

Onda Connection Library Source Code

Morten Fischer Rasmussen

April 24, 2013

Contents

| | |
|------------------------------|-----------|
| Contents | 1 |
| 1 MATLAB routines | 3 |
| 1.1 Connection | 3 |
| 1.2 Movement | 5 |
| 1.3 Movement | 8 |
| 2 C++ Library | 11 |
| 2.1 onda_lib | 11 |
| 2.2 TCP connection | 15 |

MATLAB routines

```

1 %
2 %
3 % onda_initialize([host_addr])
4 %
5 % Initializes the connection to the Onda system.
6 %
7 % host_addr: string containing the IP-address of the host. Defaults ↵
   to 10.59.44.100.
8 %
9 %
10 % By MFR,
11 % Version 1.0, 2012-04-15, Init version.
12 % Version 1.1, 2013-04-15, Added check of whether IP address is valid
13 %
14 %
15 %
16 %
17 function onda_initialize(host_addr)
18 %
19 IPv4_pattern = '^[01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d↵
   |25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d↵
   |25[0-5])$';
20 %
21 %
22 %
23 if nargin == 0
24     onda_lib('init_connection', '10.59.44.100');
25 else
26     if ~ischar(host_addr)
27         error('The host address must be a string');
28     else
29         is_valid_address = ~isempty(regexp(host_addr, IPv4_pattern))
30         if ~is_valid_address
31             error('The host address appears not to be a valid IPv4 ↵
   address');
32         end
33     end
34     onda_lib('init_connection', host_addr);
35 end
36 %
37 %
38 %
39 %
40 %
41 %
42 %
43 %
44 %
45 %
46 %
47 %
48 %
49 %
50 %
51 %
52 %
53 %
54 %
55 %
56 %
57 %
58 %
59 %
60 %
61 %
62 %
63 %
64 %
65 %
66 %
67 %
68 %
69 %
70 %
71 %
72 %
73 %
74 %
75 %
76 %
77 %
78 %
79 %
80 %
81 %
82 %
83 %
84 %
85 %
86 %
87 %
88 %
89 %
90 %
91 %
92 %
93 %
94 %
95 %
96 %
97 %
98 %
99 %
100 %
101 %
102 %
103 %
104 %
105 %
106 %
107 %
108 %
109 %
110 %
111 %
112 %
113 %
114 %
115 %
116 %
117 %
118 %
119 %
120 %
121 %
122 %
123 %
124 %
125 %
126 %
127 %
128 %
129 %
130 %
131 %
132 %
133 %
134 %
135 %
136 %
137 %
138 %
139 %
140 %
141 %
142 %
143 %
144 %
145 %
146 %
147 %
148 %
149 %
150 %
151 %
152 %
153 %
154 %
155 %
156 %
157 %
158 %
159 %
160 %
161 %
162 %
163 %
164 %
165 %
166 %
167 %
168 %
169 %
170 %
171 %
172 %
173 %
174 %
175 %
176 %
177 %
178 %
179 %
180 %
181 %
182 %
183 %
184 %
185 %
186 %
187 %
188 %
189 %
190 %
191 %
192 %
193 %
194 %
195 %
196 %
197 %
198 %
199 %
200 %
201 %
202 %
203 %
204 %
205 %
206 %
207 %
208 %
209 %
210 %
211 %
212 %
213 %
214 %
215 %
216 %
217 %
218 %
219 %
220 %
221 %
222 %
223 %
224 %
225 %
226 %
227 %
228 %
229 %
230 %
231 %
232 %
233 %
234 %
235 %
236 %
237 %
238 %
239 %
240 %
241 %
242 %
243 %
244 %
245 %
246 %
247 %
248 %
249 %
250 %
251 %
252 %
253 %
254 %
255 %
256 %
257 %
258 %
259 %
260 %
261 %
262 %
263 %
264 %
265 %
266 %
267 %
268 %
269 %
270 %
271 %
272 %
273 %
274 %
275 %
276 %
277 %
278 %
279 %
280 %
281 %
282 %
283 %
284 %
285 %
286 %
287 %
288 %
289 %
290 %
291 %
292 %
293 %
294 %
295 %
296 %
297 %
298 %
299 %
300 %
301 %
302 %
303 %
304 %
305 %
306 %
307 %
308 %
309 %
310 %
311 %
312 %
313 %
314 %
315 %
316 %
317 %
318 %
319 %
320 %
321 %
322 %
323 %
324 %
325 %
326 %
327 %
328 %
329 %
330 %
331 %
332 %
333 %
334 %
335 %
336 %
337 %
338 %
339 %
340 %
341 %
342 %
343 %
344 %
345 %
346 %
347 %
348 %
349 %
350 %
351 %
352 %
353 %
354 %
355 %
356 %
357 %
358 %
359 %
360 %
361 %
362 %
363 %
364 %
365 %
366 %
367 %
368 %
369 %
370 %
371 %
372 %
373 %
374 %
375 %
376 %
377 %
378 %
379 %
380 %
381 %
382 %
383 %
384 %
385 %
386 %
387 %
388 %
389 %
390 %
391 %
392 %
393 %
394 %
395 %
396 %
397 %
398 %
399 %
400 %
401 %
402 %
403 %
404 %
405 %
406 %
407 %
408 %
409 %
410 %
411 %
412 %
413 %
414 %
415 %
416 %
417 %
418 %
419 %
420 %
421 %
422 %
423 %
424 %
425 %
426 %
427 %
428 %
429 %
430 %
431 %
432 %
433 %
434 %
435 %
436 %
437 %
438 %
439 %
440 %
441 %
442 %
443 %
444 %
445 %
446 %
447 %
448 %
449 %
450 %
451 %
452 %
453 %
454 %
455 %
456 %
457 %
458 %
459 %
460 %
461 %
462 %
463 %
464 %
465 %
466 %
467 %
468 %
469 %
470 %
471 %
472 %
473 %
474 %
475 %
476 %
477 %
478 %
479 %
480 %
481 %
482 %
483 %
484 %
485 %
486 %
487 %
488 %
489 %
490 %
491 %
492 %
493 %
494 %
495 %
496 %
497 %
498 %
499 %
500 %
501 %
502 %
503 %
504 %
505 %
506 %
507 %
508 %
509 %
510 %
511 %
512 %
513 %
514 %
515 %
516 %
517 %
518 %
519 %
520 %
521 %
522 %
523 %
524 %
525 %
526 %
527 %
528 %
529 %
530 %
531 %
532 %
533 %
534 %
535 %
536 %
537 %
538 %
539 %
540 %
541 %
542 %
543 %
544 %
545 %
546 %
547 %
548 %
549 %
550 %
551 %
552 %
553 %
554 %
555 %
556 %
557 %
558 %
559 %
560 %
561 %
562 %
563 %
564 %
565 %
566 %
567 %
568 %
569 %
570 %
571 %
572 %
573 %
574 %
575 %
576 %
577 %
578 %
579 %
580 %
581 %
582 %
583 %
584 %
585 %
586 %
587 %
588 %
589 %
590 %
591 %
592 %
593 %
594 %
595 %
596 %
597 %
598 %
599 %
600 %
601 %
602 %
603 %
604 %
605 %
606 %
607 %
608 %
609 %
610 %
611 %
612 %
613 %
614 %
615 %
616 %
617 %
618 %
619 %
620 %
621 %
622 %
623 %
624 %
625 %
626 %
627 %
628 %
629 %
630 %
631 %
632 %
633 %
634 %
635 %
636 %
637 %
638 %
639 %
640 %
641 %
642 %
643 %
644 %
645 %
646 %
647 %
648 %
649 %
650 %
651 %
652 %
653 %
654 %
655 %
656 %
657 %
658 %
659 %
660 %
661 %
662 %
663 %
664 %
665 %
666 %
667 %
668 %
669 %
670 %
671 %
672 %
673 %
674 %
675 %
676 %
677 %
678 %
679 %
680 %
681 %
682 %
683 %
684 %
685 %
686 %
687 %
688 %
689 %
690 %
691 %
692 %
693 %
694 %
695 %
696 %
697 %
698 %
699 %
700 %
701 %
702 %
703 %
704 %
705 %
706 %
707 %
708 %
709 %
710 %
711 %
712 %
713 %
714 %
715 %
716 %
717 %
718 %
719 %
720 %
721 %
722 %
723 %
724 %
725 %
726 %
727 %
728 %
729 %
730 %
731 %
732 %
733 %
734 %
735 %
736 %
737 %
738 %
739 %
740 %
741 %
742 %
743 %
744 %
745 %
746 %
747 %
748 %
749 %
750 %
751 %
752 %
753 %
754 %
755 %
756 %
757 %
758 %
759 %
760 %
761 %
762 %
763 %
764 %
765 %
766 %
767 %
768 %
769 %
770 %
771 %
772 %
773 %
774 %
775 %
776 %
777 %
778 %
779
```

```
2 %
%
% onda terminate()
```

```
4 %  
5 % $Id: onda_terminate.m 5 2012-04-16 13:31:11Z mf $  
6 %  
7 % By MFR, 2012-04-15  
8 % Version 1.0 Init version.  
9 %  
10  
11  
12  
13  
14 function onda_terminate  
15  
16 onda_lib('terminate_connection');  
17  
18 %↵  
19 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
20 %%% onda_initialise.m ends here
```

Listing 1.2: onda_terminate.m

1.2 Movement

```

2 % onda_move_absolute(dist [,axis])
3 %
4 % dist: distance in meters to move in absolute coordinates.
5 % Must be a vector with three elements representing
6 % [dist_x, dist_y, dist_z]. When a scaler is given, ''axis''
7 % must also be set.
8 %
9 % axis: Optional to use. Can be a vector or scaler. Choses which
10 % axis to move. 0=x, 1=y, 2=z, 3=rot1, 4=rot2.
11 %
12 % By MFR,
13 % Version 1.0, 2012-04-15, Init version.
14 % Version 1.1, 2013-04-17, Changed to accept single vector argument.
15 % Version 1.2, 2013-04-22, Re-added ''axis'', now as an optional ←
16 % argument
17 %
18 %
19 %
20 function onda_move_absolute(dist, axis)
21
22 if nargin == 1
23     % assume a vector for movement in (x,y,z) is given
24     axis = [0 1 2]; %x,y,z
25 end
26 %make sure all dist-elements are numbers
27 if ~isnumeric(dist)
28     error('dist must contain only numbers.')
29 end
30 %make sure all axis-elements are numbers
31 if ~isnumeric(axis)
32     error('dist must contain only numbers.')
33 end
34 % make sure dist and axis have same length
35 if length(dist) ~= length(axis)
36     error(['axis and dist must contain the same number of ' ...
37           'elements.'])
38 end
39
40 % Send the command to the Onda system
41 for idx = 1:length(axis)
42     if axis(idx) >= 0 && axis(idx) <= 2
43         dist = dist*1000; %m to cm conversion
44     end
45     onda_lib('move_absolute', axis(idx), dist(idx));
46 end
47 end

```

Listing 1.3: onda_move_absolute.m

```

2 % onda_move_relative(dist [,axis])
3 %
4 % dist: distance in meters to move relative to current position.
5 % It must be a vector with three elements representing:
6 % [dist_x, dist_y, dist_z]. When a scaler is given, ''axis''
7 % must also be set.

```

```

8 | %
9 | % axis: Optional to use. Can be a vector or scalar. Choses which
10 | % axis to move. 0=x, 1=y, 2=z, 3=rot1, 4=rot2.
11 | %
12 | % By MFR,
13 | % Version 1.0, 2012-04-15, Init version.
14 | % Version 1.1, 2013-04-17, Changed to accept single vector argument.
15 | % Version 1.2, 2013-04-22, Re-added 'axis', now as an optional ←
16 | % argument
17 | %
18 | %
19 |
20 | function onda_move_relative(dist , axis)
21 |
22 | if nargin == 1
23 |     % assume a vector for movement in (x,y,z) is given
24 |     axis = [0 1 2]; %x,y,z
25 | end
26 | %make sure all dist-elements are numbers
27 | if ~isnumeric(dist)
28 |     error('dist must contain only numbers.')
29 | end
30 | %make sure all axis-elements are numbers
31 | if ~isnumeric(axis)
32 |     error('dist must contain only numbers.')
33 | end
34 | % make sure dist and axis have same length
35 | if length(dist) ~= length(axis)
36 |     error(['axis and dist must contain the same number of ' ...
37 |           'elements.'])
38 | end
39 |
40 |
41 | % Send the command to the Onda system
42 | for idx = 1:length(axis)
43 |     if axis(idx) >= 0 && axis(idx) <= 2
44 |         dist = dist*1000; %m to cm conversion
45 |     end
46 |     onda_lib('move_relative', axis(idx), dist(idx));
47 | end
48 |

```

Listing 1.4: onda_move_relative.m

```

1 | %
2 | %
3 | % onda_set_position(position)
4 | %
5 | % position: Vector with three elements setting the position of the
6 | % x-, y- and z-axis. In units of meters.
7 | %
8 | %
9 | % By MFR, 2012-04-15
10 | % Version 1.0 Init version.
11 | % Version 1.1, 2013-04-17, Changed to set the position for x,y,z axes←
12 | % , always.
13 | %
14 |
15 | function onda_set_position(position)
16 |

```

```

17 %make sure all position-elements are numbers
18 if ~isnumeric(position)
19     error('dist must contain only numbers.');
```

```

20 end
21 if length(position) ~= 3
22     error('''position'' must be a vector with 3 elements.');
```

```

23 end
24
25 axis = [0 1 2]; %x,y,z
26 % Send the command to the Onda system
27 for idx = 1:length(axis)
28     if axis(idx) >= 0 && axis(idx) <= 2
29         position = position*1000; %m to mm conversion
30     end
31     onda_lib('set_position', axis(idx), position(idx));
32 end
33 end

```

Listing 1.5: onda_set_position.m

```

1 %
2 %
3 % position = onda_get_position
4 %
5 % By MFR,
6 % Version 1.0, 2012-04-15, Init version.
7 % Version 1.1, 2013-04-17, Changed to return position for x,y,z axes,↵
8 %     always.
9 %
10
11 function position = onda_get_position
12
13
14 pos_x = onda_lib('get_position', 0);
15 pos_y = onda_lib('get_position', 1);
16 pos_z = onda_lib('get_position', 2);
17
18 position = [pos_x pos_y pos_z]';
19 position = position/1000; %mm to m conversion

```

Listing 1.6: onda_get_position.m

1.3 Movement

```

1 %
2 %
3 % onda_set_high_limit (axis, limit)
4 %
5 % axis:      integer or string choosing which axis to use. (usually ←
6 %           0,1,2 or 'x','y','z').
7 % limit:     limit in meters on the chosen axis.
8 %
9 %
10 % By MFR,
11 % Version 1.0, 2013-04-15, Init version.
12 % Version 1.1, 2013-04-15, Now also accepts strings to identify axes.
13 %
14
15 function onda_set_high_limit(axis, limit)
16
17 if nargin ~= 2
18     error('This function requires two inputs.');
```

Listing 1.7: onda_set_high_limit.m

```

1 %
2 %
3 % onda_get_high_limit(axis)
4 %
5 % axis:      integer choosing which axis to use. (0,1,2 or 'x','y','z'←
6 %           ')
7 %
8 %
9 % By MFR,
10 % Version 1.0, 2013-04-15, Init version.
11 % Version 1.1, 2013-04-15, Now also accepts strings to identify axes.
12 %
13
14 function limit = onda_get_high_limit(axis)
15
16 if nargin ~= 1
17     error('This function requires one input.');
```



```

    limit = onda_lib('get_high_limit', 1);
25 elseif strcmp(axis,'z') || axis == 2
    limit = onda_lib('get_high_limit', 2);
27 else
    warning('Did not identify the axis.')
29 end
31 limit = limit/1000; %mm to m conversion

```

Listing 1.8: onda_get_high_limit.m

```

1 %
2 %
3 % onda_set_low_limit(axis , limit)
4 %
5 % axis:      integer choosing which axis to use. (0,1,2 or 'x','y','z'↵
6 %            ')
7 % limit:      position limit in meters on the chosen axis.
8 %
9 % By MFR,
10 % Version 1.0, 2013-04-15, Init version.
11 % Version 1.1, 2013-04-15, Now also accepts strings to identify axes.
12 %
13
15 function onda_set_low_limit(axis , limit)
17 if nargin ~= 2
18     error('This function requires two inputs.');
```

```

19 end
21 limit = limit*1000; %m to mm conversion
23 if strcmp(axis,'x') || axis == 0
    onda_lib('set_low_limit', 0, limit);
25 elseif strcmp(axis,'y') || axis == 1
    onda_lib('set_low_limit', 1, limit);
27 elseif strcmp(axis,'z') || axis == 2
    onda_lib('set_low_limit', 2, limit);
29 else
    warning('Did not identify the axis.')
31 end

```

Listing 1.9: onda_set_low_limit.m

```

1 %
2 %
3 % onda_get_low_limit(axis)
4 %
5 % axis:      integer choosing which axis to use. (0,1,2 or 'x','y','z'↵
6 %            ')
7 %
8 % By MFR,
9 % Version 1.0, 2013-04-15, Init version.
10 % Version 1.1, 2013-04-15, Now also accepts strings to identify axes.
11 %
12
14 function onda_get_low_limit(axis , limit)

```

```
16 if nargin ~= 2
    error('This function requires two inputs.');
```

```
18 end

20 limit = limit*1000; %m to mm conversion

22 if strcmp(axis,'x') || axis == 0
    onda_lib('get_low_limit', 0);
24 elseif strcmp(axis,'y') || axis == 1
    onda_lib('get_low_limit', 1);
26 elseif strcmp(axis,'z') || axis == 2
    onda_lib('get_low_limit', 2);
28 else
    warning('Did not identify the axis.')
30 end
```

Listing 1.10: onda_get_low_limit.m

2

C++ Library

2.1 onda_lib

```
2 #ifndef C_PRINT_H
3 #define C_PRINT_H
4
5 #ifndef MATLAB_MEX_FILE
6 #define err_printf printf
7 #define info_printf printf
8 #else
9 #include "mex.h"
10 #define info_printf mexPrintf
11 #define err_printf mexErrMsgTxt
12 #endif
13
14 #endif
```

Listing 2.1: err_printf.h

```
1
2 #ifndef ONDA_POSITIONER_H_
3 #define ONDA_POSITIONER_H_
4
5
6
7 int PositionerMoveRel(int axis, float value);
8 int PositionerMoveAbs(int axis, float value);
9 int SetPosition(int axis, float value);
10 int GetPosition(int axis, float* result);
11
12 int SetPositionerLowLimit(int axis, float value);
13 int GetPositionerLowLimit(int axis, float* result);
14 int SetPositionerHighLimit(int axis, float value);
15 int GetPositionerHighLimit(int axis, float* result);
16 int GetPositionerStepsPerSecond(int axis, int* result);
17 int GetPositionerMinStepsPerSecond(int axis, int* result);
18
19 #endif /* !ONDA_POSITIONER_H_*/
```

Listing 2.2: onda_positioner.h

```
/**
2 * onda_lib.cpp
3 * This file makes the connection between MATLAB and the C++ library ↔
4 * which
5 * connects to the Onda system.
6 *
7 */
8
```

```

10 #include <cstdio>
12 #include <cstring>
14 #include <mex.h>
16 #include "onda_positioner.h"
18 #include "tcp.h"
20 #include "err_printf.h"

18 #define HOST_PORT 49999

20 void
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
22 {

24     int axis = 0, retval;
26     float value = 0;
28     float* value_p = NULL;

30     /* string buffers */
32     enum { host_addr_len = 20;
34     enum { func_str_len = 30;
36     char host_addr[host_addr_len];
38     char func_str[func_str_len];

40     /* output pointers */
42     mxArray* val_p = NULL;

44     if (nrhs < 1) err_printf("Minimum one input required.");
46     if (nlhs > 1) err_printf("Maximum one output argument allowed.");

48     /* get input vars */
50     mxGetString(prhs[0], func_str, func_str_len);
52     if (nrhs > 1) axis = (int)mxGetScalar(prhs[1]);
54     if (nrhs > 2) value = (float)mxGetScalar(prhs[2]);

56     /* set output pointer */
58     val_p = plhs[0];
60     val_p = mxCreateDoubleMatrix(1, 1, mxREAL);

62     /* Get Version of this library */
64     // TODO: Get define the version during compiling.
66     if (strcmp("version", func_str) == 0){
68         info_printf("Onda connection library, version 1.3, 2013-04-22.\n");
69     }

70     /* move relative */
72     else if (strcmp("move_relative", func_str) == 0){
74         if (nrhs != 3) err_printf("This function requires three input arguments.");

76         retval = PositionerMoveRel(axis, value);
78         if (retval){
80             err_printf("Error in Positioner move relative\n");
82             return;
84         }

```

```

70     }

72     /* move absolute */
73     else if (strcmp("move_absolute", func_str) == 0){
74         if (nrhs != 3) err_printf("This function requires three input ↵
arguments.");

76         retval = PositionerMoveAbs(axis, value);
77         if (retval) err_printf("Error in positioner move absolut.");
78     }

80     /* set position */
81     else if (strcmp("set_position", func_str) == 0){
82         if (nrhs != 3) err_printf("This function requires three input ↵
arguments.");

84         retval = SetPosition(axis, value);
85         if (retval) err_printf("Error in Set position.");
86     }

88     /* get position */
89     else if (strcmp("get_position", func_str) == 0){
90         if (nrhs != 2) err_printf("This function requires two input ↵
arguments.");

92         retval = GetPosition(axis, value_p);
93         if (retval) err_printf("Error in Get position.");
94         info_printf("Onda pos: %f\n", *value_p);
95         /* set output */
96         mxGetPr(val_p)[0] = *value_p;
97     }

98     /* Get low limit */
99     else if (strcmp("get_low_limit", func_str) == 0){
100         if (nrhs != 2) err_printf("This function requires two input ↵
arguments.");

102         retval = GetPositionerLowLimit(axis, value_p);
103         if (retval) err_printf("Onda: Error in Get low limit.");
104         /* set output */
105         mxGetPr(val_p)[0] = *value_p;
106     }

108     /* set low limit */
109     else if (strcmp("set_low_limit", func_str) == 0){
110         if (nrhs != 3) err_printf("This function requires three input ↵
arguments.");

112         retval = SetPositionerLowLimit(axis, value);
113         if (retval) err_printf("Onda: Error in Set low limit.");
114     }

116     /* Get high limit */
117     else if (strcmp("get_high_limit", func_str) == 0){
118         if (nrhs != 3) err_printf("This function requires three input ↵
arguments.");

```

```

126     retval = GetPositionerHighLimit(axis, value_p);
127     if (retval) err_printf("Error in Get high limit.");
128     info_printf("Onda: Pos low limit: %f\n", *value_p);
129     /* set output */
130     mxGetPr(val_p)[0] = *value_p;
131 }
132
133 /* Set high limit */
134 else if (strcmp("set_high_limit", func_str) == 0){
135     if (nrhs != 3) err_printf("This function requires three input ↵
136         arguments.");
137
138     retval = SetPositionerHighLimit(axis, value);
139     if (retval) err_printf("Error in Set high limit.");
140 }
141
142 /* Init connection */
143 else if (strcmp("init_connection", func_str) == 0){
144     if (nrhs != 2) err_printf("This function requires two input ↵
145         arguments.");
146
147     mxGetString(prhs[1], host_addr, host_addr_len);
148     retval = tcp_init (host_addr, HOST_PORT);
149     if (retval)
150         err_printf("Onda: Could not initialise connection.\n");
151     else
152         info_printf("Onda: connection initialised.\n");
153
154     /* automatic close connection when MATLAB clears or exits */
155     mexAtExit(tcp_auto_term);
156 }
157
158 /* terminate connection */
159 else if (strcmp("terminate_connection", func_str) == 0){
160     retval = tcp_term ();
161     if (retval)
162         err_printf("Onda: Could not terminate connection.\n");
163     else
164         info_printf("Onda connection terminated.\n");
165 }
166
167
168
169
170 /* CMD not found*/
171 else {
172     err_printf("Command not found.");
173 }
174 }

```

Listing 2.3: onda_lib.cpp

2.2 TCP connection

```

1  /*
2  ** tcp_lib.h
3  **
4  ** Made by Morten Fischer Rasmussen
5  ** Login    <mf@mf-black>
6  **
7  ** Started on Wed Jul 30 20:21:19 2008 Morten Fischer Rasmussen
8  ** Last update Wed Jul 30 20:21:19 2008 Morten Fischer Rasmussen
9  */
10
11 #ifndef TCP_H_
12 #define TCP_H_
13
14 #define ONDA_NOT_CONNECTED 0
15 #define ONDA_CONNECTED 1
16
17 /* struct contraining all connection information */
18 typedef struct connection
19 {
20     char* name;
21     int buf_size;
22     char *rx_buf;
23     char *tx_buf;
24     char* addr;
25     unsigned int port;
26     int connected;
27     int sock;
28 } connection_t ;
29
30 /* typedef struct connection Connection; */
31
32
33 /* prototypes */
34 extern int tcp_init (const char* host_addr, const int host_port);
35 extern int tcp_term (void);
36 extern void tcp_auto_term (void);
37 extern int tcp_tx (const char* send_str);
38 extern int tcp_rx (char** data_ptr);
39 extern int tcp_query (const char* send_str, char** data_ptr);
40 //extern int tcp_clear_rx_buff (void);
41
42
43 /* error numbers */
44
45 //const char *ERR_RCV_TIMEOUT_STR ="Connection timed out. No data was←
46     received.\n";
47 enum {ERR_RCV_TIMEOUT = -10};
48 enum {ERR_RCV_SYS_ERR = -11};
49 enum {ERR_RCV_NO_DATA = -12};
50 enum {ERR_RCV_TIMEOUT2 = -13};
51
52 #endif /* !TCP_H_ */

```

Listing 2.4: tcp.h

```

1  /*
2  ** tcp.cpp

```

```

3  **
4  ** This file implements the TCP connection to the Onda system.
5  ** It was originally written in C for the SARUS project.
6  **
7  ** $Id: tcp.cpp 6 2012-04-16 15:40:01Z mf $
8  **
9  ** Made by (Morten Fischer Rasmussen)
10 ** Login    <mf@mf-black>
11 **
12 ** Started on  Wed Jul 30 20:22:23 2008 Morten Fischer Rasmussen
13 ** Modified for Onda on Wed Feb 22 12:27:10 2012 Morten Fischer ←
14 **      Rasmussen
15 */
16
17 #include <stdio.h>
18 #include <sys/socket.h>
19 #include <arpa/inet.h>
20 #include <stdlib.h>
21 #include <string.h>
22 #include <unistd.h> /* provides close () */
23 #include <netinet/in.h>
24 /* #include <fcntl.h> */
25 #include <errno.h>
26 #include <sys/select.h>
27 #include <sys/time.h>
28
29 #include "tcp.h"
30 #include "err_printf.h"
31
32 #define ONDA_BUFF_SIZE 9096
33 #define ONDA_PORT 49999
34
35 /* input/output buffers */
36 static char rx_buf[ONDA_BUFF_SIZE];
37 static char tx_buf[ONDA_BUFF_SIZE];
38 static char onda_addr[30];
39
40 static connection_t onda_st = {(char*)"Onda",
41                                ONDA_BUFF_SIZE,
42                                rx_buf,
43                                tx_buf,
44                                onda_addr,
45                                ONDA_PORT,
46                                ONDA_NOT_CONNECTED,
47                                0};
48
49
50
51
52
53 /**
54  * Clears the local buffer and system buffer
55  */
56 static int
57 tcp_clear_rx_buff (void)
58 {
59     int retval;
60     int nfd = onda_st.sock + 1;
61     fd_set read_fd;
62     struct timeval timeout;
63     /* make sure pointer does not contain random value */

```



```

65     // *data_ptr = NULL;
66     if (onda_st.connected != ONDA_CONNECTED){
67     err_printf ("Error on clearing buffer: Cannot receive on non-↵
        existing connection.\n");
        return -1;
69     }

71     /* clear the struct */
    FD_ZERO (&read_fd);
73     /* add the socket to the struct/list */
    FD_SET (onda_st.sock, &read_fd);
75     /* set timeout to 0.001 sec */
    timeout.tv_sec = 0;
77     timeout.tv_usec = 1000;
    /* Wait until data is available or timeout has passed */
79     retval = select (nfds, &read_fd, NULL, NULL, &timeout);

81     if (retval > 0) /* data present in buffer */
    {
83         /* clear buffer */
        /* data available -> receive it */
85         recv (onda_st.sock, onda_st.rx_buf, onda_st.buf_size -1, 0);
    }

87     /* Reset local buffer */
89     rx_buf [0] = '\0';

91     return 0;
    }
93
95
97
99 /**
    * Creates a socket that uses TCP/IP and connects the destination adr↵
    * +port
    * Input:  -struct containing all connection info
    101 *         -pointer to a char buffer where error messages are ↵
    *         printed
    *
    * Returns: 0 on success
    *         -1 on failure
    105 */
    int
    107 tcp_init (const char *addr_host, const int port_host)
    {
    109         if (onda_st.connected == ONDA_CONNECTED)
    111         {
            err_printf ("Onda: Connection already initialised.\n");
            113             return -1;
        }
    115         sprintf(onda_st.addr, "%s", addr_host);
        onda_st.port = port_host;
    117         /* clear buffers */
        rx_buf[0] = '\0';
    119         tx_buf[0] = '\0';

    121
        struct sockaddr_in dst_server;

```

```

123     if (onda_st.connected != ONDA_NOT_CONNECTED){
125     info_printf ("Onda interface err.: Can not initialize an already ↵
        existing connection.\n",
        onda_st.name);
127     return -1;
        }
129
131     /* input validation */
        if (onda_st.port > 65535){
133     info_printf ("Failed to create socket for %s: maximum value of ↵
        destination port is 65535\n", onda_st.name);
        return -1;
135     }
        /* We assume that destination addresses are always valid (this is ↵
        not necessary true) */
137     /* Create the IP/TCP socket */
        if ((onda_st.sock = socket (PF_INET, SOCK_STREAM, IPPROTO_TCP)) <↵
            0){
139     info_printf ("Failed to create socket for %s.\nSystem returned: %s↵
        .\n", onda_st.name, strerror(errno));
        return -1;
141     }
143
        /* Construct the server sockaddr_in structure */
        memset (&dst_server, 0, sizeof (dst_server));           /* Clear ↵
        struct */
145     dst_server.sin_family = AF_INET;                           /* ↵
        Internet/IP */
        dst_server.sin_addr.s_addr = inet_addr (onda_st.addr);   /* IP ↵
        address */
147     dst_server.sin_port = htons (onda_st.port);                /* ↵
        server port */
149
        /* Establish Connection */
        if (connect (onda_st.sock, (struct sockaddr *) &dst_server, ↵
            sizeof(dst_server)) < 0){
151     info_printf ("Error: Failed to connect with %s on address: %s port:↵
        %i.\nSystem returned: %s.\n",
        onda_st.name, onda_st.addr, onda_st.port, strerror(errno));
153     return -1;
        }
        else
155     onda_st.connected = ONDA_CONNECTED;
157
        return 0;
159 }
161
163
165
167 int
tcp_term (void)
169 {
    int retval;
171     if (onda_st.connected != ONDA_CONNECTED){
        info_printf ("Onda interface err.: Can not close non existing ↵
        connection: %s.\n",

```

```

173         onda_st.name);
174     return -1;
175 }

176     retval = close(onda_st.sock);
177     if (retval == 0){
178         onda_st.connected = ONDA_NOT_CONNECTED;
179         onda_st.sock = 0;
180     } else {
181         info_printf("Onda interface: closed connection.\n");
182     } else {
183         info_printf ("Onda interface err.: Could not terminate connection.\n↵
184             nSytem returned: %s.\n", strerror(errno));
185         return retval;
186     }

187     return 0;
188 }

189

190

191

192

193

194

195 void
196 tcp_auto_term (void)
197 {
198     if (onda_st.connected == ONDA_CONNECTED)
199         tcp_term();
200
201     return;
202 }
203
204
205
206
207
208
209
210
211 /**
212  * Sends raw data using the socket handle
213  * Input:  -struct containing all connection info
214  *          -pointer to a char buffer where error messages are ↵
215  *          printed
216  *
217  * Returns: 0 on success
218  *          -1 on failure
219  */
219 int
220 tcp_tx (const char* send_str)
221 {
222     int echolen;
223     int retval;
224     /* set the maximum number of file descriptors */
225     int nfds = onda_st.sock + 1;
226     fd_set write_fd;
227     struct timeval timeout;

228     if (onda_st.connected != ONDA_CONNECTED){
229         info_printf ("Error: can not send data. Connection is non ↵
230             initialised.\n");
231         return -1;

```

```

233     }
234
235     /* Clear the receive buffers */
236     retval = tcp_clear_rx_buff();
237     if (retval)
238         return -1;
239
240     /* clear the struct */
241     FD_ZERO (&write_fd);
242     /* add the socket to the struct/list */
243     FD_SET (onda_st.sock, &write_fd);
244
245     /* set timeout to 5 sec */
246     timeout.tv_sec = 5;
247     timeout.tv_usec = 0;
248
249     /* Wait until there is room in the socket send buffer or select() ←
250        times out */
251     retval = select (nfds, NULL, &write_fd, NULL, &timeout);
252
253     switch (retval)
254     {
255         /* timeout */
256         case 0:
257             info_printf ("Unable to send data. Socket is not ready to send ←
258                          data.\n");
259             return -1;
260             break;
261
262         /* error */
263         case -1:
264             info_printf ("Unable to send data.\nSystem returned: %s\n", ←
265                          strerror(errno));
266             return -1;
267             break;
268
269         /* AOK */
270         default: break;
271     }
272
273     /* Send the string to the server */
274     /* echolen = strlen (onda_st.tx_buf); */
275     /* retval = send (onda_st.sock, onda_st.tx_buf, echolen, 0); /* ←
276        "0" for IP-protocol */
277     echolen = strlen (send_str);
278     retval = send (onda_st.sock, send_str, echolen, 0); /* "0" for ←
279        IP-protocol */
280
281     /* test for success */
282     if (retval == -1)
283     {
284         info_printf ("Unable to send data\nSystem returned: %s\n", ←
285                      strerror(errno));
286         return -1;
287     }
288     else if (retval != echolen)
289     {
290         info_printf ("Unable to send all data. Should have sent: %i, ←
291                      but only %i bytes was sent.\nSystem returned: %s\n",
292                      echolen, retval, strerror(errno));
293         return -1;

```

```

287     }
288     return 0;
289 }
290
291
292
293
294
295
296
297 /**
298  * Receives data using the socket handle
299  * Input:  -struct containing all connection info
300  *          -pointer to a char buffer where error messages are ↵
301  *          printed
302  * Returns: 0 on success
303  *          -1 on failure
304  */
305 int
306 tcp_rx (char** data_ptr)
307 {
308     int retval;
309     int nfd = onda_st.sock + 1;
310     fd_set read_fd;
311     struct timeval timeout;
312     /* make sure pointer does not contain random value */
313     // *data_ptr = NULL;
314
315     if (onda_st.connected != ONDA_CONNECTED){
316         info_printf ("Onda interface err.: Can not receive on non-existing ↵
317                     connection.\n");
318         return -1;
319     }
320
321     /* clear the struct */
322     FD_ZERO (&read_fd);
323     /* add the socket to the struct/list */
324     FD_SET (onda_st.sock, &read_fd);
325     /* set timeout to 5 sec */
326     timeout.tv_sec = 5;
327     timeout.tv_usec = 0;
328     /* Wait until data is available or timeout has passed */
329     retval = select (nfd, &read_fd, NULL, NULL, &timeout);
330
331     /* Wait a little extra time, to assure the entire string is ↵
332        received. */
333     timeout.tv_sec = 0;
334     timeout.tv_usec = 50000; /* 0.05sec */
335     select (nfd, NULL, NULL, NULL, &timeout);
336
337     switch (retval)
338     {
339         /* time out */
340         case 0:
341             return ERR_RCV_TIMEOUT;
342             break;
343
344         /* error */
345         case -1:
346             info_printf ("Error: could not receive data.\nSystem returned: ↵

```

```

        "%s.\n", strerror(errno));
        return ERR_RCV_SYS_ERR;
        break;
347
    /* AOK */
    default: break;
349
    }
351

    /* data available -> receive it */
    if ((retval = recv (onda_st.sock, onda_st.rx_buf, onda_st.↵
        buf_size -1, 0)) < 0){
353
    rx_buf [0] = '\0';
355
    /* error handling */
    info_printf("Error: could not receive data.\nSystem returned: %s.\n↵
        ", strerror(errno));
357
    /* Assure null terminated string */
    return ERR_RCV_NO_DATA;
359
    }
    else
361
    /* Assure null terminated string */
    rx_buf [retval] = '\0';
363

365
    /* set pointer to buffer */
    *data_ptr = rx_buf;
367

    return 0;
369
}
371

373
375

377 /**
    * Sends AND receives data using the socket handle
379
    * Input:  -c-string containing the TX string
    *          -pointer to a char buffer where error messages are ↵
    *          printed
381
    * Returns: 0 on success
    *          -1 on failure
383
    */
385 int
tcp_query (const char* send_str, char** receive_ptr)
387 {
    int retval;
389
    retval = tcp_tx(send_str);
    if (retval)
391
    return retval;

    /* receive string */
    retval = tcp_rx(receive_ptr);
393
    if (retval)
395
    return retval;

    return 0;
397

}
399

```

Listing 2.5: tcp.cpp