Inhaltsverzeichnis Source Code AR5000RA

```
(*************************************************************************
Remote Control of Receiver AR5000 with frequency monitoring
for ionosperic or meteor scatter observation projects
Uses one Com-Port out of COM1 to COM8. Set AR5000 to 19200 baud.
*************************************************************************

Copyright 2010 - 2012 Wolfgang Kaufmann, Algermissen, Germany

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
any later version.

This program 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 General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.

*************************************************************************)
```

program AR5000RA;

```
{$mode objfpc} {$GOTO ON} {$APPTYPE GUI} {$H+}

uses
  Classes,
  WinCrt,
  synaser,
  sysutils,
  Graph;

const
     upperx1 = 13;      // Titel-Fenster
     upperx2 = 795;
     uppery1 = 0;
     uppery2 = 64;

     rxx1 = 350;        // Receiver-Fenster
     rxx2 = 795;
     rxy1 = 72;
     rxy2 = 200;

     frex1 = 5;        // Frequenzen-Fenster
     frex2 = 272;
     frey1 = 72;
     frey2 = 128;

     sigx1 = 5;        // Signal-Strength-Fenster
     sigx2 = 272;
     sigy1 = 136;
     sigy2 = 175;

     middlex1 = 5;      // Menue-Fenster
     middlex2 = 795;
     middley1 = 208;
     middley2 = 328;

     lowerx1 = 13;      // Ein-Ausgabefenster
     lowerx2 = 795;
     lowery1 = 340;
     lowery2 = 595;

     mboxx1 = 524;      // Messagebox
     mboxx2 = 790;
```

```
     mboxy1 = 340;
     mboxy2 = 444;

     scalex1 = 5;       // Frequenzskala
     scalex2 = 795;
     scaley1 = 430;
     scaley2 = 520;

     oldlevel: integer = 0;
     einschwingzeit = 500;


var
   ser: TBlockSerial;
   bandbreite: string;
   mode: string;
   agc: string;
   volume: string;
   ant: string;
   att: string;
   hpf: string;
   lpf: string;
   auto: string;
   schrittweite: string;
   shiftstep: string;
   frequenz: string;
   vfo: string;
   squelch: string;
   key: char;
   graphresult: boolean;
   comresult: boolean;
   comport: string;
   littfont: integer;
   tscrfont: integer;
```

function power(base: double; exp: integer): double;

```
var presult: double;
    i: integer;

begin
     presult := 1;
     for i:= 1 to exp do
     begin
          presult := presult * base;
     end;
     power := presult;
end;
```

function HexToDec(hx: string): integer;

```
var
   decimal: integer;
   flag: integer;

begin
    val(hx, decimal, flag);
    if flag <> 0 then
       begin
            if hx = 'A' then decimal := 10;
            if hx = 'B' then decimal := 11;
            if hx = 'C' then decimal := 12;
```

```
            if hx = 'D' then decimal := 13;
            if hx = 'E' then decimal := 14;
            if hx = 'F' then decimal := 15;
       end;
    HexToDec := decimal;
end;
```

function GetAGC: integer;

```
var
   hexlevel: string;
   hexlow: string;
   hexhigh: string;
   declow: integer;
   dechigh: integer;
   rawlevel: integer;
   linlevel: double;

begin
     ser.SendString('LM'+ CRLF);
     delay(50);
     hexlevel := ser.RecvString(500);
     ser.purge;
     hexhigh := copy(hexlevel, 4, 1);
     hexlow := copy (hexlevel, 5, 1);
     declow := HexToDec(hexlow);
     dechigh := HexToDec(hexhigh);
     rawlevel := declow + 16 * dechigh; // 0 - 255
     (* Kalibrieren: AGC-Kennlinie aus AOR bulletin, 2003 *)
     linlevel := (0.43970456*rawlevel) + (-0.010419413*power(rawlevel,2)) +
                 (0.00012850522*power(rawlevel,3)) + (-5.8958831E-7*power(rawlevel,4))
                 + (9.6151903E-10*power(rawlevel,5));
     (* Der Ddynamikumfang beträgt 113.63858 dB *)
     GetAGC := round(linlevel);
end;
```

procedure PrintAgc(flag: boolean; thislevel: integer);

```
const
     x1pos = 8;
     y1pos = 8;
     y2pos = 16;
     printcolor = lightgray;

var
   level: integer;
   x2pos: integer;
   oldx2pos: integer;
   i: byte;
   step: byte;
   memcolor: word;
   ttext: string;

begin
    setviewport (sigx1, sigy1, sigx2, sigy2, clipoff);
    settextstyle(littfont, horizdir, 4);
    settextjustify(centertext, toptext);
    setlinestyle(solidln, solidln, normwidth);
    rectangle(0, 0, (sigx2 - sigx1), (sigy2 - sigy1));
    memcolor := getcolor;
    setcolor(printcolor);

    (* Skala unterteilen, Skalenbreite 250 Pixel *)
```

```
    for i := 0 to 10 do
        begin
        step := i * 25;
        str((i*10), ttext);
        line((x1pos+step), y2pos, (x1pos+step), (y2pos+6));
        outtextxy((x1pos+step), (y2pos+7), ttext);
        end;

    if flag then
    begin
         level := GetAGC;  // AGC-Level holen
         level := (oldlevel + level) div 2; // gleitender Mittelwert
    end
    else level := thislevel;

    (* Bildschirmposition x-Achse, nur Umfang von 100 dB darstellen,
    darüber läuft der RX in die Sättigung *)
    x2pos := x1pos + round((level / 100) * 250);
    oldx2pos := x1pos + round((oldlevel / 100) * 250);

    (* Signalstärke anzeigen *)
    if x2pos < oldx2pos then
       begin
            setfillstyle(solidfill, getbkcolor); // löschen
            bar(x2pos, y1pos, oldx2pos, y2pos);
       end
    else
       begin
            setfillstyle(solidfill, printcolor); // neuzeichnen
            bar(x1pos, y1pos, x2pos, y2pos);
       end;

    oldlevel := level;
    setcolor(memcolor);
    settextstyle(defaultfont, horizdir, 1);
    settextjustify(lefttext, bottomtext);
end;
```

function ReadNumerical(xpos,ypos: integer): string;

```
(* wenn an beliebiger Stelle des Eingabevorgangs die Taste X gedrückt
   wird, wird die Eingabe abgebrochen und ein X zurückgeliefert *)

const
     delbackspace = 8;
     box = 219;
     xstep = 8;

var
     key: char;
     count: integer;
     indata: string;
     viewport: viewporttype;
     memcolor: word;
begin
     indata := '';
     getviewsettings(viewport);
     memcolor := getcolor;
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
           key := upcase(key);
           setviewport(viewport.x1, viewport.y1, viewport.x2,
                                    viewport.y2, viewport.clip);
           (* erster Fall: Zifferntaste gedrückt *)
```

```
           if ((key >= '0') and (key <= '9')) then
               begin
                    indata := indata + key;
                    outtextxy(xpos, ypos, key);
                    xpos := xpos + xstep;
               end;
           (* zweiter Fall: Delbackspace-Taste gedrückt *)
           if key = chr(delbackspace) then
               begin
               count := length(indata);
               if count > 1 then
                     begin
                     indata := copy(indata, 1, (count-1));
                     xpos := xpos - xstep;
                     setcolor(getbkcolor);
                     outtextxy(xpos, ypos, chr(box));
                     setcolor(memcolor);
                     end;
               if count = 1 then
                     begin
                     indata := '';
                     xpos := xpos - xstep;
                     setcolor(getbkcolor);
                     outtextxy(xpos, ypos, chr(box));
                     setcolor(memcolor);
                     end;
               end;
     until (key = CR) or (key = 'X');
     If key = 'X' then ReadNumerical := 'X' else ReadNumerical := indata;
end;
```

function FormatNumerical(numval: string; minval, maxval: qword): string;

```
var
   number: qword;
   code: integer;
   digit: byte;
   position : byte;
   i : byte;
   ttext: string;

begin
      (* Eingabe auf zulässigen Bereich korrigieren *)
      val(numval, number, code);
      if number < minval then number := minval;
      if number > maxval then number := maxval;
      (* Länge des Rückgabestrings aus maxval ermitteln *)
      str(maxval, ttext);
      digit := length(ttext);
      (* Rückgabstring mit führenden Nullen erzeugen *)
      str(number, ttext);
      for i := 1 to digit do
      begin
           ttext := '0' + ttext;
      end;
      (* Rückgabestring einkürzen *)
      position := length(ttext) - digit + 1;
      FormatNumerical := copy(ttext, position, digit);
end;
```

function GetScaleIndex(f: qword): byte;

```
begin
```

```
     (* die Skalenbreite ist fest und beträgt 2100 kHz bzw. 21 MHz *)
     (* der Index ist bis 31,5 MHz ein ganzzahliges Vielfaches von 2100 kHz *)
     if f < 31500000 then
        (* Lang-, Mittel- und Kurzwelle 0 - 31,5 MHz *)
        GetScaleIndex := f div 2100000
    else
        (* UKW 87 - 108 MHz *)
        if (f >= 87000000) and (f <= 108000000) then
            GetScaleIndex := 15
        else
            GetScaleIndex := 255;
end;
```

function GiveScalePos(freq: qword; index: byte; sscale, pscale, fscale: integer): integer;

```
var ffrac: longint;

begin
     (* Lang-, Mittel- und Kurzwellenskala 0 - 31,5 MHz *)
     if index < 15 then
     begin
          ffrac := (freq - (index * fscale * 1000));  // 0 - 2.100.000 Hz
          GiveScalePos := trunc(((ffrac / (fscale * 1000)) * pscale) + sscale);
     end
     else
     (* UKW-Skala von 87 bis 108 MHz, dh. Diff = 21 MHz *)
     begin
          ffrac := freq - 87000000;  // 0 - 21.000.000 Hz
          GiveScalePos := trunc(((ffrac / (fscale * 10000)) * pscale) + sscale);
     end;
end;
```

procedure OpenCom(var flag: boolean; var comport: string);

```
const linefeed = 16;

var
   portlist: string;
   comnr: char;
   index: byte;
   j: byte;

begin
     setviewport (middlex1, middley1, middlex2, middley2, clipoff);
     portlist := getserialportnames;
     index := pos('COM',portlist);
     if index > 0 then                    // es existiert mind. ein Com-Port
     begin
          outtextxy(50, linefeed, 'These serial ports have been found:');
          j := 0;
          repeat
                inc(j);
                index := pos('COM',portlist);
                comport := copy(portlist, index, 4);
                outtextxy(50, ((j+1)*linefeed), comport);
                delete(portlist, index, 4);
          until pos('COM',portlist) = 0;

          if j > 1 then                   // es existieren mehrere Com-Ports
          begin
               outtextxy(50, ((j+3)*linefeed), 'Enter the number of the serial port');
               outtextxy(50, ((j+4)*linefeed), 'connected to the AR5000 receiver (1 - 8)');
               repeat
```

```
                    comnr := readkey;
               until (comnr >= '1') and (comnr <= '8');
               comport := 'COM' + comnr;
          end;

          clearviewport;
          ser := TBlockserial.Create;
          ser.Connect(comport);
          ser.Config(19200,8,'N',0,true,false);
          delay (200);
          flag := true;
     end
     else
     begin                                  // es existiert kein Com-Port
          flag := false;
          outtextxy(50, linefeed, 'No serial port has been found!');
          outtextxy(50, (linefeed*2), 'Press any key');
          repeat
          until keypressed;
     end;
end;
```

procedure CloseCom;

```
begin
     (* Level-Squelch ausschalten *)
     ser.SendString('DB000' + CRLF);
     ser.purge;
     (* RX freigeben *)
     ser.SendString('EX'+CRLF);
     delay(200);
     (* Comport schliessen *)
     ser.CloseSocket;
end;
```

procedure SetScreen (var flag: boolean);

```
var
   gd,gm: smallint;
   path: string;

begin
    path := '';
    detectgraph (gd, gm);
    if gd >= d8bit then
    begin
         gm := m800x600;
         initGraph (gd, gm, path);

         SetBkColor(darkgray);
         SetColor(white);
         cleardevice;
         tscrfont := installuserfont('TSCR');
         littfont := installuserfont('LITT');
         settextjustify(lefttext, bottomtext);
         settextstyle(defaultfont, horizdir, 1);
         flag := true;
    end
    else
    begin
         initgraph(gd, gm, path);
         outtextxy(8, 10, 'Graphic adapter should at least');
         outtextxy(8, 26, 'support a resolution of 800 x 600');
```

```
        outtextxy(8, 42, 'pixel with 8bit color-depth');
        repeat
        until keypressed;
        flag := false;
    end;
end;
```

procedure PrintMenue;

```
const
     linefeed = 16;
     tab = 220;

begin
    setviewport (middlex1, middley1, middlex2, middley2, clipoff);
    setfillstyle(solidfill, green);
    bar(0, 0, (middlex2 - middlex1), (middley2 - middley1));

    outtextxy(0, (linefeed*2), ' f - Frequency ');
    outtextxy(0, (linefeed*3), ' s - Search ');
    outtextxy(0, (linefeed*4), ' q - Squelch ');
    outtextxy(0, (linefeed*5), ' r - HF Att ');
    outtextxy(0, (linefeed*6), ' a - Auto-Mode ');

    outtextxy(tab, (linefeed*2), ' v - VFO ');
    outtextxy(tab, (linefeed*3), ' m - Mode ');
    outtextxy(tab, (linefeed*4), ' i - IF Bandwidth ');
    outtextxy(tab, (linefeed*5), ' p - Step Size ');
    outtextxy(tab, (linefeed*6), ' j - Step Adjust ');

    outtextxy((tab*2), (linefeed*2), ' o - Volume ');
    outtextxy((tab*2), (linefeed*3), ' g - AGC ');
    outtextxy((tab*2), (linefeed*4), ' l - AF Low Pass ');
    outtextxy((tab*2), (linefeed*5), ' h - AF High Pass ');
    outtextxy((tab*2), (linefeed*6), ' n - Antenna ');

    outtextxy((tab*3), (linefeed*2), ' t - Monitor ');
    outtextxy((tab*3), (linefeed*3), ' c - Spectrum ');
    outtextxy((tab*3), (linefeed*4), ' e - Free RX ');
    outtextxy((tab*3), (linefeed*5), ' u - About ');
    outtextxy((tab*3), (linefeed*6), ' x - Exit ');
end;
```

procedure GetRX(status: byte);

```
var
   rx: string;
   shift : byte;

begin
(*  status = 0:
    Abfrage reduziert auf die Parameter, die vom Auto-Modus verändert werden
    status = 1:
    Abfrage aller RX-Parameter
    status = 2:
    Abfrage im Search-Modus *)

    if status = 2 then shift := 3 else shift := 0;
    ser.purge;
    (* Abfrage solange wiederholen, bis ein vollständiges Ergebnis vorliegt *)
    repeat
          ser.SendString('RX'+CR);
          rx := ser.RecvString(500);
```

```
          auto := copy (rx, (27+shift), 3);
    until (auto = 'AU0') or (auto = 'AU1');
    vfo := copy (rx, (1+shift), 2);
    if status < 2 then frequenz := copy (rx, 4, 12);
    schrittweite := copy (rx, (17+shift), 8);
    mode := copy (rx, (31+shift),3);

    (* Abfrage solange wiederholen, bis ein vollständiges Ergebnis vorliegt *)
    repeat
          ser.SendString('BW'+CR);
          bandbreite := ser.RecvString(500);
    until copy(bandbreite, 1, 2) = 'BW';

    ser.SendString('HP'+CR);
    hpf := ser.RecvString(500);

    ser.SendString('LP'+CR);
    lpf := ser.RecvString(500);

    if status = 1 then
        begin
             ser.SendString('AC'+CR);
             agc := ser.RecvString(500);

             ser.SendString('VL'+CR);
             volume := ser.RecvString(500);

             ser.SendString('AN'+CR);
             ant := ser.RecvString(500);
             ant := copy(ant, 1, 3);

             ser.SendString('AT'+CR);
             att := ser.RecvString(500);
             if copy(att,3,1) = '1' then
                att := 'ATF'
             else
                att := 'AT' + copy(att, 4, 1);

             ser.SendString('DB'+CR);
             squelch := ser.RecvString(500);

             ser.SendString('SH'+CR);
             shiftstep := ser.RecvString(500);
        end;
end;
```

procedure PrintFrequencies;

```
const
     linefeed = 16;
     tab = 8;

var
   f: string;
   xshift: integer;

begin
     setviewport (frex1, frey1, frex2, frey2, clipoff);
     clearviewport;
     setlinestyle(solidln, solidln, normwidth);
     rectangle(0, 0, (frex2 - frex1), (frey2 - frey1));

     f := copy(frequenz, 3, 1) + '.' + copy(frequenz, 4, 3) + '.'+
          copy(frequenz, 7, 3) + '.' + copy(frequenz,10,3);
     outtextxy(tab, linefeed, f);
     xshift := textwidth(f) + 8;
```

```
     outtextxy((xshift+tab), linefeed, 'Hz - Receive Freq');

     f := copy(schrittweite, 3, 3) + '.' + copy(schrittweite, 6, 3);
     outtextxy(54, (linefeed*3), f);
     outtextxy((xshift+tab), (linefeed*3), 'Hz - Step Size');
end;
```

procedure PrintTitle;

```
const linefeed = 32;

begin
     setviewport (upperx1, uppery1, upperx2, uppery2, clipoff);
     settextstyle(tscrfont, horizdir, 4);
     settextjustify(centertext, bottomtext);
     outtextxy(400, linefeed, 'AR5000 Communications Receiver');
     settextstyle(defaultfont, horizdir, 1);
     settextjustify(lefttext, bottomtext);
end;
```

procedure PrintRX;

```
const
     linefeed = 16;
     tab = 240;
var
   ttext: string;
   choice: integer;

begin
    setviewport (rxx1, rxy1, rxx2, rxy2, clipoff);
    clearviewport;

    ttext :=  'VFO: ' + copy(vfo, 2, 1);
    outtextxy(0, (linefeed*1), ttext);

    moveto(0, (linefeed*2));
    outtext('Mode: ');
    val(copy(mode, 3, 1), choice);
    case choice of
         0: outtext('FM ');
         1: outtext('AM ');
         2: outtext('LSB');
         3: outtext('USB');
         4: outtext('CW ');
    end;

    moveto(0, (linefeed*3));
    outtext('IF Bandwidth: ');
    val(copy(bandbreite, 3, 1), choice);
    case choice of
         1: outtext('3 kHz');
         2: outtext('6 kHz  ');
         3: outtext('15 kHz ');
         4: outtext('30 kHz ');
         5: outtext('110 kHz');
         6: outtext('220 kHz');
    end;

    moveto(0, (linefeed*4));
    outtext('AGC: ');
    if copy(agc, 3, 1) = 'F' then
       outtext('Off   ')
```

```
    else
        begin
        val(copy(agc, 3, 1), choice);
            case choice of
                 0: outtext('Fast  ');
                 1: outtext('Middle');
                 2: outtext('Slow  ');
            end;
        end;

    ttext :=  'Volume: ' +  copy(volume,3, 3);
    outtextxy(0, (linefeed*5), ttext);

    ttext :=  'Squelch: ' + copy(squelch,4,3);
    outtextxy(0, (linefeed*6), ttext);

    moveto(tab, (linefeed*1));
    outtext('Antenna: ');
    if copy(ant, 3, 1) = '0' then
       outtext('Auto')
    else
       begin
       ttext := copy(ant, 3, 1);
       outtext(ttext);
       end;

    moveto(tab, (linefeed*2));
    outtext('HF Attenuation: ');
    if copy(att, 3, 1) = 'F' then
       outtext('Auto')
    else
        begin
             val(copy(att, 3, 1), choice);
             case choice of
                  0: outtext('0 dB ');
                  1: outtext('10 dB');
                  2: outtext('20 dB');
             end;
        end;

    moveto(tab, (linefeed*3));
    outtext('AF High Pass: ');
    val(copy(hpf, 3, 1), choice);
    case choice of
         0: outtext('0,05 kHz');
         1: outtext('0,2 kHz ');
         2: outtext('0,3 kHz ');
         3: outtext('0,4 kHz ');
    end;

    moveto(tab, (linefeed*4));
    outtext('AF Low Pass: ');
    val(copy(lpf, 3, 1), choice);
    case choice of
         0: outtext('3,0 kHz');
         1: outtext('4,0 kHz');
         2: outtext('6,0 kHz');
         3: outtext('12 KHz ');
    end;

    ttext := copy(shiftstep, 3, 3) + '.' + copy(shiftstep, 6, 3);
    ttext :=  'Step Adjust: ' +  ttext + ' Hz';
    outtextxy(tab, (linefeed*5), ttext);

    moveto(tab, (linefeed*6));
    outtext('Auto-Mode: ');
    val(copy(auto, 3, 1), choice);
    case choice of
         0: outtext('OFF');
```

```
         1: outtext('ON ');
    end;
end;
```

procedure SetVfo;

```
const linefeed = 16;
var key: char;

begin
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, (linefeed*1), 'VFO:');
     outtextxy(0, (linefeed*2), 'A / B / C / D / E');
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
           key := upcase(key);
     until ((key >= 'A') and (key <= 'E'));
     vfo := 'V' + key;
     ser.SendString(vfo + CR);
     delay(einschwingzeit);
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetAuto;

```
const linefeed = 16;
var key: char;

begin
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, (linefeed*1), 'Auto-Mode:');
     outtextxy(0, (linefeed*2), '0 - OFF');
     outtextxy(0, (linefeed*3), '1 - ON');
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
     until ((key = '0') or (key = '1'));
     auto := 'AU' + key;
     ser.SendString(auto + CRLF);
     ser.purge;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetHighPass;

```
const linefeed = 16;
var key: char;

begin
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, (linefeed*1), 'AF High Pass:');
     outtextxy(0, (linefeed*2), '0 -  50 Hz');
     outtextxy(0, (linefeed*3), '1 - 200 Hz');
```

```
     outtextxy(0, (linefeed*4), '2 - 300 Hz');
     outtextxy(0, (linefeed*5), '3 - 400 Hz');
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
     until ((key >= '0') and (key <= '3'));
     hpf := 'HP' + key;
     ser.SendString(hpf + CRLF);
     ser.purge;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetLowPass;

```
const linefeed = 16;
var key: char;

begin
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, (linefeed*1), 'AF Low Pass:');
     outtextxy(0, (linefeed*2), '0 -  3 kHz');
     outtextxy(0, (linefeed*3), '1 -  4 kHz');
     outtextxy(0, (linefeed*4), '2 -  6 kHz');
     outtextxy(0, (linefeed*5), '3 - 12 kHz');
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
     until ((key >= '0') and (key <= '3'));
     lpf := 'LP' + key;
     ser.SendString(lpf + CRLF);
     ser.purge;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetAttenuation;

```
const linefeed = 16;
var key: char;

begin
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, (linefeed*1), 'HF Attenuation:');
     outtextxy(0, (linefeed*2), '0 -  0 dB');
     outtextxy(0, (linefeed*3), '1 - 10 dB');
     outtextxy(0, (linefeed*4), '2 - 20 dB');
     outtextxy(0, (linefeed*5), 'F - Auto-Attenuation ON');
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
        key := readkey;
        key := upcase(key);
     until ((key >= '0') and (key <= '2')) or (key = 'F');
     att := 'AT' + key;
     ser.SendString(att + CRLF);
     ser.purge;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
```

```
     clearviewport;
end;
```

procedure SetAntenna;

```
const linefeed = 16;
var key: char;

begin
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, (linefeed*1), 'Antenna:');
     outtextxy(0, (linefeed*2), '0 - Auto');
     outtextxy(0, (linefeed*3), '1 - Ant 1');
     outtextxy(0, (linefeed*4), '2 - Ant 2');
     outtextxy(0, (linefeed*5), '3 - Ant 3');
     outtextxy(0, (linefeed*6), '4 - Ant 4');
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
     until (key >= '0') and (key <= '4');
     ant := 'AN' + key;
     ser.SendString(ant + CRLF);
     ser.purge;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetAgcMode;

```
const linefeed = 16;
var key: char;

begin
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, (linefeed*1), 'AGC:');
     outtextxy(0, (linefeed*2), '0 - Fast');
     outtextxy(0, (linefeed*3), '1 - Middle');
     outtextxy(0, (linefeed*4), '2 - Slow');
     outtextxy(0, (linefeed*5), 'F - Off');
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
           key := upcase(key);
     until ((key >= '0') and (key <= '2')) or (key = 'F');
     agc := 'AC' + key;
     ser.SendString(agc + CRLF);
     ser.purge;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetMode;

```
const linefeed = 16;
var key: char;
```

```
begin
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, (linefeed*1), 'Mode:');
     outtextxy(0, (linefeed*2), '0 - FM');
     outtextxy(0, (linefeed*3), '1 - AM');
     outtextxy(0, (linefeed*4), '2 - LSB');
     outtextxy(0, (linefeed*5), '3 - USB');
     outtextxy(0, (linefeed*6), '4 - CW');
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
     until (key >= '0') and (key <= '4');
     mode := 'MD' + key;
     ser.SendString(mode + CRLF);
     ser.purge;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetBandwidth;

```
const linefeed = 16;
var key: char;

begin
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, (linefeed*1), 'IF Bandwidth:');
     outtextxy(0, (linefeed*2), '1 -   3 kHz');
     outtextxy(0, (linefeed*3), '2 -   6 kHz');
     outtextxy(0, (linefeed*4), '3 -  15  kHz');
     outtextxy(0, (linefeed*5), '4 -  30  kHz');
     outtextxy(0, (linefeed*6), '5 - 110 kHz');
     outtextxy(0, (linefeed*7), '6 - 220 kHz');
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
     until (key >= '0') and (key <= '6');
     bandbreite := 'BW' + key;
     ser.SendString(bandbreite + CRLF);
     ser.purge;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetStep;

```
const
     linefeed = 16;
     tab = 220;
var key: char;

begin
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);

     outtextxy(0, (linefeed*1), 'Step Size:');
     outtextxy(0, (linefeed*2), 'a -   1 Hz');
     outtextxy(0, (linefeed*3), 'b -  10 Hz');
     outtextxy(0, (linefeed*4), 'c -  50 Hz');
     outtextxy(0, (linefeed*5), 'd - 100 Hz');
```

```
     outtextxy(0, (linefeed*6), 'e - 500 Hz');

     outtextxy(tab, (linefeed*2), 'f -  1 kHz');
     outtextxy(tab, (linefeed*3), 'g -  5 kHz');
     outtextxy(tab, (linefeed*4), 'h -  9 kHz');
     outtextxy(tab, (linefeed*5), 'i - 10 kHz');
     outtextxy(tab, (linefeed*6), 'j - 12,5 kHz');

     outtextxy((tab*2), (linefeed*2), 'k -  20 kHz');
     outtextxy((tab*2), (linefeed*3), 'l -  25 kHz');
     outtextxy((tab*2), (linefeed*4), 'm -  30 kHz');
     outtextxy((tab*2), (linefeed*5), 'n -  50 kHz');
     outtextxy((tab*2), (linefeed*6), 'o - 100 kHz');

     outtextxy((tab*3), (linefeed*2), 'p - 500 kHz');

     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
           key := upcase(key);
     until (key >= 'A') and (key <= 'P');
     case key of
          'A': schrittweite := '000001';
          'B': schrittweite := '000010';
          'C': schrittweite := '000050';
          'D': schrittweite := '000100';
          'E': schrittweite := '000500';
          'F': schrittweite := '001000';
          'G': schrittweite := '005000';
          'H': schrittweite := '009000';
          'I': schrittweite := '010000';
          'J': schrittweite := '012500';
          'K': schrittweite := '020000';
          'L': schrittweite := '025000';
          'M': schrittweite := '030000';
          'N': schrittweite := '050000';
          'O': schrittweite := '100000';
          'P': schrittweite := '500000';
     end;
     schrittweite := 'ST' + schrittweite;
     ser.SendString(schrittweite + CRLF);
     ser.purge;
     (* Step-Adjust ausschalten *)
     ser.SendString('SH000000' + CRLF);
     ser.purge;

     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetSquelch;

```
const linefeed = 16;
var   numval: string;
      ttext: string;

begin
     ttext := 'Enter Squelch Level (0 - 255): ';
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, linefeed, ttext);
     (* Eingabe-Aufforderung wiederholen falls Leerstring *)
     repeat
           numval := ReadNumerical(textwidth(ttext), linefeed);
     until length(numval) > 0;
```

```
     (* Eingabe auf erlaubte Wert prüfen und formatieren*)
     if not (numval = 'X') then
     begin
          numval := FormatNumerical(numval, 0, 255);
          squelch := 'DB' + numval;
          ser.SendString(squelch + CRLF);
          ser.purge;
     end;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetVolume;

```
const linefeed = 16;
var   numval: string;
      ttext: string;

begin
     ttext := 'Enter Volume (0 - 255): ';
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, linefeed, ttext);
     (* Eingabe-Aufforderung wiederholen falls Leerstring *)
     repeat
           numval := ReadNumerical(textwidth(ttext), linefeed);
     until length(numval) > 0;
     (* Eingabe auf erlaubte Wert prüfen und formatieren*)
     if not (numval = 'X') then
     begin
          numval := FormatNumerical(numval, 0, 255);
          volume := 'VL' + numval;
          ser.SendString(volume + CRLF);
          ser.purge;
     end;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetFrequency;

```
const linefeed = 16;
var   numval: string;
      ttext: string;

begin
     ttext := 'Enter receive frequency (kHz as integer): ';
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, linefeed, ttext);
     (* Eingabe-Aufforderung wiederholen falls Leerstring *)
     repeat
           numval := ReadNumerical(textwidth(ttext), linefeed);
     until length(numval) > 0;
     (* Eingabe auf erlaubte Wert prüfen und formatieren*)
     if not (numval = 'X') then
     begin
          numval := numval + '000'; // von kHz in Hz umwandeln
          numval := FormatNumerical(numval, 10000, 2600000000);
          frequenz := 'RF' + numval;
          ser.SendString(frequenz + CRLF);
          delay(einschwingzeit);
          ser.purge;
     end;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
```

```
end;
```

procedure PrintBand(bandstart, bandende: qword; index: byte; sscale, pscale, fscale: integer; farbe: word);

```
const y1 = 54;
      y2 = 61;

var   bstart: integer;    // pixel
      bende: integer;     // pixel
begin
      bstart := GiveScalePos(bandstart, index, sscale, pscale, fscale);
      bende := GiveScalePos(bandende, index, sscale, pscale, fscale);
      setfillstyle(solidfill, farbe);
      bar(bstart, y1, bende, y2);
end;
```

procedure AddBands(index: byte; sscale, pscale, fscale: integer);

```
const rfarbe = 49; //lightgreen
      hfarbe = 52; //lightcyan
var   bandstart: qword;   // Hz
      bandende: qword;    // Hz

begin
     case index of
     0: begin
             bandstart := 153000;         // LW
             bandende := 285000;
             PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
             bandstart := 531000;         // MW
             bandende := 1700000;
             PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
             bandstart := 1810000;        // AFU 160m
             bandende := 2000000;
             PrintBand(bandstart, bandende, index, sscale, pscale, fscale, hfarbe);
        end;
     1: begin
             bandstart := 2300000;        // 120m
             bandende := 2495000;
             PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
             bandstart := 3200000;        // 90m
             bandende := 3400000;
             PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
             bandstart := 3500000;        // AFU 80m
             bandende := 3800000;
             PrintBand(bandstart, bandende, index, sscale, pscale, fscale, hfarbe);
             bandstart := 3900000;        // 75m
             bandende := 4000000;
             PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
        end;
     2: begin
             bandstart := 4750000;        // 60 m
             bandende := 5060000;
             PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
             bandstart := 5900000;        // 49 m
             bandende := 6200000;
             PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
        end;
     3: begin
             bandstart := 7000000;        // AFU 40 m
             bandende := 7100000;
             PrintBand(bandstart, bandende, index, sscale, pscale, fscale, hfarbe);
```

```
          bandstart := 7200000;        // 41 m
          bandende := 7450000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
     end;
  4: begin
          bandstart := 9400000;        // 31 m
          bandende := 9900000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
          bandstart := 10100000;       // AFU 30 m
          bandende := 10150000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, hfarbe);
     end;
  5: begin
          bandstart := 11600000;       // 25 m
          bandende := 12100000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
     end;
  6: begin
          bandstart := 13570000;       // 22 m
          bandende := 13870000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
          bandstart := 14000000;       // AFU 20 m
          bandende := 14350000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, hfarbe);
     end;
  7: begin
          bandstart := 15100000;       // 19 m
          bandende := 15800000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
     end;
  8: begin
          bandstart := 17480000;       // 16 m
          bandende := 17900000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
          bandstart := 18068000;       // AFU 17 m
          bandende := 18168000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, hfarbe);
     end;
  9: begin
          bandstart := 18900000;       // 15 m
          bandende := 19020000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
     end;
 10: begin
          bandstart := 21000000;       // AFU 15 m
          bandende := 21450000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, hfarbe);
          bandstart := 21450000;       // 13 m
          bandende := 21850000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
      end;
 11: begin
          bandstart := 24890000;        // AFU 12 m
          bandende := 24990000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, hfarbe);
     end;
 12: begin
          bandstart := 25670000;        // 11 m
          bandende := 26100000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, rfarbe);
     end;
 13: begin
          bandstart := 28000000;        // AFU 10 m, erster Teil
          bandende := 29400000;
          PrintBand(bandstart, bandende, index, sscale, pscale, fscale, hfarbe);
     end;
 14: begin
          bandstart := 29400000;        // AFU 10, zweiter Teil
          bandende := 29700000;
```

```
            PrintBand(bandstart, bandende, index, sscale, pscale, fscale, hfarbe);
        end;
    end;

end;
```

procedure PrintSteeringKey;

```
const linefeed = 16;
      leftarrow = chr(27);
      rightarrow = chr(26);

begin
     (* Skalenanzeige löschen *)
     setviewport(scalex1, scaley1, scalex2, scaley2, clipoff);
     clearviewport;
     setviewport((lowerx1 - 8), lowery1, lowerx2, lowery2, clipoff);
     (* Schaltflächen zeichnen *)
     setfillstyle(solidfill, lightgray);
     bar(320,(linefeed*7),370,(linefeed*10));
     bar(410, (linefeed*7), 460, (linefeed*10));
     (* Pfeilzeichen ausgeben *)
     settextstyle(defaultfont, horizdir, 2);
     outtextxy(340, (linefeed*9), leftarrow);
     outtextxy (430, (linefeed*9), rightarrow);
     settextstyle(defaultfont, horizdir, 1);
end;
```

procedure PrintFrame(index: byte; sscale, pscale, fscale: integer);

```
const linefeed = 16;
      scaleback = 97; // whitegreen
      scalefront = 120; // darkgreen
      textcolor = 16; // black

var
   memcolor: word;
   f: string;
   stepscale: integer;
   i: integer;

begin
     setviewport(scalex1, scaley1, scalex2, scaley2, clipoff);
     memcolor := getcolor;

     (* Hintergrund füllen *);
     setfillstyle(solidfill, scaleback);
     bar (0, 0, (scalex2 - scalex1), (scaley2 - scaley1));
     (* Skalenbezeichner schreiben *)
     setcolor(textcolor);
     case index of
     0: begin
            outtextxy(8, (linefeed*2), '  LW');
            outtextxy(8, (linefeed*3), '  MW');
            outtextxy(8, (linefeed*4), '160m');
        end;
     1: begin
            outtextxy(8, (linefeed*1), '120m');
            outtextxy(8, (linefeed*2), ' 90m');
            outtextxy(8, (linefeed*3), ' 80m');
            outtextxy(8, (linefeed*4), ' 75m');
        end;
     2: begin
            outtextxy(8, (linefeed*3), ' 60m');
```

```
            outtextxy(8, (linefeed*4), ' 49m');
         end;
      3: begin
            outtextxy(8, (linefeed*3), ' 40m');
            outtextxy(8, (linefeed*4), ' 41m');
         end;
      4: begin
            outtextxy(8, (linefeed*3), ' 31m');
            outtextxy(8, (linefeed*4), ' 30m');
         end;
      5: outtextxy(8, (linefeed*4), ' 25m');
      6: begin
            outtextxy(8, (linefeed*3), ' 22m');
            outtextxy(8, (linefeed*4), ' 20m');
         end;
      7: outtextxy(8, (linefeed*4), ' 19m');
      8: begin
            outtextxy(8, (linefeed*3), ' 16m');
            outtextxy(8, (linefeed*4), ' 17m');
         end;
      9: outtextxy(8, (linefeed*4), ' 15m');
      10: begin
             outtextxy(8, (linefeed*3), ' 15m');
             outtextxy(8, (linefeed*4), ' 13m');
          end;
      11: outtextxy(8, (linefeed*4), ' 12m');
      12: outtextxy(8, (linefeed*4), ' 11m');
      13: outtextxy(8, (linefeed*4), ' 10m');
      14: outtextxy(8, (linefeed*4), ' 10m');
      15: outtextxy(8, (linefeed*4), ' UKW');
      end;
      (* Startfrequenz eintragen *)
      setcolor(scalefront);
      settextjustify(lefttext, bottomtext);
      if index = 15 then f := '87'
      else str((index * fscale), f);
      outtextxy((sscale + 1), (linefeed*5), f);
      (* Endfrequenz eintragen *)
      settextjustify(righttext, bottomtext);
      if index = 15 then f := '108'
      else str(((index+1) * fscale), f);
      outtextxy((sscale + pscale), (linefeed*5), f);
      (* Basislinie einzeichnen *);
      setfillstyle(solidfill, scalefront);
      bar(sscale, 50, (sscale + pscale), 65);
      (* Schrittweite Basislinien-Unterteilung (100 kHz- bzw. 1 MHz-Schritte *)
      stepscale := 735 div 21;
      i := sscale + stepscale;
      setlinestyle(solidln, solidln, thickwidth);
      repeat
            line(i, 50, i, 43);
            i := i + stepscale;
      until i >= (sscale + pscale);
      (* erste und letzte Unterteilung einzeln eintragen *)
      i := sscale + 1;
      line(i, 50, i, 43);
      i := (sscale + pscale) - 1;
      line(i, 50, i, 43);
      (* Textausgabe-Einstellungen zurücksetzen *)
      settextjustify(lefttext, bottomtext);
      setcolor(memcolor);
end;
```

procedure PrintPointer(freq: qword; index: byte; sscale, pscale, fscale: integer);

```
const pointercolor = lightred;
```

```
var pointerpos: integer;

begin
     pointerpos := GiveScalePos(freq, index, sscale, pscale, fscale);
     setfillstyle(solidfill, pointercolor);
     bar((pointerpos - 1), 0, (pointerpos + 1), (scaley2 - scaley1));
end;
```

procedure PrintScale;

```
const
     sscale = 52;     // Start der Frequenzskala
     pscale = 735;    // Länge der Frequenzskala: 735 pixel
     fscale = 2100;   // Länge der Frequenzskala: 2100 kHz (UKW: 21 MHz)

var freq: qword;
    code: integer;
    index: byte;

begin
     val(copy(frequenz, 3,10), freq, code);  // Hz
     (* Prüfen ob für Empfangsfrequenz anzeigbare Skalen existieren *)
     index := GetScaleIndex(freq);
     if index < 255 then
     begin
          PrintFrame(index, sscale, pscale, fscale);
          AddBands(index, sscale, pscale, fscale);
          PrintPointer(freq, index, sscale, pscale, fscale);
     end
     else
     begin
          PrintSteeringKey;
     end;
end;
```

procedure PrintStepStatus(multiplier: boolean);

```
const
     box = 219;
     linefeed = 16;
     status: string = '10-fold step !';

var  memcolor: word;
     i: integer;

begin
     setviewport((lowerx1 - 8), lowery1, lowerx2, lowery2, clipoff);
     memcolor := getcolor;

     (* Zeile löschen *)
     setcolor(getbkcolor);
     i := 340;
     repeat
           outtextxy(i, linefeed*4, chr(box));
           i := i + 8;
     until i > 490;

     (* Multiplier anzeigen *)
     if multiplier then
     begin
           setcolor(yellow);
           outtextxy(340, (linefeed*4), status);
     end;
```

```
     setcolor(memcolor);
end;
```

procedure IncFrequ(multiplier: boolean);

```
const stepup = chr(30);
var   i: byte;

begin
    (* Up-Freq an RX senden: bei multiplier = true mit 10facher Schrittweite  *)
    if multiplier then
    begin
         for i := 1 to 10 do ser.SendString(stepup + CR);
    end
    else ser.SendString(stepup + CR);
    (* Anzeige aktualisieren *)
    GetRX(0);
    if auto = 'AU1' then PrintRX;
    PrintFrequencies;
    PrintScale;
end;
```

procedure DecFrequ(multiplier: boolean);

```
const stepdown = chr(31);
var   i: byte;

begin
    (* Down-Freq an RX senden: bei multiplier = true mit 10facher Schrittweite *)
    if multiplier then
    begin
         for i := 1 to 10 do ser.SendString(stepdown + CR);
    end
    else ser.SendString(stepdown + CR);
    (* Anzeige aktualisieren *)
    GetRX(0);
    if auto = 'AU1' then PrintRX;
    PrintFrequencies;
    PrintScale;
end;
```

procedure TuneRX;

```
const up = 77;
      down = 75;
      menuekeys ='ACEFGHIJLMNOPQRSTUVX';
var
   code: byte;
   multiplier: boolean;

begin
     key := ' ';
     multiplier := false;
     PrintScale;
     repeat
           code := ord(key);
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
```

```
          key := upcase(key);
          if key = '*' then
             if multiplier = true then multiplier := false
             else multiplier := true;
          PrintStepStatus(multiplier);
          if ((ord(key) = up) and (code = 0)) then IncFrequ(multiplier);
          if ((ord(key) = down) and (code = 0)) then DecFrequ(multiplier);
     until ((pos(key, menuekeys) > 0) and (code <> 0));
     setviewport((lowerx1 - 8), lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure StepShift;

```
const linefeed = 16;
var   ttext: string;
      numval: string;

begin
     ttext := 'Enter Step Adjust (0 - 999999 Hz): ';
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, linefeed, ttext);
     (* Eingabe-Aufforderung wiederholen falls Leerstring *)
     repeat
          numval := ReadNumerical(textwidth(ttext), linefeed);
     until length(numval) > 0;
     (* Eingabe auf erlaubte Wert prüfen und formatieren*)
     if not (numval = 'X') then
     begin
          numval := FormatNumerical(numval, 0, 999999);
          shiftstep := 'SH' + numval;
          ser.SendString(shiftstep + CRLF);
          ser.purge;
     end;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure DoSearch;

```
const linefeed = 16;
      continue = chr(30);
var
      key: char;
      busy: string;
      hexhigh, hexlow: string;
      dechigh, declow: integer;
      rawlevel: integer;
      linlevel: double;

begin
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);

      (* Send agc-level and frequeny when Squelch is on *);
      ser.SendString('LC1' + CRLF);
      ser.purge;

      (* Search starten *)
      ser.SendString('VSA' + CR);

      (* Search steuern *)
      outtextxy(0, (linefeed*5), 'c - Continue');
      outtextxy(0, (linefeed*6), 'x - Exit');
```

```
      repeat
            repeat
                  repeat
                        busy := ser.RecvString(500);
                        (* AGC-Anzeige aktualisieren *)
                        if copy(busy,1,2) = 'LC' then
                        begin
                             hexhigh := copy(busy, 3, 1);
                             hexlow := copy (busy, 4, 1);
                             declow := HexToDec(hexlow);
                             dechigh := HexToDec(hexhigh);
                             rawlevel := declow + 16 * dechigh; // 0 - 255
                             (* Kalibrieren: AGC-Kennlinie aus AOR bulletin, 2003 *)
                             linlevel := (0.43970456*rawlevel) +
                             (-0.010419413*power(rawlevel,2)) +
                             (0.00012850522*power(rawlevel,3)) +
                             (-5.8958831E-7*power(rawlevel,4)) +
                             (9.6151903E-10*power(rawlevel,5));
                             (* Der Dynamikumfang beträgt 113.63858 dB *)
                             PrintAgc(false, round(linlevel));
                        end;
                        (* Frequenzanzeige aktualisieren *)
                        if copy(busy,1,2) = 'RF' then
                        begin
                             frequenz := busy;
                             GetRX(2);
                             PrintFrequencies;
                             PrintRX;
                        end;
                  until keypressed;
                  key := readkey;
                  key := upcase(key);
            until (key = 'C') or (key = 'X');
            if key = 'C' then  ser.SendString(continue + CR);
      until key = 'X';

      (* Mute RX *)
      ser.SendString('VL000' + CRLF);
      ser.purge;
      (* Search beenden *)
      ser.SendString('VA' + CR);
      delay(einschwingzeit);
      (* Stop sending agc-Level and frequency when Squelch is on *);
      ser.SendString('LC0' + CRLF);
      ser.purge;
      (* Set last receive frequency *);
      ser.SendString(frequenz + CRLF);
      delay(einschwingzeit);
      ser.purge;
      (* Restore Volume *)
      ser.SendString(volume + CRLF);
      ser.purge;

      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      clearviewport;
end;
```

procedure PrepSearch(fstart, fstop: string);

```
const linefeed = 16;

begin
      outtextxy(0, (linefeed*4), 'Search-Mode:');
     (* Mute RX *)
      ser.SendString('VL000' + CRLF);
      ser.purge;
```

```
     (* Start-Frequenz in VFO A setzen *)
      ser.SendString('VA' + CR);
      delay(einschwingzeit);
      fstart := 'RF' + fstart;
      ser.SendString(fstart + CRLF);
      delay(einschwingzeit);
      ser.purge;
     (* Stop-Frequenz in VFO B setzen *)
      ser.SendString('VB' + CR);
      delay(einschwingzeit);
      fstop := 'RF' + fstop;
      ser.SendString(fstop + CRLF);
      delay(einschwingzeit);
      ser.purge;
     (* Restore Volume *)
      ser.SendString(volume + CRLF);
      ser.purge;
end;
```

procedure GetSearch(var fstart, fstop: string; var flag: boolean);

```
label _Beenden;

const
     linefeed = 16;
     starttext = 'Enter lower frequency limit (kHz as integer): ';
     endtext = 'Enter upper frequency limit (kHz as integer): ';
var
   fstr: string;
   key: char;

begin
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      (* Hinweis ausgeben *)
      outtextxy(0,(linefeed*1), 'Channel search will be performed with the current');
      outtextxy(0,(linefeed*2), 'settings of VFO A and squelch. Continue (y/n)?');
      repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
           key := upcase(key);
      until (key = 'Y') or (key = 'N');
      if key = 'N' then
         begin
              flag := false;
              goto _Beenden;
         end;

      (* Startfrequenz holen *)
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      clearviewport;

      outtextxy(0, (linefeed*1), starttext);
      repeat
            fstr := ReadNumerical(textwidth(starttext), (linefeed*1));
      until length(fstr) > 0;
      if fstr = 'X' then
         begin
              flag := false;
              goto _Beenden;
         end;
      fstart := FormatNumerical((fstr + '000'), 10000, 2600000000);

      (* Endfrequenz holen *)
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
```

```
      outtextxy(0, (linefeed*2), endtext);
      repeat
            fstr := ReadNumerical(textwidth(endtext), (linefeed*2));
      until length(fstr) > 0;
      if fstr = 'X' then
         begin
              flag := false;
              goto _Beenden;
         end;
      fstop := FormatNumerical((fstr + '000'), 10000, 2600000000);

      _Beenden:
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
end;
```

procedure ChannelSearch;

```
var
   fstart: string;
   fstop: string;
   flag: boolean;

begin
     flag := true;
     GetSearch(fstart, fstop, flag);
     if flag then
     begin
          PrepSearch(fstart, fstop);
          DoSearch;
     end;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure PrintMessageA;

```
const linefeed = 16;
var   key: char;
      memcolor: word;

begin
     setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
     memcolor := getcolor;
     setcolor(yellow);
     setlinestyle(solidln, solidln, normwidth);
     rectangle(0, 0, (mboxx2 - mboxx1), (mboxy2 - mboxy1));

     outtextxy(16, (linefeed*1), 'Selected range of frequency');
     outtextxy(16, (linefeed*2), 'exceeds graphical resolution.');
     outtextxy(16, (linefeed*3), 'Maximum range is 156 MHz.');
     outtextxy(16, (linefeed*4), 'Choose other limits.');
     outtextxy(16, (linefeed*6), 'Press <Enter> to continue ..');
     setcolor(memcolor);
     repeat
          repeat
               PrintAgc(true, 0);
          until keypressed;
          key := readkey;
      until key = CR;

      setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
      clearviewport;
end;
```

procedure PrintMessageB;

```
const linefeed = 16;
var   key: char;
      memcolor: word;

begin
     setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
     memcolor := getcolor;
     setcolor(yellow);
     setlinestyle(solidln, solidln, normwidth);
     rectangle(0, 0, (mboxx2 - mboxx1), (mboxy2 - mboxy1));

     outtextxy(8, (linefeed*1), 'Selected spectral resolution');
     outtextxy(8, (linefeed*2), 'exceeds grahical resolution.');
     outtextxy(8, (linefeed*3), 'Maximum is a set of 780 frequen-');
     outtextxy(8, (linefeed*4), 'cies. Choose a lower resolution.');
     outtextxy(8, (linefeed*6), 'Press <Enter> to continue ..');
     setcolor(memcolor);
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
      until key = CR;

      setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
      clearviewport;
end;
```

procedure PrintMessageC;

```
const linefeed = 16;
var   key: char;
      memcolor: word;

begin
     setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
     memcolor := getcolor;
     setcolor(yellow);
     setlinestyle(solidln, solidln, normwidth);
     rectangle(0, 0, (mboxx2 - mboxx1), (mboxy2 - mboxy1));

     outtextxy(8, (linefeed*1), 'Selected sampling rate');
     outtextxy(8, (linefeed*2), 'falls short of integration time');
     outtextxy(8, (linefeed*3), 'with respect to number of VFOs.');
     outtextxy(8, (linefeed*4), 'Choose a lower sampling rate.');
     outtextxy(8, (linefeed*6), 'Press <Enter> to continue ..');
     setcolor(memcolor);
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
      until key = CR;

      setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
      clearviewport;
end;
```

procedure PrintMessageD;

```
const linefeed = 16;
var   key: char;
      memcolor: word;

begin
     setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
     memcolor := getcolor;
     setcolor(yellow);
     setlinestyle(solidln, solidln, normwidth);
     rectangle(0, 0, (mboxx2 - mboxx1), (mboxy2 - mboxy1));

     outtextxy(8, (linefeed*1), 'Selected cycle rate is too');
     outtextxy(8, (linefeed*2), 'short for the frequency range');
     outtextxy(8, (linefeed*3), 'with respect to resolution.');
     outtextxy(8, (linefeed*4), 'Choose a higher cycle rate.');
     outtextxy(8, (linefeed*6), 'Press <Enter> to continue ..');
     setcolor(memcolor);
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
      until key = CR;

      setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
      clearviewport;
end;
```

procedure PrintMessageE;

```
const linefeed = 16;
var   key: char;
      memcolor: word;

begin
     setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
     memcolor := getcolor;
     setcolor(yellow);
     setlinestyle(solidln, solidln, normwidth);
     rectangle(0, 0, (mboxx2 - mboxx1), (mboxy2 - mboxy1));
     outtextxy(8, (linefeed*1), 'Responsivity (classification');
     outtextxy(8, (linefeed*2), 'for central Europe):');
     outtextxy(8, (linefeed*3), '3 - very high');
     outtextxy(8, (linefeed*4), '5 - high');
     outtextxy(8, (linefeed*5), '7 - standard');
     outtextxy(8, (linefeed*6), '9 - low');
     setcolor(memcolor);
     setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
end;
```

procedure PrintMessageF;

```
const linefeed = 16;
var   key: char;
      memcolor: word;

begin
     setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
     memcolor := getcolor;
     setcolor(yellow);
     setlinestyle(solidln, solidln, normwidth);
     rectangle(0, 0, (mboxx2 - mboxx1), (mboxy2 - mboxy1));
```

```
     outtextxy(8, (linefeed*1), 'Selected sampling rate falls');
     outtextxy(8, (linefeed*2), 'short of period of observation');
     outtextxy(8, (linefeed*3), 'with respect to number of VFOs.');
     outtextxy(8, (linefeed*4), 'Choose a lower sampling rate.');
     outtextxy(8, (linefeed*6), 'Press <Enter> to continue ..');
     setcolor(memcolor);
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
      until key = CR;

      setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
      clearviewport;
end;
```

procedure GetSpectralParameter(var fstart, fstop, resolution: qword;
var ordinalres: char; var flag: boolean);

```
label _Beenden;

const
     linefeed = 16;
     starttext = 'Enter lower frequency limit (kHz as integer): ';
     endtext = 'Enter upper frequency limit (kHz as integer): ';
     (* 156000000 Hz ergeben bei 200 kHz Auflösung 780 Messpunkte *)
     maxbereich = 156000000;
     maxmesspunkte = 780;
var
   fstr: string;
   key: char;
   code: integer;

begin
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      (* Hinweis ausgeben *)
      outtextxy(0,(linefeed*1), 'Spectral analysis will be performed with the');
      outtextxy(0,(linefeed*2), 'currently selected antenna. Continue (y/n)?');
      repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
           key := upcase(key);
      until (key = 'Y') or (key = 'N');
      if key = 'N' then
         begin
              flag := false;
              goto _Beenden;
         end;

      repeat
            (* Startfrequenz holen *)
            setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
            clearviewport;

            outtextxy(0, (linefeed*1), starttext);
            repeat
                  fstr := ReadNumerical(textwidth(starttext), (linefeed*1));
            until length(fstr) > 0;
            if fstr = 'X' then
               begin
                    flag := false;
                    goto _Beenden;
               end;
```

```
            val((fstr + '000'), fstart, code); // khz -> Hz
            if fstart < 10000 then fstart := 10000;  // untere RX-Frequenzgrenze

            (* Endfrequenz holen *)
            setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
            outtextxy(0, (linefeed*2), endtext);
            repeat
                  fstr := ReadNumerical(textwidth(endtext), (linefeed*2));
            until length(fstr) > 0;
            if fstr = 'X' then
               begin
                    flag := false;
                    goto _Beenden;
               end;
            val((fstr + '000'), fstop, code);  // khz -> Hz
            if fstop > 2600000000 then fstop := 2600000000; // obere RX-Frequenzgrenze

            (* Bereichsüberprüfung *)
            if (fstop - fstart) > maxbereich then PrintMessageA;
      until (fstop - fstart) <= maxbereich;

      (* Auflösung holen *);
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      outtextxy(0, (linefeed*4), 'Resolution:');
      outtextxy(0, (linefeed*5), '1 -   2 kHz');
      outtextxy(0, (linefeed*6), '2 -  10 kHz');
      outtextxy(0, (linefeed*7), '3 -  20 kHz');
      outtextxy(0, (linefeed*8), '4 -  50 kHz');
      outtextxy(0, (linefeed*9), '5 - 100 kHz');
      outtextxy(0, (linefeed*10), '6 - 200 kHz');
      repeat
            repeat
                 repeat
                       PrintAgc(true, 0);
                 until keypressed;
                 key := readkey;
            until (key >= '1') and (key <= '6');
            ordinalres := key;
            case key of
                 '1': resolution := 2000;
                 '2': resolution := 10000;
                 '3': resolution := 20000;
                 '4': resolution := 50000;
                 '5': resolution := 100000;
                 '6': resolution := 200000;
            end;
            if (maxmesspunkte / ((fstop - fstart) / resolution)) < 1 then PrintMessageB;
       until (maxmesspunkte / ((fstop - fstart) / resolution)) >= 1 ;

      _Beenden:
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      clearviewport;
end;
```

procedure PrepSpectrum (var fstart, fstop, resolution: qword; var grafikbreite, balkenbreite, grafikstep, grid: integer);

```
var anzahlmesspunkte: integer;

begin
    anzahlmesspunkte := trunc((fstop - fstart) / resolution);
    (* fstop auf ganzzahliges Vielfaches der gewählten Auflösung korrigieren *)
    fstop := fstart + (anzahlmesspunkte * resolution);
    (* den fstop-Messwert noch berücksichtigen *)
    anzahlmesspunkte := anzahlmesspunkte + 1;
    (* Graphische Auflösung bestimmen *)
    if (anzahlmesspunkte < 33) then
```

```
        begin
            grafikstep := 24;
            balkenbreite := 20;
            grid := 2;
         end;
    if (anzahlmesspunkte > 32) and (anzahlmesspunkte < 65) then
        begin
            grafikstep := 12;
            balkenbreite := 9;
            grid := 5;
        end;
    if (anzahlmesspunkte > 64) and (anzahlmesspunkte < 130) then
        begin
            grafikstep := 6;
            balkenbreite := 4;
            grid := 10;
        end;
    if (anzahlmesspunkte > 129) and (anzahlmesspunkte < 260) then
        begin
            grafikstep := 3;
            balkenbreite := 1;
            grid := 10;
        end;
    if (anzahlmesspunkte > 259) and (anzahlmesspunkte < 390) then
        begin
            grafikstep := 2;
            balkenbreite := 0;
            grid := 20;
        end;
    if (anzahlmesspunkte > 389) then
        begin
            grafikstep := 1;
            balkenbreite := 0;
            grid := 50;
        end;
    grafikbreite := (anzahlmesspunkte * grafikstep) - (grafikstep - balkenbreite);
end;
```

procedure PrintGrid(fstart, fstop, resolution: qword; grafikbreite, balkenbreite, grafikstep, grid: integer);

```
var   x: integer;
      fstr: string;

begin
     (* Plotfenster: x bis max. 780 pixel, y = 200 pixel hoch *)
     setlinestyle(solidln, solidln, normwidth);
     rectangle(0, 20, grafikbreite, 220);
     (* Frequenz-Achse unterteilen *)
     setlinestyle(dottedln, dottedln, normwidth);
     x := (balkenbreite div 2);
     repeat
           line(x, 20, x, 220);
           x := x + (grafikstep * grid);
     until x >= grafikbreite;
     (* Frequenzachse beschriften *)
     settextjustify(lefttext, toptext);
     str((fstart div 1000), fstr); // Hz -> kHz
     outtextxy(0, 225, fstr);

     settextjustify(righttext, toptext);
     str((fstop div 1000), fstr); // Hz -> kHz
     outtextxy(grafikbreite, 225, fstr);

     settextjustify(centertext, bottomtext);
     str(((resolution div 1000) * grid), fstr);
```

```
     fstr := fstr + ' kHz/div';
     outtextxy((grafikbreite div 2), 15, fstr);

     (* Ursprüngliche Text-Einstellungen wiederherstellen *)
     settextjustify(lefttext, bottomtext);
end;
```

procedure SetSpectralMode(ordinalres: char);

```
begin
     ser.SendString('MD1' + CRLF);
     ser.purge;

     ser.SendString('BW' + ordinalres + CRLF);
     ser.purge;

     ser.SendString('AC0' + CRLF);
     ser.purge;

     ser.SendString('VL000' + CRLF);
     ser.purge;

     ser.SendString('AT0' + CRLF);
     ser.purge;

     ser.SendString('DB000' + CRLF);
     ser.purge;
end;
```

procedure DoSpectralScan(fstart, resolution: qword;
grafikbreite, balkenbreite, grafikstep: integer; var key: char);

```
var
   index: integer;
   n: byte;
   decimal: array[0..4] of integer;
   stdabw: single;
   y: integer;
   f: qword;          // Hz
   fstr: string;      // Hz

begin
     setviewport(lowerx1, (lowery1+20), grafikbreite, (lowery1+220), clipoff);
     setfillstyle(solidfill, getcolor);
     index :=  0;
     f := fstart;
     repeat
           (* Frequenz setzen *)
           str(f, fstr);
           fstr := '0000000000' + fstr;
           fstr := copy(fstr, (length(fstr) - 9), 10);
           ser.SendString('RF' + fstr + CRLF);
           delay(einschwingzeit);
           ser.purge;
           (* Störungen aus der Messung eliminieren:                    *)
           (* Messung solange wiederholen, bis Standardabweichung < 3 ist *)
           repeat
                 (* Mittelwertbildung aus 4 AGC-Messwerten *)
                 decimal[0] := 0;
                 for n := 1 to 4 do
                 begin
                      decimal[n] := GetAGC;
                      decimal[0] := decimal[0] + decimal[n];
                 end;
```

```
                decimal[0] := decimal[0] div 4;
                (* Standardabweichung berechnen *)
                stdabw := 0;
                for n := 1 to 4 do stdabw := stdabw + sqr(decimal[n] - decimal[0]);
                stdabw := sqrt(stdabw / 3);
           until stdabw < 3;
           (* Balken zeichnen, nur Umfang von 100 dB darstellen,
           darüber läuft der RX in die Sättigung *)
           y := round(200 * (decimal[0] / 100));
           bar(index, 200,(index + balkenbreite), (200 - y));
           index := index + grafikstep;
           (* Frequenz erhöhen *)
           f := f + resolution;
           (* Abbruch durch User ? *)
           if keypressed then
           begin
                key := readkey;
                key := upcase(key);
           end;
     until (index >= grafikbreite) or (key = 'X');
end;
```

procedure RestoreMode;

```
begin
     ser.SendString(mode + CRLF);
     ser.purge;

     ser.SendString(bandbreite + CRLF);
     ser.purge;

     ser.SendString(agc + CRLF);
     ser.purge;

     ser.SendString(auto + CRLF);
     ser.purge;

     ser.SendString(frequenz + CRLF);
     delay(einschwingzeit);
     ser.purge;

     ser.SendString(volume + CRLF);
     ser.purge;

     ser.SendString(att + CRLF);
     ser.purge;

     ser.SendString(squelch + CRLF);
     ser.purge;
end;
```

procedure DoSpectralAnalysis;

```
var
   fstart: qword;
   fstop: qword;
   resolution: qword;
   ordinalres: char;
   grafikbreite: integer;
   balkenbreite: integer;
   grafikstep: integer;
   grid: integer;
   flag: boolean;
   key: char;
```

```
begin
     flag := true;
     key := ' ';
     GetSpectralParameter(fstart, fstop, resolution, ordinalres, flag);
     if flag then
     begin
          PrepSpectrum (fstart, fstop, resolution, grafikbreite,
                        balkenbreite, grafikstep, grid);
          PrintGrid(fstart, fstop, resolution, grafikbreite,
                    balkenbreite, grafikstep, grid);
          SetSpectralMode(ordinalres);
          DoSpectralScan(fstart, resolution, grafikbreite,
                         balkenbreite, grafikstep, key);
          RestoreMode;
          if key <> 'X' then
          repeat
                repeat
                      PrintAgc(true, 0);
                until keypressed;
                key := readkey;
                key := upcase(key);
          until key = 'X';
          setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
          clearviewport;
     end;
end;
```

procedure PrintAbout;

```
const linefeed = 16;
var key: char;

begin
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, (linefeed*1), 'AR5000 Radio Control v.0.70 - Copyright 2010 -2012 Wolfgang
Kaufmann');
     outtextxy(0, (linefeed*3), 'This program is distributed in the hope that it will be
useful,');
     outtextxy(0, (linefeed*4), 'but WITHOUT ANY WARRANTY; without even the implied
warranty');
     outtextxy(0, (linefeed*5), 'of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.');
     outtextxy(0, (linefeed*6), 'See the GNU General Public License for more details.');
     outtextxy(0, (linefeed*8), 'This program is free software: you can redistribute it and/or
modify');
     outtextxy(0, (linefeed*9), 'it under the terms of the GNU General Public License as
published by');
     outtextxy(0, (linefeed*10), 'the Free Software Foundation, either version 3 of the
License');
     outtextxy(0, (linefeed*11), 'or any later version.');
     outtextxy(0, (linefeed*13), 'Press <X>');
     repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
           key := upcase(key);
     until key = 'X';
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure DoAsk(var key: char);

```
const linefeed = 16;
var   memcolor: word;

begin
     setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
     memcolor := getcolor;
     setcolor(lightred);
     setlinestyle(solidln, solidln, normwidth);
     rectangle(0, 0, (mboxx2 - mboxx1), (mboxy2 - mboxy1));

     outtextxy(32, (linefeed*2), 'Exit programm?');
     outtextxy(32, (linefeed*4), 'Y / N');
     setcolor(memcolor);
     repeat
           key := readkey;
           key := upcase(key);
     until (key = 'Y') or (key = 'N');

     setviewport(mboxx1, mboxy1, mboxx2, mboxy2, clipoff);
     clearviewport;
end;
```

function GetKey: char;

```
begin
     GetKey := chr(0);
     (*Tastaturpuffer leeren, zuletzt gedrückte Taste ausgeben*)
     repeat
           if keypressed then GetKey := upcase(readkey);
     until not keypressed;
end;
```

function AddTime(ttime: TDateTime; t:single; var flag: boolean): TDateTime;

```
var hh, mm, ss, ms: word;
    tt: single;
    fraction: single;

begin
     flag := false;
     decodetime(ttime, hh, mm, ss, ms);
     (*Uhrzeit des aktuellen Tages umwandeln in Dezimal-Stunden *)
     tt := hh + (mm / 60) + (ss / 3600);
     (*Addieren*)
     tt := tt + t;
     (*Dezimal-Stunden umwandeln in Uhrzeit*)
     hh := trunc(tt);
     fraction := frac(tt);
     mm := trunc(fraction * 60);
     fraction := frac(fraction * 60);
     ss := trunc(fraction * 60);
     (*Prüfen, ob Datumsgrenze überschritten wurde*)
     if hh >= 24 then
     begin
          hh := hh - 24;
          flag := true;
     end;
     AddTime := encodetime(hh, mm, ss, ms);
end;
```

function MakeParameterSet(nrvfo: integer): string;

```
var ttext: string;
    letter: char;

begin
     (*VFO einstellen*)
     letter := chr(nrvfo + 64);
     vfo := 'V' + letter;
     ser.SendString(vfo + CR);
     delay(einschwingzeit);
     (*Parameter des eingestellten VFO abfragen*)
     GetRX(1);
     ttext := vfo + ': ' + frequenz + ' ' + bandbreite + ' ' + agc + ' ' + att;
     MakeParameterSet := '"' + ttext + '"';
end;
```

function MakeFrequDataSet(tstart: TDateTime; cvfo: char; m: integer;
agcmean, sdev: single): string;

```
var ttext: string;
    convert: string;

begin
     ttext := DateToStr(tstart) + ',';              // Datum
     ttext := ttext + TimeToStr(tstart) + ',';      // Startzeit des Messung
     ttext := ttext + cvfo + ',';                   // VFO
     str(m, convert);
     ttext := ttext + convert + ',';                // Anzahl Samples
     str(agcmean:7:3, convert);
     ttext := ttext + convert + ',';                // Mittelwert
     str(sdev:7:3, convert);
     ttext := ttext + convert;                      // Standardabweichung
     MakeFrequDataSet := ttext;
end;
```

function MakeBandDataSet(tstart: TDateTime; f: qword; agcmean, sdev: single): string;

```
var ttext: string;
    convert: string;

begin
     ttext := DateToStr(tstart) + ',';              // Datum
     ttext := ttext + TimeToStr(tstart) + ',';      // Startzeit der Messung bei f
     str(f, convert);
     ttext := ttext + convert + ',';                // Frequenz f
     str(agcmean:7:3, convert);
     ttext := ttext + convert + ',';                // Mittelwert
     str(sdev:7:3, convert);
     ttext := ttext + convert;                      // Standardabweichung
     MakeBandDataSet := ttext;
end;
```

function MakeSfericsDataSet(tstop: TDateTime; cvfo: char; minsum: integer): string;

```
var ttext: string;
    convert: string;
```

```
begin
     ttext := DateToStr(tstop) + ',';               // Datum
     ttext := ttext + TimeToStr(tstop) + ',';       // Stopzeit des 60s Zyklus
     ttext := ttext + cvfo + ',';                   // VFO
     str(minsum, convert);                          // Minutensumme
     ttext := ttext + convert;
     MakeSfericsDataSet := ttext;
end;
```

procedure DoWait(wartezeit: longint; var key: char);

```
(*delay verarbeitet nur Werte bis 65535*)

var minuten: byte;
    millisekunden: word;

begin
     (*1 Minute = 60 s = 60000 ms*)
     minuten := trunc(wartezeit / 60000);
     millisekunden := trunc(60000 * frac(wartezeit / 60000));
     while minuten > 0 do
     begin
          delay(60000);
          dec(minuten);
          if keypressed then key := GetKey;
          if key = 'Q' then exit;
     end;
     delay(millisekunden);
end;
```

procedure RunFrequObservation (filename: string; numvfo: integer; itime, srate: longint);

```
const header = '"Date","Time","VFO","Samples","AGC","Stdev"';
      linefeed = 16;

var agcdata: array[1..110] of integer;
    cvfo: char;
    n, m, p: integer;
    counter: longint;
    timer: longint;
    agcmean: single;
    sdev: single;
    ttext: string;
    textt: string;
    tstart: TDateTime;
    datafile: text;
    key: char;
    vol: string;

begin
     (*Start-Anzeige*)
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
     outtextxy(0,(linefeed*2), 'Observation is running (Stop with <Q>):');

     (*Mute*)
     vol := volume;
     ser.SendString('VL000' + CRLF);
     ser.purge;

     (*Datei öffnen*)
     assign(datafile, filename);
     rewrite(datafile);
```

```
(*Header in Datei schreiben*)
writeln(datafile,header);

(*VFO A einstellen*)
ser.SendString('VA' + CR);
delay(einschwingzeit);

(*Messung beginnen*)
repeat
      timer := 0;                    //Verstrichene Zeit eines Zyklus messen
      cvfo := 'A';
      for n := 1 to numvfo do
      begin
           (*VFO einstellen*)
           if numvfo > 1 then
           begin
               ser.SendString('V' + cvfo + CR);
               delay(einschwingzeit);
               ser.purge;
               timer := timer + einschwingzeit;
           end;

           (*AGC-Werte holen für die Dauer von itime*)
           m := 0;
           counter := 0;
           tstart := now;
           repeat
                 inc(m);
                 agcdata[m] := GetAgc;
                 counter := counter + 50;           //GetAGC benötigt 50 ms
           until counter >= itime;
           timer := timer + counter;

           (*Mittelwert und Standardabweichung berechnen*)
           agcmean := 0;
           for p := 1 to m do
           begin
                agcmean := agcmean + agcdata[p];
           end;
           agcmean := agcmean / m;

           sdev := 0;
           for p := 1 to m do
           begin
                sdev := sdev + sqr(agcdata[p] - agcmean);
           end;
           sdev := sqrt((1 / (m - 1)) * sdev);

           (*Datensatz bilden*)
           ttext := MakeFrequDataSet(tstart, cvfo, m, agcmean, sdev);

           (*Datensatz anzeigen*)
           clearviewport;
           outtextxy(0,(linefeed*2), 'Observation is running (Stop with <Q>):');
           outtextxy(0,(linefeed*4),ttext);

           (* Einzeldaten anzeigen für Kontrollzwecke*)
           //for p := 1 to m do
           //begin
           //     str(agcdata[p],textt);
           //     outtextxy(((p-1)*26),(linefeed*5),textt);
           //end;

           (*Datensatz auf Disk schreiben*)
           writeln(datafile, ttext);

           (*VFO weiterschalten*)
           cvfo := succ(cvfo);
```

```
          end;

          (*Warten bis zum nächsten Messzyklus oder Messung beenden*)
          key := GetKey;
          if key <> 'Q' then if (srate - timer) > 0 then DoWait(srate - timer, key);
          if keypressed then key := GetKey;
     until key = 'Q';

     (*Empfangsparameter in Datei schreiben*)
     writeln(datafile);
     for p := 1 to numvfo do
     begin
          ttext := MakeParameterSet(p);
          writeln(datafile, ttext);
     end;

     (*Datei schließen*)
     close(datafile);

     (*Restore Volume*)
     volume := vol;
     ser.SendString(volume + CRLF);
     ser.purge;
end;
```

procedure RunSfericsObservation (filename: string; numvfo: integer; rlimit: single);

```
const header = '"Date","Time","VFO","Strokes/min"';
      linefeed = 16;

var agcdata: array[1..2] of integer;
    signalrise: array[1..1000] of byte;
    cvfo: char;
    n, p, o: integer;
    timer: longint;
    index: integer;
    minsum: integer;
    ttext: string;
    textt: string;
    tstop: TDateTime;
    datafile: text;
    key: char;
    vol: string;

begin
     (*Info ausgeben*)
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
     outtextxy(0,(linefeed*2), 'Observation is running (Stop with <Q>):');
     outtextxy(0,(linefeed*4), 'First sampling (60 seconds) - please wait');

     (*Mute*)
     vol := volume;
     ser.SendString('VL000' + CRLF);
     ser.purge;

     (*Datei öffnen*)
     assign(datafile, filename);
     rewrite(datafile);

     (*Header in Datei schreiben*)
     writeln(datafile,header);

     (*VFO A einstellen*)
     ser.SendString('VA' + CR);
     delay(einschwingzeit);
```

```
(*Messung beginnen*)
repeat
      cvfo := 'A';
      for n := 1 to numvfo do
      begin
           timer := 0;
           index := 1;

           (*VFO einstellen*)
           if numvfo > 1 then
           begin
               ser.SendString('V' + cvfo + CR);
               delay(einschwingzeit);
           end;

           (*60 s Messzyklus für den eingestellten VFO*)
           agcdata[1] := 0;
           o := 0;
           repeat
                 (*AGC-Werte holen*)
                 agcdata[2] := GetAgc;
                 timer := timer + 62;              //GetAGC benötigt ca.62 ms

                 //Testausgabe der ersten AGC-Werte:
                 //str(agcdata[2],textt);
                 //outtextxy(o,(linefeed*6),textt);
                 //o := o + 25;

                 (*Flanke detektieren*)
                 if (agcdata[1]<agcdata[2]) and ((agcdata[2]-agcdata[1])>rlimit)
                     then
                     signalrise[index] := 1
                     else
                     signalrise[index] := 0;

                 agcdata[1] := agcdata[2];
                 inc(index);
            until timer >= 60000;                  //-> Messzyklus von 60 s
            tstop := now;                          // Ortszeit Messzyklus-Ende

           (*Summe der steigenden Flanken bilden*)
           minsum := 0;
           for p := 2 to (index - 1) do
           begin
                minsum := minsum + signalrise[p];
           end;

           (*Datensatz bilden*)
           ttext := MakeSfericsDataSet(tstop, cvfo, minsum);

           (*Datensatz anzeigen*)
           clearviewport;
           outtextxy(0,(linefeed*2), 'Observation is running (Stop with <Q>):');
           outtextxy(0,(linefeed*4),ttext);

           (*Datensatz auf Disk schreiben*)
           writeln(datafile, ttext);

           (*VFO weiterschalten*)
           cvfo := succ(cvfo);
      end;

      (*Messung beenden?*)
      key := GetKey;
until key = 'Q';

(*Empfangsparameter in Datei schreiben*)
writeln(datafile);
```

```
     for p := 1 to numvfo do
     begin
          ttext := MakeParameterSet(p);
          str(rlimit:3:1, textt);
          ttext := copy(ttext, 1, (length(ttext)-1)) + ' THR' + textt + '"';
          writeln(datafile, ttext);
     end;

     (*Datei schließen*)
     close(datafile);

     (*Restore Volume*)
     volume := vol;
     ser.SendString(volume + CRLF);
     ser.purge;
end;
```

procedure RunWeakSignalObs(numvfo: integer; itime, srate: longint);

```
const linefeed = 16;

var cvfo: char;
    n: integer;
    counter: longint;
    timer: longint;
    key: char;
    vol: string;
    ttext: string;

begin
     (*Start-Anzeige*)
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
     str((itime div 1000),ttext);

     (*Mute*)
     vol := volume;
     ser.SendString('VL000' + CRLF);
     ser.purge;

     (*Start observation*)
     repeat
          timer := 0;                    //Verstrichene Zeit eines Zyklus messen
          cvfo := 'A';
          for n := 1 to numvfo do
          begin
               (*VFO einstellen*)
               ser.SendString('V' + cvfo + CR);
               delay(einschwingzeit);
               timer := timer + einschwingzeit;

               (*Anzeige*)
               clearviewport;
               outtextxy(0,(linefeed*2), 'Observation is running (Stop with <Q>):');
               outtextxy(0,(linefeed*4), ('Active VFO: '+cvfo+' for '+ttext+' seconds'));

               (*Wait*)
               ser.SendString(vol + CRLF);
               ser.purge;
               DoWait(itime, key);
               timer := timer + itime;
               ser.SendString('VL000' + CRLF);
               ser.purge;
               if key = 'Q' then exit;

               (*VFO weiterschalten*)
```

```
                cvfo := succ(cvfo);
            end;

            (*Anzeige*)
            clearviewport;
            outtextxy(0,(linefeed*2), 'Observation is running (Stop with <Q>):');
            outtextxy(0,(linefeed*4), 'Active VFO: Waiting for the next cycle of sampling' );

            (*Warten bis zum nächsten Zyklus oder Beenden*)
            if keypressed then key := GetKey;
            if key <> 'Q' then if (srate - timer) > 0 then DoWait(srate - timer, key);
            if keypressed then key := GetKey;
        until key = 'Q';

        (*Restore volume*)
        ser.SendString(vol + CRLF);
        ser.purge;
end;
```

procedure MakeFilename(var fname: string);

```
var ThisMoment: TDateTime;
    ttext: string;

begin
    ThisMoment := Now;
    ttext := DateToStr(ThisMoment);
    fname := copy(ttext,1,2) + copy(ttext,4,2) + copy(ttext,9,2);  // ddmmyy
    ttext := TimeToStr(ThisMoment);
    fname := fname + copy(ttext,1,2);  // hh
    fname := fname + '.csv'
end;
```

procedure SetFrequObservation;

```
label _Beenden;

const linefeed = 16;
      box = 219;
      xstep = 8;
      tab = 220;

var   key: char;
      numvfo: integer;
      result: integer;
      itime: longint;          // Millisekunden
      n: integer;
      m: integer;
      srate: longint;          // Millisekunden
      ttext: string;
      filename: string;
      memcolor: word;
      mtime: longint;          //Millisekunden

begin
     (* Hinweis ausgeben *)
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      outtextxy(0,(linefeed*1), 'Frequency observation will be performed with the');
      outtextxy(0,(linefeed*2), 'current settings of VFO A - E. Continue (y/n)?');
      repeat
           repeat
                PrintAgc(true, 0);
           until keypressed;
```

```
      key := readkey;
      key := upcase(key);
 until (key = 'Y') or (key = 'N');
 if key = 'N' then goto _Beenden;
 setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
 clearviewport;
 memcolor := getcolor;

(*Anzahl zu beobachtender VFOs:*)
outtextxy(0, (linefeed*1), 'Number of VFOs (1 - 5):');
repeat
      key := readkey;
until (key >= '1') and (key <= '5');
outtextxy(192, (linefeed*1), key);
val(key, numvfo, result);

(*Integrationszeitraum*)
outtextxy(0, (linefeed*3), 'Integration time:');
outtextxy(0, (linefeed*4), 'a - 0.1 s');
outtextxy(0, (linefeed*5), 'b - 0.2 s');
outtextxy(0, (linefeed*6), 'c - 0.5 s');
outtextxy(0, (linefeed*7), 'd - 1.0 s');
outtextxy(0, (linefeed*8), 'e - 2.0 s');
outtextxy(0, (linefeed*9), 'f - 5.0 s');
repeat
      key := readkey;
      key := upcase(key);
until (key >= 'A') and (key <= 'F');
case key of
     'A': itime := 100;
     'B': itime := 200;
     'C': itime := 500;
     'D': itime := 1000;
     'E': itime := 2000;
     'F': itime := 5000;
end;
str(itime, ttext);
outtextxy(144, (linefeed*3), (ttext + ' ms'));

(*Menue-Einträge löschen*)
setcolor(getbkcolor);
for n := 4 to 9 do
begin
     for m := 0 to 20 do outtextxy((m*xstep), (linefeed*n), chr(box));
end;
setcolor(memcolor);

(*Messdauer ermitteln*)
mtime := itime * numvfo;
if numvfo > 1 then mtime := mtime + (numvfo * einschwingzeit);

(*Sampling-Rate*)
outtextxy(0, (linefeed*5), 'Sampling rate:');
outtextxy(0, (linefeed*7), 'a -   0 s');
outtextxy(0, (linefeed*8), 'b - 0.5 s');
outtextxy(0, (linefeed*9), 'c -   1 s');
outtextxy(0, (linefeed*10), 'd -   2 s');
outtextxy(0, (linefeed*11), 'e -   5 s');
outtextxy(0, (linefeed*12), 'f -  10 s');
outtextxy(tab, (linefeed*7), 'g -  20 s');
outtextxy(tab, (linefeed*8), 'h -  50 s');
outtextxy(tab, (linefeed*9), 'i - 100 s');
outtextxy(tab, (linefeed*10), 'j - 200 s');
outtextxy(tab, (linefeed*11), 'k - 500 s');
repeat
      repeat
            key := readkey;
            key := upcase(key);
      until (key >= 'A') and (key <= 'K');
```

```
          case key of
               'A': srate := 0;
               'B': srate := 500;
               'C': srate := 1000;
               'D': srate := 2000;
               'E': srate := 5000;
               'F': srate := 10000;
               'G': srate := 20000;
               'H': srate := 50000;
               'I': srate := 100000;
               'J': srate := 200000;
               'K': srate := 500000;
          end;
          if ((mtime > srate) and (srate > 0)) then
          begin
               PrintMessageC;
               setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
          end;
     until ((mtime <= srate) or (srate = 0));
     str(srate, ttext);
     outtextxy(120, (linefeed*5), (ttext + ' ms'));

     (*Menue-Einträge löschen*)
     setcolor(getbkcolor);
     for n := 7 to 12 do
     begin
          for m := 0 to (tab+20) do outtextxy((m*xstep), (linefeed*n), chr(box));
     end;
     setcolor(memcolor);

     (*Dateinamen erstellen *)
     outtextxy(0, (linefeed*7), 'Save data to Disk (A - Z):');
     repeat
          key := readkey;
          key := upcase(key);
     until (key >= 'A') and (key <= 'Z');
     MakeFilename(filename);
     filename := key + ':\' + filename;
     outtextxy(216, (linefeed*7), key);

     (*Rückfrage*)
     outtextxy(0, (linefeed*9), 'Start (Y/N):');
     repeat
          key := readkey;
          key := upcase(key);
     until (key = 'Y') or (key = 'N');
     if key = 'Y' then RunFrequObservation(filename, numvfo, itime, srate);

     _Beenden:
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetSferics;

```
label _Beenden;

const linefeed = 16;
      box = 219;
      xstep = 8;

var   key: char;
      numvfo: integer;
      result: integer;
      filename: string;
      rlimit: single;
```

```
      n,m: integer;
      ttext: string;

begin
     (* Hinweis ausgeben *)
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      outtextxy(0,(linefeed*1), 'Sferics observation will be performed with the');
      outtextxy(0,(linefeed*2), 'current settings of VFO A - E. Continue (y/n)?');
      repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
           key := upcase(key);
      until (key = 'Y') or (key = 'N');
      if key = 'N' then goto _Beenden;
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      clearviewport;

     (*Anzahl zu beobachtender VFOs:*)
     outtextxy(0, (linefeed*1), 'Number of VFOs (1 - 5):');
     repeat
           key := readkey;
     until (key >= '1') and (key <= '5');
     outtextxy(192, (linefeed*1), key);
     val(key, numvfo, result);

     (*Schwellenwert in dB festlegen*)
     outtextxy(0, (linefeed*3), 'Responsivity (1 - 9):');
     PrintMessageE;
     repeat
           key := readkey;
     until (key >= '1') and (key <= '9');
     val(key, rlimit);

     (*MessageE löschen*)
     clearviewport;
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(176, (linefeed*3), key);

     (*Dateinamen erstellen *)
     outtextxy(0, (linefeed*5), 'Save data to Disk (A - Z):');
     repeat
           key := readkey;
           key := upcase(key);
     until (key >= 'A') and (key <= 'Z');
     MakeFilename(filename);
     filename := key + ':\' + filename;
     outtextxy(216, (linefeed*5), key);

     (*Rückfrage*)
     outtextxy(0, (linefeed*7), 'Start (Y/N):');
     repeat
           key := readkey;
           key := upcase(key);
     until (key = 'Y') or (key = 'N');
     if key = 'Y' then RunSfericsObservation(filename, numvfo, rlimit);

     _Beenden:
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure SetWeakSignalObs;

```
label _Beenden;
```

```
const linefeed = 16;
      box = 219;
      xstep = 8;

var   key: char;
      numvfo: integer;
      result: integer;
      itime: longint;          // Millisekunden
      srate: longint;          // Millisekunden
      ttext: string;
      mtime: longint;          //Millisekunden
      numval: string;
      memcolor: word;
      m: integer;

begin
     (* Hinweis ausgeben *)
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      outtextxy(0,(linefeed*1), 'Weak signal observation will be performed with');
      outtextxy(0,(linefeed*2), 'the current settings of VFO A - E and volume.');
      outtextxy(0,(linefeed*3), 'Continue (y/n)?');
      repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
           key := upcase(key);
      until (key = 'Y') or (key = 'N');
      if key = 'N' then goto _Beenden;
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      clearviewport;
      memcolor := getcolor;

     (*Anzahl zu beobachtender VFOs:*)
     outtextxy(0, (linefeed*1), 'Number of VFOs (1 - 5):');
     repeat
           key := readkey;
     until (key >= '1') and (key <= '5');
     outtextxy(192, (linefeed*1), key);
     val(key, numvfo, result);

     (*Get period of observation*)
     ttext := 'Enter period of observation in sec (1 - 900): ';
     outtextxy(0, (linefeed*3), ttext);
     repeat
           numval := ReadNumerical(textwidth(ttext), linefeed*3);
     until (length(numval) > 0) and (numval <> 'X');

     (* Eingabe auf erlaubte Wert prüfen und formatieren*)
     numval := FormatNumerical(numval, 1, 900);
     val(numval, itime, result);
     itime := itime * 1000;                              // s -> ms

     (* Alten Eintrag löschen*)
     setcolor(getbkcolor);
     for m := (length(ttext)*xstep) to 500 do outtextxy(m, (linefeed*3), chr(box));
     setcolor(memcolor);

     (*Neuer Eintrag*)
     outtextxy((length(ttext)*xstep), (linefeed*3), numval);

     (*Messdauer ermitteln*)
     mtime := itime * numvfo;
     if numvfo > 1 then mtime := mtime + (numvfo * einschwingzeit);

     (*Get sampling-rate*)
     ttext := 'Enter rate of sampling in min (0 - 120): ';
     outtextxy(0, (linefeed*5), ttext);
```

```
     repeat
           (* Eingabe-Aufforderung wiederholen falls Leerstring *)
           repeat
                 numval := ReadNumerical(textwidth(ttext), linefeed*5);
           until (length(numval) > 0) and (numval <> 'X');

           (* Eingabe auf erlaubte Wert prüfen und formatieren*)
           numval := FormatNumerical(numval, 0, 120);
           val(numval, srate, result);
           srate := srate * 60 * 1000;                      // min -> ms

           (* Alten Eintrag löschen*)
           setcolor(getbkcolor);
           for m := (length(ttext)*xstep) to 500 do outtextxy(m, (linefeed*5), chr(box));
           setcolor(memcolor);

           (*Neuer Eintrag*)
           outtextxy((length(ttext)*xstep), (linefeed*5), numval);

           if (mtime > srate) and (srate > 0) then
                 begin
                      PrintMessageF;
                      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
                      (*Einträge löschen*)
                      setcolor(getbkcolor);
                      for m := 41 to 500 do outtextxy((m*xstep), (linefeed*5), chr(box));
                      setcolor(memcolor);
                 end;
     until (mtime <= srate) or (srate = 0);

     (*Rückfrage*)
     outtextxy(0, (linefeed*7), 'Start (Y/N):');
     repeat
           key := readkey;
           key := upcase(key);
     until (key = 'Y') or (key = 'N');
     if key = 'Y' then RunWeakSignalObs(numvfo, itime, srate);

     _Beenden:
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
end;
```

procedure RunBandObservation(filename: string; fstart, fstop, resolution: qword;

```
                              cycle: single);

const header = '"Date","Time","Frequency","AGC","Stdev"';
      linefeed = 16;

var
   n: byte;
   decimal: array[1..10] of integer;
   agcmean, stdabw: single;
   f: qword;          // Hz
   fstr: string;      // Hz
   datafile: text;
   ttext: string;
   textt: string;
   tstart, dstart: TDateTime;
   tnext: TDateTime;
   key: char;
   nextday: boolean;

begin
     (*Start-Anzeige*)
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
```

```
      clearviewport;
      outtextxy(0,(linefeed*2), 'Observation is running (Stop with <Q>, Break with <X>):');

      (*Datei öffnen*)
      assign(datafile, filename);
      rewrite(datafile);

      (*Header in Datei schreiben*)
      writeln(datafile,header);

      repeat
            f := fstart;
            dstart := date;
            tnext := AddTime(time, cycle, nextday);
            repeat
                  (* Frequenz setzen *)
                  str(f, fstr);
                  fstr := '0000000000' + fstr;
                  fstr := copy(fstr, (length(fstr) - 9), 10);
                  ser.SendString('RF' + fstr + CRLF);
                  delay(einschwingzeit);
                  ser.purge;

                  (* Messung durchführen *)
                  (* Mittelwertbildung aus 10 AGC-Messwerten, Dauer 500 ms *)
                  tstart := now;
                  agcmean := 0;
                  for n := 1 to 10 do
                  begin
                       decimal[n] := GetAGC;                //Messdauer 50 ms
                       agcmean := agcmean + decimal[n];
                  end;
                  agcmean := agcmean / 10;
                  (* Standardabweichung berechnen *)
                  stdabw := 0;
                  for n := 1 to 10 do stdabw := stdabw + sqr(decimal[n] - agcmean);
                  stdabw := sqrt(stdabw / 9);

                  (*Datensatz bilden*)
                  ttext := MakeBandDataSet(tstart, f, agcmean, stdabw);

                  (*Datensatz anzeigen*)
                  clearviewport;
                  outtextxy(0,(linefeed*2), 'Observation is running (Stop with <Q>, Break with
<X>):');
                  outtextxy(0,(linefeed*4),ttext);

                  (* Einzeldaten anzeigen für Kontrollzwecke*)
                  //for n := 1 to 10 do
                  //begin
                  //     str(decimal[n],textt);
                  //     outtextxy(((n-1)*26),(linefeed*5),textt);
                  //end;

                  (*Datensatz auf Disk schreiben*)
                  writeln(datafile, ttext);

                  (* Frequenz erhöhen *)
                  f := f + resolution;

                  (* sofortiger Abbruch: X, Beenden nach Banddurchlauf: Q  *)
                  if keypressed then key := GetKey;
            until (f > fstop) or (key = 'X');
            (*Leerzeile nach Messzyklus einschieben*)
            writeln(datafile);

            (*Warten bis zum Beginn des nächsten Messzyklus*)
            if nextday then
            repeat
```

```
                if keypressed then key := GetKey;
            until (date > dstart) or (key = 'X') or (key = 'Q');

            repeat
                if keypressed then key := GetKey;
            until (time >= tnext) or (key = 'X') or (key = 'Q');
      until (key = 'X') or (key = 'Q');

      (*Datei schließen*)
      close(datafile);
end;
```

procedure SetBandObservation;

```
label _Beenden;

const
     linefeed = 16;
     starttext = 'Enter lower frequency limit (kHz as integer): ';
     endtext = 'Enter upper frequency limit (kHz as integer): ';
     box = 219;
     xstep = 8;
var
   fstr: string;
   key: char;
   code: integer;
   fstart, fstop: qword;
   resolution: qword;
   ordinalres: char;
   filename: string;
   ttext: string;
   memcolor: word;
   n,m: integer;
   cycle: single;
   mtime: longint;

begin
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);

      (* Hinweis ausgeben *)
      outtextxy(0,(linefeed*1), 'Spectral analysis will be performed with the');
      outtextxy(0,(linefeed*2), 'currently selected antenna. Continue (y/n)?');
      repeat
           repeat
                PrintAgc(true, 0);
           until keypressed;
           key := readkey;
           key := upcase(key);
      until (key = 'Y') or (key = 'N');
      if key = 'N' then goto _Beenden;
      memcolor := getcolor;

      (* Startfrequenz holen *)
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      clearviewport;

      outtextxy(0, (linefeed*1), starttext);
      repeat
            fstr := ReadNumerical(textwidth(starttext), (linefeed*1));
      until length(fstr) > 0;
      if fstr = 'X' then goto _Beenden;
      val((fstr + '000'), fstart, code); // khz -> Hz
      if fstart < 10000 then fstart := 10000;  // untere RX-Frequenzgrenze

      (* Endfrequenz holen *)
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
```

```
outtextxy(0, (linefeed*2), endtext);
repeat
      fstr := ReadNumerical(textwidth(endtext), (linefeed*2));
until length(fstr) > 0;
if fstr = 'X' then goto _Beenden;
val((fstr + '000'), fstop, code);  // khz -> Hz
if fstop > 2600000000 then fstop := 2600000000; // obere RX-Frequenzgrenze

(* Auflösung holen *);
setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
outtextxy(0, (linefeed*4), 'Resolution:');
outtextxy(0, (linefeed*5), '1 -   2 kHz');
outtextxy(0, (linefeed*6), '2 -  10 kHz');
outtextxy(0, (linefeed*7), '3 -  20 kHz');
outtextxy(0, (linefeed*8), '4 -  50 kHz');
outtextxy(0, (linefeed*9), '5 - 100 kHz');
outtextxy(0, (linefeed*10), '6 - 200 kHz');
repeat
     key := readkey;
until (key >= '1') and (key <= '6');
ordinalres := key;
case key of
    '1': resolution := 2000;           // Hz
    '2': resolution := 10000;
    '3': resolution := 20000;
    '4': resolution := 50000;
    '5': resolution := 100000;
    '6': resolution := 200000;
end;
str((resolution div 1000), ttext);
outtextxy(104, (linefeed*4), (ttext + ' kHz'));

(*Menue-Einträge löschen*)
setcolor(getbkcolor);
for n := 5 to 10 do
begin
    for m := 0 to 20 do outtextxy((m*xstep), (linefeed*n), chr(box));
end;
setcolor(memcolor);

(*Zeitlicher Abstand der einzelnen Messzyklen*)
outtextxy(0, (linefeed*6), 'Cycle rate:');
outtextxy(0, (linefeed*7), 'a - 0.1 h');
outtextxy(0, (linefeed*8), 'b - 0.2 h');
outtextxy(0, (linefeed*9), 'c - 0.5 h');
outtextxy(0, (linefeed*10), 'd -   1 h');
outtextxy(0, (linefeed*11), 'e -   2 h');
outtextxy(0, (linefeed*12), 'f -   6 h');
repeat
     repeat
          key := readkey;
          key := upcase(key);
     until (key >= 'A') and (key <= 'F');
     case key of
          'A': cycle := 0.1;          // h
          'B': cycle := 0.2;
          'C': cycle := 0.5;
          'D': cycle := 1;
          'E': cycle := 2;
          'F': cycle := 6;
     end;
     (*Eine Messung pro Frequenz dauert ca. 1 s*)
     mtime := (fstop - fstart) div resolution;
     mtime := abs(mtime);              // s
     if mtime > trunc(cycle * 3600) then
     begin
          PrintMessageD;
          setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     end;
```

```
      until mtime <= trunc(cycle * 3600);
      str(cycle:3:1, ttext);
      outtextxy(104, (linefeed*6), (ttext + ' h'));

      (*Menue-Einträge löschen*)
      setcolor(getbkcolor);
      for n := 7 to 12 do
      begin
          for m := 0 to 20 do outtextxy((m*xstep), (linefeed*n), chr(box));
      end;
      setcolor(memcolor);

      (*Dateinamen erstellen *)
      outtextxy(0, (linefeed*8), 'Save data to Disk (A - Z):');
      repeat
           key := readkey;
           key := upcase(key);
      until (key >= 'A') and (key <= 'Z');
      MakeFilename(filename);
      filename := key + ':\' + filename;
      outtextxy(216, (linefeed*8), key);

      (*Receiver-Parameter setzen*)
      SetSpectralMode(ordinalres);

      (*Rückfrage*)
      outtextxy(0, (linefeed*10), 'Start (Y/N):');
      repeat
            key := readkey;
            key := upcase(key);
      until (key = 'Y') or (key = 'N');
      if key = 'Y' then RunBandObservation(filename, fstart, fstop, resolution, cycle);

      (*Vorherige Einstellungen widerherstellen*)
      RestoreMode;

      _Beenden:
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      clearviewport;
end;
```

procedure SelectObservationMode;

```
const linefeed = 16;

var key: char;

begin
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      outtextxy(0,(linefeed*1), 'a - Observe up to five distinct frequencies');
      outtextxy(0,(linefeed*2), 'b - Observe a frequency band');
      outtextxy(0,(linefeed*3), 'c - Observe sferics');
      outtextxy(0,(linefeed*4), 'd - Observe weak singnals (audio observation)');
      repeat
           repeat
                 PrintAgc(true, 0);
           until keypressed;
           key := readkey;
           key := upcase(key);
      until (key >= 'A') and (key <= 'D');
      setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
      clearviewport;
      case key of
           'A': SetFrequObservation;
           'B': SetBandObservation;
           'C': SetSferics;
```

```
           'D': SetWeakSignalObs;
      end;
end;
```

procedure FreeReceiver;

```
const linefeed = 16;

var key: char;
    memcolor: word;

begin
     memcolor := getcolor;
     (* Clear upper windows *)
     setviewport (frex1, uppery1, upperx2, rxy2, clipoff);
     clearviewport;
     (* Free RX *)
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     outtextxy(0, (linefeed*2), 'Receiver is now under manual control!');
     CloseCom;
     (* Return to remote control again *)
     outtextxy(0, (linefeed*12), 'Press <X>');
     repeat
          (*Farbwechsel Warnmeldung*)
          repeat
                setcolor(lightred);
                outtextxy(0, (linefeed*5), 'Before returning to remote control');
                outtextxy(0, (linefeed*6), 'make sure the receiver is in VFO mode!');
                delay (500);
                setcolor(memcolor);
                outtextxy(0, (linefeed*5), 'Before returning to remote control');
                outtextxy(0, (linefeed*6), 'make sure the receiver is in VFO mode!');
                delay (500);
          until keypressed;
          key := readkey;
          key := upcase(key);
     until key = 'X';
     ser := TBlockserial.Create;
     ser.Connect(comport);
     ser.Config(19200,8,'N',0,true,false);
     delay (200);
     setviewport(lowerx1, lowery1, lowerx2, lowery2, clipoff);
     clearviewport;
     (* Restore *)
     PrintTitle;
     ser.SendString(squelch + CRLF);
     ser.purge;
end;


(*-------------------------------------------------------------------------*)
(*                              Main Program                               *)
(*-------------------------------------------------------------------------*)

begin
    SetScreen(graphresult);
    if graphresult then
    begin
        OpenCom(comresult, comport);
        if comresult then
        begin
            PrintTitle;
            PrintMenue;
            repeat
                  repeat
                      GetRX(1);          // AR5000 Settings holen
```

```
                    PrintRX;           // Receiver-Settings anzeigen
                    PrintFrequencies;  // aktuelle Empfangsfrequenz anzeigen
                    TuneRX;            // Tuning-Modus einschalten
                    case key of
                         'V': SetVfo;
                         'A': SetAuto;
                         'H': SetHighPass;
                         'L': SetLowPass;
                         'R': SetAttenuation;
                         'N': SetAntenna;
                         'G': SetAgcMode;
                         'M': SetMode;
                         'I': SetBandwidth;
                         'P': SetStep;
                         'Q': SetSquelch;
                         'O': SetVolume;
                         'F': SetFrequency;
                         'U': PrintAbout;
                         'S': ChannelSearch;
                         'C': DoSpectralAnalysis;
                         'J': StepShift;
                         'T': SelectObservationMode;
                         'E': FreeReceiver;
                    end;
               until key = 'X';
          DoAsk(key);                          // Sicherheitsabfrage
          until key = 'Y';
          CloseCom;
     end;
end;
closegraph;
end.
```