Descriptions of existing test cases

test/Board.cpp	Tim
test/Toolchain.cpp	Tim
test/BoardDevice	Pontus
test/BoardView	Lukas
test/Polyfills.cpp	Marek
test/LibManagement	Max

Board

Board - Board contracts

- Test whether the status concept of the board works and whether the validation status of the board view is correctly returned after every state change.
- Following actions:
  - Create Toolchain
  - Create sketch
  - Compile sketch
  - Create Board
  - Several state changes:
    - Board has to be clean
    - BoardView is not valid
    - Then doing all the actions and check every time for correct board status and validity of Board View
    - …
    - Board is running and valid
    - …
    - Board is clean again and invalid

Board - Board exit-notify

- The used sketch throws an exception in the loop method (intentionally implemented). The test checks whether the exit_notify method is called correctly. This is done by waiting for the execution of this function. The body of the function is ex.set_value(ec); When the method call is not done in 5 seconds, the test fails. The future's return code also has to not be equal to zero (exit_notify correctly triggered and returning an exit code).

Board - Mixed INO/C++ sources

- Test whether a sketch can still be compiled when the sketch is a INO file but also has a function call to a C++ class in it (see sketch directory in test folder and compare “noop” with “with_cxx”)

Toolchain

Toolchain - Toolchain invalid

- Toolchain is created referring to an empty directory
- The method check_suitable_environment needs to return an error code due to the wrongly configured toolchain. Therefore, it says REQUIRE(t c.check_suitable:environment()), even though this is a bit misleading because it feels like it is a good thing to require check_suitable_environment, but it is not...

Toolchain - Toolchain valid

- Toolchain is correctly configured with the resource directory referring to SMCE_PATH
- Besides the suitable environment, the resource directory has to be set correctly and the Cmake-Path has to be configured as well.

Board View

BoardView - BoardView GPIO

- Tests that the GPIO pins work as expected.
- A board is created with two pins of type Digital- and Analog Driver.
- Tests are performed to check that the pins exist and that it is possible to read to and from them, and not when it should not be possible.
- This test case is a **good** example of how pins are created and used.

BoardView - BoardView UART

- Checks that the UART communication works as expected.
- A board configuration is created with uart_channels. This is later attached to the board.
- Tests are performed to check that a uart exists, that rx and tx within the uart exist, and that it is possible to have a communication between rx and tx.
- An output array is created and sent from the rx pin (why rx? shouldn't it be tx?). The tx pin will receive the values sent from rx in a while loop of “ticks”.
- Tests are performed to check that data is correct after the communication.

BoardView - RGB444 cvt

- A board is created and configured with a frame buffer.
- Variables of height, width, array in, array expected_out are created and the values of height and width are added to the created frame buffer.
- A test is then performed to check that loading the framebuffer with the values from the “in” array is possible.
- The next test reads the value from the frame buffer and compares it with the expected output. Should be the same.

- The same kind of test is then performed with a new height and width (both equal to 2).
- The same kind of tests are performed two more times, once again with different input values and expected output of the frame buffer.

Board Device

BoardDevice - Basic board device

- Setups a toolchain and checks that it is valid
- Creates a PluginManifest stating that it needs the device "TestUDD"
- Creates a sketch with the previously created PluginManifest, I do not understand this part: `.genbind_devices = { std::cref(TestUDD::specification) }`
- It then compiles the sketch and checks that no error occurred during the compilation
- Creates a board with an empty BoardConfig except for **.board_devices =** `{{std::cref(TestUDD::specification), 2}}`
- Requires that configuring the board with the BoardConfig, attaching the sketch, starting the board, and that the BoardView works without errors
- Checks that two devices are returned by TestUDD::getObjects(boardview), then it tries loading data from both devices. The loading has to be finished (checked by seeing if size is correct, 10 and 65500) before the ticks reach 8192.

LibManagement

Invalid manifests processing

- Creates invalid manifests and checks to see that the system returns code 1 (fails) when trying to run it.
  - first It uses the method GENERATE to pick out the name for the invalid manifest that will be used in the test. The test will be run once for each manifest name in GENERATE. Each manifest here represents a bad input on one of the variables in the PluginManifest.
  - In the next step the corresponding manifest is loaded into a newly created manifests folder.
  - Then it starts a shell on the path of the copied Manifest, runs the cmake file of the manifest and stores the printout from the shell.
  - Finally it checks to see that the shell finished with a code not equal to zero meaning that the shell had an error. Invalid manifests shouldn't run without error.

Valid manifest processing

- Writes a valid plugin Manifest and then checks to see that it runs successfully.
  - It begins by generating a new directory with a randomly generated name.
  - Then it generates an inner directory called manifests.
  - Then it instantiates an analogWrite pluginManifest pulled from github.
  - Then it generates a cmake file inside the manifests directory using the previously instantiated pluginManifest and calls it ArduinoRemote. According to the docs the write_manifest method only generates a lib for the manifest.

        But it also does other things. I believe it is generating the cmake file for the manifest.
    - From line 129 to 144 I have a hard time following what is going on.
    - From line 146 to 156 I'm again confused. They create a CmakeLists file inside the earlier created lib and then add an empty sketch and a bunch of cmake instructions to it. I think it is creating an executable cmake project to test the manifest with.
    - The last step is running the project with a shell where it stores the output. The output is then required to be zero for the test to pass.

Board remote preproc lib

- Creates a toolchain and a sketch and tries to compile the sketch using the toolchain requiring it to succeed. It checks whether the collaboration with a remote "legacy-preproc-librar" is working.
    - First it creates a toolchain and requires it to return false when asked if it is a suitable environment.
    - Then it creates a sketch targeting a nano arduino board(don't know why), with MQTT (don't know why).
    - then it lastly tries to compile the sketch using the toolchain and requires it to return false/0 which means the compilation succeeded. If build fails the build log is printed.

Patch lib

- This test creates a toolchain, a sketch and a board and compiles these to see if they run successfully. I think the purpose of the test is to patch the system during runtime maybe? It has a path to a patch but I don't see where the actual patching is done. I'm guessing this is done automatically when the board is started/configured.
    - Like with the last test it creates a toolchain and requires that it isn't yet a suitable environment.
    - Then it creates an analog write pluginManifest with code from github in the same way as it did in the valid manifest processing test.
    - Then it creates a sketch config with the plugin manifest.
    - The sketch config is then used to create a sketch.
    - It then tries to compile the sketch with the toolchain created earlier. The compile is required to return 0, which means the compile succeeded.
    - After this it creates a board and a boardconfig and then requires that the board can successfully be configured with the boardconfig.
    - It attaches the sketch to the board and requires it to be successful.
    - Then it requires that starting the board is successful.
    - then it pulls out pin0 from the board and requires it to exist.
    - Then it tests that pin0 will return 42 within 16384 ticks of length 1ms.
    - Finally it requires that the board will stop successfully.

Polyfills

I didn't know what "polyfill" means, I found out that polyfill is "is code that implements a feature on "web browsers" (or I guess on any medium) that do not support the feature".

“Polyfills allow web developers to use an API regardless of whether or not it is supported by a browser, and usually with minimal overhead”. “Polyfilling is providing a polyfill for a feature”.

I guess this means that we implement a feature, like Wifi and SD, on a board using the legacy libraries - in other words, we implement a feature that the board does not generally support.

Wifi intended use
This test case checks if the wifi and mqtt preproc libraries can be successfully compiled on a sketch.

- Creates a toolchain in the smce path, checks to see that this isn’t a suitable environment
- Creates a sketch called “sk”, with a SKETCHES_PATH “wifi” and two parameters - board name (fqbn) being “arduino:avr:nano” and legacy_preproc_libs being “Wifi” and “MQTT”. These two represent the two legacy libraries that we want to use
- Compile the sketch and save it to “ec” variable
- If the “ec” variable is true and therefore the sketch was not successfully compiled, we print build log as error output
- Check that “ec” is false, meaning the compile was successful.

SD polyfill
This test checks whether the SD polyfilling is working. It includes not just compiling the sketch with the SD library, but also tests the correct workflow of the filesystem.

- Create a toolchain in the smce path, check for any errors using check_sustainable environment.
- Create a sketch called “sk” with a SKETCHES_PATH “sd_fs” and two parameters - board name (fqbn) being “arduino:avr:nano” and legacy_preproc_libs being “SD”.
- Compile the sketch and save it to “ec” variable
- If “ec” is true,and therefore the sketch was not successfully compiled, we print build log as error output.
- Check that “ec” is false, meaning the compile was successful.
- Create an empty board “br”.
- Create a board config “bc” with three parameters - pins, gpio_drivers, and sd_cards.
- Configure the board “br” using the board config “bc” parameters, check that it was configured.
- If there exists a STORAGE_PATH in the filesystem, remove everything from that directory, create a new directory STORAGE_PATH.
- Attach the sketch “sk” to the board “br” and check.
- Start the board “br” and check.
- Save the digital pin on the board (that was configured using board config) in the “d0” variable.
- Check that there exists a “foo” storage path.
- Check that there is a directory in the path called “foo”.
- Check that there exists a “bar” storage path.
- Check that there is a directory in the path called “bar”.
- Check that there exists a “bar/baz” storage path.
- Check that “bar/baz” is a regular file (?) in the filesystem.

- Create an ifstream named baz reading from storage path bar/baz.C
- Check that we have connection to the “baz” file and can perform read and write.
- Read from “baz”, get the first string and save it in “s”.
- Read the string s and check that it equals “quxx”.