

A Practical Guide to Automated Testing

May 30th 2018

Andrew Jeffery
andrewrj@au1.ibm.com

Assumptions

- C or C++
- Autotools-based project
- Icov
 - Ubuntu: ``sudo apt install lcov``
- Use of Google Test and Google Mock libraries

phosphor-led-sysfs

<https://gerrit.openbmc-project.xyz/#/q/project:openbmc/phosphor-led-sysfs+topic:testing>

Automated testing...

- Is automated code review
 - Less work to do once the tests are in place
- Provides confidence code is working as expected
- Enforces expected behaviour in the face of change

But only for the execution paths
that your tests touch!

The first step

<https://gerrit.openbmc-project.xyz/#/c/10827/>

```
diff --git a/configure.ac b/configure.ac
index 6a08be339e16..211e4efe1b02 100644
--- a/configure.ac
+++ b/configure.ac
@@ -29,6 +29,8 @@ PKG_CHECK_MODULES([PHOSPHOR_DBUS_INTERFACES], [phosphor-dbus-interfaces], [AC_M
AX_PTHREAD([GTEST_CPPFLAGS="-DGTEST_HAS_PTHREAD=1"], [GTEST_CPPFLAGS="-DGTEST_HAS_PTHREAD=0"])
AC_SUBST(GTEST_CPPFLAGS)
```

+AX_CODE_COVERAGE

```
+
AC_ARG_VAR(BUSNAME, [The Dbus busname to own])
AS_IF([test "x$BUSNAME" == "x"], [BUSNAME="xyz.openbmc_project.LED.Controller"])
AC_DEFINE_UNQUOTED([BUSNAME], ["$BUSNAME"], [The Dbus busname to own])
```

The first step

<https://gerrit.openbmc-project.xyz/#/c/10827/>

```
diff --git a/Makefile.am b/Makefile.am
index 3c42e07184d7..a5e625fc9160 100644
--- a/Makefile.am
+++ b/Makefile.am
@@ -9,3 +9,10 @@ phosphor_ledcontroller_LDFLAGS = $(SDBUSPLUS_LIBS) \
        $(PHOSPHOR_DBUS_INTERFACES_LIBS)
 phosphor_ledcontroller_CFLAGS = $(SDBUSPLUS_CFLAGS) \
        $(PHOSPHOR_DBUS_INTERFACES_CFLAGS)
+
+@CODE_COVERAGE_RULES@
+
+check_PROGRAMS =
+XFAIL_TESTS =
+
+TESTS = $(check_PROGRAMS)
```

The first result

```
$ ./configure --enable-code-coverage && make check-code-coverage
```

```
...
```

```
=====  
Testsuite summary for phosphor-led-sysfs 1.0  
=====
```

```
# TOTAL: 0  
# PASS: 0  
# SKIP: 0  
# XFAIL: 0  
# FAIL: 0  
# XPASS: 0  
# ERROR: 0  
=====
```

```
make[3]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
```

```
make[2]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
```

```
make[1]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
```

```
make[1]: Entering directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
```

```
LCOV --capture phosphor-led-sysfs-1.0-coverage.info
```

```
geninfo: WARNING: no .gcda files found in . - skipping!
```

```
LCOV --remove /tmp/*
```

```
lcov: ERROR: no valid records found in tracefile phosphor-led-sysfs-1.0-coverage.info.tmp
```

```
Makefile:1329: recipe for target 'code-coverage-capture' failed
```

```
make[1]: *** [code-coverage-capture] Error 255
```

```
make[1]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
```

```
Makefile:1323: recipe for target 'check-code-coverage' failed
```

```
make: *** [check-code-coverage] Error 2
```


phosphor-led-sysfs

```
$ git ls-files | grep '\.[ch]pp'
```

```
argument.cpp
```

```
argument.hpp
```

```
controller.cpp
```

```
physical.cpp
```

```
physical.hpp
```

```
$
```

phosphor-led-sysfs

- **argument.[ch]pp**: Argument parsing
- **controller.cpp**: main(), orchestrates everything
- **physical.[ch]pp**: Testable business logic!
 - Converts Dbus API to kernel LED sysfs ABI

Testing physical.cpp

<https://gerrit.openbmc-project.xyz/#/c/10829/>

```
diff --git a/Makefile.am b/Makefile.am
```

```
index a5e625fc9160..429415b54688 100644
```

```
--- a/Makefile.am
```

```
+++ b/Makefile.am
```

```
@@ -15,4 +15,6 @@ phosphor_ledcontroller_CFLAGS = $(SDBUSPLUS_CFLAGS) \
```

```
check_PROGRAMS =
```

```
XFAIL_TESTS =
```

```
+include test/Makefile.am.include
```

```
+
```

```
TESTS = $(check_PROGRAMS)
```

Testing physical.cpp

<https://gerrit.openbmc-project.xyz/#/c/10829/>

```
diff --git a/test/Makefile.am.include b/test/Makefile.am.include
new file mode 100644
index 000000000000..32b97f79c6ef
--- /dev/null
+++ b/test/Makefile.am.include
@@ -0,0 +1,13 @@
+GTEST_LIBS = -lgtest -lgtest_main -lgmock
+AM_CPPFLAGS = $(CODE_COVERAGE_CPPFLAGS) $(PTHREAD_CPPFLAGS) $(SDBUSPLUS_CPPFLAGS) $(PHOSPHOR_DBUS_INTERFACES_CPPFLAGS)
+AM_CFLAGS = $(CODE_COVERAGE_CFLAGS) $(PTHREAD_CFLAGS) $(SDBUSPLUS_CFLAGS) $(PHOSPHOR_DBUS_INTERFACES_CFLAGS)
+AM_CXXFLAGS = $(CODE_COVERAGE_CXXFLAGS) $(PTHREAD_CXXFLAGS) $(SDBUSPLUS_CXXFLAGS) $(PHOSPHOR_DBUS_INTERFACES_CXXFLAGS)
+AM_LDFLAGS = $(CODE_COVERAGE_LIBS) $(SDBUSPLUS_LIBS) $(PHOSPHOR_DBUS_INTERFACES_LIBS) $(GTEST_LIBS) $(PTHREAD_CFLAGS)
+
+test_physical_SOURCES = physical.cpp %reldir%/physical.cpp
+test_physical_CPPFLAGS = $(AM_CPPFLAGS) $(GTEST_CPPFLAGS)
+
+check_PROGRAMS += %reldir%/physical
```

Testing physical.[ch]pp

```
diff --git a/test/physical.cpp b/test/physical.cpp
new file mode 100644
index 000000000000..511bbfa4f70e
--- /dev/null
+++ b/test/physical.cpp
@@ -0,0 +1,36 @@
+#include <gtest/gtest.h>
+#include <sdbusplus/bus.hpp>
+
+#include "physical.hpp"
+
+constexpr auto LED_OBJ = "/foo/bar/led";
+constexpr auto LED_SYSFS = "/sys/class/leds/test";
+
+using Action = sdbusplus::xyz::openbmc_project::Led::server::Physical::Action;
+
+TEST(Physical, ctor)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    phosphor::led::Physical led(bus, LED_OBJ, LED_SYSFS);
+}
+
+TEST(Physical, off)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    phosphor::led::Physical led(bus, LED_OBJ, LED_SYSFS);
+    led.state(Action::Off);
+}
+
+TEST(Physical, on)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    phosphor::led::Physical led(bus, LED_OBJ, LED_SYSFS);
+    led.state(Action::On);
+}
+
+TEST(Physical, blink)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    phosphor::led::Physical led(bus, LED_OBJ, LED_SYSFS);
+    led.state(Action::Blink);
+}
```

The second result

```
$ ./configure --enable-code-coverage && make check-code-coverage
```

```
...
```

```
=====  
Testsuite summary for phosphor-led-sysfs 1.0  
=====
```

```
# TOTAL: 1
```

```
# PASS: 1
```

```
# SKIP: 0
```

```
# XFAIL: 0
```

```
# FAIL: 0
```

```
# XPASS: 0
```

```
# ERROR: 0  
=====
```

```
make[3]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
```

```
make[2]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
```

```
make[1]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
```

```
make[1]: Entering directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
```

```
LCOV --capture phosphor-led-sysfs-1.0-coverage.info
```

```
LCOV --remove /tmp/*
```

```
GEN phosphor-led-sysfs-1.0-coverage
```

```
file:///home/andrew/src/openbmc/phosphor-led-sysfs/phosphor-led-sysfs-1.0-coverage/index.html
```

```
make[1]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
```

The second result

LCOV - phosphor-led-sysfs

Person 1

file:///home/andrew/src/openbmc/phosphor-led-sysfs/phosphor-led-sysfs-1.0-coverage/index.html

LCOV - code coverage report

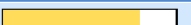
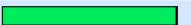
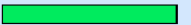




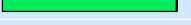
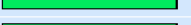
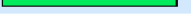
Current view: **top level**

Test: **phosphor-led-sysfs-1.0 Code Coverage**

Date: **2018-05-26 22:33:23**

Legend: Rating: **low: < 75 %** **medium: >= 75 %** **high: >= 90 %**

	Hit	Total	Coverage
Lines:	220	258	85.3 %
Functions:	49	63	77.8 %

Directory	Line Coverage	Functions
/home/andrew/src/openbmc/phosphor-led-sysfs	 79.4 % 50 / 63	92.9 % 13 / 14
/home/andrew/src/openbmc/phosphor-led-sysfs/test	 100.0 % 19 / 19	71.4 % 10 / 14
/usr/include/c++/7	 100.0 % 14 / 14	- 0 / 0
/usr/include/c++/7/bits	 90.0 % 81 / 90	100.0 % 8 / 8
/usr/include/c++/7/ext	 34.8 % 8 / 23	50.0 % 1 / 2
/usr/include/gtest	 66.7 % 2 / 3	66.7 % 2 / 3
/usr/include/gtest/internal	 100.0 % 5 / 5	66.7 % 8 / 12
/usr/local/include/sdbusplus	 100.0 % 23 / 23	100.0 % 4 / 4
/usr/local/include/sdbusplus/server	 100.0 % 17 / 17	75.0 % 3 / 4
/usr/local/include/xyz/openbmc_project/Led/Physical	 100.0 % 1 / 1	0.0 % 0 / 2

Generated by: [LCOV version 1.13](#)

The second result

LCOV - phosphor-led-sysfs

Person 1

file:///home/andrew/src/openbmc/phosphor-led-sysfs/phosphor-led-sysfs-1.0-coverage/home/andrew/...

LCOV - code coverage report



Current view: [top level](#) - /home/andrew/src/openbmc/phosphor-led-sysfs

Test: phosphor-led-sysfs-1.0 Code Coverage

Date: 2018-05-26 22:33:23

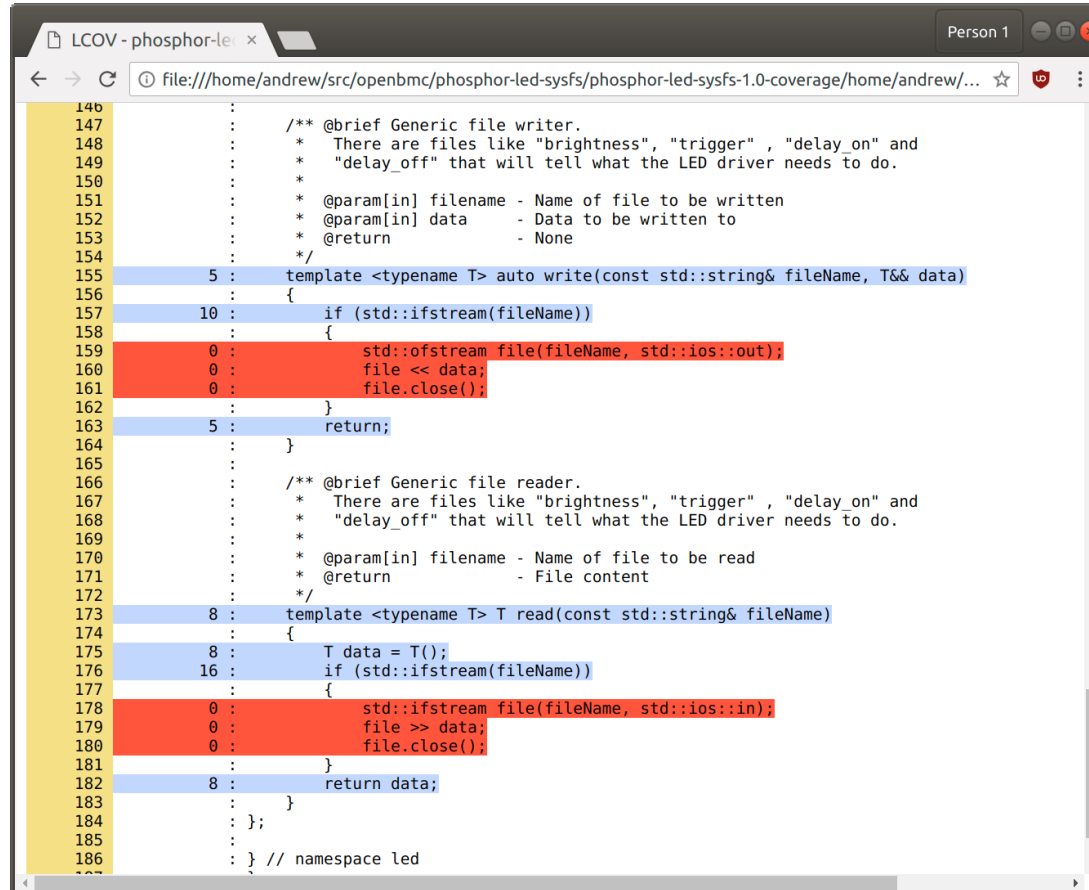
	Hit	Total	Coverage
Lines:	50	63	79.4 %
Functions:	13	14	92.9 %

Legend: Rating: low: < 75 % medium: >= 75 % high: >= 90 %

Filename	Line Coverage (show details)	Functions
physical.cpp	 83.7 % 36 / 43	100.0 % 6 / 6
physical.hpp	 70.0 % 14 / 20	87.5 % 7 / 8

Generated by: [LCOV version 1.13](#)

What we missed: physical.hpp



```
146 :  
147 : /** @brief Generic file writer.  
148 : * There are files like "brightness", "trigger", "delay_on" and  
149 : * "delay_off" that will tell what the LED driver needs to do.  
150 : *  
151 : * @param[in] filename - Name of file to be written  
152 : * @param[in] data - Data to be written to  
153 : * @return - None  
154 : */  
155 : 5 : template <typename T> auto write(const std::string& fileName, T&& data)  
156 : : {  
157 : 10 : if (std::ifstream(fileName))  
158 : : {  
159 : 0 : std::ofstream file(fileName, std::ios::out);  
160 : 0 : file << data;  
161 : 0 : file.close();  
162 : : }  
163 : 5 : return;  
164 : : }  
165 : :  
166 : : /** @brief Generic file reader.  
167 : * There are files like "brightness", "trigger", "delay_on" and  
168 : * "delay_off" that will tell what the LED driver needs to do.  
169 : *  
170 : * @param[in] filename - Name of file to be read  
171 : * @return - File content  
172 : */  
173 : 8 : template <typename T> T read(const std::string& fileName)  
174 : : {  
175 : : T data = T();  
176 : 16 : if (std::ifstream(fileName))  
177 : : {  
178 : 0 : std::ifstream file(fileName, std::ios::in);  
179 : 0 : file >> data;  
180 : 0 : file.close();  
181 : : }  
182 : 8 : return data;  
183 : : }  
184 : : };  
185 : :  
186 : : } // namespace led
```

What we missed: physical.cpp

```
LCOV - phosphor-le x Person 1
file:///home/andrew/src/openbmc/phosphor-led-sysfs/phosphor-led-sysfs-1.0-coverage/home/andrew/...
26 4 : void Physical::setInitialState()
27 : {
28 :     // Control files in /sys/class/leds/<led-name>
29 8 :     brightCtrl = path + BRIGHTNESS;
30 8 :     blinkCtrl = path + BLINKCTRL;
31 :
32 8 :     delayOnCtrl = path + DELAYON;
33 8 :     delayOffCtrl = path + DELAYOFF;
34 :
35 :     // 1. read /sys/class/leds/name/trigger
36 :     // 2. If its 'timer', then its blinking.
37 :     // 2.1: On blink, use delay on and delay off into dutyOn
38 :     // 3. If its 'none', then read brightness. 255 means, its ON, else OFF.
39 :
40 8 :     auto trigger = read<std::string>(blinkCtrl);
41 4 :     if (trigger == "timer")
42 :     {
43 :         // LED is blinking. Get the delay on and delay off and compute
44 :         // DutyCycle. sfsfs values are in strings. Need to convert 'em over to
45 :         // integer.
46 0 :     auto delayOn = std::stoi(read<std::string>(delayOnCtrl));
47 0 :     auto delayOff = std::stoi(read<std::string>(delayOffCtrl));
48 :
49 :         // Calculate frequency and then percentage ON
50 0 :     frequency = delayOn + delayOff;
51 0 :     auto factor = frequency / 100;
52 0 :     auto dutyOn = delayOn / factor;
53 :
54 :     // Update.
55 0 :     this->dutyOn(dutyOn);
56 :     }
57 :     else
58 :     {
59 :         // This is hardcoded for now. This will be changed to this->frequency()
60 :         // when frequency is implemented.
61 :         // TODO
62 4 :     frequency = 1000;
63 :
64 :     // LED is either ON or OFF
65 8 :     auto brightness = read<std::string>(brightCtrl);
66 16 :     if (brightness == std::string(ASSERT))
67 :     {
68 :         // LED is in Solid ON
69 0 :     sdbusplus::xyz::openbmc_project::Led::server::Physical::state(
70 :         Action::On);
71 :     }
72 :     else
73 :     {
74 :         // LED is in OFF state
75 4 :     sdbusplus::xyz::openbmc_project::Led::server::Physical::state(
76 :         Action::Off);
77 :     }
78 : }
79 4 : return;
```

The unhelpful design of Physical

- 1) read() and write() are non-virtual functions
- 2) read() is called by the constructor
- 3) read() and write() are template functions
- 4) There is a better choice of abstraction

sysfs.[ch]pp

<https://gerrit.openbmc-project.xyz/#/c/10830/>

```
diff --git a/sysfs.hpp b/sysfs.hpp
new file mode 100644
index 000000000000..e88cfd09465b
--- /dev/null
+++ b/sysfs.hpp
@@ -0,0 +1,39 @@
+namespace phosphor {
+namespace led {
+class SysfsLed
+{
+ public:
+ SysfsLed(std::experimental::filesystem::path&& root) : root(root) {}
+ virtual ~SysfsLed() {}
+ virtual unsigned long getBrightness();
+ virtual void setBrightness(unsigned long value);
+ virtual unsigned long getMaxBrightness();
+ virtual std::string getTrigger();
+ virtual void setTrigger(std::string trigger);
+ virtual unsigned long getDelayOn();
+ virtual void setDelayOn(unsigned long ms);
+ virtual unsigned long getDelayOff();
+ virtual void setDelayOff(unsigned long ms);
+ protected:
+ std::experimental::filesystem::path root;
+};
+}
+}
```

```
diff --git a/sysfs.cpp b/sysfs.cpp
new file mode 100644
index 000000000000..89b85e125c08
--- /dev/null
+++ b/sysfs.cpp
@@ -0,0 +1,86 @@
+namespace fs = std::experimental::filesystem;
+namespace phosphor {
+namespace led {
+template <typename T> T get_sysfs_attr(fs::path&& path);
+template <typename T> void set_sysfs_attr(fs::path&& path, T& value)
+
+
+unsigned long SysfsLed::getBrightness()
+{
+ return get_sysfs_attr<unsigned long>(root / BRIGHTNESS);
+}
+
+void SysfsLed::setBrightness(unsigned long brightness)
+{
+ set_sysfs_attr<unsigned long>(root / BRIGHTNESS, brightness);
+}
+
+unsigned long SysfsLed::getMaxBrightness()
+{
+ return get_sysfs_attr<unsigned long>(root / MAX_BRIGHTNESS);
+}
+
+...
```

test/sysfs.cpp

<https://gerrit.openbmc-project.xyz/#/c/10830/>

```
+class FakeSysfsLed : public phosphor::led::SysfsLed
+{
+ public:
+  static FakeSysfsLed create()
+  {
+    const char* const tmpl = "/tmp/FakeSysfsLed.XXXXXX";
+    char buffer[MAXPATHLEN] = {0};
+
+    strncpy(buffer, tmpl, strlen(tmpl));
+    char* dir = mkdtemp(buffer);
+    if (!dir)
+      throw std::system_error(errno, std::system_category());
+
+    return FakeSysfsLed(fs::path(dir));
+  }
+
+  ~FakeSysfsLed()
+  {
+    fs::remove_all(root);
+  }
+
+ private:
+  FakeSysfsLed(fs::path&& path) : SysfsLed(std::move(path))
+  {
+    std::string attrs[4] = {BRIGHTNESS, TRIGGER, DELAY_ON, DELAY_OFF};
+    for (auto attr : attrs)
+    {
+      fs::path p = root / attr;
+      std::ofstream f(p, std::ios::out);
+      f.exceptions(f.failbit);
+    }
+
+    fs::path p = root / MAX_BRIGHTNESS;
+    std::ofstream f(p, std::ios::out);
+    f.exceptions(f.failbit);
+    f << MAX_BRIGHTNESS_VAL;
+  }
+};

+TEST(Sysfs, getBrightness)
+{
+  constexpr auto brightness = 127;
+  FakeSysfsLed fsl = FakeSysfsLed::create();
+
+  fsl.setBrightness(brightness);
+  ASSERT_EQ(brightness, fsl.getBrightness());
+}

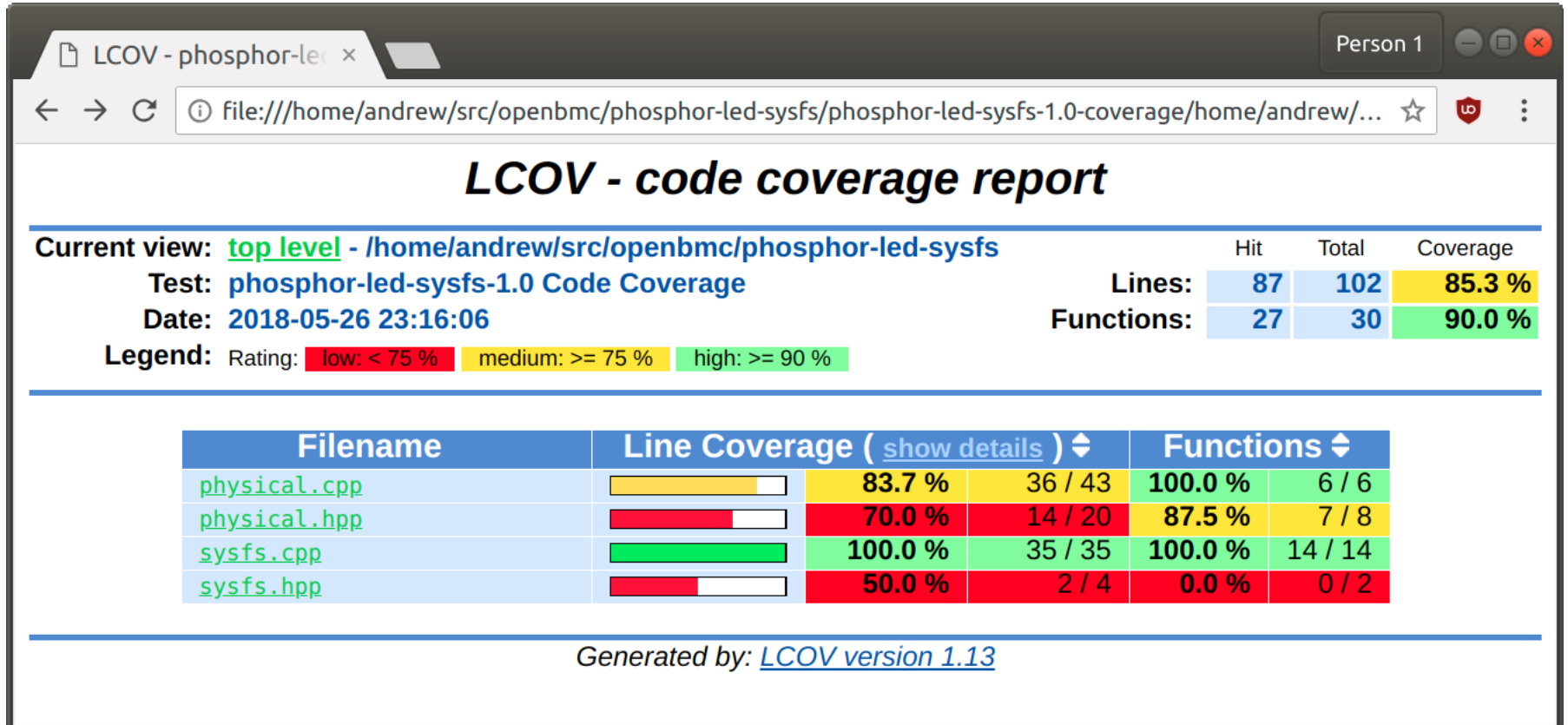
+TEST(Sysfs, getMaxBrightness)
+{
+  FakeSysfsLed fsl = FakeSysfsLed::create();
+
+  ASSERT_EQ(MAX_BRIGHTNESS_VAL, fsl.getMaxBrightness());
+}

+TEST(Sysfs, getTrigger)
+{
+  constexpr auto trigger = "none";
+  FakeSysfsLed fsl = FakeSysfsLed::create();
+
+  fsl.setTrigger(trigger);
+  ASSERT_EQ(trigger, fsl.getTrigger());
+}

+TEST(Sysfs, getDelayOn)
+{
+  constexpr auto delayOn = 250;
+  FakeSysfsLed fsl = FakeSysfsLed::create();
+
+  fsl.setDelayOn(delayOn);
+  ASSERT_EQ(delayOn, fsl.getDelayOn());
+}

+TEST(Sysfs, getDelayOff)
+{
+  constexpr auto delayOff = 750;
+  FakeSysfsLed fsl = FakeSysfsLed::create();
+
+  fsl.setDelayOff(delayOff);
+  ASSERT_EQ(delayOff, fsl.getDelayOff());
+}
```

The third result



Integrate SysfsLed into Physical

<https://gerrit.openbmc-project.xyz/#/c/10831/>

```
@@ -70,12 +42,12 @@ class Physical : public sdbusplus::server::object::object<
 * @param[in] ledPath - sysfs path where this LED is exported
 */
Physical(sdbusplus::bus::bus& bus, const std::string& objPath,
-     const std::string& ledPath) :
+     SysfsLed& led) :

    sdbusplus::server::object::object<
        sdbusplus::xyz::openbmc_project::Led::server::Physical>(
            bus, objPath.c_str(), true),
-     path(ledPath)
+     led(led)
    {
        // Suppose this is getting launched as part of BMC reboot, then we
        // need to save what the micro-controller currently has.
@@ -96,25 +68,13 @@ class Physical : public sdbusplus::server::object::object<
 /** @brief File system location where this LED is exposed
 * Typically /sys/class/leds/<Led-Name>
 */
-     std::string path;
+     SysfsLed& led;

 /** @brief Frequency range that the LED can operate on.
 * Will be removed when frequency is put into interface
 */
 uint32_t frequency;
```

```
-
- /** @brief Generic file writer.
- * There are files like "brightness", "trigger", "delay_on" and
- * "delay_off" that will tell what the LED driver needs to do.
- *
- * @param[in] filename - Name of file to be written
- * @param[in] data - Data to be written to
- * @return - None
- */
- template <typename T> auto write(const std::string& fileName, T&& data)
- {
-     if (std::ifstream(fileName))
-     {
-         std::ofstream file(fileName, std::ios::out);
-         file << data;
-         file.close();
-     }
-     return;
- }
-
- /** @brief Generic file reader.
- * There are files like "brightness", "trigger", "delay_on" and
- * "delay_off" that will tell what the LED driver needs to do.
- *
- * @param[in] filename - Name of file to be read
- * @return - File content
- */
- template <typename T> T read(const std::string& fileName)
- {
-     T data = T();
-     if (std::ifstream(fileName))
-     {
-         std::ifstream file(fileName, std::ios::in);
-         file >> data;
-         file.close();
-     }
-     return data;
- }
```

The fourth result

LCOV - phosphor-led-sysfs

Person 1

file:///home/andrew/src/openbmc/phosphor-led-sysfs/phosphor-led-sysfs-1.0-coverage/home/a...

LCOV - code coverage report


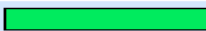
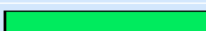

Current view: [top level](#) - /home/andrew/src/openbmc/phosphor-led-sysfs

Test: **phosphor-led-sysfs-1.0 Code Coverage**

Date: **2018-05-28 09:35:40**

Legend: Rating: low: < 75 % medium: >= 75 % high: >= 90 %

	Hit	Total	Coverage
Lines:	75	84	89.3 %
Functions:	21	25	84.0 %

Filename	Line Coverage (show details)	Functions
physical.cpp	 81.6 % 31 / 38	100.0 % 6 / 6
physical.hpp	 100.0 % 7 / 7	33.3 % 1 / 3
sysfs.cpp	 100.0 % 35 / 35	100.0 % 14 / 14
sysfs.hpp	 50.0 % 2 / 4	0.0 % 0 / 2

Generated by: [LCOV version 1.13](#)

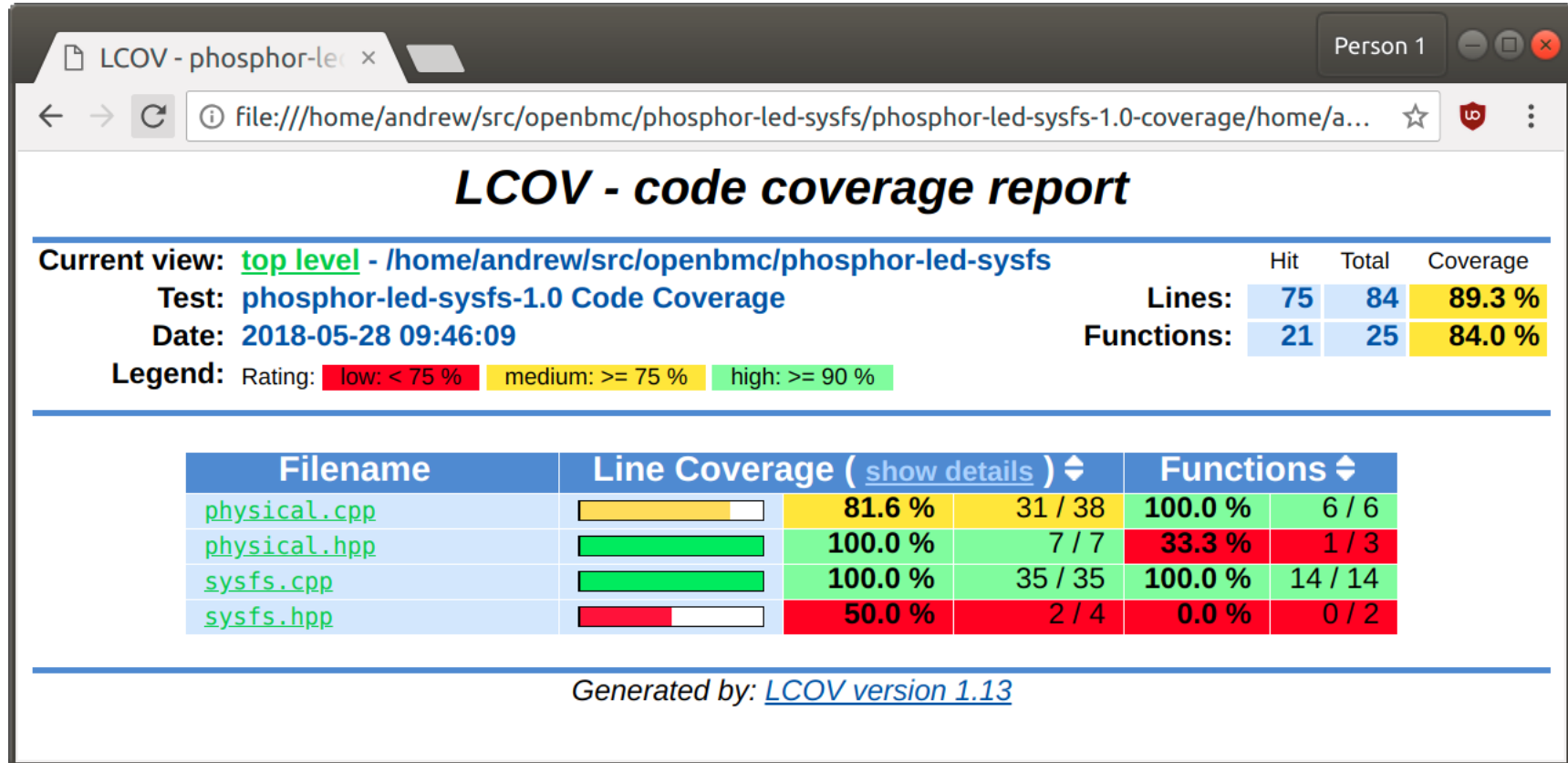
Mocking SysfsLed

<https://gerrit.openbmc-project.xyz/#/c/10832/>

```
diff --git a/test/physical.cpp b/test/physical.cpp
index fd1157e9a115..eab8ac9b03dc 100644
--- a/test/physical.cpp
+++ b/test/physical.cpp
@@ -1,25 +1,87 @@
#include <gtest/gtest.h>
+#include <gmock/gmock.h>
...
+class MockLed : public phosphor::led::SysfsLed
+{
+ public:
+ /* Use a no-args ctor here to avoid headaches with {Nice,Strict}Mock */
+ MockLed() : SysfsLed(...) {}
+ virtual ~MockLed() { ... }
+
+ MOCK_METHOD0(getBrightness, unsigned long());
+ MOCK_METHOD1(setBrightness, void(unsigned long value));
+ MOCK_METHOD0(getMaxBrightness, unsigned long());
+ MOCK_METHOD0(getTrigger, std::string());
+ MOCK_METHOD1(setTrigger, void(std::string trigger));
+ MOCK_METHOD0(getDelayOn, unsigned long());
+ MOCK_METHOD1(setDelayOn, void(unsigned long ms));
+ MOCK_METHOD0(getDelayOff, unsigned long());
+ MOCK_METHOD1(setDelayOff, void(unsigned long ms));
+};
```

```
+using ::testing::NiceMock;
+using ::testing::Return;
+
+TEST(Physical, ctor)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    - phosphor::led::SysfsLed led = phosphor::led::SysfsLed(fs::path(LED_SYSFS));
+    + /* NiceMock ignores calls to methods with no expectations defined */
+    + NiceMock<MockLed> led;
+    + ON_CALL(led, getTrigger()).WillByDefault(Return("none"));
+    phosphor::led::Physical phy(bus, LED_OBJ, led);
+}
+
+TEST(Physical, off)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    - phosphor::led::SysfsLed led = phosphor::led::SysfsLed(fs::path(LED_SYSFS));
+    + NiceMock<MockLed> led;
+    + ON_CALL(led, getTrigger()).WillByDefault(Return("none"));
+    phosphor::led::Physical phy(bus, LED_OBJ, led);
+    phy.state(Action::Off);
+}
+@@ -27,7 +89,8 @@ TEST(Physical, off)
+TEST(Physical, on)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    - phosphor::led::SysfsLed led = phosphor::led::SysfsLed(fs::path(LED_SYSFS));
+    + NiceMock<MockLed> led;
+    + ON_CALL(led, getTrigger()).WillByDefault(Return("none"));
+    phosphor::led::Physical phy(bus, LED_OBJ, led);
+    phy.state(Action::On);
+}
+@@ -35,7 +98,9 @@ TEST(Physical, on)
+TEST(Physical, blink)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    - phosphor::led::SysfsLed led = phosphor::led::SysfsLed(fs::path(LED_SYSFS));
+    + NiceMock<MockLed> led;
+    + ON_CALL(led, getTrigger()).WillByDefault(Return("none"));
+    + ON_CALL(led, getDelayOn()).WillByDefault(Return(500));
+    phosphor::led::Physical phy(bus, LED_OBJ, led);
+    phy.state(Action::Blink);
+}
+}
```

The fifth result



The un-hit paths in Physical.cpp

```
LCOV - phosphor-le x Person 1
file:///home/andrew/src/openbmc/phosphor-led-sysfs/phosphor-led-sysfs-1.0-coverage/home/a...
22 : namespace led
23 : {
24 :
25 : /** @brief Populates key parameters */
26 4 : void Physical::setInitialState()
27 : {
28 :     // 1. read /sys/class/leds/name/trigger
29 :     // 2. If its 'timer', then its blinking.
30 :     // 2.1: On blink, use delay_on and delay off into dutyOn
31 :     // 3. If its 'none', then read brightness. 255 means, its ON, else OFF.
32 :
33 8 :     auto trigger = led.getTrigger();
34 4 :     if (trigger == "timer")
35 :     {
36 :         // LED is blinking. Get the delay_on and delay_off and compute
37 :         // DutyCycle. sfsfs values are in strings. Need to convert 'em over to
38 :         // integer.
39 0 :         auto delayOn = led.getDelayOn();
40 0 :         auto delayOff = led.getDelayOff();
41 :
42 :         // Calculate frequency and then percentage ON
43 0 :         frequency = delayOn * delayOff;
44 0 :         auto factor = frequency / 100;
45 0 :         auto dutyOn = delayOn / factor;
46 :
47 :         // Update.
48 0 :         this->dutyOn(dutyOn);
49 :     }
50 :     else
51 :     {
52 :         // This is hardcoded for now. This will be changed to this->frequency()
53 :         // when frequency is implemented.
54 :         // TODO
55 4 :         frequency = 1000;
56 :
57 :         // LED is either ON or OFF
58 4 :         auto brightness = led.getBrightness();
59 4 :         if (brightness == ASSERT)
60 :         {
61 :             // LED is in Solid ON
62 0 :             sdbusplus::xyz::openbmc_project::Led::server::Physical::state(
63 :                 Action::On);
64 :         }
65 :         else
66 :         {
67 :             // LED is in OFF state
68 4 :             sdbusplus::xyz::openbmc_project::Led::server::Physical::state(
69 :                 Action::Off);
70 :         }
71 :     }
72 4 :     return;
73 : }
74 :
75 : /** @brief Overloaded State Property Setter function */
76 3 : auto Physical::state(Action value) -> Action
```

Hitting ‘trigger == “timer”’

<https://gerrit.openbmc-project.xyz/#/c/10833/>

```
diff --git a/test/physical.cpp b/test/physical.cpp
index eab8ac9b03dc..a853dcd748aa 100644
--- a/test/physical.cpp
+++ b/test/physical.cpp
@@ -68,7 +68,7 @@ using ::testing::NiceMock;
using ::testing::Return;
using ::testing::Throw;
```

```
-TEST(Physical, ctor)
```

```
+TEST(Physical, ctor_none_trigger)
```

```
{
    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
    /* NiceMock ignores calls to methods with no expectations
    defined */
@@ -77,6 +77,16 @@ TEST(Physical, ctor)
    phosphor::led::Physical phy(bus, LED_OBJ, led);
}
```

```
+TEST(Physical, ctor_timer_trigger)
```

```
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    NiceMock<MockLed> led;
+    EXPECT_CALL(led, getTrigger()).WillOnce(Return("timer"));
+    EXPECT_CALL(led, getDelayOn()).WillOnce(Return(500));
+    EXPECT_CALL(led, getDelayOff()).WillOnce(Return(500));
+    phosphor::led::Physical phy(bus, LED_OBJ, led);
+}
+
```

The sixth result

The screenshot shows a web browser displaying an LCOV code coverage report. The title is "LCOV - code coverage report". Below the title, it shows the current view as "top level - /home/andrew/src/openbmc/phosphor-led-sysfs", the test as "phosphor-led-sysfs-1.0 Code Coverage", and the date as "2018-05-28 10:02:16". A legend indicates that a rating of low (< 75%) is red, medium (>= 75%) is yellow, and high (>= 90%) is green. A summary table shows 81 lines hit out of 84 total (96.4% coverage) and 21 functions hit out of 25 total (84.0% coverage). A detailed table lists coverage for four files: physical.cpp (97.4% line, 100.0% function), physical.hpp (100.0% line, 33.3% function), sysfs.cpp (100.0% line, 100.0% function), and sysfs.hpp (50.0% line, 0.0% function). The report was generated by LCOV version 1.13.

LCOV - code coverage report

Current view: [top level](#) - /home/andrew/src/openbmc/phosphor-led-sysfs

Test: **phosphor-led-sysfs-1.0 Code Coverage**

Date: **2018-05-28 10:02:16**

Legend: Rating: low: < 75 % medium: >= 75 % high: >= 90 %

Filename	Line Coverage (show details)	Functions
physical.cpp	<div style="width:97.4%;"><div style="width:97.4%;"></div></div> 97.4 % 37 / 38 100.0 % 6 / 6	<div style="width:100%;"><div style="width:100%;"></div></div> 100.0 % 7 / 7 33.3 % 1 / 3
physical.hpp	<div style="width:100%;"><div style="width:100%;"></div></div> 100.0 % 35 / 35 100.0 % 14 / 14	<div style="width:50%;"><div style="width:50%;"></div></div> 50.0 % 2 / 4 0.0 % 0 / 2
sysfs.cpp	<div style="width:100%;"><div style="width:100%;"></div></div> 100.0 %	
sysfs.hpp	<div style="width:50%;"><div style="width:50%;"></div></div> 50.0 %	

Generated by: [LCOV version 1.13](#)

The screenshot shows a C++ source code file with LCOV coverage annotations. The code is for a class named Physical, which is part of the phosphor-led-sysfs project. The code includes comments and function definitions. The LCOV annotations are shown as colored bars next to the code lines, indicating the coverage status of each line. The code is as follows:

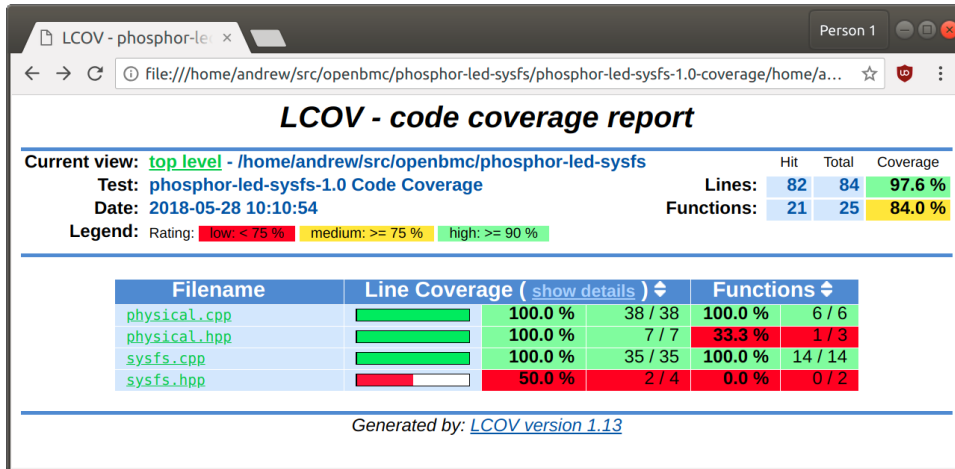
```
24 :  
25 :  
26 : /** @brief Populates key parameters */  
27 : void Physical::setInitialState()  
28 : {  
29 :     // 1. read /sys/class/leds/name/trigger  
30 :     // 2. If its 'timer', then its blinking.  
31 :     // 2.1: On blink, use delay_on and delay_off into dutyOn  
32 :     // 3. If its 'none', then read brightness. 255 means, its ON, else OFF.  
33 :  
34 :     10 : auto trigger = led.getTrigger();  
35 :     5 : if (trigger == "timer")  
36 :     {  
37 :         // LED is blinking. Get the delay_on and delay_off and compute  
38 :         // DutyCycle. sfsfs values are in strings. Need to convert 'em over to  
39 :         // integer.  
40 :         1 : auto delayOn = led.getDelayOn();  
41 :         1 : auto delayOff = led.getDelayOff();  
42 :  
43 :         // Calculate frequency and then percentage ON  
44 :         1 : frequency = delayOn + delayOff;  
45 :         1 : auto factor = frequency / 100;  
46 :         1 : auto dutyOn = delayOn / factor;  
47 :  
48 :         // Update.  
49 :         1 : this->dutyOn(dutyOn);  
50 :     }  
51 :     else  
52 :     {  
53 :         // This is hardcoded for now. This will be changed to this->frequency()  
54 :         // when frequency is implemented.  
55 :         // TODO  
56 :         4 : frequency = 1000;  
57 :  
58 :         // LED is either ON or OFF  
59 :         4 : auto brightness = led.getBrightness();  
60 :         4 : if (brightness == ASSERT)  
61 :         {  
62 :             // LED is in Solid ON  
63 :             0 : sdbusplus::xyz::openbmc_project::Led::server::Physical::state(  
64 :                 Action::On);  
65 :         }  
66 :         else  
67 :         {  
68 :             // LED is in OFF state  
69 :             4 : sdbusplus::xyz::openbmc_project::Led::server::Physical::state(  
70 :                 Action::Off);  
71 :         }  
72 :     }  
73 :     5 : return;  
74 : }  
75 :  
76 : /** @brief Overloaded State Property Setter function */  
77 : 3 : auto Physical::state(Action value) -> Action  
78 : {  
79 :     // Obtain current operation
```

Hitting 'brightness == ASSERT'

<https://gerrit.openbmc-project.xyz/#/c/10834/>

```
diff --git a/test/physical.cpp b/test/physical.cpp
index a853dcd748aa..9bb8424bee8f 100644
--- a/test/physical.cpp
+++ b/test/physical.cpp
@@ -114,3 +114,13 @@ TEST(Physical, blink)
     phosphor::led::Physical phy(bus, LED_OBJ, led);
     phy.state(Action::Blink);
 }
+
+TEST(Physical, ctor_none_trigger_asserted_brightness)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    NiceMock<MockLed> led;
+    EXPECT_CALL(led, getTrigger()).WillRepeatedly(Return("none"));
+    constexpr auto val = phosphor::led::ASSERT;
+    EXPECT_CALL(led, getBrightness()).WillRepeatedly(Return(val));
+    phosphor::led::Physical phy(bus, LED_OBJ, led);
+}
```

The seventh result



The screenshot shows the LCOV code coverage report for the phosphor-led-sysfs project. The report title is "LCOV - code coverage report". The current view is "top level - /home/andrew/src/openbmc/phosphor-led-sysfs". The test is "phosphor-led-sysfs-1.0 Code Coverage" and the date is "2018-05-28 10:10:54". The legend indicates that low coverage is < 75%, medium is >= 75%, and high is >= 90%. The summary table shows 82 lines hit out of 84 total (97.6% coverage) and 21 functions hit out of 25 total (84.0% coverage). A detailed table lists coverage for physical.cpp, physical.hpp, sysfs.cpp, and sysfs.hpp.

LCOV - code coverage report

Current view: [top level](#) - /home/andrew/src/openbmc/phosphor-led-sysfs

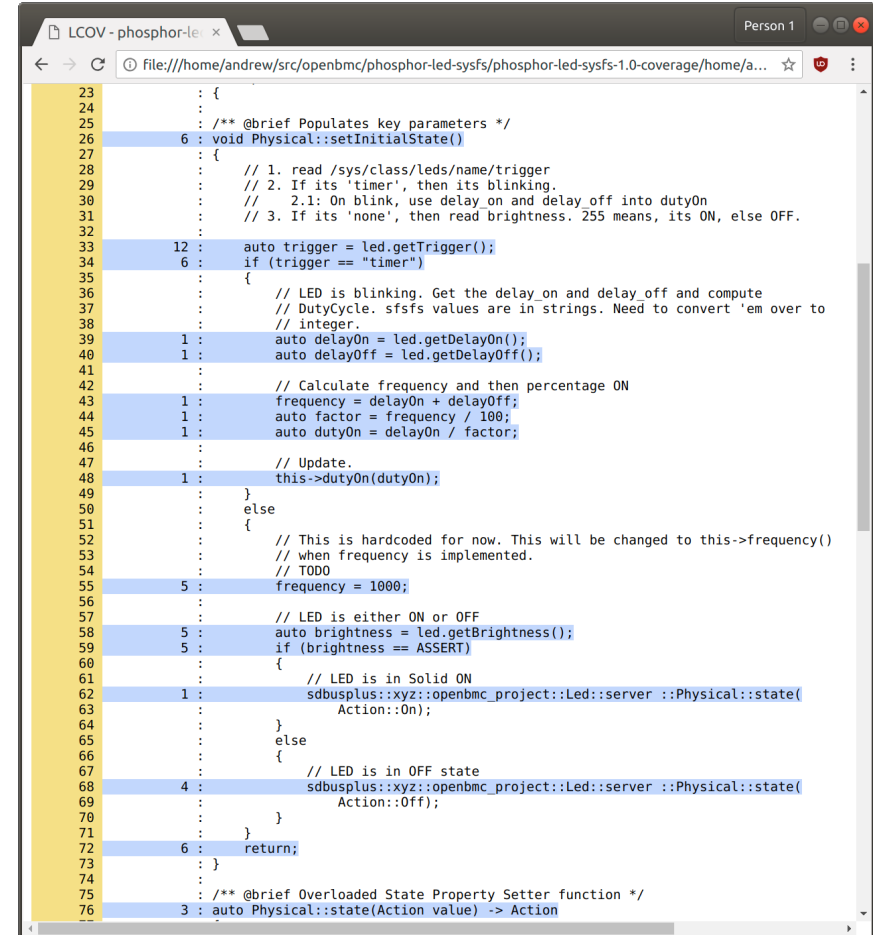
Test: phosphor-led-sysfs-1.0 Code Coverage

Date: 2018-05-28 10:10:54

Legend: Rating: ■ low: < 75 % ■ medium: >= 75 % ■ high: >= 90 %

Filename	Line Coverage (show details)	Functions
physical.cpp	100.0 % 38 / 38	100.0 % 6 / 6
physical.hpp	100.0 % 7 / 7	33.3 % 1 / 3
sysfs.cpp	100.0 % 35 / 35	100.0 % 14 / 14
sysfs.hpp	50.0 % 2 / 4	0.0 % 0 / 2

Generated by: [LCOV version 1.13](#)



The screenshot shows a C++ source code file for the Physical class. The code includes comments and function definitions for setting initial state and updating duty cycle. The code is as follows:

```
23 : {
24 :
25 :     /** @brief Populates key parameters */
26 :     6 : void Physical::setInitialState()
27 :     : {
28 :         // 1. read /sys/class/leds/name/trigger
29 :         // 2. If its 'timer', then its blinking.
30 :         // 2.1: On blink, use delay_on and delay_off into dutyOn
31 :         // 3. If its 'none', then read brightness. 255 means, its ON, else OFF.
32 :
33 :     12 : auto trigger = led.getTrigger();
34 :     6 : if (trigger == "timer")
35 :     : {
36 :         // LED is blinking. Get the delay_on and delay_off and compute
37 :         // DutyCycle. sfsfs values are in strings. Need to convert 'em over to
38 :         // integer.
39 :     1 : auto delayOn = led.getDelayOn();
40 :     1 : auto delayOff = led.getDelayOff();
41 :
42 :         // Calculate frequency and then percentage ON
43 :     1 : frequency = delayOn + delayOff;
44 :     1 : auto factor = frequency / 100;
45 :     1 : auto dutyOn = delayOn / factor;
46 :
47 :         // Update.
48 :     1 : this->dutyOn(dutyOn);
49 :     }
50 :     else
51 :     : {
52 :         // This is hardcoded for now. This will be changed to this->frequency()
53 :         // when frequency is implemented.
54 :         // TODO
55 :     5 : frequency = 1000;
56 :
57 :         // LED is either ON or OFF
58 :     5 : auto brightness = led.getBrightness();
59 :     5 : if (brightness == ASSERT)
60 :     : {
61 :         // LED is in Solid ON
62 :     1 : sdbusplus::xyz::openbmc_project::Led::server::Physical::state(
63 :         : Action::On);
64 :     }
65 :     else
66 :     : {
67 :         // LED is in OFF state
68 :     4 : sdbusplus::xyz::openbmc_project::Led::server::Physical::state(
69 :         : Action::Off);
70 :     }
71 : }
72 : 6 : return;
73 : }
74 :
75 : /** @brief Overloaded State Property Setter function */
76 : 3 : auto Physical::state(Action value) -> Action
```

Enforce behaviour with Sensing

<https://gerrit.openbmc-project.xyz/#/c/10836/>

```
diff --git a/test/physical.cpp b/test/physical.cpp
index fade5c8c3bd9..bbb51438c722 100644
--- a/test/physical.cpp
+++ b/test/physical.cpp
@@ -110,7 +110,10 @@ TEST(Physical, blink)
 {
     sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
     NiceMock<MockLed> led;
-   ON_CALL(led, getTrigger()).WillByDefault(Return("none"));
+   EXPECT_CALL(led, getTrigger()).WillOnce(Return("none"));
+   EXPECT_CALL(led, setTrigger("timer"));
+   EXPECT_CALL(led, setDelayOn(500));
+   EXPECT_CALL(led, setDelayOff(500));
     phosphor::led::Physical phy(bus, LED_OBJ, led);
     phy.state(Action::Blink);
 }
```


Branch Coverage

make **CODE_COVERAGE_BRANCH_COVERAGE=1** check-code-coverage

LCOV - code coverage report

Current view: [top level](#) - /home/andrew/src/openbmc/phosphor-led-sysfs

Test: **phosphor-led-sysfs-1.0 Code Coverage**

Date: **2018-05-28 10:33:39**

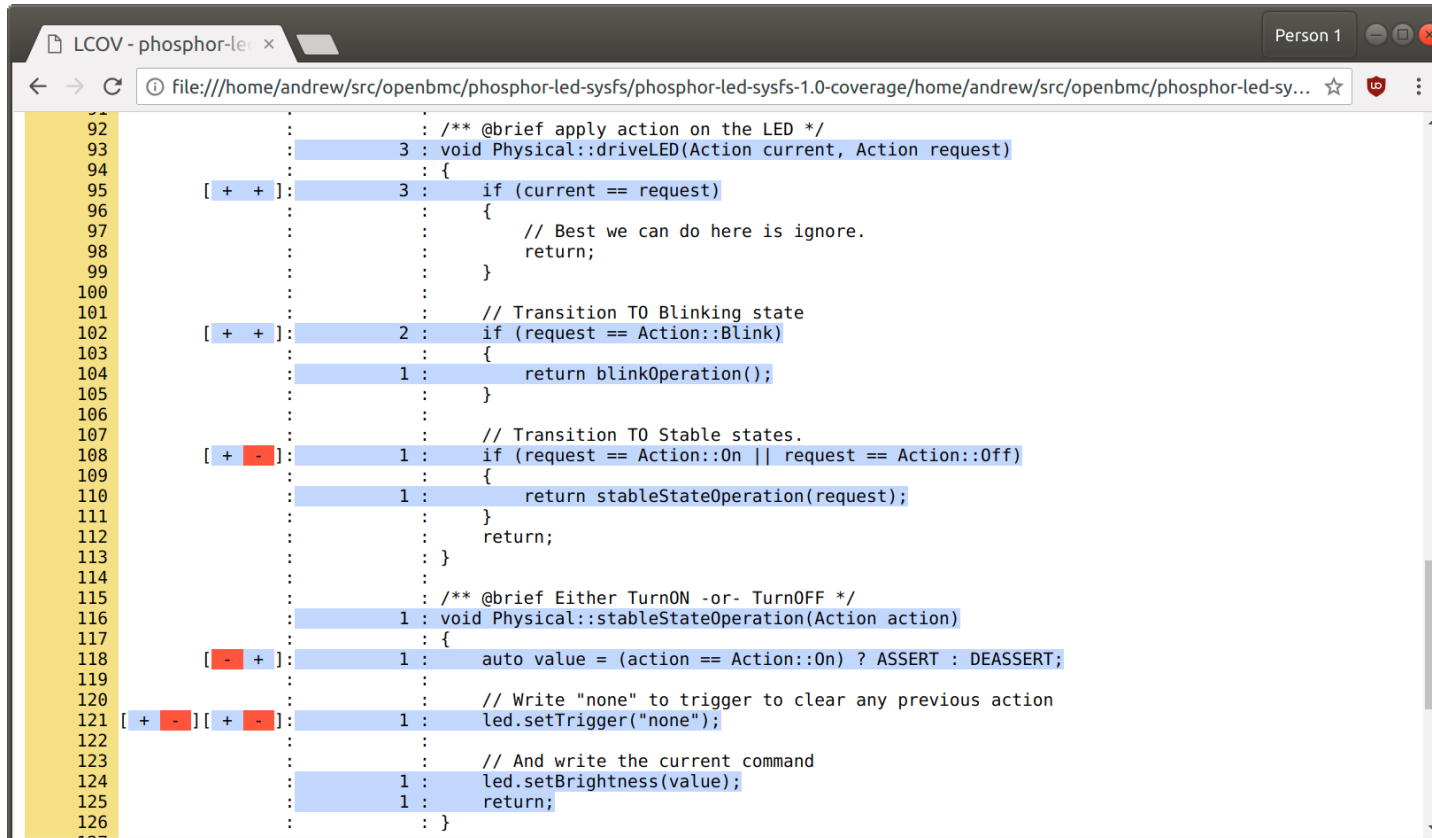
Legend: Rating: low: < 75 % medium: >= 75 % high: >= 90 %

	Hit	Total	Coverage
Lines:	81	83	97.6 %
Functions:	21	25	84.0 %
Branches:	44	78	56.4 %

Filename	Line Coverage (show details)	Functions	Branches
physical.cpp	<div style="width: 100%; height: 10px; background-color: green;"></div> 100.0 % 37 / 37	100.0 % 6 / 6	66.7 % 20 / 30
physical.hpp	<div style="width: 100%; height: 10px; background-color: green;"></div> 100.0 % 7 / 7	33.3 % 1 / 3	50.0 % 1 / 2
sysfs.cpp	<div style="width: 100%; height: 10px; background-color: green;"></div> 100.0 % 35 / 35	100.0 % 14 / 14	50.0 % 22 / 44
sysfs.hpp	<div style="width: 50%; height: 10px; background-color: red;"></div> 50.0 % 2 / 4	0.0 % 0 / 2	50.0 % 1 / 2

Generated by: [LCOV version 1.13](#)

Branch Coverage



```
92 :           /** @brief apply action on the LED */
93 :           3 : void Physical::driveLED(Action current, Action request)
94 :           : {
95 : [ + + ] :           3 :     if (current == request)
96 :           :           {
97 :           :             // Best we can do here is ignore.
98 :           :             return;
99 :           :           }
100 :           :
101 :           :           // Transition TO Blinking state
102 : [ + + ] :           2 :     if (request == Action::Blink)
103 :           :           {
104 :           :           1 :       return blinkOperation();
105 :           :           }
106 :           :
107 :           :           // Transition TO Stable states.
108 : [ + - ] :           1 :     if (request == Action::On || request == Action::Off)
109 :           :           {
110 :           :           1 :       return stableStateOperation(request);
111 :           :           }
112 :           :           return;
113 :           :           }
114 :           :
115 :           :           /** @brief Either TurnON -or- TurnOFF */
116 :           1 : void Physical::stableStateOperation(Action action)
117 :           : {
118 : [ - + ] :           1 :     auto value = (action == Action::On) ? ASSERT : DEASSERT;
119 :           :
120 :           :           // Write "none" to trigger to clear any previous action
121 : [ + - ] [ + - ] :           1 :     led.setTrigger("none");
122 :           :
123 :           :           // And write the current command
124 :           :           1 :     led.setBrightness(value);
125 :           :           1 :     return;
126 :           :           }
```

Covering un-hit branches

<https://gerrit.openbmc-project.xyz/#/c/10836/>

```
diff --git a/test/physical.cpp b/test/physical.cpp
index 156a2e5e87d1..fade5c8c3bd9 100644
--- a/test/physical.cpp
+++ b/test/physical.cpp
@@ -64,6 +64,7 @@ class MockLed : public phosphor::led::SysfsLed
     MOCK_METHOD1(setDelayOff, void(unsigned long ms));
 };

+using ::testing::InSequence;
using ::testing::NiceMock;
using ::testing::Return;
using ::testing::Throw;
@@ -123,3 +124,20 @@ TEST(Physical, ctor_none_trigger_asserted_brightness)
     EXPECT_CALL(led, getBrightness()).WillRepeatedly(Return(val));
     phosphor::led::Physical phy(bus, LED_OBJ, led);
 }
+
+TEST(Physical, on_to_off)
+{
+    InSequence s;
+
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    NiceMock<MockLed> led;
+    EXPECT_CALL(led, getTrigger()).Times(1).WillOnce(Return("none"));
+    unsigned long deasserted = phosphor::led::DEASSERT;
+    EXPECT_CALL(led, getBrightness()).WillOnce(Return(deasserted));
+    unsigned long asserted = phosphor::led::ASSERT;
+    EXPECT_CALL(led, setBrightness(asserted));
+    EXPECT_CALL(led, setBrightness(deasserted));
+    phosphor::led::Physical phy(bus, LED_OBJ, led);
+    phy.state(Action::On);
+    phy.state(Action::Off);
+}
```

The eighth result



The screenshot shows a web browser window titled "LCOV - phosphor-led-sy...". The address bar contains the file path: `file:///home/andrew/src/openbmc/phosphor-led-sysfs/phosphor-led-sysfs-1.0-coverage/home/andrew/src/openbmc/phosphor-led-sy...`. The browser window displays C code with LCOV coverage statistics. The code is as follows:

```
92 :           : /** @brief apply action on the LED */
93 :           : 5 : void Physical::driveLED(Action current, Action request)
94 :           : {
95 : [+ +] : 5 :   if (current == request)
96 :           :   {
97 :           :     // Best we can do here is ignore.
98 :           :     return;
99 :           :   }
100 :          :
101 :          :   // Transition TO Blinking state
102 : [+ +] : 4 :   if (request == Action::Blink)
103 :          :   {
104 :          :     1 :   return blinkOperation();
105 :          :   }
106 :          :
107 :          :   // Transition TO Stable states.
108 : [+ -] : 3 :   if (request == Action::On || request == Action::Off)
109 :          :   {
110 :          :     3 :   return stableStateOperation(request);
111 :          :   }
112 :          :   return;
113 :          : }
114 :          :
115 :          : /** @brief Either TurnON -or- TurnOFF */
116 :          : 3 : void Physical::stableStateOperation(Action action)
117 :          : {
118 : [+ +] : 3 :   auto value = (action == Action::On) ? ASSERT : DEASSERT;
119 :          :
120 :          :   // Write "none" to trigger to clear any previous action
121 : [+ -][+ -] : 3 :   led.setTrigger("none");
122 :          :
123 :          :   // And write the current command
124 :          :   3 :   led.setBrightness(value);
125 :          :   3 :   return;
126 :          : }
```

Now that tests are in place...

- physical: Conform to LED class kernel ABI
- physical: Avoid unreachable statement in `driveLED()`
- physical: Cleanup unnecessary variables
- physical: Rework commentary for brevity
- physical: 'frequency' is really periodicity

Ratcheting up the pressure

<https://gerrit.openbmc-project.xyz/#/c/10828/>

```
diff --git a/bootstrap.sh b/bootstrap.sh
index 50b75b7ee911..11c8ae9f96c0 100755
--- a/bootstrap.sh
+++ b/bootstrap.sh
@@ -1,10 +1,20 @@
#!/bin/sh

+set -eu
+
AUTOCONF_FILES="Makefile.in aclocal.m4 ar-lib autom4te.cache compile \
config.guess config.h.in config.sub configure depcomp install-sh \
ltmain.sh missing *libtool test-driver"

-case $1 in
+BOOTSTRAP_MODE=""
+
+if [ $# -gt 0 ];
+then
+  BOOTSTRAP_MODE="${1}"
+  shift 1
+fi
+
+case ${BOOTSTRAP_MODE} in
  clean)
    test -f Makefile && make maintainer-clean
    for file in ${AUTOCONF_FILES}; do
@@ -15,4 +25,17 @@ case $1 in
esac

autoreconf -i
-echo "Run './configure ${CONFIGURE_FLAGS} && make'"
+
+case ${BOOTSTRAP_MODE} in
+  dev)
+    FLAGS="-fsanitize=address -fsanitize=leak -fsanitize=undefined -Wall -Werror"
+    ./configure CFLAGS="${FLAGS}" CXXFLAGS="${FLAGS}" --enable-code-coverage "$@"
+    ;;
+  *)
+    echo "Run './configure ${CONFIGURE_FLAGS} && make'"
+    ;;
+esac
```

Code Sanitizers

- `-fsanitize=address`
 - Immediate abort and report on bad memory accesses
- `-fsanitize=leak`
 - Memory leak report on application termination
- `-fsanitize=undefined`
 - Immediate abort and report on undefined behaviour

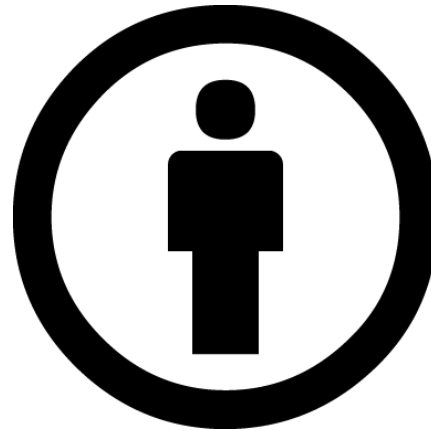
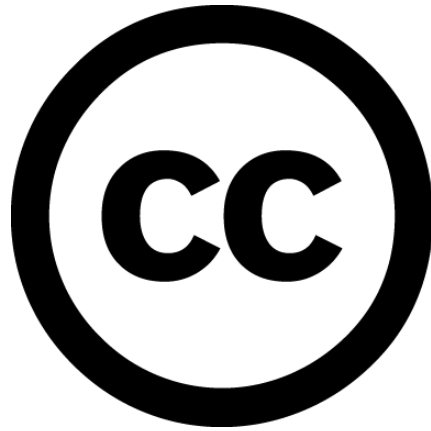
Wrapping up...

- Automated testing is automated code review
 - Accelerates future development through characterisation
 - Good coverage drastically reduces risk of (invasive) changes
- You need to instrument your tests
 - Use code coverage metrics to guide your testing
- Rework abstractions to increase coverage
 - Dependency injection allows for mocks and thus sensing
- Use branch coverage analysis to find missed paths
- Use sanitizers so your tests fail fast, hard and informatively
- Use this presentation to adopt tests in your code!

Recommended Reading

- [Working Effectively with Legacy Code](#)
- [Design Patterns: Elements of Reusable Object-Oriented Software](#)

Please attribute IBM Corp when using or adapting this work



Except where otherwise noted, this work is licensed under

<https://creativecommons.org/licenses/by/4.0/>