```cpp
#include "ADC.h"

namespace Hardware
{
    /*! Constructor
    \param pin and ADCPin type indicating which analogue pin to use
    */
    ADC::ADC(ADCPin pin)
    {
        this->Pin = pin;
        switch (pin)
        {
        case Hardware::ADC::ADC0:
            adcpath = ADC0_PATH;
            break;
        case Hardware::ADC::ADC1:
            adcpath = ADC1_PATH;
            break;
        case Hardware::ADC::ADC2:
            adcpath = ADC2_PATH;
            break;
        case Hardware::ADC::ADC3:
            adcpath = ADC3_PATH;
            break;
        case Hardware::ADC::ADC4:
            adcpath = ADC4_PATH;
            break;
        case Hardware::ADC::ADC5:
            adcpath = ADC5_PATH;
            break;
        case Hardware::ADC::ADC6:
            adcpath = ADC6_PATH;
            break;
        case Hardware::ADC::ADC7:
            adcpath = ADC7_PATH;
            break;
        }

        MinIntensity = 0;
        MaxIntensity = 4096;
```

```
41      }
42
43      /*! De-constructor*/
44      ADC::~ADC() { }
45
46      /*! Reads the current voltage in the pin
47      \return an integer between 0 and 4096
48      */
49      int ADC::GetCurrentValue()
50      {
51          int retVal = StringToNumber<int>(Read(adcpath));
52          Intensity = (float)(retVal - MinIntensity)/(4096 - (MinIntensity + (4096 - MaxIntensity)));
53          return retVal;
54      }
55
56      /*! Set the current voltage at the pin as the minimum voltage*/
57      void ADC::SetMinIntensity()
58      {
59          MinIntensity = StringToNumber<int>(Read(adcpath));
60      }
61
62      void ADC::SetMaxIntensity()
63      {
64          MaxIntensity = StringToNumber<int>(Read(adcpath));
65      }
66
67      /*! Threading enabled polling of the analogue pin
68      \param callback the function which should be called when polling indicates a change CallbackType
69      \return 0
70      */
71      int ADC::WaitForValueChange(CallbackType callback)
72      {
73          threadRunning = true;
74          callbackFunction = callback;
75          if (pthread_create(&thread, NULL, &threadedPollADC, static_cast<void*>(this)))
76          {
77              threadRunning = false;
78              throw Exception::FailedToCreateGPIOPollingThreadException();
79          }
80          return 0;
81      }
82
83
```

```
84      \return the current value
85      */
86      int ADC::WaitForValueChange()
87      {
88          int fd, i, epollfd, count = 0;
89          struct epoll_event ev;
90          epollfd = epoll_create(1);
91          if (epollfd == -1)
92          {
93              throw Exception::FailedToCreateGPIOPollingThreadException("GPIO: Failed to create epollfd!");
94          }
95          if ((fd = open(adcpath.c_str(), O_RDONLY | O_NONBLOCK)) == -1) { throw Exception::ADCReadException();   }
96          ev.events = EPOLLIN;
97          ev.data.fd = fd;
98
99          if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) { throw Exception::FailedToCreateGPIOPollingThreadException("ADC:
              Failed to add control interface!"); }
100
101         while (count <= 1)
102         {
103             i = epoll_wait(epollfd, &ev, 1, -1);
104             if (i == -1)
105             {
106                 close(fd);
107                 return -1;
108             }
109             else { count++; }
110         }
111         close(fd);
112         return StringToNumber<int>(Read(adcpath));
113     }
114
115     /*! friendly function to start the threading*/
116     void *threadedPollADC(void *value)
117     {
118         ADC *adc = static_cast<ADC*>(value);
119         while (adc->threadRunning)
120         {
121             adc->callbackFunction(adc->WaitForValueChange());
122             usleep(200000);
123         }
124     }
```

```
125 }
```