```c
/*
 * 2013-2014 (c) Simtec Buergel AG
 * All rights reserved
 * Rev. 2, 12.12.2013
 * Rev. 3, 17.10.2014, some comments added
 *
 * Decodes some important PSS-8 and ADP-5.5 data messages.
 * Status messages not implemented yet.
 * Use a RS485 to USB converter as described at:
 * http://www.swiss-airdata.com/blog/?p=13
 *
 * User must set correct serial device.
 *
 * Compiled and tested with MinGW (gcc)
 * http://sourceforge.net/projects/mingwbuilds/
 *
 * Example code only. No error handling.
 * Use at own risk.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 *
 * Simtec Buergel AG has no obligation to provide maintenance, support,
 * updates, enhancements, or modifications.
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <stdint.h>
#include <stdbool.h>
#include <windows.h>
#include <winbase.h>

// Replace with COM1, COM2, etc
// \\.\CNCB0 is used by com0com
#define DEVICE "\\\\.\\CNCB0"

// Baud-rate
#define BAUDRATE 230400

// Start of header
#define SOH 0x01

// Linefeed
#define LF  0x0a

// Carriage return
#define CR  0x0d

// Windows file handle
static HANDLE hCom = INVALID_HANDLE_VALUE;

/* Opens and initializes the serial interface. */
static bool serial_open()
```

```
58  {
59      bool fSuccess;
60      DCB dcb;
61
62      hCom = CreateFile(DEVICE, GENERIC_READ | GENERIC_WRITE,
63              0, NULL, OPEN_EXISTING, 0, 0/*NULL*/);
64
65      if (hCom == INVALID_HANDLE_VALUE) {
66          printf("Cannot open %s\n", DEVICE);
67          return FALSE;
68      }
69
70      COMMTIMEOUTS timeouts;
71      fSuccess = GetCommTimeouts(hCom, &timeouts);
72      if (!fSuccess) {
73          printf("Cannot get timeouts on %s\n", DEVICE);
74          return FALSE;
75      }
76
77      // No (minimal) blocking!
78      timeouts.ReadIntervalTimeout = 1;
79      timeouts.ReadTotalTimeoutMultiplier = 0;
80      timeouts.ReadTotalTimeoutConstant = 1;
81      timeouts.WriteTotalTimeoutMultiplier = 0;
82      timeouts.WriteTotalTimeoutConstant = 0;
83      fSuccess = SetCommTimeouts(hCom, &timeouts);
84      if (!fSuccess) {
85          printf("Cannot set timeouts on %s\n", DEVICE);
86          return FALSE;
87      }
88
89      fSuccess = GetCommState(hCom, &dcb);
90      if (!fSuccess) {
91          printf("Cannot comm-state on %s\n", DEVICE);
92          return FALSE;
93      }
94
95      dcb.BaudRate = BAUDRATE;
96      dcb.ByteSize = 8;
97      dcb.Parity = NOPARITY;
98      dcb.StopBits = ONESTOPBIT;
99      dcb.fRtsControl = RTS_CONTROL_DISABLE;
100     fSuccess = SetCommState(hCom, &dcb);
101     if (!fSuccess) {
102         printf("Cannot set comm-state on %s\n", DEVICE);
103         return FALSE;
104     }
105
106     return TRUE;
107 };
108
109 /* Closes the serial interface. */
110 static bool serial_close()
111 {
112     if (hCom != INVALID_HANDLE_VALUE) {
113         CloseHandle(hCom);
114         hCom = INVALID_HANDLE_VALUE;
```

```
115             return TRUE;
116         } else {
117             return FALSE;
118         }
119     };
120
121     /* Read a couple of bytes from the serial line. */
122     static int serial_read(byte buffer[], int size)
123     {
124         DWORD cnt = 0;
125
126         BOOL fSucess = ReadFile(hCom, buffer, size, &cnt, NULL);
127
128         if (!fSucess) {
129             printf("Cannot read from serial %s\n", DEVICE);
130             return 0;
131         }
132
133         return cnt;
134     }
135
136     static void print_help()
137     {
138         printf("\n");
139         printf("A simple terminal program to print PSS-8 and ADP-5.5 messages\n");
140         printf("(c) 2013, Simtec Buergel AG\n");
141         printf("\n");
142     }
143
144     static char* flag_str[] = {
145         "valid",
146         "range+",
147         "range-",
148         "invalid+",
149         "invalid-",
150         "invalid"
151     };
152
153     /* Decode message. */
154     static void decode_and_print_message(byte msg[])
155     {
156         const char deg = (char) 0xF8;
157
158         // Extract label and flag from second byte
159         int label = msg[1] & 0xF;
160         int flag = (msg[1] >> 4) & 0x7;
161
162         // Mark end of string and convert ASCII-hex to integer
163         msg[10] = '\0';
164         long long bits = strtoll((char*) &msg[2], NULL, 16);
165
166         // Cast integer-bits to float
167         float ans = *(float*) &bits;
168
169         //
170         // Print message
171         //
```

```
172         switch (label) {
173             case 1:
174                 printf("Qc   = %9.1f [Pa]   (%s)\n", ans, flag_str[flag]);
175                 break;
176             case 2:
177                 printf("Ps   = %9.1f [Pa]   (%s)\n", ans, flag_str[flag]);
178                 break;
179             case 3:
180                 // ADP-5.5 only
181                 printf("AoA  = %9.3f [%c]    (%s)\n", ans, deg, flag_str[flag]);
182                 break;
183             case 4:
184                 // ADP-5.5 only
185                 printf("AoS  = %9.3f [%c]    (%s)\n", ans, deg, flag_str[flag]);
186                 break;
187             case 5:
188                 printf("CAS  = %9.2f [m/s] (%s)\n", ans, flag_str[flag]);
189                 break;
190             case 6:
191                 printf("TAS  = %9.2f [m/s] (%s)\n", ans, flag_str[flag]);
192                 break;
193             case 7:
194                 printf("Hp   = %9.1f [m]   (%s)\n", ans, flag_str[flag]);
195                 break;
196             case 8:
197                 printf("Mach = %9.3f [-]   (%s)\n", ans, flag_str[flag]);
198                 break;
199             case 9:
200                 printf("SAT  = %9.1f [%cC]  (%s)\n", ans, deg, flag_str[flag]);
201                 break;
202             case 10:
203                 printf("TAT  = %9.1f [%cC]  (%s)\n", ans, deg, flag_str[flag]);
204                 break;
205         }
206     }
207
208     static void read_and_decode_and_print_message()
209     {
210         int pos = 0;
211         byte buf = 0;
212         byte msg[11];
213
214         // Process as long a user does not hit any key
215         while ((pos < 11) && !kbhit()) {
216             if (serial_read(&buf, 1) == 1) {
217                 if (buf == SOH) {
218                     // start a new message if a SOH is found
219                     pos = 0;
220                     msg[pos++] = buf;
221                 } else if (buf == CR && pos == 10) {
222                     // full data message collected, decode and print now
223                     msg[pos++] = buf;
224                     decode_and_print_message(msg);
225                     break;
226                 } else {
227                     // add data to message
228                     msg[pos++] = buf;
```

```
229                 }
230
231                 if (pos == 2 && msg[1] == 0x8F) {
232                     // This is a status message add a new
233                     // line to separate message output
234                     printf("\n");
235                 }
236             }
237         }
238     }
239
240     int main(int argc, char** argv)
241     {
242         print_help();
243
244         if (serial_open()) {
245
246             printf("Starting on %s @ B%d\n", DEVICE, BAUDRATE);
247             printf("Hit any key to exit\n\n");
248
249             while (!kbhit()) {
250                 read_and_decode_and_print_message();
251             }
252
253             serial_close();
254         }
255
256         return EXIT_SUCCESS;
257     }
258
```