data.
.
not
 *                      recognized by rig.
 */
static int
gs232_transaction (ROT *rot, const char *cmdstr,
                                char *data, size_t data_len)
{
    struct rot_state *rs;
    int retval;
    int retry_read = 0;
    char replybuf[BUFSZ];

    rs = &rot->state;

transaction_write:

    serial_flush(&rs->rotport);

    if (cmdstr) {
        retval = write_block(&rs->rotport, cmdstr, strlen(cmdstr));
        if (retval != RIG_OK)
            goto transaction_quit;
    }

    /* Always read the reply to know whether the cmd went OK */
    if (!data)
            data = replybuf;
    if (!data_len)
            data_len = BUFSZ;

    memset(data,0,data_len);
    retval = read_string(&rs->rotport, data, data_len, REPLY_EOM, strlen(REPLY_EOM));
    if (retval < 0) {
        if (retry_read++ < rot->state.rotport.retry)
            goto transaction_write;
        goto transaction_quit;
    }

#if 0
    /* Check that command termination is correct */
    if (strchr(REPLY_EOM, data[strlen(data)-1])==NULL) {
ed '%s'\n", __FUNCTION__, data);
        if (retry_read++ < rig->state.rotport.retry)
            goto transaction_write;
        retval = -RIG_EPROTO;
        goto transaction_quit;
    }
#endif

    if (data[0] == '?') {
            /* Invalid command */
            rig_debug(RIG_DEBUG_VERBOSE, "%s: Error for '%s': '%s'\n",
                                    __FUNCTION__, cmdstr, data);
            retval = -RIG_EPROTO;
            goto transaction_quit;
    }

    retval = RIG_OK;
transaction_quit:
    return retval;
}



/*
 * write-only transaction, no data returned by controller
 */
static int
gs232_wo_transaction (ROT *rot, const char *cmdstr,
                                char *data, size_t data_len)
{
    return write_block(&rot->state.rotport, cmdstr, strlen(cmdstr));
}


static int
gs232_rot_set_position(ROT *rot, azimuth_t az, elevation_t el)
{
    char cmdstr[64];
    int retval;
    unsigned u_az, u_el;

    rig_debug(RIG_DEBUG_TRACE, "%s called: %f %f\n", __FUNCTION__, az, el);

    if (az < 0.0) az += 360.0; /* added to ensure proper use with DHBW Azimut-Rotor */
    u_az = (unsigned)rint(az);
    u_el = (unsigned)rint(el);

    sprintf(cmdstr, "W%03u %03u" EOM, u_az, u_el);
    retval = gs232_wo_transaction(rot, cmdstr, NULL, 0);

    if (retval != RIG_OK)
        return retval;

    return RIG_OK;
}

static int
gs232_rot_get_position(ROT *rot, azimuth_t *az, elevation_t *el)
{
    char posbuf[32];
    int retval;

    rig_debug(RIG_DEBUG_TRACE, "%s called\n", __FUNCTION__);

    retval = gs232_transaction(rot, "C2" EOM, posbuf, sizeof(posbuf));
```

```c
    if (retval != RIG_OK || strlen(posbuf) < 10)
        return retval;

    /* parse */
    if (sscanf(posbuf+2, "%f", az) != 1) {
        rig_debug(RIG_DEBUG_ERR, "%s: wrong reply '%s'\n", __FUNCTION__, posbuf);
        return -RIG_EPROTO;
    }
    if (*az > 180.0) *az -= 360.0; /* added to ensure proper use with DHBW Azimut-Rotor */
    if (sscanf(posbuf+7, "%f", el) != 1) {
        rig_debug(RIG_DEBUG_ERR, "%s: wrong reply '%s'\n", __FUNCTION__, posbuf);
        return -RIG_EPROTO;
    }

    rig_debug(RIG_DEBUG_TRACE, "%s: (az, el) = (%.1f, %.1f)\n",
               __FUNCTION__, *az, *el);

    return RIG_OK;
}

static int
gs232_rot_stop(ROT *rot)
{
    int retval;

    rig_debug(RIG_DEBUG_TRACE, "%s called\n", __FUNCTION__);

    /* All Stop */
    retval = gs232_wo_transaction(rot, "S" EOM, NULL, 0);
    if (retval != RIG_OK)
        return retval;

    return RIG_OK;
}


**** */
/*
 * Generic GS232 (not A, not B) rotator capabilities.
 */

const struct rot_caps gs232_rot_caps = {
  .rot_model =      ROT_MODEL_GS232,
  .model_name =     "GS-232",
  .mfg_name =       "Yaesu/Kenpro",
  .version =        "0.1",
  .copyright =      "LGPL",
  .status =         RIG_STATUS_BETA,
  .rot_type =       ROT_TYPE_AZEL,
  .port_type =      RIG_PORT_SERIAL,
  .serial_rate_min =   150,
  .serial_rate_max =   9600,
  .serial_data_bits =  8,
  .serial_stop_bits =  1,
  .serial_parity =     RIG_PARITY_NONE,
  .serial_handshake =  RIG_HANDSHAKE_NONE,
  .write_delay =  0,
  .post_write_delay =  0,
  .timeout =  400,
  .retry =  3,

  .min_az =     -180.0, r */
  .max_az =     450.0,  /* vary according to rotator type */
  .min_el =     0.0,
  .max_el =     180.0,

  .get_position =  gs232_rot_get_position,
  .set_position =  gs232_rot_set_position,
  .stop =              gs232_rot_stop,
};


**** */
/*
 * F1TE Tracker, GS232 withtout position feedback
 *
 * id=39
 */

const struct rot_caps f1tetracker_rot_caps = {
  .rot_model =      ROT_MODEL_F1TETRACKER,
  .model_name =     "GS232/F1TE Tracker",
  .mfg_name =       "F1TE",
  .version =        "0.1",
  .copyright =      "LGPL",
  .status =         RIG_STATUS_BETA,
  .rot_type =       ROT_TYPE_AZEL,
  .port_type =      RIG_PORT_SERIAL,
  .serial_rate_min =   150,
  .serial_rate_max =   9600,
  .serial_data_bits =  8,
  .serial_stop_bits =  1,
  .serial_parity =     RIG_PARITY_NONE,
  .serial_handshake =  RIG_HANDSHAKE_NONE,
  .write_delay =  0,
  .post_write_delay =  0,
  .timeout =  400,
  .retry =  0,

  .min_az =     0.0,
  .max_az =     360.0,  /* vary according to rotator type */
  .min_el =     0.0,
  .max_el =     180.0,

  .get_position =  NULL,    /* no position feedback available */
  .set_position =  gs232_rot_set_position,
#if 0
  .stop =              gs232_rot_stop,
#endif
};
```