libSMCE_System_comprehension

Definitions from Dimitris article:

- "SMCE is comprised of two parts: The "front-end" (smce-gd) and the "back-end" (libSMCE)."
- "libSMCE is the library responsible for compiling and running the Arduino sketches on your computer. This cross-platform C++ library provides its consumers the ability to compile and execute Arduino sketches on a hosted environment, with bindings to its virtual I/O ports to allow the host application to interact with its child sketches."
- "smce-gd depends on libSMCE to visualize the sketch execution in a colorful virtual 3D world and the interaction with the surrounding environment. Unless you are planning to do something "smart", extend or contribute to the project etc, smce-gd is the software you shall primarily care about. For this tutorial, I will be using the 1.3.1 release of smce-gd, running on Ubuntu 20.04."

Class?	**Who?**
BoardConfig	**Lukas**
BoardView	**Max**
Toolchain	**Pontus**
Sketch	**Pontus**
SketchConf	**Marek**
PluginManifest	**Marek**
BoardData	**Tim**
SharedBoardData	**Tim**

<u>Board:</u> **Representing the simulated hardware of Arduino.**

- m_status:Status
    - clean, configured, running, suspended, stopped
- m_conf_opt:BoardConfig?
    - Member for board configuration

- m_sketch_ptr
    - Pointer to attached sketch
- m_exit_notfiy
    - Function to say what is supposed to happen, when the sketch exits unexpected
- status()
    - Return the current status of the board.
- view()
    - Returns BoardView (Configuration of elements we are working with).
    - Only returns something when the board is running or suspended. If that is the case, a BoardView object is created by passing the shared board data.
- attach_sketch(sketch:Sketch)
    - Attaches the passed sketch to the board.
- get_sketch()
    - returns the private pointer to the sketch (m_sketch_ptr)
- tick()
    - Called with every tick of the processor. Checks whether the sketch is still running.
    - If not, sweep (cleanup) is done, the board is set to status STOP and m_exit_notify is called with the sketch exit code.
- reset()
    - Does reap and sweep and resets all variables like the sketch pointer.
    - Sets the status of the board to CLEAN.
- configure(bconf: BoardConfig)
    - When the board is not configured yet, it has to be clean, otherwise the configuration is not possible.
    - If it is already configured, it can always be reconfigured.
    - The parameter value is put into the private member m_conf_opt and the status of the board is set to CONFIGURED.
- start()
    - It has to be configured, otherwise it cannot be started. If it is not configured, then it has to be in another state than STOPPED.
    - When there is no sketch or the sketch is not compiled, it is also not possible to start the board.
    - Then there is a third check whether all devices are bound.
    - If all is fine, the spawn is done and the status of the board is set to RUNNING.
- suspend()
    - We can only suspend when the board is already running.
    - Process is suspended (OS specific)
    - Status of the board is set to SUSPENDED
- resume()
    - We can only resume when the board is suspended.
    - Process is resumed (OS specific)
    - Status of the board is set to RUNNING
- terminate()
    - We can only terminate when the board is running or suspended.
    - Board is reaped
    - Status of the board is set to STOPPED

- stop()
    - At the moment it just calls terminate(), it is a FIXME in the code.
- do_spawn()
- do_reap()
- do_sweep()

BoardView: Acts as an interface towards the board. (I think) But it has no direct connection to Board as far as I can see. It only uses BoardData which I assume indirectly connects it to the Board? Anyway, all the calls on BoardView eventually make it through to BoardData. the BoardView class (which is only one of many classes in the file) is the "main" class of the file and contains instances of most of the other classes. Would be nice to know more about how the BoardView and Board are connected and what BoardData does in this context. All the other classes mostly do low-level operations that are a bother to read through.

- The frontend version only calls on the methods of the backend without doing anything special on its own as far as I can see.
-

- Class VirtualAnalogDriver (analog driver for GPIO pin)
    - VirtualPin is friend class (VirtualPin has access to VirtualAnalogDriver)
    - m_bdat, a BoardData pointer.
    - m_idx, a std::size_t that contains the current index for read/write operations on the driver.
    - exists(), if m_bdat == true (?) && if m_bdat contains more pins than m_idx: return true.
    - can_read(), if exists() && the pin at m_idx "can analog read": return true.
    - can_write(), if exists() && the pin at m_idx "can analog write": return true.
    - read(), if exists(): return value at pin m_idx, Otherwise return 0.
    - void write(std:uint16_t), if exists(): write input to pin at m_idx.
- Class VirtualDigitalDriver
    - VirtualPin is friend class (VirtualPin has access to VirtualAnalogDriver)
    - m_bdat, a BoardData pointer.
    - m_idx, a std::size_t that contains the current index for read/write operations on the driver.
    - exists(), if m_bdat == true (?) && if m_bdat contains more pins than m_idx: return true.
    - can_read(), if exists() && the pin at m_idx "can digital read": return true.
    - can_write(), if exists() && the pin at m_idx "can digital write": return true.
    - read(), if exists() && loading value at pin m_idx is successfull: return true.
    - void write(bool), if exists(): write input to pin at m_idx.
- Class VirtualPin
    - friends with VirtualPins, (VirtualPins has access to VirtualPin)
    - m_bdat, a pointer to BoardData
    - m_idx
    - enum DataDirection, can be in or out.
    - exists(), if m_bdat == true (?) && if m_bdat contains more pins than m_idx: return true.
    - locked(), if exists() && the active driver at pin at m_idx != gpio (?).
    - set_direction(DataDirectition), if exists
    - get_direction(),
    - digital(), it returns a list with m_bdat and m_idx I think...
    - analog(), it returns a list with m_bdat and m_idx I think...

- Class VirtualPins
    -
- Class VirtualUartBuffer
- Class VirtualUart
- Class VirtualUarts
- Class FrameBuffer
- Class FrameBuffers
- Class BoardView
    - This is where it all comes together. This class holds instances of the others and thus creates a centralized point where other classes can interact with the board (maybe).
- Class VirtualUarts::Iterator
-

<u>Toolchain</u> **(used to configure and build sketches):**

- Toolchain(stdfs::path resources_dir) - **Constructor**
  - Has to be created with a given path to the SMCE resources directory (*SMCE_Resorces.zip*)

- m_res_dir
  - path to the SMCE resources directory (*SMCE_Resorces.zip*)
- m_cmake_path
  - defaults to simply “cmake”, this probably only has to change if ran on a system without CMake on path
- m_build_log
- m_build_log_mtx
- compile(Sketch& sketch)
  - Checks that given sketch exists
  - Compiles sketch by running do_configure() and do_build() on the sketch
- do_configure(Sketch& sketch)
  - Private method used while compiling a sketch
  - Creates a /tmp dir in resources directory, writes plugin manifests
  - Creates? a CMake file with a bunch of flags taken from the sketch: “/RtResources/SMCE/share/CMake/Scripts/ConfigureSketch.cmake”
- do_build(Sketch& sketch)
  - Private method used while compiling a sketch
  - Builds the sketch with CMake to the tmp directory created in *do_configure()*
- resource_dir()
  - getter for the path to the SMCE resources directory
- cmake_path()
  - getter for the path to cmake
- build_log()
- check_suitable_environment()
  - Checks whether CMake is on path

<u>Sketch:</u>

- Sketch(stdfs::path source, SketchConfig conf) - **Constructor**
  - Takes path to the sketch and a *SketchConfig* object
  - Removes existing tmp directory if present
  - Do not really understand the rest of the constructor

- m_uuid
  - Unique ID generated when instanced
- m_conf
  - A *SketchConfig* object used when being compiled, passed through the constructor
- m_source
  - Path to the sketch in the filesystem
- m_tmpdir

    - Path to the tmp directory where sketch-related files will be written during compile
- m_executable
    - Path to compiled sketch?
- m_built
    - Boolean indicating whether the sketch has been built/compiled yet
- get_source()
    - getter for m_source
- is_compiled()
    - getter for m_built
- get_uuid()
    - getter for m_uuid

BoardConfig <<Struct>>: **Represents all the pins etc. “on the blue board”**

https://www.electronicwings.com/arduino/basics

- **BoardConfig**
    - **GpioDrivers**
        - **DigitalDriver**
            - bool: board_read
            - bool: board_write
        - **AnalogDriver**
            - bool: board_read
            - bool: board_write
        - **pin_id**
            - The id of pins
        - **digital_driver**
            - Based on the struct DigitalDriver, representing the digital ports on the board.
        - **analog_driver**
            - Based on the structAnalogDriver, representing the analog ports on the board.
    - **UartChannel => Communication protocol**
        - **rx_pin_override**
            - The receiver pin on the board.
        - **tx_pin_override**
            - The transmitter pin on the board.
        - **baud_rate = 9600**
            - Number of changes in signal per second. I.e. how fast the communication will be. baud rate = bit rate in this case.
    - **SecureDigitalStorage**
        - Secure storage? Have a path to the root directory.
    - **FrameBuffer**
        -
    - **BoardDevice**
- **vector: pins**
    - Vector of all the GPIO pins on the board (digital+analog).

- **vector<GpioDriver> gpio_drivers**
    - The GPIO drivers to apply on all existing pins.
- **vector<UartChannel> uart_channels**
    - Representing the uart (usart) communication on the board. The board will have two pins, one transmitter and one receiver. The start bit is 0 (low) and the end is 1 (high). The communication will happen in between the high and low (1/0).
-

<u>SketchConf</u>

· This class serves as a configuration class for sketch building

· It utilizes the ArduinoLibrary for preprocessing as well as plugins from the PluginManifest class

· Has only attributes, no methods

· **Struct SMCE_API SketchConfig**

o Library to pull from the Arduino library manager

o **Struct Arduino Library**

§ **String name;**

· (Library name as found in the install command)

§ **String version;**

· (Version string; empty if latest)

· **String fqbn;**

o Fully-qualified board name that the sketch is targeting

· **Vector<std::string> extra_board_uris;**

o Extra board.txt URIs for ArduinoCLI

· **Vector<ArduinoLibrary> legacy_preproc_libs**

o Libraries to use during legacy preprocessing

· **Vector<PluginManifest> plugins**

o Plugins to compile with

PluginManifest

- This class serves as a container for all the Arduino plugins used by the system
- It has several attributes and one method
- *(clang-format off)* Includes an **enum class Defaults** with:

o arduino, /// src/** is sources, src/ is incdir, no linkdir

  - single_dir, /// ./* is sources, ./ is incdir, ./ is linkdir
  - c, /// src/* is sources, include/ is incdir, lib is linkdir
  - none, /// empty
  - cmake, /// do not generate a target and do add_subdirectory

- *(clang-format on)* Includes several attributes:
  - std::string name;
  - std::string version;
  - std::vector<std::string> depends; /// required plugins
  - std::string uri; /// file:// of the source-root or http[s]:// of the tar/zip archive
  - std::string patch_uri; /// same as above but for patching
  - Defaults defaults;
  - std::vector<std::string> incdirs;
  - std::vector<std::string> sources;
  - std::vector<std::string> linkdirs;
  - std::vector<std::string> linklibs;
  - bool development = false; /// set to true to CONFIGURE_DEPENDS all the globs
- Includes a method:
  - **Write_manifest**
    - PluginManifest and location path as a parameter

§ Creates the directory for the manifest

§ If there is an error, write an error message

- The CPP file also includes **two templates**
    - I believe the first one creates a cmake_list or writes into it.
    - The purpose of the second one is not clear yet.

SharedBoardData and BoardData:

- These classes are both internal classes and therefore in the internal package. This means that they cannot be directly used by the frontend.
- SharedBoardData is a container that has an object of type BoardData and additionally an attribute of type boost::interprocess::managed_shared_memory
- => SharedBoardData is a container for including the boost functionality. What I understand is that boost helps with managing memory segments that can be shared between processes. This article could for example help: https://www.boost.org/doc/libs/1_55_0/doc/html/interprocess/managed_memory_segments.html#interprocess.managed_memory_segments.making_ipc_easy.managed_memory_segments_intro
- BoardData includes 4 structs:
    - Pin
    - UartChannel
    - Direct Storage
    - Frame Buffer
- There are 4 lists of objects of these structs in BoardData
- While Board is the blue board and BoardConf is the configuration for running a sketch, BoardData contains information about the data flow and the configuration of all elements on the Arduino Board. (?)
- The class BoardData is also very boost oriented. For example, many attributes in the structs are of a boost type or of either IpcAtomicValue or IpcMovableMutex and these two classes are also extended from boost classes.