

```
1: // Copyright 2017 Patrick Muldoon
2: #include "intouch.hpp"
3: #include <string>
4: #include <iostream>
5: #include <boost/date_time/posix_time/posix_time.hpp>
6:
7: Intouch::Intouch(std::string start_line, std::string _filename, unsigned
int line_number) {
8:     filename= _filename;
9:     success = false;
10:    start_line_number = line_number;
11:    end_line_number = 0;
12:    end_time = "";
13:    boot_time = 0;
14:
15:
16:    boost::smatch sm;
17:    boost::regex_match(start_line, sm, StartRegex);
18:    start_time = sm[1] + "-" + sm[2] + "-" + sm[3] + " " + sm[4]
19:                + ":" + sm[5] + ":" + sm
[6];
20: }
21:
22: void Intouch::BootSuccess(std::string successful_line, unsigned int line_
number) {
23:     success = true;
24:     end_line_number = line_number;
25:
26:     boost::smatch sm;
27:     boost::regex_match(successful_line, sm, SucceededRegex);
28:     end_time = sm[1] + "-" + sm[2] + "-" + sm[3] + " " + sm[4] + ":"
+ sm[5]
29:                + ":" + sm[6];
30:
31:     boot_time = Time_Elapsed();
32: }
33:
34: std::ostream& operator<< (std::ostream &out, const Intouch &it) {
35:     out << "=== Device Boot ===" << std::endl;
36:     out << it.start_line_number << "(" << it.filename << "): "
<< it.start_time << " Boot Start" << std::endl;
38:
39:     if(it.success == true) {
40:         out << it.end_line_number << "(" << it.filename << "): "
<< it.end_time
41:         << " Boot Completed" << std::endl;
42:         out << "\tBoot Time: " << it.boot_time << "ms" << std::en
dl;
43:     }else {
44:         out << "**** Incomplete Boot ****" << std::endl;
45:     }
46:
47:     return out;
48: }
49:
50: std::ostream& operator<< (std::ostream &out, const Services &service)
51: {
52:     out << "\t" << service.service_name << std::endl;
53:     out << "\t\tStart: " << service.start_line_number << "(" << servi
ce.filename
54:     << ")" << std::endl;
55:     if(service.success == true) {
56:         out << "\t\tCompleted: " << service.end_line_number << "
(" << service.filename
57:         << ")" << std::endl;
```

```
58:             out << "\\t\\tElapsed Time: " << service.boot_time << " ms"
<< std::endl;
59:         } else {
60:             out << "\\t\\tCompleted: Not completed" << "(" << service.f
ilename << ")" << std::endl;
61:             out << "\\t\\tElapsed Time: " << std::endl;
62:         }
63:         return out;
64:     }
65:
66: unsigned int Intouch::Time_Elapsed() {
67:     boost::posix_time::ptime start;
68:     start = (boost::posix_time::time_from_string(start_time));
69:     boost::posix_time::ptime end;
70:     end = (boost::posix_time::time_from_string(end_time));
71:     boost::posix_time::time_duration total;
72:     total = end - start;
73:     return total.total_milliseconds();
74: }
75:
76: Services::Services(std::string start_line, std::string _filename, unsigne
d int line_number)
77: {
78:     filename = _filename;
79:     success = false;
80:     start_line_number = line_number;
81:     end_line_number = 0;
82:     boot_time = "";
83:
84:     boost::smatch sm;
85:     boost::regex_match(start_line, sm, ServiceStart);
86:     service_name = sm[1];
87:
88:
89: }
90:
91: void Services::ServiceBoot(std::string successful_line, unsigned int line
_number)
92: {
93:     success = true;
94:     end_line_number = line_number;
95:     boost::smatch sm;
96:     boost::regex_match(successful_line, sm, ServiceSuccess);
97:     boot_time = sm[3];
98: }
99:
100: Softload::Softload(std::string start_line, std::string _filename, unsigne
d int line_number)
101: {
102:     filename= _filename;
103:     success = false;
104:     start_line_number = line_number;
105:     end_line_number = 0;
106:     end_time_soft = "";
107:     boot_time = 0;
108:
109:
110:     boost::smatch sm;
111:     boost::regex_match(start_line, sm, SoftLoadBegin);
112:     start_time_soft = sm[1] + " " + sm[2] + " " + sm[3] + ":" + sm[4]
+ ":" + sm[5];
113:     boost::smatch z;
114:     boost::regex_match(start_line, z, SoftLoadBegin);
115:     begin = z[3] + ":" + z[4] + ":" + z[5];
116: }
```

```
117:
118: void Softload::Originalver(std::string successful_line)
119: {
120:     boost::smatch sm;
121:     boost::regex_match(successful_line, sm, Original);
122:     oldSoftLoad = sm[6] + "." + sm[7] + "." + sm[8] + "-" + sm[9];
123: }
124:
125: void Softload::Newver(std::string successful_line)
126: {
127:     boost::smatch sm;
128:     boost::regex_match(successful_line, sm, New);
129:     newSoftLoad = sm[6] + "." + sm[7] + "." + sm[8] + "-" + sm[9];
130: }
131:
132: void Softload::SoftloadSuccess(std::string successful_line, unsigned int
line_number)
133: {
134:     success = true;
135:     end_line_number = line_number;
136:
137:     boost::smatch sm;
138:     boost::regex_match(successful_line, sm, SoftLoadEnd);
139:     end_time_soft = sm[1] + " " + sm[2] + " " + sm[3] + ":" + sm[4] +
":" + sm[5];
140:     boost::smatch z;
141:     boost::regex_match(successful_line, z, SoftLoadEnd);
142:     stop = z[3] + ":" + z[4] + ":" + z[5];
143:     //boot_time = Time();
144:
145: }
146:
147: unsigned int Softload::Time()
148: {
149:     std::cout << "breaking here\n";
150:     std::cout << getStartTime() << " " << getEndTime() << std::endl;
151:     std::cout << getBegin() << " " << getStop() << std::endl;
152:     boost::posix_time::ptime x(boost::posix_time::time_from_string(be
gin));
153:     std::cout << "broke\n";
154:     boost::posix_time::ptime z(boost::posix_time::time_from_string(en
d_time_soft));
155:     boost::posix_time::time_duration total;
156:     total = z - x;
157:     return total.total_milliseconds();
158: }
```