AtomicX

Generated by Doxygen 1.9.3

Chapter 1

Arduino procedures

1.1 Install arduino-cli

1.2 Useful commands

```
arduino-cli board details -b arduino:avr:uno --list-programmers
arduino-cli board details "$1"
arduino-cli board listall
arduino-cli board listall esp8266:esp8266:wifi
arduino-cli board listall avr
```

1.3 Compiling and uploading

```
arduino-cli compile -b arduino:avr:nano:cpu=atmega328  --upload -p "$1" "$2"
# upload AVR using USB-ASP
arduino-cli compile -b arduino:avr:nano:cpu=atmega328  --upload -P usbasp  "$1"
# upload using avrisp XSP
arduino-cli compile -b arduino:avr:nano:cpu=atmega328  --upload -P avrisp -p "$1" "$2"
arduino-cli compile -u -p "$1"  -b esp32:esp32:heltec_wifi_kit_32  "$2"
arduino-cli compile -u  -p "$1" -b
       Arduino_STM32:STM32F1:genericSTM32F103C:upload_method=jlinkMethod,opt=o3std "$2"
arduino-cli compile -v -b esp8266:esp8266:wifi_kit_8:baud=921600 -upload -p "<serial>" "<arduino>"
arduino-cli compile -b arduino:mbed_nano:nano33ble -upload -p "<serial>" "<arduino>"
# Using ST-LINK
arduino-cli compile -v -b STMicroelectronics:stm32:GenF1:pnum=BLUEPILL_F103C8,usb=CDCgen -upload -p "."
       "<arduino>"
```

Despite the lack of feature parity at the moment, Arduino CLI provides many of the features you can find in the Arduino IDE. Let's see some examples.

1.3.1 LIST CPUS and other options from a board

```
arduino-cli board details -b STMicroelectronics:stm32:GenF1
arduino-cli board details -b arduino:avr:nano
```

1.4 Before you start

`arduino-cli` is a container of commands and each command has its own dedicated help text that can be shown with the `help` command like this:

```
$ arduino-cli help core
Arduino Core operations.
Usage:
    arduino-cli core [command]
```

```
Examples:
    ./arduino-cli core update-index
Available Commands:
    download     Downloads one or more cores and corresponding tool dependencies.
    install      Installs one or more cores and corresponding tool dependencies.
    list         Shows the list of installed platforms.
    search       Search for a core in Boards Manager.
    uninstall    Uninstalls one or more cores and corresponding tool dependencies if no more used.
    update-index Updates the index of cores.
    upgrade      Upgrades one or all installed platforms to the latest version.
Flags:
    -h, --help   help for core
Global Flags:
        --additional-urls strings   Additional URLs for Boards Manager.
        --config-file string        The custom config file (if not specified the default will be used).
        --format string             The output format, can be [text|json]. (default "text")
        --log-file string           Path to the file where logs will be written.
        --log-format string         The output format for the logs, can be [text|json].
        --log-level string          Messages with this level and above will be logged.
    -v, --verbose                   Print the logs on the standard output.
Use "arduino-cli core [command] --help" for more information about a command.
```

1.5 Create a configuration file

Arduino CLI doesn't strictly require a configuration file to work because the command line interface provides any possible functionality. However, having one can spare you a lot of typing when issuing a command, so let's go ahead and create it with:

```
$ arduino-cli config init
Config file written: /home/luca/.arduino15/arduino-cli.yaml
```

If you inspect the contents of `arduino-cli.yaml`, you'll find the available options with their respective default values. For more information, see the configuration documentation.

1.6 Create a new sketch

To create a new sketch named `MyFirstSketch` in the current directory, run the following command:

```
$ arduino-cli sketch new MyFirstSketch
Sketch created in: /home/luca/MyFirstSketch
```

A sketch is a folder containing assets like source files and libraries; the `new` command creates for you a .ino file called `MyFirstSketch.ino` containing Arduino boilerplate code:

```
$ cat $HOME/MyFirstSketch/MyFirstSketch.ino
void setup() {
}
void loop() {
}
```

At this point you can use your favourite file editor or IDE to open the file `$HOME/MyFirstSketch/MyFirst↩Sketch.ino` and change the code like this:

```
void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

1.7 Connect the board to your PC

The first thing to do upon a fresh install is to update the local cache of available platforms and libraries by running:

```
$ arduino-cli core update-index
Updating index: package_index.json downloaded
```

After connecting the board to your PC by using the USB cable, you should be able to check whether it's been recognized by running:

```
$ arduino-cli board list
Port         Type              Board Name              FQBN                 Core
/dev/ttyACM1 Serial Port (USB) Arduino/Genuino MKR1000 arduino:samd:mkr1000 arduino:samd
```

In this example, the MKR1000 board was recognized and from the output of the command you see the platform core called `arduino:samd` is the one that needs to be installed to make it work.

If you see an `Unknown` board listed, uploading should still work as long as you identify the platform core and use the correct FQBN string. When a board is not detected for whatever reason, you can list all the supported boards and their FQBN strings by running the following:

```
$ arduino-cli board listall mkr
Board Name              FQBN
Arduino MKR FOX 1200    arduino:samd:mkrfox1200
Arduino MKR GSM 1400    arduino:samd:mkrgsm1400
Arduino MKR WAN 1300    arduino:samd:mkrwan1300
Arduino MKR WiFi 1010   arduino:samd:mkrwifi1010
Arduino MKRZERO         arduino:samd:mkrzero
Arduino/Genuino MKR1000 arduino:samd:mkr1000
```

1.8 Install the core for your board

To install the `arduino:samd` platform core, run the following:

```
$ arduino-cli core install arduino:samd
Downloading tools...
arduino:arm-none-eabi-gcc@4.8.3-2014q1 downloaded
arduino:bossac@1.7.0 downloaded
arduino:openocd@0.9.0-arduino6-static downloaded
arduino:CMSIS@4.5.0 downloaded
arduino:CMSIS-Atmel@1.1.0 downloaded
arduino:arduinoOTA@1.2.0 downloaded
Downloading cores...
arduino:samd@1.6.19 downloaded
Installing tools...
Installing platforms...
Results:
arduino:samd@1.6.19 - Installed
arduino:arm-none-eabi-gcc@4.8.3-2014q1 - Installed
arduino:bossac@1.7.0 - Installed
arduino:openocd@0.9.0-arduino6-static - Installed
arduino:CMSIS@4.5.0 - Installed
arduino:CMSIS-Atmel@1.1.0 - Installed
arduino:arduinoOTA@1.2.0 - Installed
```

Now verify we have installed the core properly by running:

```
$ arduino-cli core list
ID              Installed       Latest  Name
arduino:samd    1.6.19          1.6.19  Arduino SAMD Boards (32-bits ARM Cortex-M0+)
```

Great! Now we are ready to compile and upload the sketch.

1.9 Adding 3rd party cores

If your board requires 3rd party core packages to work, you can list the URLs to additional package indexes in the Arduino CLI configuration file.

For example, to add the ESP8266 core, edit the configuration file and change the `board_manager` settings as follows:

```
board_manager:
  additional_urls:
    - https://arduino.esp8266.com/stable/package_esp8266com_index.json
```

If you have your package indexes locally installed, you can list their file path in the Arduino CLI configuration file.

For example, to add the NRF52832 core, edit the configuration file and change the `board_manager` settings as follows:

```
board_manager:
  additional_urls:
    - https://arduino.esp8266.com/stable/package_esp8266com_index.json
    - file:///absolute/path/to/your/package_nrf52832_index.json
```

From now on, commands supporting custom cores will automatically use the additional URL from the configuration file:

```
$ arduino-cli core update-index
Updating index: package_index.json downloaded
Updating index: package_esp8266com_index.json downloaded
Updating index: package_nrf52832_index.json
Updating index: package_index.json downloaded
$ arduino-cli core search esp8266
ID              Version Name
esp8266:esp8266 2.5.2   esp8266
```

Alternatively, you can pass a link to the additional package index file with the `--additional-urls` option, that has to be specified every time and for every command that operates on a 3rd party platform core, for example:

```
$ arduino-cli  core update-index --additional-urls
       https://arduino.esp8266.com/stable/package_esp8266com_index.json
Updating index: package_esp8266com_index.json downloaded
$ arduino-cli core search esp8266 --additional-urls
       https://arduino.esp8266.com/stable/package_esp8266com_index.json
ID              Version Name
esp8266:esp8266 2.5.2   esp8266
```

The same applies to the additional package index file provided by file paths:

```
$ arduino-cli  core update-index --additional-urls
       file:///absolute/path/to/your/package_esp8266com_index.json
Updating index: package_esp8266com_index.json downloaded
$ arduino-cli core search esp8266 --additional-urls
       file:///absolute/path/to/your/package_esp8266com_index.json
ID              Version Name
esp8266:esp8266 2.5.2   esp8266
```

1.10 Compile and upload the sketch

To compile the sketch you run the `compile` command, passing the proper FQBN string:

```
$ arduino-cli compile --fqbn arduino:samd:mkr1000 MyFirstSketch
Sketch uses 9600 bytes (3%) of program storage space. Maximum is 262144 bytes.
```

To upload the sketch to your board, run the following command, using the serial port your board is connected to:

```
$ arduino-cli upload -p /dev/ttyACM0 --fqbn arduino:samd:mkr1000 MyFirstSketch
No new serial port detected.
Atmel SMART device 0x10010005 found
Device       : ATSAMD21G18A
Chip ID      : 10010005
Version      : v2.0 [Arduino:XYZ] Dec 20 2016 15:36:43
Address      : 8192
Pages        : 3968
Page Size    : 64 bytes
Total Size   : 248KB
Planes       : 1
Lock Regions : 16
Locked       : none
Security     : false
Boot Flash   : true
BOD          : true
BOR          : true
Arduino      : FAST_CHIP_ERASE
Arduino      : FAST_MULTI_PAGE_WRITE
Arduino      : CAN_CHECKSUM_MEMORY_BUFFER
Erase flash
done in 0.784 seconds
Write 9856 bytes to flash (154 pages)
[==============================] 100% (154/154 pages)
done in 0.069 seconds
Verify 9856 bytes of flash with checksum.
Verify successful
done in 0.009 seconds
CPU reset.
```

1.11 Add libraries

If you need to add more functionalities to your sketch, chances are some of the libraries available in the Arduino ecosystem already provide what you need. For example, if you need a debouncing strategy to better handle button inputs, you can try searching for the `debouncer` keyword:

```
$ arduino-cli lib search debouncer
Name: "Debouncer"
    Author: hideakitai
    Maintainer: hideakitai
    Sentence: Debounce library for Arduino
    Paragraph: Debounce library for Arduino
    Website: https://github.com/hideakitai
    Category: Timing
    Architecture: *
    Types: Contributed
    Versions: [0.1.0]
Name: "FTDebouncer"
    Author: Ubi de Feo
    Maintainer: Ubi de Feo, Sebastian Hunkeler
    Sentence: An efficient, low footprint, fast pin debouncing library for Arduino
    Paragraph: This pin state supervisor manages debouncing of buttons and handles transitions between LOW
       and HIGH state, calling a function and notifying your code of which pin has been activated or
       deactivated.
    Website: https://github.com/ubidefeo/FTDebouncer
    Category: Uncategorized
    Architecture: *
    Types: Contributed
    Versions: [1.3.0]
Name: "SoftTimer"
    Author: Balazs Kelemen <prampec+arduino@gmail.com>
    Maintainer: Balazs Kelemen <prampec+arduino@gmail.com>
    Sentence: SoftTimer is a lightweight pseudo multitasking solution for Arduino.
    Paragraph: SoftTimer enables higher level Arduino programing, yet easy to use, and lightweight. You are
       often faced with the problem that you need to do multiple tasks at the same time. In SoftTimer, the
       programmer creates Tasks that runs periodically. This library comes with a collection of handy tools
       like blinker, pwm, debouncer.
    Website: https://github.com/prampec/arduino-softtimer
    Category: Timing
    Architecture: *
    Types: Contributed
    Versions: [3.0.0, 3.1.0, 3.1.1, 3.1.2, 3.1.3, 3.1.5, 3.2.0]
```

Our favourite is `FTDebouncer`, let's install it by running:

```
$ arduino-cli lib install FTDebouncer
FTDebouncer depends on FTDebouncer@1.3.0
Downloading FTDebouncer@1.3.0...
FTDebouncer@1.3.0 downloaded
Installing FTDebouncer@1.3.0...
Installed FTDebouncer@1.3.0
```

1.12 Using the <tt>daemon</tt> mode and the gRPC interface

Arduino CLI can be launched as a gRPC server via the `daemon` command.

The `client_example` folder contains a sample client code that shows how to interact with the gRPC server. Available services and messages are detailed in the gRPC reference pages.

To provide observability for the gRPC server activities besides logs, the `daemon` mode activates and exposes by default a `Prometheus` endpoint (`http://localhost:9090/metrics`) that can be fetched for metrics data like:

```
# TYPE daemon_compile counter
daemon_compile{buildProperties="",exportFile="",fqbn="arduino:samd:mkr1000",installationID="ed6f1f22-1fbe-4b1f-84be-84d035b63
       1 1580385724726
# TYPE daemon_board_list counter
daemon_board_list{installationID="ed6f1f22-1fbe-4b1f-84be-84d035b6369c",success="true"} 1 1580385724833
```

The metrics settings are exposed via the `metrics` section in the CLI configuration:

```
metrics:
  enabled: true
  addr: :9090
```

1.13 Configuration keys

- board_manager
  - additional_urls - the URLs to any additional Boards Manager package index files needed for your boards platforms.
- daemon - options related to running Arduino CLI as a gRPC server.
  - port - TCP port used for gRPC client connections.
- directories - directories used by Arduino CLI.
  - data - directory used to store Boards/Library Manager index files and Boards Manager platform installations.
  - downloads - directory used to stage downloaded archives during Boards/Library Manager installations.
  - user - the equivalent of the Arduino IDE's "sketchbook" directory. Library Manager installations are made to the libraries subdirectory of the user directory.
- library - configuration options relating to Arduino libraries.
  - enable_unsafe_install - set to true to enable the use of the --git-url and --zip-file flags with `arduino-cli lib install`. These are considered "unsafe" installation methods because they allow installing files that have not passed through the Library Manager submission process.
- logging - configuration options for Arduino CLI's logs.
  - file - path to the file where logs will be written.
  - format - output format for the logs. Allowed values are text or json.
  - level - messages with this level and above will be logged. Valid levels are: trace, debug, info, warn, error, fatal, panic.
- metrics - settings related to the collection of data used for continued improvement of Arduino CLI.
  - addr - TCP port used for metrics communication.
  - enabled - controls the use of metrics.
- sketch - configuration options relating to Arduino sketches.
  - always_export_binaries - set to true to make `arduino-cli compile` always save binaries to the sketch folder. This is the equivalent of using the --export-binaries flag.
- updater - configuration options related to Arduino CLI updates
  - enable_notification - set to false to disable notifications of new Arduino CLI releases, defaults to true

1.14 Configuration methods

Arduino CLI may be configured in three ways:

1. Command line flags
1. Environment variables
1. Configuration file

If a configuration option is configured by multiple methods, the value set by the method highest on the above list overwrites the ones below it.

If a configuration option is not set, Arduino CLI uses a default value.

`arduino-cli config dump` displays the current configuration values.

1.14.1 Command line flags

Arduino CLI's command line flags are documented in the command line help and the Arduino CLI command reference.

1.14.1.1 Example

Setting an additional Boards Manager URL using the `--additional-urls` command line flag:

```
$ arduino-cli core update-index --additional-urls
      https://downloads.arduino.cc/packages/package_staging_index.json
```

1.14.2 Environment variables

All configuration options can be set via environment variables. The variable names start with `ARDUINO`, followed by the configuration key names, with each component separated by _. For example, the `ARDUINO_DIRECTORIES_USER` environment variable sets the `directories.user` configuration option.

On Linux or macOS, you can use the `export command` to set environment variables. On Windows cmd, you can use the `set command`.

1.14.2.1 Example

Setting an additional Boards Manager URL using the `ARDUINO_BOARD_MANAGER_ADDITIONAL_URLS` environment variable:

```
$ export
      ARDUINO_BOARD_MANAGER_ADDITIONAL_URLS=https://downloads.arduino.cc/packages/package_staging_index.json
```

1.14.3 Configuration file

`arduino-cli config init` creates or updates a configuration file with the current configuration settings.

This allows saving the options set by command line flags or environment variables. For example:

```
arduino-cli config init --additional-urls https://downloads.arduino.cc/packages/package_staging_index.json
```

1.14.3.1 File name

The configuration file must be named `arduino-cli`, with the appropriate file extension for the file's format.

1.14.3.2 Supported formats

`arduino-cli config init` creates a YAML file, however a variety of common formats are supported:

- `JSON`
- `TOML`
- `YAML`
- `Java properties file`
- `HCL`
- envfile
- `INI`

1.14.3.3 Locations

Configuration files in the following locations are recognized by Arduino CLI:

1. Location specified by the `--config-file` command line flag

1. Current working directory

1. Any parent directory of the current working directory (more immediate parents having higher precedence)

1. Arduino CLI data directory (as configured by `directories.data`)

If multiple configuration files are present, the one highest on the above list is used. Configuration files are not combined.

The location of the active configuration file can be determined by running the command:
```
arduino-cli config dump --verbose
```

1.14.3.4 Example

Setting an additional Boards Manager URL using a YAML format configuration file:
```
board_manager:
  additional_urls:
    - https://downloads.arduino.cc/packages/package_staging_index.json
```

Doing the same using a TOML format file:
```
[board_manager]
additional_urls = [ "https://downloads.arduino.cc/packages/package_staging_index.json" ]
```

This is the specification for Arduino sketches.

The programs that run on Arduino boards are called "sketches". This term was inherited from Processing, upon which the Arduino IDE and the core API were based.

1.15 Sketch folders and files

The sketch root folder name and code file names must start with a basic letter (`A-Z` or `a-z`) or number (`0-9`), followed by basic letters, numbers, underscores (_), dots (`.`) and dashes (`-`). The maximum length is 63 characters.

Support for names starting with a number was added in Arduino IDE 1.8.4.

1.15.1 Sketch root folder

Because many Arduino sketches only contain a single .ino file, it's easy to think of that file as the sketch. However, it is the folder that is the sketch. The reason is that sketches may consist of multiple code files and the folder is what groups those files into a single program.

1.15.2 Primary sketch file

Every sketch must contain a `.ino` file with a file name matching the sketch root folder name.

`.pde` is also supported but **deprecated** and will be removed in the future, using the `.ino` extension is strongly recommended.

1.15.3 Additional code files

Sketches may consist of multiple code files.

The following extensions are supported:

- .ino - `Arduino language` files.
- .pde - Alternate extension for Arduino language files. This file extension is also used by Processing sketches. .ino is recommended to avoid confusion. `**.pde` extension is deprecated and will be removed in the future.**
- .cpp - C++ files.
- .c - C Files.
- .S - Assembly language files.
- .h - Header files.
- .tpp, .ipp - Header files (available from Arduino CLI 0.19.0).

For information about how each of these files and other parts of the sketch are used during compilation, see the Sketch build process documentation.

1.15.4 <tt>src</tt> subfolder

The contents of the `src` subfolder are compiled recursively. Unlike the code files in the sketch root folder, these files are not shown as tabs in the IDEs.

This is useful for files you don't want to expose to the sketch user via the IDE's interface. It can be used to bundle libraries with the sketch in order to make it a self-contained project.

Arduino language files under the `src` folder are not supported.

- In Arduino IDE 1.6.5-r5 and older, no recursive compilation was done.
- In Arduino IDE 1.6.6 - 1.6.9, recursive compilation was done of all subfolders of the sketch folder.
- In Arduino IDE 1.6.10 and newer, recursive compilation is limited to the `src` subfolder of the sketch folder.

1.15.5 <tt>data</tt> subfolder

The `data` folder is used to add additional files to the sketch, which will not be compiled.

Files added to the sketch via the Arduino IDE's **Sketch** > **Add File...** are placed in the `data` folder.

The Arduino IDE's **File** > **Save As...** only copies the code files in the sketch root folder and the full contents of the `data` folder, so any non-code files outside the `data` folder are stripped.

1.15.6 Metadata

Arduino CLI and Arduino Web Editor use a file named sketch.json, located in the sketch root folder, to store sketch metadata.

The `cpu` key contains the board configuration information. This can be set via `arduino-cli board attach` or by selecting a board in the Arduino Web Editor while the sketch is open. With this configuration set, it is not necessary to specify the `--fqbn` or `--port` flags to the `arduino-cli compile` or `arduino-cli upload` commands when compiling or uploading the sketch.

The `included_libs` key defines the library versions the Arduino Web Editor uses when the sketch is compiled. This is Arduino Web Editor specific because all versions of all the Library Manager libraries are pre-installed in Arduino Web Editor, while only one version of each library may be installed when using the other Arduino development software.

1.15.7 Secrets

Arduino Web Editor has a `"Secret tab" feature` that makes it easy to share sketches without accidentally exposing sensitive data (e.g., passwords or tokens). The Arduino Web Editor automatically generates macros for any identifier in the sketch which starts with `SECRET_` and contains all uppercase characters.

When you download a sketch from Arduino Web Editor that contains a Secret tab, the empty `#define` directives for the secrets are in a file named arduino_secrets.h, with an `#include` directive to that file at the top of the primary sketch file. This is hidden when viewing the sketch in Arduino Web Editor.

1.15.8 Documentation

Image and text files in common formats which are present in the sketch root folder are displayed in tabs in the Arduino Web Editor.

1.15.9 Sketch file structure example

```
Foo
|_ arduino_secrets.h
|_ Abc.ino
|_ Def.cpp
|_ Def.h
|_ Foo.ino
|_ Ghi.c
|_ Ghi.h
|_ Jkl.h
|_ Jkl.S
|_ sketch.json
|_ data
|  |_ Schematic.pdf
|_ src
   |_ SomeLib
      |_ library.properties
      |_ src
         |_ SomeLib.h
         |_ SomeLib.cpp
```

1.16 Sketchbook

The Arduino IDE provides a "sketchbook" folder (analogous to Arduino CLI's "user directory"). In addition to being the place where user libraries and manually installed platforms are installed, the sketchbook is a convenient place to store sketches. Sketches in the sketchbook folder appear under the Arduino IDE's **File** > **Sketchbook** menu. However, there is no requirement to store sketches in the sketchbook folder.

1.17 Library/Boards Manager links

A URI in a comment in the form `http://librarymanager#SEARCH_TERM` will open a search for SEARCH_TERM in `Library Manager` when clicked in the Arduino IDE.

A URI in a comment in the form `http://boardsmanager#SEARCH_TERM` will open a search for SEARCH_TERM in `Boards Manager` when clicked in the Arduino IDE.

This can be used to offer the user an easy way to install dependencies of the sketch.

This feature is only available when using the Arduino IDE, so be sure to provide supplementary documentation to help the users of other development software install the sketch dependencies.

This feature was added in Arduino IDE 1.6.9.

1.17.1 Example

```
 {c++}
// install the Arduino SAMD Boards platform to add support for your MKR WiFi 1010 board
// if using the Arduino IDE, click here: http://boardsmanager#SAMD
// install the WiFiNINA library via Library Manager
// if using the Arduino IDE, click here: http://librarymanager#WiFiNINA
#include <WiFiNINA.h>
```

1.18 See also

- Sketch build process documentation
- `Style guide for example sketches`

1.19 Boards

gustavocampos@GUSTAVOs-MBP .ssh % arduino-cli core list ID Installed Latest Name arduino:avr 1.8.3 1.8.↩ 4 Arduino AVR Boards arduino:megaavr 1.8.7 1.8.7 Arduino megaAVR Boards arduino:sam 1.6.12 1.6.12 Arduino SAM Boards (32-bits ARM Cortex-M3) arduino:samd 1.8.11 1.8.12 Arduino SAMD Boards (32-bits ARM Cortex-↩ M0+) esp32:esp32 1.0.6 1.0.6 ESP32 Arduino esp8266:esp8266 3.0.2 3.0.2 ESP8266 Boards (3.0.2) Maixduino↩ :k210 0.3.11 0.3.11 Maixduino

1.20 repos

```
 https://raw.githubusercontent.com/sparkfun/Arduino_Boards/master/IDE_↩
Board_Manager/package_sparkfun_index.json https://dl.espressif.com/dl/package↩
_esp32_index.json     http://arduino.esp8266.com/stable/package_esp8266com↩
_index.json      https://raw.githubusercontent.com/stm32duino/BoardManager↩
Files/master/package_stmicroelectronics_index.json https://raw.githubusercontent.↩
com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.↩
json http://dl.sipeed.com/MAIX/Maixduino/package_Maixduino_k210_index.json
```

1.21 Full TODAY's boards

Board Name	FQBN
3D printer boards	STMicroelectronics:stm32:3dprinter
4D Systems gen4 IoD Range	esp8266:esp8266:gen4iod
AI Thinker ESP32-CAM	esp32:esp32:esp32cam
ALKS ESP32	esp32:esp32:alksesp32
ATtiny24/44/84	attiny:avr:ATtinyX4
ATtiny25/45/85	attiny:avr:ATtinyX5
Adafruit Circuit Playground	arduino:avr:circuitplay32u4cat
Adafruit Circuit Playground Express	arduino:samd:adafruit_circuitplayground_m0
Adafruit ESP32 Feather	esp32:esp32:featheresp32
Adafruit Feather HUZZAH ESP8266	esp8266:esp8266:huzzah
Amperka WiFi Slot	esp8266:esp8266:wifi_slot
Arduino	esp8266:esp8266:arduino-esp8266
Arduino BT	arduino:avr:bt
Arduino Due (Native USB Port)	arduino:sam:arduino_due_x
Arduino Due (Programming Port)	arduino:sam:arduino_due_x_dbg
Arduino Duemilanove or Diecimila	arduino:avr:diecimila
Arduino Esplora	arduino:avr:esplora
Arduino Ethernet	arduino:avr:ethernet
Arduino Fio	arduino:avr:fio
Arduino Gemma	arduino:avr:gemma
Arduino Industrial 101	arduino:avr:chiwawa
Arduino Leonardo	arduino:avr:leonardo
Arduino Leonardo ETH	arduino:avr:leonardoeth
Arduino M0	arduino:samd:mzero_bl
Arduino M0 Pro (Native USB Port)	arduino:samd:mzero_pro_bl
Arduino M0 Pro (Programming Port)	arduino:samd:mzero_pro_bl_dbg
Arduino MKR FOX 1200	arduino:samd:mkrfox1200
Arduino MKR GSM 1400	arduino:samd:mkrgsm1400
Arduino MKR NB 1500	arduino:samd:mkrnb1500
Arduino MKR Vidor 4000	arduino:samd:mkrvidor4000
Arduino MKR WAN 1300	arduino:samd:mkrwan1300
Arduino MKR WAN 1310	arduino:samd:mkrwan1310
Arduino MKR WiFi 1010	arduino:samd:mkrwifi1010
Arduino MKR1000	arduino:samd:mkr1000
Arduino MKRZERO	arduino:samd:mkrzero
Arduino Mega ADK	arduino:avr:megaADK
Arduino Mega or Mega 2560	arduino:avr:mega
Arduino Micro	arduino:avr:micro
Arduino Mini	arduino:avr:mini
Arduino NANO 33 IoT	arduino:samd:nano_33_iot
Arduino NG or older	arduino:avr:atmegang
Arduino Nano	arduino:avr:nano
Arduino Nano 33 BLE	arduino:mbed_nano:nano33ble
Arduino Nano Every	arduino:megaavr:nona4809
Arduino Nano RP2040 Connect	arduino:mbed_nano:nanorp2040connect
Arduino Pro or Pro Mini	arduino:avr:pro
Arduino Robot Control	arduino:avr:robotControl
Arduino Robot Motor	arduino:avr:robotMotor
Arduino Tian	arduino:samd:tian
Arduino Uno	arduino:avr:uno
Arduino Uno Mini	arduino:avr:unomini
Arduino Uno WiFi	arduino:avr:unowifi
Arduino Uno WiFi Rev2	arduino:megaavr:uno2018
Arduino Yún	arduino:avr:yun
Arduino Yún Mini	arduino:avr:yunmini
Arduino Zero (Native USB Port)	arduino:samd:arduino_zero_native
Arduino Zero (Programming Port)	arduino:samd:arduino_zero_edbg
BPI-BIT	esp32:esp32:bpi-bit
Blues Wireless boards	STMicroelectronics:stm32:BluesW
D-duino-32	esp32:esp32:d-duino-32
DOIT ESP-Mx DevKit (ESP8285)	esp8266:esp8266:espmxdevkit
DOIT ESP32 DEVKIT V1	esp32:esp32:esp32doit-devkit-v1
DOIT ESPduino32	esp32:esp32:esp32doit-espduino
Digistump Oak	esp8266:esp8266:oak
Discovery	STMicroelectronics:stm32:Disco
Dongsen Tech Pocket 32	esp32:esp32:pocket_32
ESP32 Dev Module	esp32:esp32:esp32
ESP32 FM DevKit	esp32:esp32:fm-devkit
ESP32 Pico Kit	esp32:esp32:pico32
ESP32 Wrover Module	esp32:esp32:esp32wrover
ESP32vn IoT Uno	esp32:esp32:esp32vn-iot-uno
ESPDuino (ESP-13 Module)	esp8266:esp8266:espduino
ESPea32	esp32:esp32:espea32
ESPectro Core	esp8266:esp8266:espectro
ESPectro32	esp32:esp32:espectro32
ESPino (ESP-12 Module)	esp8266:esp8266:espino
ESPresso Lite 1.0	esp8266:esp8266:espresso_lite_v1
ESPresso Lite 2.0	esp8266:esp8266:espresso_lite_v2
ET-Board	esp32:esp32:ET-Board
Elecgator boards	STMicroelectronics:stm32:Elecgator
Electronic SweetPeas - ESP320	esp32:esp32:esp320
Electronic speed controllers	STMicroelectronics:stm32:ESC_board
Eval	STMicroelectronics:stm32:Eval
FireBeetle-ESP32	esp32:esp32:firebeetle32

Frog Board ESP32	esp32:esp32:frogboard
Garatronic-McHobby	STMicroelectronics:stm32:Garatronic
Generic ESP8266 Module	esp8266:esp8266:generic
Generic ESP8285 Module	esp8266:esp8266:esp8285
Generic Flight Controllers	STMicroelectronics:stm32:GenFlight
Generic STM32F0 series	STMicroelectronics:stm32:GenF0
Generic STM32F1 series	STMicroelectronics:stm32:GenF1
Generic STM32F2 series	STMicroelectronics:stm32:GenF2
Generic STM32F3 series	STMicroelectronics:stm32:GenF3
Generic STM32F4 series	STMicroelectronics:stm32:GenF4
Generic STM32F7 series	STMicroelectronics:stm32:GenF7
Generic STM32G0 series	STMicroelectronics:stm32:GenG0
Generic STM32G4 series	STMicroelectronics:stm32:GenG4
Generic STM32H7 Series	STMicroelectronics:stm32:GenH7
Generic STM32L0 series	STMicroelectronics:stm32:GenL0
Generic STM32L1 series	STMicroelectronics:stm32:GenL1
Generic STM32L4 series	STMicroelectronics:stm32:GenL4
Generic STM32L5 series	STMicroelectronics:stm32:GenL5
Generic STM32U5 series	STMicroelectronics:stm32:GenU5
Generic STM32WB series	STMicroelectronics:stm32:GenWB
Generic STM32WL series	STMicroelectronics:stm32:GenWL
HONEYLemon	esp32:esp32:honeylemon
Heltec WiFi Kit 32	esp32:esp32:heltec_wifi_kit_32
Heltec WiFi LoRa 32	esp32:esp32:heltec_wifi_lora_32
Heltec WiFi LoRa 32(V2)	esp32:esp32:heltec_wifi_lora_32_V2
Heltec Wireless Stick	esp32:esp32:heltec_wireless_stick
Heltec Wireless Stick Lite	esp32:esp32:heltec_wireless_stick_lite
Hornbill ESP32 Dev	esp32:esp32:hornbill32dev
Hornbill ESP32 Minima	esp32:esp32:hornbill32minima
IMBRIOS LOGSENS_V1P1	esp32:esp32:imbrios-logsens-v1p1
INEX OpenKB	esp32:esp32:OpenKB
ITEAD Sonoff	esp8266:esp8266:sonoff
IntoRobot Fig	esp32:esp32:intorobot-fig
Invent One	esp8266:esp8266:inventone
KITS ESP32 EDU	esp32:esp32:kits-edu
LOLIN D32	esp32:esp32:d32
LOLIN D32 PRO	esp32:esp32:d32_pro
LOLIN(WEMOS) D1 R2 & mini	esp8266:esp8266:d1_mini
LOLIN(WEMOS) D1 mini (clone)	esp8266:esp8266:d1_mini_clone
LOLIN(WEMOS) D1 mini Lite	esp8266:esp8266:d1_mini_lite
LOLIN(WEMOS) D1 mini Pro	esp8266:esp8266:d1_mini_pro
LOLIN(WeMos) D1 R1	esp8266:esp8266:d1
Labplus mPython	esp32:esp32:mPython
LamLoei AIoT DaaN Board	Maixduino:k210:aiotdaan
Lifely Agrumino Lemon v4	esp8266:esp8266:agruminolemon
LilyPad Arduino	arduino:avr:lilypad
LilyPad Arduino USB	arduino:avr:LilyPadUSB
Linino One	arduino:avr:one
LoPy	esp32:esp32:lopy
LoPy4	esp32:esp32:lopy4
LoRa boards	STMicroelectronics:stm32:LoRa
M5Stack-ATOM	esp32:esp32:m5stack-atom
M5Stack-Core-ESP32	esp32:esp32:m5stack-core-esp32
M5Stack-Core2	esp32:esp32:m5stack-core2
M5Stack-CoreInk	esp32:esp32:m5stack-coreink
M5Stack-FIRE	esp32:esp32:m5stack-fire
M5Stack-Timer-CAM	esp32:esp32:m5stack-timer-cam
M5Stick-C	esp32:esp32:m5stick-c
MGBOT IOTIK 32A	esp32:esp32:mgbot-iotik32a
MGBOT IOTIK 32B	esp32:esp32:mgbot-iotik32b
MH ET LIVE ESP32DevKIT	esp32:esp32:mhetesp32devkit
MH ET LIVE ESP32MiniKit	esp32:esp32:mhetesp32minikit
MagicBit	esp32:esp32:magicbit
Metro ESP-32	esp32:esp32:metro_esp-32
Microduino-CoreESP32	esp32:esp32:CoreESP32
Midatronics boards	STMicroelectronics:stm32:Midatronics
Nano32	esp32:esp32:nano32
Node32s	esp32:esp32:node32s
NodeMCU 0.9 (ESP-12 Module)	esp8266:esp8266:nodemcu
NodeMCU 1.0 (ESP-12E Module)	esp8266:esp8266:nodemcuv2
NodeMCU-32S	esp32:esp32:nodemcu-32s
Noduino Quantum	esp32:esp32:quantum
Nucleo-144	STMicroelectronics:stm32:Nucleo_144
Nucleo-32	STMicroelectronics:stm32:Nucleo_32
Nucleo-64	STMicroelectronics:stm32:Nucleo_64
ODROID ESP32	esp32:esp32:odroid_esp32
OLIMEX ESP32-DevKit-LiPo	esp32:esp32:esp32-DevKitLipo
OLIMEX ESP32-EVB	esp32:esp32:esp32-evb
OLIMEX ESP32-GATEWAY	esp32:esp32:esp32-gateway
OLIMEX ESP32-PoE	esp32:esp32:esp32-poe
OLIMEX ESP32-PoE-ISO	esp32:esp32:esp32-poe-iso
OROCA EduBot	esp32:esp32:oroca_edubot
Olimex MOD-WIFI-ESP8266(-DEV)	esp8266:esp8266:modwifi
Onehorse ESP32 Dev Module	esp32:esp32:onehorse32dev
Phoenix 1.0	esp8266:esp8266:phoenix_v1
Phoenix 2.0	esp8266:esp8266:phoenix_v2
Piranha ESP-32	esp32:esp32:piranha_esp-32

ProtoCentral HealthyPi 4	esp32:esp32:healthypi4
Pycom GPy	esp32:esp32:gpy
S.ODI Ultra v1	esp32:esp32:S_ODI_Ultra
STM32MP1 series coprocessor	STMicroelectronics:stm32:STM32MP1
Schirmilabs Eduino WiFi	esp8266:esp8266:eduinowifi
Seeed Wio Link	esp8266:esp8266:wiolink
Senses's WEIZEN	esp32:esp32:sensesiot_weizen
Silicognition wESP32	esp32:esp32:wesp32
Sipeed Maix Bit Board	Maixduino:k210:bit
Sipeed Maix Bit-Mic Board	Maixduino:k210:bitm
Sipeed Maix Go Board	Maixduino:k210:go
Sipeed Maix One Dock Board	Maixduino:k210:m1
Sipeed Maixduino Board	Maixduino:k210:mduino
SparkFun Blynk Board	esp8266:esp8266:blynk
SparkFun ESP32 Thing	esp32:esp32:esp32thing
SparkFun ESP32 Thing Plus	esp32:esp32:esp32thing_plus
SparkFun ESP8266 Thing	esp8266:esp8266:thing
SparkFun ESP8266 Thing Dev	esp8266:esp8266:thingdev
SparkFun LoRa Gateway 1-Channel	esp32:esp32:sparkfun_lora_gateway_1-channel
SweetPea ESP-210	esp8266:esp8266:esp210
T-Beam	esp32:esp32:t-beam
TTGO LoRa32-OLED V1	esp32:esp32:ttgo-lora32-v1
TTGO LoRa32-OLED v2.1.6	esp32:esp32:ttgo-lora32-v21new
TTGO T-Watch	esp32:esp32:twatch
TTGO T1	esp32:esp32:ttgo-t1
TTGO T7 V1.3 Mini32	esp32:esp32:ttgo-t7-v13-mini32
TTGO T7 V1.4 Mini32	esp32:esp32:ttgo-t7-v14-mini32
ThaiEasyElec's ESPino	esp8266:esp8266:espinotee
ThaiEasyElec's ESPino32	esp32:esp32:espino32
TinyPICO	esp32:esp32:tinypico
Turta IoT Node	esp32:esp32:turta_iot_node
VintLabs ESP32 Devkit	esp32:esp32:vintlabs-devkit-v1
WEMOS D1 MINI ESP32	esp32:esp32:d1_mini32
WEMOS LOLIN32	esp32:esp32:lolin32
WEMOS LOLIN32 Lite	esp32:esp32:lolin32-lite
WeMos WiFi&Bluetooth Battery	esp32:esp32:WeMosBat
WiFi Kit 8	esp8266:esp8266:wifi_kit_8
WiFiduino	esp8266:esp8266:wifiduino
WiFiduino32	esp32:esp32:wifiduino32
WiPy 3.0	esp32:esp32:wipy3
Widora AIR	esp32:esp32:widora-air
WifInfo	esp8266:esp8266:wifinfo
XinaBox CW01	esp8266:esp8266:cw01
XinaBox CW02	esp32:esp32:cw02
u-blox NINA-W10 series (ESP32)	esp32:esp32:nina_w10

1.22 Hints for bluepill STM32F103

Exemple compilation/upload:

```
arduino-cli compile -v -b STMicroelectronics:stm32:GenF1:pnum=BLUEPILL_↩
F103C8,usb=CDCgen -upload -p "." .
```

If you see the error: 'Error during Upload: Property 'upload.tool.serial' is undefined` please, Add the following line anywhere in the STM32 boards.txt file, otherwise: `GenF1.menu.upload_method.swdMethod.↩
upload.tool.default=stm32CubeProg`

Chapter 2

AtomicX

Version 1.2.1 release

What is AtomicX? AtomicX is a general purpose **cooperative** thread lib for embedded applications (single core or confined within other RTOS) that allows you partition your application "context" (since core execution) into several controlled context using cooperative thread. So far here nothing out of the ordinary, right? Lets think again:

2.1 Backlog and updates

2.1.1 Implementations from Work on progress

2.1.2 Version 1.2.1

- Adding Dynamic Nice, now it is possible to let the kernel set the best performance for your thread, for this `SetNice(*initial nice*)` and than `SetDynamicNice(true)` in the constructor of your thread. The kernel will be able to always adjust your thread for Best performance, but, it will leave no room for sleeps between threads, increasing power consumption, it is powerful but use it carefully.
- Added `YieldNow()` the higher priority context change, it will allow other threads to work, but will, also return faster than others
- `**smartSemaphore**`, Used to compliance with RII, once used in the thread context, it takes a semaphore to be initialized and expose the same methods, although it manages the local context, and ones it it gets out of context, due to leaving {} or a functions, for example the semaphore shared context is released if ever taken during the smartSemaphore instantiated object life cycle. The same is available for `mutex`, called `smartMutex`, follows the same principle.
- **IMPORTANT**, Introducing Semaphores, `atomicx::semaphore(<How many shared>)`, now you can use methods (`acquire()` or `acquire(timeout)`) and `release()` along with `GetCount`, `GetMaxAcquired`, `GetWaitCount` and static method `GetMax` to return the maximum shared you can use to instantiate. Examples for Arduino and PC where also introduced and fully tested.
- Introducing `atomicx::Timeout`, this will help tracking a timeout over time, using methods `IsTimedout` and `GetRemaining` and `GetDurationSince`. Special use case, if the timeout value is zero, Is↩ Timedout will always return false.
- **IMPORTANT NOTIFICATION** `atomicx::lock` has been renamed to `atomicx::mutex` for consistency, all methods are the same.
- **Improvement** Added a contructor for self-manager start to define a start size and increase pace. For example: a thread starts with 150 bytes and increase pace of 10, but used stack was 200, the kernel will do 200 + 10 (increase pace) to give it room to work. The default value is (1)
```
atomicx(size_t nStackSize, int nStackIncreasePace=1);
```

2.1.3 Version 1.2.0

- **INTRODUCING** Self managed stack, now it is possible to have self-managed stack memory for any threads, no need to define stack size... (although use it with care) just by not providing a stack memory, AtomicX will automatically switch the tread to self-managed, to do just use atomicx() default constructor instead.

  Notes:

  - It will only entries the stack enough to hold what is needed if the used stack is greater than the stack memory managed.
  - No decrease of the stack size was added to this release.
  - In case your thread is not able to resize the stack, if it needs more, StackOverflowHandle is called.

Examples:

- Ardunino/Simple
- avrAutoRobotController

Explicitly added the pc example shown here to to examples/pc as simple along with makefile for it. Also updated it to have an example of Self-managed stack memory as well.

2.1.4 Version 1.1.3

- Added a Thermal Camera Demo ported from CorePartition but now fully object oriented
- **POWERFUL**: Now `Wait`Notify` will accept a new parameter called subType, the name gives no clue but it is really powerfull it allows developer to create custom Types of notifications, that same strategy is used when syncNotify is called and get blocked until a timeout occur or a wait functions is used by another thread.

2.1.5 Version 1.1.2

- **Important* `Notify` was split into `Notify` and `SyncNotify` to avoid compilation ambiguity reported for some boards, all the examples have been migrated to use one of those accordingly and tested against all supported processors.

2.1.6 Version 1.1.1

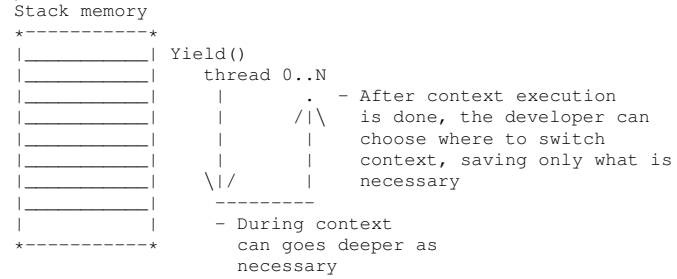
- *PLEASE NOTE* **No Spin Lock what so ever in this Kernel**, it is working fully based on Notification event along with message transportation.
- `NOTIFY` are now able to sync, if a atomicx_time is provided, Notify will wait for a specific signal to inform a `Wait` for refVar/Tag is up. This is a important feature toward using WAIT/Notify reliably, while your thread can do other stuffs on idle moment
- `avrRobotController` simulator for Arduino, is introduced, to show real inter process communication, it will open a terminal and both commands are available: `system` - To show Memory, Threads and motor status and `move <flot motor A> <flot motor B> <flot motor C>`

2.1.7 Version 1.1.0

- `finish()` method will be call every time `run()` is returned, this allow special cases like eventual threads to self-destroy itself, otherwise the object would be only a memory leak.... see examples on `main.cpp`
- `smartMutex` RAII compliance, allow mutex or shared mutex to be auto release on object destruction.
- **IMPORTANT** Now Notifications (Wait/Notify) can be timedout. if Tick based time is given, the waiting procedure will only stay blocked during it. (NO SPIN LOCK, REAL STATE BLOCK)
- **IMPORTANT** `LookForWaitings` block for timeout time will a wait for specific refVar/tag is available, otherwise timeout, can be used sync wait and notify availability
- **IMPORTANT** Now `Wait/Notify Tags`, used to give meaning/channel to a notification can be se to "all tags" if `Tag` is zero, otherwise it will respect refVar/Tag

2.1.8 Version 1.0.0

- **DOES NOT DISPLACE STACK, IT WILL STILL AVAILABLE FOR PROCESSING**, the *Stack Page* will only hold a backup of the most necessary information needed, allowing stacks in few bites most if the time. This implementation if highly suitable for Microcontrollers like ATINY85, for example, that only has 512 bites, and you can have 5 or more threads doing things for you, only backup the most important context information.
  - *IMPORTANT*: DO NOT USE CONTEXT MEMORY POINTER to exchange information to other threads, wait/notify and etc. All threads will use the *dafault stack memory* to execute, instead use Global variables, allocated memory or atomicx_smart_ptr objects.
- Since it implements Cooperative thread every execution will atomic between *atomicx* thrteads.
- AtomicX **DOES NOT DISPLACE STACK**, yes, it will use a novel technique that allow you to use full stack memory freely, and once done, just call `Yield()` to switch the context.
  1. Allow you to use all your stack during thread execution and only switch once back to an appropriate place



- Due to the **zero stack-displacement** technology, developers can ensure minimal stack memory page, allowing ultra sophisticated designes and execution stack diving and only backing up to the stack memory page what is necessary.
- Full feature for IPC (*Inter Process Communication*)
  - Thread safe Queues for data/object transporting.
  - EVERY Smart Lock can transport information (atomicx::message)
  - Message is composed by "size_t `atomix::message` and a "size_t tag" ∗ This novel concept of "tag"s for an atomicx::message gives the message meaning. ∗ Since <tt>atomicx::message</tt> uses <tt>size_t</tt> messages can also transport pointers ∗ Smart Locks can Lock and Shared Lock in the same object, making ∗ Full QUEUE capable to transport objects. ∗ Full feature for IPN (↩ Inter Process Notification) ∗ Thread can wait for an event to happen. ∗ On event notification a <tt>atomix::message</tt> can be sent/received ∗ A message broker based on observer pattern ∗ A thread can use <tt>WaitBroker Message</tt> to wait for any specifc topic asynchronously.

∗ Instead of having a <tt>Subcrib</tt> call, the developer will provide a <tt>IsSubscribed</tt> method that the kernel will use to determine if the object/thread is subscribed to a given topic. ∗ Broker uses <tt>atomicx::message</tt> to transport information. For inter process Object transport, please use atomicx::queue. ∗ ALL WAIT actions will block the thread, on kernel level (setting thread to a waiting state), until the notification occurs. Alternatively the notification can be transport a <tt>atomicx::message</tt> structure (tag/message) ∗ WAIT and NOTIFY (one or all) will use any pointer as the signal input, virtually any valid address pointer can be used. IMPORTANT: Unless you know what you are doing, do NOT use context pointer (execution stack memory), use a global or allocated memory instead (including <tt>atomicx::smart_prt</tt>) ∗ All Notifications or Publish functions will provide a Safe version, that different from the pure functions, will not trigger a context change and the function will only fully take effect onces the context is changed in the current thread where the interrupt request happened. ∗ IMPORTANT since all threads will be executed in the "_default_" stack memory, it will not be jailed in the stack size memory page, DO NOT USE STACK ADDRESS TO COMMUNICATE with another threads, use only global or alloced memory pointers to communicate ∗ IMPORTANT In order to operate with precision, specialise ticks by providing either <tt>atomicx_time Atomicx_GetTick (void)</tt> and <tt>void Atomicx_SleepTick(atomicx_time nSleep)</tt> to work within the timeframe (milleseconds, nanoseconds, seconds.. etc). Since AtomicX, also, provice, Sleep Tick functionality (to handle idle time), depending on the sleep time, to developer can redude the processor overall consuption to minimal whenever it is not necessary. ∗ Since it will be provided by the developer, it gives the possibility to use external clocks, hardware sleep or lower consumptions and fine tune power and resource usages. ∗ If not specialization is done, the source code will use a simple and non-deterministic loop cycle to count ticks. @code // // main.cpp // atomicx // // Created by GUSTAVO CAMPOS on 28/08/2021. // #include <unistd.h> #include <sys/time.h> #include <unistd.h> #include <cstring> #include <cstdint> #include <iostream> #include <setjmp.h> #include <string> #include "atomicx.hpp" using namespace thread; #ifdef FAKE_TIMER uint nCounter=0; #endif void ListAllThreads(); /∗ ∗ Define the default ticket granularity ∗ to milliseconds or round tick if -DFAKE_TICKER ∗ is provided on compilation ∗/ atomicx_time Atomicx_GetTick (void) { #ifndef FAKE_TIMER usleep (20000); struct timeval tp; gettimeofday (&tp, NULL); return (atomicx_time)tp.tv_sec ∗ 1000 + tp.tv_usec / 1000; #else nCounter++; return nCounter; #endif } /∗ ∗ Sleep for few Ticks, since the default ticket granularity ∗ is set to Milliseconds (if -DFAKE_TICKET provide will it will ∗ be context switch countings), the thread will sleep for ∗ the amount of time needed till next thread start. ∗/ void Atomicx_SleepTick(atomicx_time nSleep) { #ifndef FAKE_TIMER usleep ((useconds_t)nSleep ∗ 1000); #else while (nSleep); usleep(100); #endif } /∗ ∗ Object that implements thread with self-managed (dynamic) stack size ∗/ class SelfManagedThread : public atomicx { public: SelfManagedThread(atomicx_time nNice) : atomicx() { SetNice(nNice); } ~SelfManagedThread() { std::cout << "Deleting " << GetName() << ": " << (size_t) this << std::endl; } void run() noexcept override { size_t nCount=0; do { std::cout << __FUNCTION__ << ", Executing " << GetName() << ": " << (size_t) this << ", Counter: " << nCount << std::endl << std::flush; nCount++; } while (Yield()); } void StackOverflowHandler (void) noexcept override { std::cout << __FUNCTION__ << ":" << GetName() << "_" << (size_t) this << ": needed: " << GetUsedStackSize() << ", allocated: " << GetStackSize() << std::endl; } const char∗ GetName (void) override { return "Self-Managed Thread"; } }; /∗ ∗ Object that implements thread ∗/ class Thread : public atomicx { public: Thread(atomicx_time nNice) : atomicx(stack) { SetNice(nNice); } ~Thread() { std::cout << "Deleting " << GetName() << ": " << (size_t) this << std::endl; } void run() noexcept override { size_t nCount=0; do { std::cout << __FUNCTION__ << ", Executing " << GetName() << ": " << (size_t) this << ", Counter: " << nCount << std::endl << std::flush; nCount++; } while (Yield()); } void StackOverflowHandler (void) noexcept override { std::cout << __FUNCTION__ << ":" << GetName() << "_" << (size_t) this << ": needed: " << GetUsedStackSize() << ", allocated: " << GetStackSize() << std::endl; } const char∗ GetName (void) override { return "Thread"; } private: uint8_t stack[1024]=""; //Static initialization to avoid initialization order problem };

int main() { Thread t1(200); Thread t2(500);

SelfManagedThread st1(200);

// This must creates threads and destroy on leaving {} context { Thread t3_1(0); Thread t3_2(0); Thread t3_3(0);

// since those objects will be destroied here // they should never start and AtomicX should // transparently clean it from the execution list }

Thread t4(1000);

atomicx::Start(); }

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 thread Namespace Reference

Data Structures

- class atomicx

Chapter 8

Data Structure Documentation

8.1 thread::atomicx::aiterator Class Reference

`#include <atomicx.hpp>`

Public Member Functions

- aiterator ()=delete
- aiterator (atomicx *ptr)

  atomicx based constructor
- atomicx & operator* () const
- atomicx * operator-> ()
- aiterator & operator++ ()

Friends

- bool operator== (const aiterator &a, const aiterator &b)
- bool operator!= (const aiterator &a, const aiterator &b)

8.1.1 Detailed Description

8.1.1.1 ITERATOR FOR THREAD LISTING

8.1.2 Constructor & Destructor Documentation

8.1.2.1 aiterator() [1/2]

```
thread::atomicx::aiterator::aiterator ( )  [delete]
```

8.1.2.2 aiterator() [2/2]

```
thread::atomicx::aiterator::aiterator (
            atomicx * ptr )
```

atomicx based constructor

Parameters

ptr	atomicx pointer to iterate

8.1.3 Member Function Documentation

8.1.3.1 operator∗()

```
atomicx & thread::atomicx::aiterator::operator* ( ) const
```

8.1.3.2 operator++()

```
atomicx::aiterator & thread::atomicx::aiterator::operator++ ( )
```

8.1.3.3 operator->()

```
atomicx * thread::atomicx::aiterator::operator-> ( )
```

8.1.4 Friends And Related Function Documentation

8.1.4.1 operator"!=

```
bool operator!= (
            const aiterator & a,
            const aiterator & b )  [friend]
```

8.1.4.2 operator==

```
bool operator== (
            const aiterator & a,
            const aiterator & b )  [friend]
```

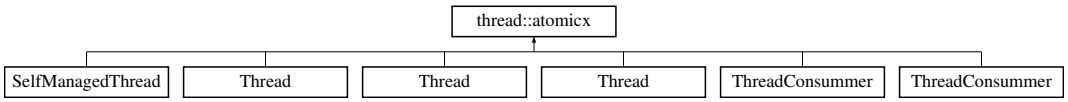
The documentation for this class was generated from the following files:

- atomicx/atomicx.hpp
- atomicx/atomicx.cpp

8.2 thread::atomicx Class Reference

`#include <atomicx.hpp>`

Inheritance diagram for thread::atomicx:



Data Structures

- class aiterator
- struct Message
- class mutex
- class queue
- class semaphore
- class smart_ptr
- class smartMutex

  RII compliance lock/shared lock to auto unlock on destruction.
- class smartSemaphore
- class Timeout

  Timeout Check object.

Public Types

- enum class aTypes : uint8_t {
  start =1 , running =5 , now =6 , stop =10 ,
  lock =50 , wait =55 , subscription =60 , sleep =100 ,
  stackOverflow =255 }
- enum class aSubTypes : uint8_t {
  error =10 , ok , look , wait ,
  timeout }
- enum class NotifyType : uint8_t { one = 0 , all = 1 }

Public Member Functions

- aiterator begin (void)

  Get the beggining of the list.
- aiterator end (void)

  Get the end of the list.
- virtual ~atomicx (void)

  virtual destructor of the atomicx
- size_t GetID (void)

  Get the current thread ID.
- size_t GetStackSize (void)

  Get the Max Stack Size for the thread.
- atomicx_time GetNice (void)

  Get the Nice the current thread.
- size_t GetUsedStackSize (void)

  Get the Used Stack Size for the thread since the last context change cycle.
- atomicx_time GetCurrentTick (void)

  Get the Current Tick using the ported tick granularity function.

Static Public Member Functions

- static atomicx * GetCurrent ()

  Get the Current thread in execution.
- static bool Start (void)

  Once it is call the process blocks execution and start all threads.

If GetName was not overloaded by the derived thread implementation

Get the Name object

Returns

const char* name in plain c string

a standard name will be returned.

- virtual const char * GetName (void)
- atomicx_time GetTargetTime (void)

  Get next moment in ported tick granularity the thread will be due to return.
- int GetStatus (void)

  Get the current thread status.
- int GetSubStatus (void)

  Get the current thread sub status.
- size_t GetReferenceLock (void)

  Get the Reference Lock last used to lock the thread.
- size_t GetTagLock (void)

  Get the last tag message posted.
- void SetNice (atomicx_time nice)

  Set the Nice of the thread.
- template<typename T , size_t N>
  atomicx (T(&stack)[N])

  Construct a new atomicx thread.
- atomicx (size_t nStackSize=0, int nStackIncreasePace=1)

  Construct a new atomicx object and set initial auto stack and increase pace.
- virtual void run (void) noexcept=0

  The pure virtual function that runs the thread loop.
- virtual void StackOverflowHandler (void) noexcept=0

  Handles the StackOverflow of the current thread.
- virtual void finish () noexcept

  Called right after run returns, can be used to self-destroy the object and other maintenance actions.
- bool IsStackSelfManaged (void)

  Return if the current thread's stack memory is automatic.
- bool Yield (atomicx_time nSleep=ATOMICX_TIME_MAX)

  Foce the context change explicitly.
- atomicx_time GetLastUserExecTime ()

  Get the Last Execution of User Code.
- size_t GetStackIncreasePace (void)

  Get the Stack Increase Pace value.
- void YieldNow (void)

  Trigger a high priority NOW, caution it will always execute before normal yield.
- void SetDynamicNice (bool status)

  Set the Dynamic Nice on and off.
- bool IsDynamicNiceOn ()

  Get Dynamic Nice status.
- uint32_t GetTopicID (const char *pszTopic, size_t nKeyLenght)

  calculate the Topic ID for a given topic text
- template<typename T >
  bool LookForWaitings (T &refVar, size_t nTag, size_t hasAtleast, atomicx_time waitFor)

  Sync with thread call for a wait (refVar,nTag)

- template<typename T >
  bool LookForWaitings (T &refVar, size_t nTag, atomicx_time waitFor)

  Sync with thread call for a wait (refVar,nTag)
- template<typename T >
  bool IsWaiting (T &refVar, size_t nTag=0, size_t hasAtleast=1, aSubTypes asubType=aSubTypes::wait)

  Check if there are waiting threads for a given reference pointer and tag value.
- template<typename T >
  size_t HasWaitings (T &refVar, size_t nTag=0, aSubTypes asubType=aSubTypes::wait)

  Report how much waiting threads for a given reference pointer and tag value are there.
- template<typename T >
  bool Wait (size_t &nMessage, T &refVar, size_t nTag=0, atomicx_time waitFor=0, aSubTypes asub↩
  Type=aSubTypes::wait)

  Blocks/Waits a notification along with a message and tag from a specific reference pointer.
- template<typename T >
  bool Wait (T &refVar, size_t nTag=0, atomicx_time waitFor=0, aSubTypes asubType=aSubTypes::wait)

  Blocks/Waits a notification along with a tag from a specific reference pointer.
- template<typename T >
  size_t SafeNotify (size_t &nMessage, T &refVar, size_t nTag=0, NotifyType notifyAll=NotifyType::one, aSubTypes asubType=aSubTypes::wait)

  Safely notify all Waits from a specific reference pointer along with a message without triggering context change.
- template<typename T >
  size_t Notify (size_t &nMessage, T &refVar, size_t nTag=0, NotifyType notifyAll=NotifyType::one, aSubTypes asubType=aSubTypes::wait)

  Notify all Waits from a specific reference pointer along with a message and trigger context change if at least one wait thread got notified.
- template<typename T >
  size_t Notify (size_t &&nMessage, T &refVar, size_t nTag=0, NotifyType notifyAll=NotifyType::one, aSubTypes asubType=aSubTypes::wait)
- template<typename T >
  size_t SyncNotify (size_t &nMessage, T &refVar, size_t nTag=0, atomicx_time waitForWaitings=0, NotifyType notifyAll=NotifyType::one, aSubTypes asubType=aSubTypes::wait)

  SYNC Waits for at least one Wait call for a given reference pointer along with a message and trigger context change.
- template<typename T >
  size_t SyncNotify (size_t &&nMessage, T &refVar, size_t nTag=0, atomicx_time waitForWaitings=0, NotifyType notifyAll=NotifyType::one, aSubTypes asubType=aSubTypes::wait)
- template<typename T >
  size_t SafeNotify (T &refVar, size_t nTag=0, NotifyType notifyAll=NotifyType::one, aSubTypes asub↩
  Type=aSubTypes::wait)

  Safely notify all Waits from a specific reference pointer without triggering context change.
- template<typename T >
  size_t SyncNotify (T &refVar, size_t nTag, atomicx_time waitForWaitings=0, NotifyType notify↩
  All=NotifyType::one, aSubTypes asubType=aSubTypes::wait)

  SYNC Waits for at least one Wait call for a given reference pointer and trigger context change.
- template<typename T >
  size_t Notify (T &refVar, size_t nTag=0, NotifyType notifyAll=NotifyType::one, aSubTypes asub↩
  Type=aSubTypes::wait)

  Notify all Waits from a specific reference pointer and trigger context change if at least one wait thread got notified.
- bool WaitBrokerMessage (const char *pszKey, size_t nKeyLenght, Message &message)

  Block and wait for message from a specific topic string.
- bool WaitBrokerMessage (const char *pszKey, size_t nKeyLenght)

  Block and wait for a notification from a specific topic string.
- bool Publish (const char *pszKey, size_t nKeyLenght, const Message message)

  Publish a message for a specific topic string and trigger a context change if any delivered.
- bool SafePublish (const char *pszKey, size_t nKeyLenght, const Message message)

Safely Publish a message for a specific topic string DO NOT trigger a context change if any delivered.

- bool Publish (const char *pszKey, size_t nKeyLenght)

  Publish a notification for a specific topic string and trigger a context change if any delivered.
- bool SafePublish (const char *pszKey, size_t nKeyLenght)

  Safely Publish a notification for a specific topic string DO NOT trigger a context change if any delivered.
- bool HasSubscriptions (const char *pszTopic, size_t nKeyLenght)

  Check if there is subscryption for a specific Topic String.
- bool HasSubscriptions (uint32_t nKeyID)

  Check if there is subscryption for a specific Topic ID.
- virtual bool BrokerHandler (const char *pszKey, size_t nKeyLenght, Message &message)

  Default broker handler for a subscribed message.
- virtual bool IsSubscribed (const char *pszKey, size_t nKeyLenght)

  Specialize and gives power to decide if a topic is subscrybed on not.
- void SetStackIncreasePace (size_t nIncreasePace)

  Set the Stack Increase Pace object.

8.2.1 Member Enumeration Documentation

8.2.1.1 aSubTypes

```
enum class thread::atomicx::aSubTypes : uint8_t [strong]
```

Enumerator

error	
ok	
look	
wait	
timeout	

8.2.1.2 aTypes

```
enum class thread::atomicx::aTypes : uint8_t [strong]
```

8.2.1.3 STATE MACHINE TYPES

Enumerator

start	
running	
now	
stop	
lock	
wait	
subscription	
sleep	
stackOverflow	

8.2.1.4 NotifyType

```
enum class thread::atomicx::NotifyType :  uint8_t  [strong]
```

Enumerator

one	
all	

8.2.2 Constructor & Destructor Documentation

8.2.2.1 ~atomicx()

```
thread::atomicx::~atomicx (
            void  )  [virtual]
```

virtual destructor of the atomicx
PUBLIC OBJECT METHOS

8.2.2.2 atomicx() [1/2]

```
template<typename T , size_t N>
thread::atomicx::atomicx (
            T(&) stack[N] )  [inline]
```

Construct a new atomicx thread.

Template Parameters

T	Stack memory page type
N	Stack memory page size

8.2.2.3 atomicx() [2/2]

```
thread::atomicx::atomicx (
            size_t nStackSize = 0,
            int nStackIncreasePace = 1 )
```

Construct a new atomicx object and set initial auto stack and increase pace.

Parameters

nStackSize	Initial Size of the stack
nStackIncreasePace	defalt=1, The increase pace on each resize

8.2.3 Member Function Documentation

8.2.3.1 begin()

```
atomicx::aiterator thread::atomicx::begin (
            void  )
```

Get the beggining of the list.

Returns

aiterator

8.2.3.2 BrokerHandler()

```
virtual bool thread::atomicx::BrokerHandler (
            const char * pszKey,
            size_t nKeyLenght,
            Message & message )  [inline], [protected], [virtual]
```

Default broker handler for a subscribed message.

Parameters

pszKey	The Topic C string
nKeyLenght	The Topic C string size in bytes
message	The atomicx::message payload received

Returns

true signify it was correctly processed

Note

Can be overloaded by the derived by the derived thread implementation and specialized, otherwise a empty function will be called instead

8.2.3.3 end()

```
atomicx::aiterator thread::atomicx::end (
            void  )
```

Get the end of the list.

Returns

aiterator

8.2.3.4 finish()

```
virtual void thread::atomicx::finish ( )  [inline], [virtual], [noexcept]
```

Called right after run returns, can be used to self-destroy the object and other maintenance actions.

Note

if not implemented a default "empty" call is used instead

8.2.3.5 GetCurrent()

```
atomicx * thread::atomicx::GetCurrent ( )  [static]
```

Get the Current thread in execution.

Returns

atomicx* thread

8.2.3.6 GetCurrentTick()

```
atomicx_time thread::atomicx::GetCurrentTick (
            void  )
```

Get the Current Tick using the ported tick granularity function.

Returns

atomicx_time based on the ported tick granularity

8.2.3.7 GetID()

```
size_t thread::atomicx::GetID (
            void  )
```

Get the current thread ID.

Returns

size_t Thread ID number

8.2.3.8 GetLastUserExecTime()

```
atomicx_time thread::atomicx::GetLastUserExecTime ( )
```

Get the Last Execution of User Code.

Returns

atomicx_time

8.2.3.9 GetName()

```
const char * thread::atomicx::GetName (
            void  )  [virtual]
```

Reimplemented in ThreadConsummer, Thread, SelfManagedThread, Thread, ThreadConsummer, and Thread.

8.2.3.10 GetNice()

```
atomicx_time thread::atomicx::GetNice (
            void  )
```

Get the Nice the current thread.

Returns

atomicx_time the number representing the nice and based on the ported tick granularity.

8.2.3.11 GetReferenceLock()

```
size_t thread::atomicx::GetReferenceLock (
            void  )
```

Get the Reference Lock last used to lock the thread.

Returns

size_t the lock_id (used my wait)

8.2.3.12 GetStackIncreasePace()

```
size_t thread::atomicx::GetStackIncreasePace (
            void  )
```

Get the Stack Increase Pace value.

8.2.3.13 GetStackSize()

```
size_t thread::atomicx::GetStackSize (
            void  )
```

Get the Max Stack Size for the thread.

Returns

size_t size in bytes

8.2.3.14 GetStatus()

```
int thread::atomicx::GetStatus (
            void  )
```

Get the current thread status.

Returns

int use atomicx::aTypes

8.2.3.15 GetSubStatus()

```
int thread::atomicx::GetSubStatus (
            void  )
```

Get the current thread sub status.

Returns

int use atomicx::aTypes

8.2.3.16 GetTagLock()

```
size_t thread::atomicx::GetTagLock (
            void  )
```

Get the last tag message posted.

Returns

size_t atomicx::message::tag value

8.2.3.17 GetTargetTime()

```
atomicx_time thread::atomicx::GetTargetTime (
            void  )
```

Get next moment in ported tick granularity the thread will be due to return.

Returns

atomicx_time based on the ported tick granularity

8.2.3.18 GetTopicID()

```
uint32_t thread::atomicx::GetTopicID (
            const char * pszTopic,
            size_t nKeyLenght )  [protected]
```

calculate the Topic ID for a given topic text

Parameters

pszTopic	Topic Text in C string
nKeyLenght	Size, in bytes + zero terminated char

Returns

uint32_t The calculated topic ID

8.2.3.19 GetUsedStackSize()

```
size_t thread::atomicx::GetUsedStackSize (
            void  )
```

Get the Used Stack Size for the thread since the last context change cycle.

Returns

size_t size in bytes

8.2.3.20 HasSubscriptions() [1/2]

```
bool thread::atomicx::HasSubscriptions (
            const char * pszTopic,
            size_t nKeyLenght )  [protected]
```

Check if there is subscryption for a specific Topic String.

Parameters

pszTopic	The Topic string in C string
nKeyLenght	The Topic C string length in bytes

Returns

true if any substriction is found, otherwise false

8.2.3.21 HasSubscriptions() [2/2]

```
bool thread::atomicx::HasSubscriptions (
            uint32_t nKeyID )  [protected]
```

Check if there is subscryption for a specific Topic ID.

Parameters

nKeyID	The Topic ID uint32_t

Returns

true if any substriction is found, otherwise false

8.2.3.22 HasWaitings()

```
template<typename T >
size_t thread::atomicx::HasWaitings (
            T & refVar,
            size_t nTag = 0,
            aSubTypes asubType = aSubTypes::wait )  [inline], [protected]
```

Report how much waiting threads for a given reference pointer and tag value are there.

Template Parameters

T	Type of the reference pointer

Parameters

refVar	The reference pointer used a a notifier
nTag	The size↩ _t tag that will give meaning to the notification, if nTag == 0 mean all bTag for the refVar
asubType	Type of the notification, only use it if you know what you are doing, it creates a different type of wait/notify, deafault == a↩ Sub↩ Type↩ ::wait

Returns

true

Note

This is a powerful tool since it create layers of waiting within the same reference pointer

8.2.3.23 IsDynamicNiceOn()

```
bool thread::atomicx::IsDynamicNiceOn ( )
```

Get Dynamic Nice status.

Returns

true if dynamic nice is on otherwise off

8.2.3.24 IsStackSelfManaged()

```
bool thread::atomicx::IsStackSelfManaged (
            void  )
```

Return if the current thread's stack memory is automatic.

8.2.3.25 IsSubscribed()

```
virtual bool thread::atomicx::IsSubscribed (
            const char * pszKey,
            size_t nKeyLenght )  [inline], [protected], [virtual]
```

Specialize and gives power to decide if a topic is subscrybed on not.

Parameters

pszKey	The Topic C String
nKeyLenght	The Topic C String size in bytes

Returns

true if the given topic was subscribed, otherwise false.

8.2.3.26 IsWaiting()

```
template<typename T >
bool thread::atomicx::IsWaiting (
            T & refVar,
            size_t nTag = 0,
            size_t hasAtleast = 1,
            aSubTypes asubType = aSubTypes::wait )  [inline], [protected]
```

Check if there are waiting threads for a given reference pointer and tag value.

Template Parameters

T	Type of the reference pointer

Parameters

refVar	The reference pointer used a a notifier
nTag	The size↩ _t tag that will give meaning to the notification, if nTag == 0 mean all bTag for the refVar
asubType	Type of the notification, only use it if you know what you are doing, it creates a different type of wait/notify, deafault == a↩ Sub↩ Type↩ ::wait

Returns

true

Note

This is a powerful tool since it create layers of waiting within the same reference pointer

8.2.3.27 LookForWaitings() [1/2]

```
template<typename T >
bool thread::atomicx::LookForWaitings (
            T & refVar,
            size_t nTag,
            atomicx_time waitFor )  [inline], [protected]
```

Sync with thread call for a wait (refVar,nTag)

Template Parameters

T	Type of the reference pointer

Parameters

refVar	The reference pointer
nTag	The notification meaning, if nTag == 0 means wait all refVar regardless
waitFor	default=0, if 0 wait indefinitely, otherwise wait for custom tick granularity times

Returns

true There is thread waiting for the given refVar/nTag

8.2.3.28 LookForWaitings() [2/2]

```
template<typename T >
bool thread::atomicx::LookForWaitings (
            T & refVar,
            size_t nTag,
            size_t hasAtleast,
            atomicx_time waitFor )  [inline], [protected]
```

Sync with thread call for a wait (refVar,nTag)

8.2.3.29 SMART WAIT/NOTIFY IMPLEMENTATION

Template Parameters

T	Type of the reference pointer

Parameters

refVar	The reference pointer
nTag	The notification meaning, if nTag == 0 means wait all refVar regardless
waitFor	default=0, if 0 wait indefinitely, otherwise wait for custom tick granularity times

Parameters

hasAtleast	define how minimal Wait calls to report true

Returns

true There is thread waiting for the given refVar/nTag

8.2.3.30 Notify() [1/3]

```
template<typename T >
size_t thread::atomicx::Notify (
            size_t && nMessage,
            T & refVar,
            size_t nTag = 0,
            NotifyType notifyAll = NotifyType::one,
            aSubTypes asubType = aSubTypes::wait )  [inline], [protected]
```

8.2.3.31 Notify() [2/3]

```
template<typename T >
size_t thread::atomicx::Notify (
            size_t & nMessage,
            T & refVar,
            size_t nTag = 0,
            NotifyType notifyAll = NotifyType::one,
            aSubTypes asubType = aSubTypes::wait )  [inline], [protected]
```

Notify all Waits from a specific reference pointer along with a message and trigger context change if at least one wait thread got notified.

Template Parameters

T	Type of the reference pointer

Parameters

nMessage	The size_t message to be sent

Parameters

refVar	The reference pointer used a a notifier
nTag	The size_t tag that will give meaning to the notification, if nTag == 0 means notify all refVar regardless
notifyAll	default = false, and only the fist available refVar Waiting thread will be notified, if true all available refVar waiting thread will be notified.

Parameters

asubType	Type of the notification, only use it if you know what you are doing, it creates a different type of wait/notify, deafault == aSubType::wait

Returns

true if at least one got notified, otherwise false.

8.2.3.32 Notify() [3/3]

```
template<typename T >
size_t thread::atomicx::Notify (
            T & refVar,
            size_t nTag = 0,
            NotifyType notifyAll = NotifyType::one,
            aSubTypes asubType = aSubTypes::wait )  [inline], [protected]
```

Notify all Waits from a specific reference pointer and trigger context change if at least one wait thread got notified.

Template Parameters

T	Type of the reference pointer

Parameters

refVar	The reference pointer used a a notifier

Parameters

nTag	The size↩ _t tag that will give meaning to the notification, if nTag == 0 means notify all refVar regardless
notifyAll	default = false, and only the fist available refVar Waiting thread will be notified, if true all available refVar waiting thread will be notified.

Parameters

asubType	Type of the notification, only use it if you know what you are doing, it creates a different type of wait/notify, deafault == aSubType::wait

Returns

true if at least one got notified, otherwise false.

8.2.3.33 Publish() [1/2]

```
bool thread::atomicx::Publish (
            const char * pszKey,
            size_t nKeyLenght ) [protected]
```

Publish a notification for a specific topic string and trigger a context change if any delivered.

Parameters

pszKey	The Topic string
nKeyLenght	The size of the topic string in bytes

Returns

true if at least one thread has received a message

8.2.3.34 Publish() [2/2]

```
bool thread::atomicx::Publish (
            const char * pszKey,
            size_t nKeyLenght,
            const Message message ) [protected]
```

Publish a message for a specific topic string and trigger a context change if any delivered.

Parameters

pszKey	The Topic string
nKeyLenght	The size of the topic string in bytes
message	the atomicx::message structure with message and tag

Returns

true if at least one thread has received a message

8.2.3.35 run()

```
virtual void thread::atomicx::run (
            void ) [pure virtual], [noexcept]
```

The pure virtual function that runs the thread loop.

Note

REQUIRED implementation and once it returns it will execute finish method

Implemented in Thread, SelfManagedThread, Thread, Thread, ThreadConsummer, and ThreadConsummer.

8.2.3.36 SafeNotify() [1/2]

```
template<typename T >
size_t thread::atomicx::SafeNotify (
            size_t & nMessage,
            T & refVar,
            size_t nTag = 0,
            NotifyType notifyAll = NotifyType::one,
            aSubTypes asubType = aSubTypes::wait ) [inline], [protected]
```

Safely notify all Waits from a specific reference pointer along with a message without triggering context change.

Template Parameters

T	Type of the reference pointer

Parameters

nMessage	The size_t message to be sent
refVar	The reference pointer used a a notifier
nTag	The size_t tag that will give meaning to the notification, if nTag == 0 means notify all refVar regardless

Parameters

notifyAll	default = false, and only the fist available refVar Waiting thread will be notified, if true all available refVar waiting thread will be notified.
asubType	Type of the notification, only use it if you know what you are doing, it creates a different type of wait/notify, deafault == a↩ Sub↩ Type↩ ::wait

Returns

true if at least one got notified, otherwise false.

8.2.3.37 SafeNotify() [2/2]

```
template<typename T >
size_t thread::atomicx::SafeNotify (
```

```
T & refVar,
size_t nTag = 0,
NotifyType notifyAll = NotifyType::one,
aSubTypes asubType = aSubTypes::wait )  [inline], [protected]
```

Safely notify all Waits from a specific reference pointer without triggering context change.

Template Parameters

T	Type of the reference pointer

Parameters

refVar	The reference pointer used a a notifier
nTag	The size_t tag that will give meaning to the notification, if nTag == 0 means notify all refVar regardless

Parameters

notifyAll	default = false, and only the fist available refVar Waiting thread will be notified, if true all available refVar waiting thread will be notified.
asubType	Type of the notification, only use it if you know what you are doing, it creates a different type of wait/notify, deafault == a↩ Sub↩ Type↩ ::wait

Returns

true if at least one got notified, otherwise false.

8.2.3.38 SafePublish() [1/2]

```
bool thread::atomicx::SafePublish (
```

```
            const char * pszKey,
            size_t nKeyLenght )  [protected]
```

Safely Publish a notification for a specific topic string DO NOT trigger a context change if any delivered.

Parameters

pszKey	The Topic string
nKeyLenght	The size of the topic string in bytes

Returns

true if at least one thread has received a message

Note

Ideal for been used with interrupt request

8.2.3.39 SafePublish() [2/2]

```
bool thread::atomicx::SafePublish (
            const char * pszKey,
            size_t nKeyLenght,
            const Message message )  [protected]
```

Safely Publish a message for a specific topic string DO NOT trigger a context change if any delivered.

Parameters

pszKey	The Topic string
nKeyLenght	The size of the topic string in bytes
message	the atomicx::message structure with message and tag

Returns

true if at least one thread has received a message

Note

Ideal for been used with interrupt request

8.2.3.40 SetDynamicNice()

```
void thread::atomicx::SetDynamicNice (
            bool status )
```

Set the Dynamic Nice on and off.

Parameters

status	True for on otherwsize off

8.2.3.41 SetNice()

```
void thread::atomicx::SetNice (
            atomicx_time nice )
```

Set the Nice of the thread.

Parameters

nice	in atomicx↩ _time refer- ence based on the ported tick granu- larity

8.2.3.42 SetStackIncreasePace()

```
void thread::atomicx::SetStackIncreasePace (
            size_t nIncreasePace )  [protected]
```

Set the Stack Increase Pace object.

Parameters

nIncreasePace	The new stack increase pace value

8.2.3.43 StackOverflowHandler()

```
virtual void thread::atomicx::StackOverflowHandler (
            void  )  [pure virtual], [noexcept]
```

Handles the StackOverflow of the current thread.

Note

REQUIRED

Implemented in ThreadConsummer, Thread, SelfManagedThread, Thread, ThreadConsummer, and Thread.

8.2.3.44 Start()

```
bool thread::atomicx::Start (
            void  )  [static]
```

Once it is call the process blocks execution and start all threads.

Returns

false if it was destried by dead lock (all threads locked)

8.2.3.45 SyncNotify() [1/3]

```
template<typename T >
size_t thread::atomicx::SyncNotify (
            size_t && nMessage,
            T & refVar,
            size_t nTag = 0,
            atomicx_time waitForWaitings = 0,
            NotifyType notifyAll = NotifyType::one,
            aSubTypes asubType = aSubTypes::wait )  [inline], [protected]
```

8.2.3.46 SyncNotify() [2/3]

```
template<typename T >
size_t thread::atomicx::SyncNotify (
            size_t & nMessage,
            T & refVar,
            size_t nTag = 0,
            atomicx_time waitForWaitings = 0,
            NotifyType notifyAll = NotifyType::one,
            aSubTypes asubType = aSubTypes::wait )  [inline], [protected]
```

SYNC Waits for at least one Wait call for a given reference pointer along with a message and trigger context change.

Template Parameters

T	Type of the reference pointer

Parameters

nMessage	The size_t message to be sent
refVar	The reference pointer used a a notifier
nTag	The size_t tag that will give meaning to the notification, if nTag == 0 means notify all refVar regardless

Parameters

waitForWaitings	default=0 (waiting for Waiting calls) othersize wait for Wait commands compatible with the paramenters (Sync call).
notifyAll	default = false, and only the fist available refVar Waiting thread will be notified, if true all available refVar waiting thread will be notified.

Parameters

asubType	Type of the notification, only use it if you know what you are doing, it creates a different type of wait/notify, deafault == aSubType::wait

Returns

true if at least one got notified, otherwise false.

8.2.3.47 SyncNotify() [3/3]

```
template<typename T >
size_t thread::atomicx::SyncNotify (
            T & refVar,
            size_t nTag,
            atomicx_time waitForWaitings = 0,
            NotifyType notifyAll = NotifyType::one,
            aSubTypes asubType = aSubTypes::wait )  [inline], [protected]
```

SYNC Waits for at least one Wait call for a given reference pointer and trigger context change.

Template Parameters

T	Type of the reference pointer

Parameters

refVar	The reference pointer used a a notifier

Parameters

nTag	The size_t tag that will give meaning to the notification, if nTag == 0 means notify all refVar regardless
waitForWaitings	default=0 (waiting for Waiting calls) othersize wait for Wait commands compatible with the paramenters (Sync call).

Parameters

notifyAll	default = false, and only the fist available refVar Waiting thread will be notified, if true all available refVar waiting thread will be notified.
asubType	Type of the notification, only use it if you know what you are doing, it creates a different type of wait/notify, deafault == a↩ Sub↩ Type↩ ::wait

Returns

true if at least one got notified, otherwise false.

8.2.3.48 Wait() [1/2]

```
template<typename T >
bool thread::atomicx::Wait (
```

```
size_t & nMessage,
T & refVar,
size_t nTag = 0,
atomicx_time waitFor = 0,
aSubTypes asubType = aSubTypes::wait )  [inline], [protected]
```

Blocks/Waits a notification along with a message and tag from a specific reference pointer.

Template Parameters

T	Type of the reference pointer

Parameters

nMessage	the size_t message to be received
refVar	the reference pointer used as a notifier
nTag	the size_t tag that will give meaning to the the message, if nTag == 0 means wait all refVar regardless

Parameters

waitFor	default==0 (undefinitly), How log to wait for a notification based on atomicx_time
asubType	Type of the notification, only use it if you know what you are doing, it creates a different type of wait/notify, deafault == aSubType::wait

Returns

true if it was successfully received.

8.2.3.49 Wait() [2/2]

```
template<typename T >
bool thread::atomicx::Wait (
            T & refVar,
            size_t nTag = 0,
            atomicx_time waitFor = 0,
            aSubTypes asubType = aSubTypes::wait )  [inline], [protected]
```

Blocks/Waits a notification along with a tag from a specific reference pointer.

Template Parameters

T	Type of the reference pointer

Parameters

refVar	the reference pointer used as a notifier
nTag	the size_t tag that will give meaning to the the message, if nTag == 0 means wait all refVar regardless
waitFor	default==0 (undefinitly), How log to wait for a notification based on atomicx_time

Parameters

asubType	Type of the notification, only use it if you know what you are doing, it creates a different type of wait/notify, deafault == a↩ Sub↩ Type↩ ::wait

Returns

true if it was successfully received.

8.2.3.50 WaitBrokerMessage() [1/2]

```
bool thread::atomicx::WaitBrokerMessage (
            const char * pszKey,
            size_t nKeyLenght ) [protected]
```

Block and wait for a notification from a specific topic string.

Parameters

pszKey	The Topic string
nKeyLenght	The size of the topic string in bytes

Returns

true if it was successfully received, otherwise false

8.2.3.51 WaitBrokerMessage() [2/2]

```
bool thread::atomicx::WaitBrokerMessage (
            const char * pszKey,
            size_t nKeyLenght,
            Message & message ) [protected]
```

Block and wait for message from a specific topic string.

8.2.3.52 SMART BROKER IMPLEMENTATION

Parameters

pszKey	The Topic string
nKeyLenght	The size of the topic string in bytes
message	the atomicx::message structure with message and tag

Returns

true if it was successfully received, otherwise false

8.2.3.53 Yield()

```
bool thread::atomicx::Yield (
            atomicx_time nSleep = ATOMICX_TIME_MAX )
```

Foce the context change explicitly.

Parameters

nSleep	default is ATOMICX_TIME_MAX, otherwise it will override the nice and sleep for n custom tick granularity

Returns

true if the context came back correctly, otherwise false

8.2.3.54 YieldNow()

```
void thread::atomicx::YieldNow (
            void  )
```

Trigger a high priority NOW, caution it will always execute before normal yield.

8.2.4 Field Documentation

8.2.4.1 autoStack

```
bool thread::atomicx::autoStack
```

8.2.4.2 dynamicNice

```
bool thread::atomicx::dynamicNice
```

The documentation for this class was generated from the following files:

- atomicx/atomicx.hpp
- atomicx/atomicx.cpp

8.3 thread::atomicx::Message Struct Reference

```
#include <atomicx.hpp>
```

Data Fields

- size_t tag
- size_t message

8.3.1 Detailed Description

PROTECTED METHODS, THOSE WILL BE ONLY ACCESSIBLE BY THE THREAD ITSELF

8.3.2 Field Documentation

8.3.2.1 message

```
size_t thread::atomicx::Message::message
```

8.3.2.2 tag

```
size_t thread::atomicx::Message::tag
```

The documentation for this struct was generated from the following file:

- atomicx/atomicx.hpp

8.4 thread::atomicx::mutex Class Reference

```
#include <atomicx.hpp>
```

Public Member Functions

- void Lock ()

  Exclusive/binary lock the smart lock.
- void Unlock ()

  Release the exclusive lock.
- void SharedLock ()

  Shared Lock for the smart Lock.
- void SharedUnlock ()

  Release the current shared lock.
- size_t IsShared ()

  Check how many shared locks are accquired.
- bool IsLocked ()

  Check if a exclusive lock has been already accquired.

8.4.1 Detailed Description

8.4.1.1 SMART LOCK IMPLEMENTATION

8.4.2 Member Function Documentation

8.4.2.1 IsLocked()

`bool thread::atomicx::mutex::IsLocked ( )`
Check if a exclusive lock has been already accquired.

Returns

true if yes, otherwise false

8.4.2.2 IsShared()

`size_t thread::atomicx::mutex::IsShared ( )`
Check how many shared locks are accquired.

Returns

size_t Number of threads holding shared locks

8.4.2.3 Lock()

`void thread::atomicx::mutex::Lock ( )`
Exclusive/binary lock the smart lock.

Note

Once Lock() methos is called, if any thread held a shared lock, the Lock will wait for it to finish in order to acquire the exclusive lock, and all other threads that needs to a shared lock will wait till Lock is accquired and released.

8.4.2.4 SharedLock()

`void thread::atomicx::mutex::SharedLock ( )`
Shared Lock for the smart Lock.

Note

Shared lock can only be accquired if no Exclusive lock is waiting or already accquired a exclusive lock, In contrast, if at least one thread holds a shared lock, any exclusive lock can only be accquired once it is released.

8.4.2.5 SharedUnlock()

`void thread::atomicx::mutex::SharedUnlock ( )`
Release the current shared lock.

8.4.2.6 Unlock()

`void thread::atomicx::mutex::Unlock ( )`
Release the exclusive lock.
The documentation for this class was generated from the following files:

- atomicx/atomicx.hpp
- atomicx/atomicx.cpp

8.5 thread::atomicx::queue< T >::QItem Class Reference

Queue Item object.
`#include <atomicx.hpp>`

Public Member Functions

- QItem ()=delete
- QItem (T &qItem)

  Queue Item constructor.
- T & GetItem ()

  Get the current object in the QItem.

Protected Member Functions

- void SetNext (QItem &qItem)

  Set Next Item in the Queue list.
- QItem * GetNext ()

  Get the Next QItem object, if any.

Friends

- class queue

8.5.1 Detailed Description

template<typename T>
class thread::atomicx::queue< T >::QItem

Queue Item object.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 QItem() [1/2]

```
template<typename T >
thread::atomicx::queue< T >::QItem::QItem ( )  [delete]
```

8.5.2.2 QItem() [2/2]

```
template<typename T >
thread::atomicx::queue< T >::QItem::QItem (
            T & qItem )  [inline]
```

Queue Item constructor.

Parameters

qItem	Obj tem- plate type T

8.5.3 Member Function Documentation

8.5.3.1 GetItem()

```
template<typename T >
T & thread::atomicx::queue< T >::QItem::GetItem ( )  [inline]
```

Get the current object in the QItem.

Returns

T& The template type T object

8.5.3.2 GetNext()

```
template<typename T >
QItem * thread::atomicx::queue< T >::QItem::GetNext ( )  [inline], [protected]
```

Get the Next QItem object, if any.

Returns

QItem* A valid QItem pointer otherwise nullptr

8.5.3.3 SetNext()

```
template<typename T >
void thread::atomicx::queue< T >::QItem::SetNext (
            QItem & qItem )  [inline], [protected]
```

Set Next Item in the Queue list.

Parameters

qItem	QItem that holds a Queue element

8.5.4 Friends And Related Function Documentation

8.5.4.1 queue

```
template<typename T >
friend class queue  [friend]
```

The documentation for this class was generated from the following file:

- atomicx/atomicx.hpp

8.6 thread::atomicx::queue< T > Class Template Reference

```
#include <atomicx.hpp>
```

Data Structures

- class QItem

Queue Item object.

Public Member Functions

- queue ()=delete
- queue (size_t nQSize)

  Thread Safe Queue constructor.
- bool PushBack (T item)

  Push an object to the end of the queue, if the queue is full, it waits till there is a space.
- bool PushFront (T item)

  Push an object to the beggining of the queue, if the queue is full, it waits till there is a space.
- T Pop ()

  Pop an Item from the beggining of queue. Is no object there is no object in the queue, it waits for it.
- size_t GetSize ()

  Get the number of the objects in the queue.
- size_t GetMaxSize ()

  Get the Max number of object in the queue can hold.
- bool IsFull ()

  Check if the queue is full.

8.6.1 Detailed Description

template<typename T>
class thread::atomicx::queue< T >

8.6.1.1 QUEUE FOR IPC IMPLEMENTATION

8.6.2 Constructor & Destructor Documentation

8.6.2.1 queue() [1/2]

```
template<typename T >
thread::atomicx::queue< T >::queue ( )  [delete]
```

8.6.2.2 queue() [2/2]

```
template<typename T >
thread::atomicx::queue< T >::queue (
            size_t nQSize )  [inline]
```

Thread Safe Queue constructor.

Parameters

nQSize	Max number of objects to hold

8.6.3 Member Function Documentation

8.6.3.1 GetMaxSize()

```
template<typename T >
size_t thread::atomicx::queue< T >::GetMaxSize ( )  [inline]
```

Get the Max number of object in the queue can hold.

Returns

size_t The max number of object

8.6.3.2 GetSize()

```
template<typename T >
size_t thread::atomicx::queue< T >::GetSize ( )  [inline]
```

Get the number of the objects in the queue.

Returns

size_t Number of the objects in the queue

8.6.3.3 IsFull()

```
template<typename T >
bool thread::atomicx::queue< T >::IsFull ( )  [inline]
```

Check if the queue is full.

Returns

true for yes, otherwise false

8.6.3.4 Pop()

```
template<typename T >
T thread::atomicx::queue< T >::Pop ( )  [inline]
```

Pop an Item from the beggining of queue. Is no object there is no object in the queue, it waits for it.

Returns

T return the object stored.

8.6.3.5 PushBack()

```
template<typename T >
bool thread::atomicx::queue< T >::PushBack (
            T item )  [inline]
```

Push an object to the end of the queue, if the queue is full, it waits till there is a space.

Parameters

item	The object to be pushed into the queue

Returns

true if it was able to push a object in the queue, false otherwise

8.6.3.6 PushFront()

```
template<typename T >
bool thread::atomicx::queue< T >::PushFront (
            T item )  [inline]
```

Push an object to the beggining of the queue, if the queue is full, it waits till there is a space.

Parameters

item	The object to be pushed into the queue

Returns

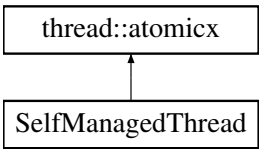
true if it was able to push a object in the queue, false otherwise

The documentation for this class was generated from the following file:

- atomicx/atomicx.hpp

8.7 SelfManagedThread Class Reference

Inheritance diagram for SelfManagedThread:



Public Member Functions

- SelfManagedThread (atomicx_time nNice)
- ~SelfManagedThread ()
- void run () noexcept override

  The pure virtual function that runs the thread loop.
- void StackOverflowHandler (void) noexcept override

  Handles the StackOverflow of the current thread.
- const char * GetName (void) override

Additional Inherited Members

8.7.1 Constructor & Destructor Documentation

8.7.1.1 SelfManagedThread()

```
SelfManagedThread::SelfManagedThread (
            atomicx_time nNice )  [inline]
```

8.7.1.2 ~SelfManagedThread()

```
SelfManagedThread::~SelfManagedThread ( )  [inline]
```

8.7.2 Member Function Documentation

8.7.2.1 GetName()

```
const char * SelfManagedThread::GetName (
            void  )  [inline], [override], [virtual]
```

Reimplemented from thread::atomicx.

8.7.2.2 run()

```
void SelfManagedThread::run ( )  [inline], [override], [virtual], [noexcept]
```

The pure virtual function that runs the thread loop.

Note

REQUIRED implementation and once it returns it will execute finish method

Implements thread::atomicx.

8.7.2.3 StackOverflowHandler()

```
void SelfManagedThread::StackOverflowHandler (
            void  )  [inline], [override], [virtual], [noexcept]
```

Handles the StackOverflow of the current thread.

Note

REQUIRED

Implements thread::atomicx.
The documentation for this class was generated from the following file:

- examples/pc/simple/simple.cpp

8.8 thread::atomicx::semaphore Class Reference

```
#include <atomicx.hpp>
```

Public Member Functions

- semaphore (size_t nMaxShared)

  Construct a new semaphore with MaxShared allowed.
- bool acquire (atomicx_time nTimeout=0)

  Acquire a shared lock context, if already on max shared allowed, wait till one is release or timeout.
- void release ()

  Releases one shared lock.

- size_t GetCount ()

  Get How many shared locks at a given moment.
- size_t GetWaitCount ()

  Get how many waiting threads for accquiring context.
- size_t GetMaxAcquired ()

  Get the Max Acquired Number.

Static Public Member Functions

- static size_t GetMax ()

  Get the maximun accquired context possible.

8.8.1 Detailed Description

8.8.1.1 SEMAPHORES IMPLEMENTATION

8.8.2 Constructor & Destructor Documentation

8.8.2.1 semaphore()

```
thread::atomicx::semaphore::semaphore (
            size_t nMaxShared )
```

Construct a new semaphore with MaxShared allowed.

Parameters

nMaxShred	Max shared lock

8.8.3 Member Function Documentation

8.8.3.1 acquire()

```
bool thread::atomicx::semaphore::acquire (
            atomicx_time nTimeout = 0 )
```

Acquire a shared lock context, if already on max shared allowed, wait till one is release or timeout.

Parameters

nTimeout	default = 0 (indefinitely), How long to wait of acquiring

Returns

true if it acquired the context, otherwise timeout returns false

8.8.3.2 GetCount()

`size_t thread::atomicx::semaphore::GetCount ( )`
Get How many shared locks at a given moment.

Returns

size_t Number of shared locks

8.8.3.3 GetMax()

`size_t thread::atomicx::semaphore::GetMax ( ) [static]`
Get the maximun accquired context possible.

Returns

size_t

8.8.3.4 GetMaxAcquired()

`size_t thread::atomicx::semaphore::GetMaxAcquired ( )`
Get the Max Acquired Number.

Returns

size_t The max acquired context number

8.8.3.5 GetWaitCount()

`size_t thread::atomicx::semaphore::GetWaitCount ( )`
Get how many waiting threads for accquiring context.

Returns

size_t Number of waiting threads

8.8.3.6 release()

`void thread::atomicx::semaphore::release ( )`
Releases one shared lock.
The documentation for this class was generated from the following files:

- atomicx/atomicx.hpp
- atomicx/atomicx.cpp

8.9 thread::atomicx::smart_ptr< T > Class Template Reference

`#include <atomicx.hpp>`

Public Member Functions

- smart_ptr (T *p)

  smart pointer constructor
- smart_ptr (const smart_ptr< T > &sa)

  smart pointer overload constructor
- smart_ptr< T > & operator= (const smart_ptr< T > &sa)

  Smart pointer Assignment operator.
- ~smart_ptr (void)

  Smart pointer destructor.
- T * operator-> (void)

  Smart pointer access operator.
- T & operator& (void)

  Smart pointer access operator.
- bool IsValid (void)

  Check if the referece still valid.
- size_t GetRefCounter (void)

  Get the Ref Counter of the managed pointer.

8.9.1 Detailed Description

template<typename T>
class thread::atomicx::smart_ptr< T >

8.9.1.1 SUPLEMENTAR SMART_PTR IMPLEMENTATION

8.9.2 Constructor & Destructor Documentation

8.9.2.1 smart_ptr() [1/2]

```
template<typename T >
thread::atomicx::smart_ptr< T >::smart_ptr (
            T * p )  [inline]
```

smart pointer constructor

Parameters

p	pointer type T to be managed

8.9.2.2 smart_ptr() [2/2]

```
template<typename T >
thread::atomicx::smart_ptr< T >::smart_ptr (
            const smart_ptr< T > & sa )  [inline]
```

smart pointer overload constructor

Parameters

sa	Smart pointer reference

8.9.2.3 ~smart_ptr()

```
template<typename T >
thread::atomicx::smart_ptr< T >::~smart_ptr (
            void ) [inline]
```

Smart pointer destructor.

8.9.3 Member Function Documentation

8.9.3.1 GetRefCounter()

```
template<typename T >
size_t thread::atomicx::smart_ptr< T >::GetRefCounter (
            void ) [inline]
```

Get the Ref Counter of the managed pointer.

Returns

size_t How much active references

8.9.3.2 IsValid()

```
template<typename T >
bool thread::atomicx::smart_ptr< T >::IsValid (
            void ) [inline]
```

Check if the referece still valid.

Returns

true if the reference still not null, otherwise false

8.9.3.3 operator&()

```
template<typename T >
T & thread::atomicx::smart_ptr< T >::operator& (
            void ) [inline]
```

Smart pointer access operator.

Returns

T* Reference for the managed object T

8.9.3.4 operator->()

```
template<typename T >
T * thread::atomicx::smart_ptr< T >::operator-> (
            void  )  [inline]
```

Smart pointer access operator.

Returns

T* Pointer for the managed object T

8.9.3.5 operator=()

```
template<typename T >
smart_ptr< T > & thread::atomicx::smart_ptr< T >::operator= (
            const smart_ptr< T > & sa )  [inline]
```

Smart pointer Assignment operator.

Parameters

sa	Smart poiter refer- ence

Returns

smart_ptr<T>& smart pointer this reference.

The documentation for this class was generated from the following file:

- atomicx/atomicx.hpp

8.10 thread::atomicx::smartMutex Class Reference

RII compliance lock/shared lock to auto unlock on destruction.

`#include <atomicx.hpp>`

Public Member Functions

- smartMutex ()=delete
- smartMutex (mutex &lockObj)

  Construct a new Smart Lock object based a existing lock.
- ~smartMutex ()

  Destroy and release the smart lock taken.
- bool SharedLock ()

  Accquire a SharedLock.
- bool Lock ()

  Accquire a exclusive Lock.
- size_t IsShared ()

  Check how many shared locks are accquired.
- bool IsLocked ()

  Check if a exclusive lock has been already accquired.

8.10.1 Detailed Description

RII compliance lock/shared lock to auto unlock on destruction.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 smartMutex() [1/2]

```
thread::atomicx::smartMutex::smartMutex ( )  [delete]
```

8.10.2.2 smartMutex() [2/2]

```
thread::atomicx::smartMutex::smartMutex (
            mutex & lockObj )
```

Construct a new Smart Lock object based a existing lock.

Parameters

lockObj	the existing lock object

8.10.2.3 ~smartMutex()

```
thread::atomicx::smartMutex::~smartMutex ( )
```

Destroy and release the smart lock taken.

8.10.3 Member Function Documentation

8.10.3.1 IsLocked()

```
bool thread::atomicx::smartMutex::IsLocked ( )
```

Check if a exclusive lock has been already accquired.

Returns

true if yes, otherwise false

8.10.3.2 IsShared()

```
size_t thread::atomicx::smartMutex::IsShared ( )
```

Check how many shared locks are accquired.

Returns

size_t Number of threads holding shared locks

8.10.3.3 Lock()

```
bool thread::atomicx::smartMutex::Lock ( )
```

Accquire a exclusive Lock.

Returns

true if accquired, false if another accquisition was already done

8.10.3.4 SharedLock()

```
bool thread::atomicx::smartMutex::SharedLock ( )
```

Accquire a SharedLock.

Returns

true if accquired, false if another accquisition was already done

The documentation for this class was generated from the following files:

- atomicx/atomicx.hpp
- atomicx/atomicx.cpp

8.11 thread::atomicx::smartSemaphore Class Reference

```
#include <atomicx.hpp>
```

Public Member Functions

- smartSemaphore (atomicx::semaphore &sem)

  Acquire and managed the semaphore.
- smartSemaphore ()=delete
- ~smartSemaphore ()

  Destroy the smart Semaphore while releasing it.
- bool acquire (atomicx_time nTimeout=0)

  Acquire a shared lock context, if already on max shared allowed, wait till one is release or timeout.
- void release ()

  Releases one shared lock.
- size_t GetCount ()

  Get How many shared locks at a given moment.
- size_t GetWaitCount ()

  Get how many waiting threads for accquiring context.
- size_t GetMaxAcquired ()

  Get the Max Acquired Number.
- bool IsAcquired ()

  Report if the smartSemaphore has acquired a shared context.

Static Public Member Functions

- static size_t GetMax ()

  Get the maximun accquired context possible.

8.11.1 Constructor & Destructor Documentation

8.11.1.1 smartSemaphore() [1/2]

```
thread::atomicx::smartSemaphore::smartSemaphore (
            atomicx::semaphore & sem )
```

Acquire and managed the semaphore.

Parameters

sem	base semaphore

8.11.1.2 smartSemaphore() [2/2]

```
thread::atomicx::smartSemaphore::smartSemaphore ( )  [delete]
```

8.11.1.3 ~smartSemaphore()

```
thread::atomicx::smartSemaphore::~smartSemaphore ( )
```

Destroy the smart Semaphore while releasing it.

8.11.2 Member Function Documentation

8.11.2.1 acquire()

```
bool thread::atomicx::smartSemaphore::acquire (
            atomicx_time nTimeout = 0 )
```

Acquire a shared lock context, if already on max shared allowed, wait till one is release or timeout.

Parameters

nTimeout	default = 0 (indefinitely), How long to wait of acquiring

Returns

true if it acquired the context, otherwise timeout returns false

8.11.2.2 GetCount()

```
size_t thread::atomicx::smartSemaphore::GetCount ( )
```

Get How many shared locks at a given moment.

Returns

size_t Number of shared locks

8.11.2.3 GetMax()

```
static size_t thread::atomicx::smartSemaphore::GetMax ( )  [static]
```

Get the maximun accquired context possible.

Returns

size_t

8.11.2.4 GetMaxAcquired()

`size_t thread::atomicx::smartSemaphore::GetMaxAcquired ( )`
Get the Max Acquired Number.

Returns

size_t The max acquired context number

8.11.2.5 GetWaitCount()

`size_t thread::atomicx::smartSemaphore::GetWaitCount ( )`
Get how many waiting threads for accquiring context.

Returns

size_t Number of waiting threads

8.11.2.6 IsAcquired()

`bool thread::atomicx::smartSemaphore::IsAcquired ( )`
Report if the smartSemaphore has acquired a shared context.

Returns

true if it has successfully acquired a shared context otherwise false

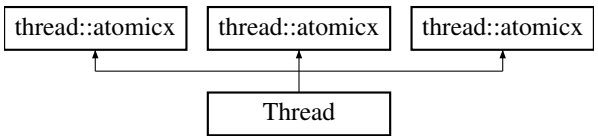
8.11.2.7 release()

`void thread::atomicx::smartSemaphore::release ( )`
Releases one shared lock.
The documentation for this class was generated from the following files:

- atomicx/atomicx.hpp
- atomicx/atomicx.cpp

8.12 Thread Class Reference

Inheritance diagram for Thread:



Public Member Functions

- Thread (atomicx_time nNice, const char *pszName)
- ~Thread ()
- void run () noexcept override

  The pure virtual function that runs the thread loop.
- void StackOverflowHandler (void) noexcept override

  Handles the StackOverflow of the current thread.
- const char * GetName (void) override

- Thread (atomicx_time nNice)
- ~Thread ()
- void run () noexcept override

  The pure virtual function that runs the thread loop.
- void StackOverflowHandler (void) noexcept override

  Handles the StackOverflow of the current thread.
- const char * GetName (void) override
- Thread (atomicx_time nNice, const char *pszName)
- ~Thread ()
- void run () noexcept override

  The pure virtual function that runs the thread loop.
- void StackOverflowHandler (void) noexcept override

  Handles the StackOverflow of the current thread.
- const char * GetName (void) override

Additional Inherited Members

8.12.1 Constructor & Destructor Documentation

8.12.1.1 Thread() [1/3]

```
Thread::Thread (
            atomicx_time nNice,
            const char * pszName )  [inline]
```

8.12.1.2 ~Thread() [1/3]

```
Thread::~Thread ( )  [inline]
```

8.12.1.3 Thread() [2/3]

```
Thread::Thread (
            atomicx_time nNice )  [inline]
```

8.12.1.4 ~Thread() [2/3]

```
Thread::~Thread ( )  [inline]
```

8.12.1.5 Thread() [3/3]

```
Thread::Thread (
            atomicx_time nNice,
            const char * pszName )  [inline]
```

8.12.1.6 ~Thread() [3/3]

```
Thread::~Thread ( )  [inline]
```

8.12.2 Member Function Documentation

8.12.2.1 GetName() [1/3]

```
const char * Thread::GetName (
            void  )  [inline], [override], [virtual]
```

Reimplemented from thread::atomicx.

8.12.2.2 GetName() [2/3]

```
const char * Thread::GetName (
            void  )  [inline], [override], [virtual]
```

Reimplemented from thread::atomicx.

8.12.2.3 GetName() [3/3]

```
const char * Thread::GetName (
            void  )  [inline], [override], [virtual]
```

Reimplemented from thread::atomicx.

8.12.2.4 run() [1/3]

```
void Thread::run ( )  [inline], [override], [virtual], [noexcept]
```

The pure virtual function that runs the thread loop.

Note

REQUIRED implementation and once it returns it will execute finish method

Implements thread::atomicx.

8.12.2.5 run() [2/3]

```
void Thread::run ( )  [inline], [override], [virtual], [noexcept]
```

The pure virtual function that runs the thread loop.

Note

REQUIRED implementation and once it returns it will execute finish method

Implements thread::atomicx.

8.12.2.6 run() [3/3]

```
void Thread::run ( )  [inline], [override], [virtual], [noexcept]
```

The pure virtual function that runs the thread loop.

Note

REQUIRED implementation and once it returns it will execute finish method

Implements thread::atomicx.

8.12.2.7 StackOverflowHandler() [1/3]

```
void Thread::StackOverflowHandler (
            void  )  [inline], [override], [virtual], [noexcept]
```

Handles the StackOverflow of the current thread.

Note

REQUIRED

Implements thread::atomicx.

8.12.2.8 StackOverflowHandler() [2/3]

```
void Thread::StackOverflowHandler (
            void  )  [inline], [override], [virtual], [noexcept]
```

Handles the StackOverflow of the current thread.

Note

REQUIRED

Implements thread::atomicx.

8.12.2.9 StackOverflowHandler() [3/3]

```
void Thread::StackOverflowHandler (
            void  )  [inline], [override], [virtual], [noexcept]
```

Handles the StackOverflow of the current thread.

Note

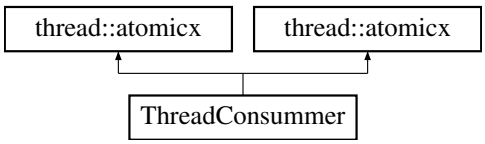
REQUIRED

Implements thread::atomicx.
The documentation for this class was generated from the following files:

- examples/pc/semaphore/semaphore.cpp
- examples/pc/simple/simple.cpp
- main.cpp

8.13 ThreadConsummer Class Reference

Inheritance diagram for ThreadConsummer:



Public Member Functions

- ThreadConsummer ()=delete
- ThreadConsummer (atomicx_time nNice, const char *pszName)
- ~ThreadConsummer ()
- void run (void) noexcept override

  The pure virtual function that runs the thread loop.
- void StackOverflowHandler (void) noexcept override

  Handles the StackOverflow of the current thread.
- const char * GetName (void) override
- ThreadConsummer ()=delete
- ThreadConsummer (atomicx_time nNice, const char *pszName)
- ~ThreadConsummer ()

- void run (void) noexcept override

  The pure virtual function that runs the thread loop.
- void StackOverflowHandler (void) noexcept override

  Handles the StackOverflow of the current thread.
- const char * GetName (void) override

Additional Inherited Members

8.13.1 Constructor & Destructor Documentation

8.13.1.1 ThreadConsummer() [1/4]

```
ThreadConsummer::ThreadConsummer ( )  [delete]
```

8.13.1.2 ThreadConsummer() [2/4]

```
ThreadConsummer::ThreadConsummer (
            atomicx_time nNice,
            const char * pszName )  [inline]
```

8.13.1.3 ~ThreadConsummer() [1/2]

```
ThreadConsummer::~ThreadConsummer ( )  [inline]
```

8.13.1.4 ThreadConsummer() [3/4]

```
ThreadConsummer::ThreadConsummer ( )  [delete]
```

8.13.1.5 ThreadConsummer() [4/4]

```
ThreadConsummer::ThreadConsummer (
            atomicx_time nNice,
            const char * pszName )  [inline]
```

8.13.1.6 ~ThreadConsummer() [2/2]

```
ThreadConsummer::~ThreadConsummer ( )  [inline]
```

8.13.2 Member Function Documentation

8.13.2.1 GetName() [1/2]

```
const char * ThreadConsummer::GetName (
            void  )  [inline], [override], [virtual]
```

Reimplemented from thread::atomicx.

8.13.2.2 GetName() [2/2]

```
const char * ThreadConsummer::GetName (
            void  )  [inline], [override], [virtual]
```

Reimplemented from thread::atomicx.

8.13.2.3 run() [1/2]

```
void ThreadConsummer::run (
            void  )  [inline], [override], [virtual], [noexcept]
```

The pure virtual function that runs the thread loop.

Note

REQUIRED implementation and once it returns it will execute finish method

Implements thread::atomicx.

8.13.2.4 run() [2/2]

```
void ThreadConsummer::run (
            void  )  [inline], [override], [virtual], [noexcept]
```

The pure virtual function that runs the thread loop.

Note

REQUIRED implementation and once it returns it will execute finish method

Implements thread::atomicx.

8.13.2.5 StackOverflowHandler() [1/2]

```
void ThreadConsummer::StackOverflowHandler (
            void  )  [inline], [override], [virtual], [noexcept]
```

Handles the StackOverflow of the current thread.

Note

REQUIRED

Implements thread::atomicx.

8.13.2.6 StackOverflowHandler() [2/2]

```
void ThreadConsummer::StackOverflowHandler (
            void  )  [inline], [override], [virtual], [noexcept]
```

Handles the StackOverflow of the current thread.

Note

REQUIRED

Implements thread::atomicx.

The documentation for this class was generated from the following files:

- examples/pc/semaphore/semaphore.cpp
- main.cpp

8.14 thread::atomicx::Timeout Class Reference

Timeout Check object.

`#include <atomicx.hpp>`

Public Member Functions

- Timeout ()=delete
- Timeout (atomicx_time nTimoutValue)

  Construct a new Timeout object.
- void Set (atomicx_time nTimoutValue)

  Set a timeout from now.
- bool IsTimedout ()

  Check wether it has timeout.
- atomicx_time GetRemaining ()

  Get the remaining time till timeout.
- atomicx_time GetDurationSince (atomicx_time startTime)

  Get the Time Since specific point in time.

8.14.1 Detailed Description

Timeout Check object.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 Timeout() [1/2]

```
thread::atomicx::Timeout::Timeout ( )  [delete]
```

8.14.2.2 Timeout() [2/2]

```
thread::atomicx::Timeout::Timeout (
            atomicx_time nTimoutValue )
```

Construct a new Timeout object.

Parameters

nTimoutValue	Timeout value to be calculated

Note

To decrease the amount of memory, Timeout does not save the start time. Special use case: if nTimeoutValue == 0, IsTimedout is always false.

8.14.3 Member Function Documentation

8.14.3.1 GetDurationSince()

```
atomicx_time thread::atomicx::Timeout::GetDurationSince (
            atomicx_time startTime )
```

Get the Time Since specific point in time.

Parameters

startTime	The specific point in time

Returns

atomicx_time How long since the point in time

Note

To decrease the amount of memory, Timeout does not save the start time.

8.14.3.2 GetRemaining()

`atomicx_time thread::atomicx::Timeout::GetRemaining ( )`

Get the remaining time till timeout.

Returns

atomicx_time Remaining time till timeout, otherwise 0;

8.14.3.3 IsTimedout()

`bool thread::atomicx::Timeout::IsTimedout ( )`

Check wether it has timeout.

Returns

true if it timeout otherwise 0

8.14.3.4 Set()

```
void thread::atomicx::Timeout::Set (
            atomicx_time nTimoutValue )
```

Set a timeout from now.

Parameters

nTimoutValue	timeout in atomicx_time

The documentation for this class was generated from the following files:

- atomicx/atomicx.hpp
- atomicx/atomicx.cpp

Chapter 9

File Documentation

9.1 atomicx/atomicx.cpp File Reference

```
#include "atomicx.hpp"
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdint.h>
#include <setjmp.h>
#include <stdlib.h>
```

Namespaces

- namespace thread

Macros

- #define POLY 0x8408

Functions

- void yield (void)

9.1.1 Macro Definition Documentation

9.1.1.1 POLY

```
#define POLY 0x8408
```

9.1.2 Function Documentation

9.1.2.1 yield()

```
void yield (
            void  )
```

9.2 atomicx/atomicx.hpp File Reference

```
#include <stdint.h>
#include <stdlib.h>
#include <setjmp.h>
```

Data Structures

- class thread::atomicx
- class thread::atomicx::Timeout

  Timeout Check object.
- class thread::atomicx::aiterator
- class thread::atomicx::smart_ptr< T >
- class thread::atomicx::queue< T >
- class thread::atomicx::queue< T >::QItem

  Queue Item object.
- class thread::atomicx::semaphore
- class thread::atomicx::smartSemaphore
- class thread::atomicx::mutex
- class thread::atomicx::smartMutex

  RII compliance lock/shared lock to auto unlock on destruction.
- struct thread::atomicx::Message

Namespaces

- namespace thread

Macros

- #define ATOMICX_VERSION "1.2.1"
- #define ATOMIC_VERSION_LABEL "AtomicX v" ATOMICX_VERSION " built at " __TIMESTAMP__
- #define ATOMICX_TIME_MAX ((atomicx_time) ~0)

Typedefs

- using atomicx_time = uint32_t

Functions

- void yield (void)
- atomicx_time Atomicx_GetTick (void)

  Implement the custom Tick acquisition.
- void Atomicx_SleepTick (atomicx_time nSleep)

  Implement a custom sleep, usually based in the same GetTick granularity.

9.2.1 Macro Definition Documentation

9.2.1.1 ATOMIC_VERSION_LABEL

```
#define ATOMIC_VERSION_LABEL "AtomicX v" ATOMICX_VERSION " built at " __TIMESTAMP__
```

9.2.1.2 ATOMICX_TIME_MAX

```
#define ATOMICX_TIME_MAX ((atomicx_time) ~0)
```

9.2.1.3 ATOMICX_VERSION

```
#define ATOMICX_VERSION "1.2.1"
```

9.2.2 Typedef Documentation

9.2.2.1 atomicx_time

```
using atomicx_time = uint32_t
```

9.2.3 Function Documentation

9.2.3.1 Atomicx_GetTick()

```
atomicx_time Atomicx_GetTick (
            void  )
```

Implement the custom Tick acquisition.

Returns

atomicx_time

9.2.3.2 Atomicx_SleepTick()

```
void Atomicx_SleepTick (
            atomicx_time nSleep )
```

Implement a custom sleep, usually based in the same GetTick granularity.

Parameters

nSleep	How long custom tick to wait

Note

This function is particularly special, since it give freedom to tweak the processor power consuption if necessary

9.2.3.3 yield()

```
void yield (
            void  )
```

9.3 atomicx.hpp

Go to the documentation of this file.

```
1 //
2 //  atomic.hpp
3 //  atomic
4 //
5 //  Created by GUSTAVO CAMPOS on 29/08/2021.
6 //
7
8 #ifndef atomic_hpp
9 #define atomic_hpp
10
11 #include <stdint.h>
12 #include <stdlib.h>
13 #include <setjmp.h>
14
15 /* Official version */
16 #define ATOMICX_VERSION "1.2.1"
17 #define ATOMIC_VERSION_LABEL "AtomicX v" ATOMICX_VERSION " built at " __TIMESTAMP__
18
19 using atomicx_time = uint32_t;
20
21 #define ATOMICX_TIME_MAX ((atomicx_time) ~0)
22
23 extern "C"
24 {
25     extern void yield(void);
26 }
27
33 extern atomicx_time Atomicx_GetTick(void);
34
43 extern void Atomicx_SleepTick(atomicx_time nSleep);
44
45 namespace thread
46 {
47     class atomicx
48     {
49     public:
50
56         enum class aTypes : uint8_t
57         {
58             start=1,
59             running=5,
60             now=6,
61             stop=10,
62             lock=50,
63             wait=55,
64             subscription=60,
65             sleep=100,
66             stackOverflow=255
67         };
68
69         enum class aSubTypes : uint8_t
70         {
71             error=10,
72             ok,
73             look,
74             wait,
75             timeout
76         };
77
78         enum class NotifyType : uint8_t
79         {
80             one = 0,
81             all = 1
82         };
83
88         class Timeout
89         {
90             public:
91                 Timeout () = delete;
92
102                 Timeout (atomicx_time nTimoutValue);
103
109                 void Set(atomicx_time nTimoutValue);
110
116                 bool IsTimedout();
117
123                 atomicx_time GetRemaining();
124
135                 atomicx_time GetDurationSince(atomicx_time startTime);
136
137              private:
138                 atomicx_time m_timeoutValue = 0;
139          };
```

```
145         class aiterator
146         {
147         public:
148             aiterator() = delete;
149 
155             aiterator(atomicx* ptr);
156 
157             /*
158              * Access operator
159              */
160             atomicx& operator*() const;
161             atomicx* operator->();
162 
163             /*
164              * Movement operator
165              */
166             aiterator& operator++();
167 
168             /*
169              * Binary operators
170              */
171             friend bool operator== (const aiterator& a, const aiterator& b){ return a.m_ptr ==
    b.m_ptr;};
172             friend bool operator!= (const aiterator& a, const aiterator& b){ return a.m_ptr !=
    b.m_ptr;};
173 
174         private:
175             atomicx* m_ptr;
176         };
177 
183         aiterator begin(void);
184 
190         aiterator end(void);
191 
197         template <typename T> class smart_ptr
198         {
199         public:
200 
206             smart_ptr(T* p) : pRef (new reference {p, 1})
207             { }
208 
214             smart_ptr(const smart_ptr<T>& sa)
215             {
216                 pRef = sa.pRef;
217                 pRef->nRC++;
218             }
219 
227             smart_ptr<T>& operator=(const smart_ptr<T>& sa)
228             {
229                 if (pRef != nullptr && pRef->nRC > 0)
230                 {
231                     pRef->nRC--;
232                 }
233 
234                 pRef = sa.pRef;
235 
236                 if (pRef != nullptr)
237                 {
238                     pRef->nRC++;
239                 }
240 
241                 return *this;
242             }
243 
247             ~smart_ptr(void)
248             {
249                 if (pRef != nullptr)
250                 {
251                     if (--pRef->nRC == 0)
252                     {
253                         delete pRef->pReference;
254                         delete pRef;
255                     }
256                     else
257                     {
258                         pRef->nRC--;
259                     }
260                 }
261             }
262 
268             T* operator-> (void)
269             {
270                 return pRef->pReference;
271             }
272 
278             T& operator& (void)
279             {
```

```
280                 return *pRef->pReference;
281             }
282 
288             bool IsValid(void)
289             {
290                 return pRef == nullptr ? false : pRef->pReference == nullptr ? false : true;
291             }
292 
298             size_t GetRefCounter(void)
299             {
300                 if (pRef != nullptr)
301                 {
302                     return pRef->nRC;
303                 }
304 
305                 return 0;
306             }
307 
308         private:
309 
310             smart_ptr(void) = delete;
311             struct reference
312             {
313                 T* pReference ;
314                 size_t nRC;
315             };
316 
317             reference* pRef=nullptr;
318         };
319 
326         template<typename T>
327         class queue
328         {
329         public:
330 
331             queue() = delete;
332 
338             queue(size_t nQSize):m_nQSize(nQSize), m_nItens{0}
339             {}
340 
349             bool PushBack(T item)
350             {
351                 if (m_nItens >= m_nQSize)
352                 {
353                     if (atomicx::GetCurrent() != nullptr)
354                     {
355                         atomicx::GetCurrent()->Wait(*this,1);
356                     }
357                     else
358                     {
359                         return false;
360                     }
361                 }
362 
363                 QItem* pQItem = new QItem(item);
364 
365                 if (m_pQIStart == nullptr)
366                 {
367                     m_pQIStart = m_pQIEnd = pQItem;
368                 }
369                 else
370                 {
371                     m_pQIEnd->SetNext(*pQItem);
372                     m_pQIEnd = pQItem;
373                 }
374 
375                 m_nItens++;
376 
377                 if (atomicx::GetCurrent() != nullptr)
378                 {
379                     atomicx::GetCurrent()->Notify(*this,0);
380                 }
381 
382                 return true;
383             }
384 
385 
394             bool PushFront(T item)
395             {
396                 if (m_nItens >= m_nQSize)
397                 {
398                     if (atomicx::GetCurrent() != nullptr)
399                     {
400                         atomicx::GetCurrent()->Wait(*this,1);
401                     }
402                     else
403                     {
```

```
404                         return false;
405                     }
406                 }
407 
408                 QItem* pQItem = new QItem(item);
409 
410                 if (m_pQIStart == nullptr)
411                 {
412                     m_pQIStart = m_pQIEnd = pQItem;
413                 }
414                 else
415                 {
416                     pQItem->SetNext(*m_pQIStart);
417                     m_pQIStart = pQItem;
418                 }
419 
420                 m_nItens++;
421 
422                 if (atomicx::GetCurrent() != nullptr)
423                 {
424                     atomicx::GetCurrent()->Notify    (*this,0);
425                 }
426 
427                 return true;
428             }
429 
436             T Pop()
437             {
438                 if (m_nItens == 0)
439                 {
440                     atomicx::GetCurrent()->Wait(*this,0);
441                 }
442 
443                 T pItem = m_pQIStart->GetItem();
444 
445                 QItem* p_tmpQItem = m_pQIStart;
446 
447                 m_pQIStart = m_pQIStart->GetNext();
448 
449                 delete p_tmpQItem;
450 
451                 m_nItens--;
452 
453                 if (atomicx::GetCurrent() != nullptr)
454                 {
455                     atomicx::GetCurrent()->Notify(*this,1);
456                 }
457 
458                 return pItem;
459             }
460 
466             size_t GetSize()
467             {
468                 return m_nItens;
469             }
470 
476             size_t GetMaxSize()
477             {
478                 return m_nQSize;
479             }
480 
486             bool IsFull()
487             {
488                 return m_nItens >= m_nQSize;
489             }
490 
491         protected:
492 
496             class QItem
497             {
498             public:
499                 QItem () = delete;
500 
506                 QItem(T& qItem) : m_qItem(qItem), m_pNext(nullptr)
507                 {}
508 
514                 T& GetItem()
515                 {
516                     return m_qItem;
517                 }
518 
519             protected:
520                 friend class queue;
521 
527                 void SetNext (QItem& qItem)
528                 {
529                     m_pNext = &qItem;
```

```
530                 }
531
537                 QItem* GetNext ()
538                 {
539                     return m_pNext;
540                 }
541
542             private:
543
544                 T m_qItem;
545                 QItem* m_pNext;
546             };
547
548         private:
549             size_t m_nQSize;
550             size_t m_nItens;
551
552             QItem* m_pQIEnd = nullptr;
553             QItem* m_pQIStart = nullptr;
554
555         };
556
563         class semaphore
564         {
565             public:
571                 semaphore(size_t nMaxShared);
572
580                 bool acquire(atomicx_time nTimeout = 0);
581
585                 void release();
586
592                 size_t GetCount();
593
599                 size_t GetWaitCount();
600
606                 size_t GetMaxAcquired();
607
613                 static size_t GetMax ();
614
615             private:
616                 size_t m_counter=0;
617                 size_t m_maxShared;
618         };
619
620         class smartSemaphore
621         {
622             public:
628                 smartSemaphore (atomicx::semaphore& sem);
629                 smartSemaphore () = delete;
633                 ~smartSemaphore();
634
642                 bool acquire(atomicx_time nTimeout = 0);
643
647                 void release();
648
654                 size_t GetCount();
655
661                 size_t GetWaitCount();
662
668                 size_t GetMaxAcquired();
669
675                 static size_t GetMax ();
676
682                 bool IsAcquired ();
683
684             private:
685             semaphore& m_sem;
686             bool bAcquired = false;
687         };
688
695         /* The stamart mutex implementation */
696         class mutex
697         {
698         public:
707             void Lock();
708
712             void Unlock();
713
721             void SharedLock();
722
726             void SharedUnlock();
727
733             size_t IsShared();
734
740             bool IsLocked();
741
742         protected:
```

```
743         private:
744             size_t nSharedLockCount=0;
745             bool bExclusiveLock=false;
746         };
747 
752         class smartMutex
753         {
754             public:
755                 smartMutex() = delete;
756 
762                 smartMutex (mutex& lockObj);
763 
767                 ~smartMutex();
768 
774                 bool SharedLock();
775 
781                 bool Lock();
782 
788                 size_t IsShared();
789 
795                 bool IsLocked();
796 
797             private:
798 
799             mutex& m_lock;
800             uint8_t m_lockType = '\0';
801         };
802 
810         virtual ~atomicx(void);
811 
817         static atomicx* GetCurrent();
818 
824         static bool Start(void);
825 
831         size_t GetID(void);
832 
838         size_t GetStackSize(void);
839 
846         atomicx_time GetNice(void);
847 
853         size_t GetUsedStackSize(void);
854 
860         atomicx_time GetCurrentTick(void);
861 
870         virtual const char* GetName(void);
871 
877         atomicx_time GetTargetTime(void);
878 
884         int GetStatus(void);
885 
891         int GetSubStatus(void);
892 
898         size_t GetReferenceLock(void);
899 
905         size_t GetTagLock(void);
906 
912         void SetNice (atomicx_time nice);
913 
920         template<typename T, size_t N> atomicx(T (&stack)[N]) : m_context{}, m_stackSize{N},
    m_stack((volatile uint8_t*) stack)
921         {
922            SetDefaultParameters();
923 
924             AddThisThread();
925         }
926 
933         atomicx(size_t nStackSize=0, int nStackIncreasePace=1);
934 
940         virtual void run(void) noexcept = 0;
941 
947         virtual void StackOverflowHandler(void) noexcept = 0;
948 
954         virtual void finish() noexcept
955         {
956             return;
957         }
958 
962         bool IsStackSelfManaged(void);
963 
971         bool Yield(atomicx_time nSleep=ATOMICX_TIME_MAX);
972 
978         atomicx_time GetLastUserExecTime();
979 
983         size_t GetStackIncreasePace(void);
984 
988         void YieldNow (void);
989 
```

```
995         void SetDynamicNice(bool status);
996
1002        bool IsDynamicNiceOn();
1003
1007      private:
1008
1013        void SetDefaultParameters ();
1014
1015        template<typename T> void SetWaitParammeters (T& refVar, size_t nTag=0, aSubTypes asubType =
    aSubTypes::wait)
1016        {
1017            m_TopicId = 0;
1018            m_pLockId = (uint8_t*)&refVar;
1019            m_aStatus = aTypes::wait;
1020            m_aSubStatus = asubType;
1021
1022            m_lockMessage.tag = nTag;
1023            m_lockMessage.message = 0;
1024        }
1025
1038        template<typename T> size_t SafeNotifier(size_t& nMessage, T& refVar, size_t nTag, aSubTypes
    subType, NotifyType notifyAll=NotifyType::one)
1039        {
1040            size_t nRet = 0;
1041
1042            for (auto& thr : *this)
1043            {
1044                if (thr.m_aSubStatus == subType && thr.m_aStatus == aTypes::wait && thr.m_pLockId ==
    (void*) &refVar && nTag == thr.m_lockMessage.tag)
1045                {
1046                    thr.m_TopicId = 0;
1047                    thr.m_aStatus = aTypes::now;
1048                    thr.m_nTargetTime = 0;
1049                    thr.m_pLockId = nullptr;
1050
1051                    thr.m_lockMessage.message = nMessage;
1052                    thr.m_lockMessage.tag = nTag;
1053
1054                    nRet++;
1055
1056                    if (notifyAll == NotifyType::one)
1057                    {
1058                        break;
1059                    }
1060                }
1061            }
1062
1063            return nRet;
1064        }
1065
1075        template<typename T> size_t SafeNotifyLookWaitings(T& refVar, size_t nTag)
1076        {
1077            size_t message=0;
1078
1079            return SafeNotifier(message, refVar, nTag, aSubTypes::look, NotifyType::all);
1080        }
1081
1085      protected:
1086
1087        struct Message
1088        {
1089            size_t tag;
1090            size_t message;
1091        };
1092
1101        uint32_t GetTopicID (const char* pszTopic, size_t nKeyLenght);
1102
1120        template<typename T> bool LookForWaitings(T& refVar, size_t nTag, size_t hasAtleast,
    atomicx_time waitFor)
1121        {
1122            Timeout timeout (waitFor);
1123
1124            while ((waitFor == 0 || timeout.IsTimedout () == false) && IsWaiting(refVar, nTag,
    hasAtleast) == false)
1125            {
1126                SetWaitParammeters (refVar, nTag, aSubTypes::look);
1127
1128                Yield(waitFor);
1129
1130                m_lockMessage = {0,0};
1131
1132                if (m_aSubStatus == aSubTypes::timeout)
1133                {
1134                    return false;
1135                }
1136
1137                // Decrease the timeout time to slice the remaining time otherwise break it
```

```
1138                if (waitFor  == 0 || (waitFor = timeout.GetRemaining ()) == 0)
1139                {
1140                    break;
1141                }
1142            }
1143
1144            return (timeout.IsTimedout ()) ? false : true;
1145        }
1146
1157        template<typename T> bool LookForWaitings(T& refVar, size_t nTag, atomicx_time waitFor)
1158        {
1159            if (IsWaiting(refVar, nTag) == false)
1160            {
1161                SetWaitParammeters (refVar, nTag, aSubTypes::look);
1162
1163                Yield(waitFor);
1164
1165                m_lockMessage = {0,0};
1166
1167                if (m_aSubStatus == aSubTypes::timeout)
1168                {
1169                    return false;
1170                }
1171            }
1172
1173            return true;
1174        }
1175
1189        template<typename T> bool IsWaiting(T& refVar, size_t nTag=0, size_t hasAtleast = 1, aSubTypes
    asubType = aSubTypes::wait)
1190        {
1191            hasAtleast = hasAtleast == 0 ? 1 : hasAtleast;
1192
1193            for (auto& thr : *this)
1194            {
1195                if (thr.m_aSubStatus == asubType && thr.m_aStatus == aTypes::wait && thr.m_pLockId ==
    (void*) &refVar && (thr.m_lockMessage.tag == nTag))
1196                {
1197                    if ((--hasAtleast) == 0)
1198                    {
1199                        return true;
1200                    }
1201                }
1202            }
1203
1204            return false;
1205        }
1206
1220        template<typename T> size_t HasWaitings(T& refVar, size_t nTag=0, aSubTypes asubType =
    aSubTypes::wait)
1221        {
1222            size_t nCounter = 0;
1223
1224            for (auto& thr : *this)
1225            {
1226                if (thr.m_aSubStatus == asubType && thr.m_aStatus == aTypes::wait && thr.m_aStatus ==
    aTypes::wait && thr.m_pLockId == (void*) &refVar && (thr.m_lockMessage.tag == nTag))
1227                {
1228                    nCounter++;
1229                }
1230            }
1231
1232            return nCounter;
1233        }
1234
1247        template<typename T> bool Wait(size_t& nMessage, T& refVar, size_t nTag=0, atomicx_time
    waitFor=0, aSubTypes asubType = aSubTypes::wait)
1248        {
1249            SafeNotifyLookWaitings(refVar, nTag);
1250
1251            SetWaitParammeters (refVar, nTag, asubType);
1252
1253            m_lockMessage.tag = nTag;
1254
1255            Yield(waitFor);
1256
1257            bool bRet = false;
1258
1259            if (m_aSubStatus != aSubTypes::timeout)
1260            {
1261                nMessage = m_lockMessage.message;
1262                bRet = true;
1263            }
1264
1265            m_lockMessage = {0,0};
1266
1267            m_aSubStatus = aSubTypes::ok;
```

```
1268
1269            return bRet;
1270        }
1271
1284        template<typename T> bool Wait(T& refVar, size_t nTag=0, atomicx_time waitFor=0, aSubTypes
    asubType = aSubTypes::wait)
1285        {
1286            SafeNotifyLookWaitings(refVar, nTag);
1287
1288            SetWaitParammeters (refVar, nTag, asubType);
1289
1290            m_lockMessage.tag = nTag;
1291
1292            Yield(waitFor);
1293
1294            bool bRet = false;
1295
1296            if (m_aSubStatus != aSubTypes::timeout)
1297            {
1298                bRet = true;
1299            }
1300
1301            m_lockMessage = {0,0};
1302            m_aSubStatus = aSubTypes::ok;
1303
1304            return bRet;
1305        }
1306
1321        template<typename T> size_t SafeNotify(size_t& nMessage, T& refVar, size_t nTag=0,  NotifyType
    notifyAll=NotifyType::one, aSubTypes asubType = aSubTypes::wait)
1322        {
1323            return SafeNotifier(nMessage, refVar, nTag, asubType, notifyAll);
1324        }
1325
1340        template<typename T> size_t Notify(size_t& nMessage, T& refVar, size_t nTag=0, NotifyType
    notifyAll=NotifyType::one, aSubTypes asubType = aSubTypes::wait)
1341        {
1342            size_t bRet = SafeNotify (nMessage, refVar, nTag, notifyAll, asubType);
1343
1344            if (bRet) Yield(0);
1345
1346            return bRet;
1347        }
1348
1349        template<typename T> size_t Notify(size_t&& nMessage, T& refVar, size_t nTag=0, NotifyType
    notifyAll=NotifyType::one, aSubTypes asubType = aSubTypes::wait)
1350        {
1351            size_t bRet = SafeNotify (nMessage, refVar, nTag, notifyAll, asubType);
1352
1353            if (bRet) Yield(0);
1354
1355            return bRet;
1356        }
1357
1373        template<typename T> size_t SyncNotify(size_t& nMessage, T& refVar, size_t nTag=0, atomicx_time
    waitForWaitings=0, NotifyType notifyAll=NotifyType::one, aSubTypes asubType = aSubTypes::wait)
1374        {
1375            if (LookForWaitings (refVar, nTag, waitForWaitings) == false)
1376            {
1377                return 0;
1378            }
1379
1380            size_t bRet = SafeNotify (nMessage, refVar, nTag, notifyAll, asubType);
1381
1382            if (bRet) Yield(0);
1383
1384            return bRet;
1385        }
1386
1387        template<typename T> size_t SyncNotify(size_t&& nMessage, T& refVar, size_t nTag=0,
    atomicx_time waitForWaitings=0, NotifyType notifyAll=NotifyType::one, aSubTypes asubType =
    aSubTypes::wait)
1388        {
1389            if (LookForWaitings (refVar, nTag, waitForWaitings) == false)
1390            {
1391                return 0;
1392            }
1393
1394            size_t bRet = SafeNotify (nMessage, refVar, nTag, notifyAll, asubType);
1395
1396            if (bRet) Yield(0);
1397
1398            return bRet;
1399        }
1400
1414        template<typename T> size_t SafeNotify(T& refVar, size_t nTag=0, NotifyType
    notifyAll=NotifyType::one, aSubTypes asubType = aSubTypes::wait)
```

```
1415         {
1416              size_t message=0;
1417              return SafeNotifier (message, refVar, nTag, asubType, notifyAll);
1418         }
1419 
1434         template<typename T> size_t SyncNotify(T& refVar, size_t nTag, atomicx_time waitForWaitings=0,
     NotifyType notifyAll=NotifyType::one, aSubTypes asubType = aSubTypes::wait)
1435         {
1436             if (LookForWaitings (refVar, nTag, waitForWaitings) == false)
1437             {
1438                 return 0;
1439             }
1440 
1441             size_t bRet = SafeNotify(refVar, nTag, notifyAll, asubType);
1442 
1443             if (bRet) Yield(0);
1444 
1445             return bRet;
1446         }
1447 
1461         template<typename T> size_t Notify(T& refVar, size_t nTag=0, NotifyType
     notifyAll=NotifyType::one, aSubTypes asubType = aSubTypes::wait)
1462         {
1463             size_t bRet = SafeNotify(refVar, nTag, notifyAll, asubType);
1464 
1465             if (bRet) Yield(0);
1466 
1467             return bRet;
1468         }
1469 
1485         bool WaitBrokerMessage (const char* pszKey, size_t nKeyLenght, Message& message);
1486 
1495         bool WaitBrokerMessage (const char* pszKey, size_t nKeyLenght);
1496 
1506         bool Publish (const char* pszKey, size_t nKeyLenght, const Message message);
1507 
1519         bool SafePublish (const char* pszKey, size_t nKeyLenght, const Message message);
1520 
1529         bool Publish (const char* pszKey, size_t nKeyLenght);
1530 
1541         bool SafePublish (const char* pszKey, size_t nKeyLenght);
1542 
1551         bool HasSubscriptions (const char* pszTopic, size_t nKeyLenght);
1552 
1560         bool HasSubscriptions (uint32_t nKeyID);
1561 
1574         virtual bool BrokerHandler(const char* pszKey, size_t nKeyLenght, Message& message)
1575         {
1576             (void) pszKey; (void) nKeyLenght; (void) message;
1577             return false;
1578         }
1579 
1588         virtual bool IsSubscribed (const char* pszKey, size_t nKeyLenght)
1589         {
1590             (void) pszKey; (void) nKeyLenght;
1591 
1592             return false;
1593         }
1594 
1600         void SetStackIncreasePace(size_t nIncreasePace);
1601 
1602     private:
1603 
1607         void AddThisThread();
1608 
1612         void RemoveThisThread();
1613 
1623         uint16_t crc16(const uint8_t* pData, size_t nSize, uint16_t nCRC);
1624 
1631         static bool SelectNextThread(void);
1632 
1637         atomicx* m_paNext = nullptr;
1638         atomicx* m_paPrev = nullptr;
1639 
1640         jmp_buf m_context;
1641 
1642         size_t m_stackSize=0;
1643         size_t m_stacUsedkSize=0;
1644         size_t m_stackIncreasePace=1;
1645 
1646         Message m_lockMessage = {0,0};
1647 
1648         atomicx_time m_nTargetTime=0;
1649         atomicx_time m_nice=0;
1650         atomicx_time m_LastUserExecTime=0;
1651         atomicx_time m_lastResumeUserTime=0;
1652 
```

```
1653         uint32_t m_TopicId=0;
1654
1655         aTypes  m_aStatus = aTypes::start;
1656         aSubTypes m_aSubStatus = aSubTypes::ok;
1657
1658         volatile uint8_t* m_stack;
1659         volatile uint8_t* m_pStaskStart=nullptr;
1660         volatile uint8_t* m_pStaskEnd=nullptr;
1661
1662         uint8_t* m_pLockId=nullptr;
1663
1664         struct
1665         {
1666             bool autoStack : 1;
1667             bool dynamicNice : 1;
1668         } m_flags = {};
1669     };
1670 }
1671
1672 #endif /* atomicx_hpp */
```

9.4 examples/Arduino/avrAutoRobotController/avrAutoRobot↩ Controller.ino File Reference

9.5 examples/Arduino/avrAutoRobotController/UpgradeUsbAsp.txt File Reference

Variables

- Programmer Day subscribers SUBSCRIBE This video is about upgrading USBasp programmer with latest firmware This upgrade will fix cannot set sck period issue

9.5.1 Variable Documentation

9.5.1.1 issue

```
Programmer Day subscribers SUBSCRIBE This video is about upgrading USBasp programmer with
latest firmware This upgrade will fix cannot set sck period issue
```

9.6 examples/Arduino/pubsublock/pubsublock.ino File Reference

9.7 examples/Arduino/semaphore/semaphore.ino File Reference

9.8 examples/Arduino/sharedlock/sharedlock.ino File Reference

9.9 examples/Arduino/simple/simple.ino File Reference

9.10 examples/Arduino/ThermalCameraDemo/ThermalCameraDemo.ino File Reference

9.11 examples/pc/semaphore/semaphore.cpp File Reference

```
#include <unistd.h>
#include <sys/time.h>
#include <cstring>
#include <cstdint>
#include <iostream>
#include <setjmp.h>
```

```
#include <string>
#include "atomicx.hpp"
```

Data Structures

- class ThreadConsummer
- class Thread

Functions

- void ListAllThreads ()
- atomicx_time Atomicx_GetTick (void)

  Implement the custom Tick acquisition.
- void Atomicx_SleepTick (atomicx_time nSleep)

  Implement a custom sleep, usually based in the same GetTick granularity.
- int main ()

Variables

- size_t nCounter = 0
- size_t nGlobalCount = 0
- atomicx::semaphore sem (2)
- Thread t1 (500, "Producer 1")
- ThreadConsummer e1 (500, "Consumer 1")

9.11.1 Function Documentation

9.11.1.1 Atomicx_GetTick()

```
atomicx_time Atomicx_GetTick (
            void  )
```

Implement the custom Tick acquisition.

Returns

atomicx_time

9.11.1.2 Atomicx_SleepTick()

```
void Atomicx_SleepTick (
            atomicx_time nSleep )
```

Implement a custom sleep, usually based in the same GetTick granularity.

Parameters

nSleep	How long custom tick to wait

Note

This function is particularly special, since it give freedom to tweak the processor power consuption if necessary

9.11.1.3 ListAllThreads()

```
void ListAllThreads ( )
```

9.11.1.4 main()

```
int main ( )
```

9.11.2 Variable Documentation

9.11.2.1 e1

```
ThreadConsummer e1(500, "Consumer 1") (
            500 ,
            "Consumer 1"  )
```

9.11.2.2 nCounter

```
size_t nCounter = 0
```

9.11.2.3 nGlobalCount

```
size_t nGlobalCount = 0
```

9.11.2.4 sem

```
atomicx::semaphore sem(2) (
            2  )
```

9.11.2.5 t1

```
Thread t1(500, "Producer 1") (
            500 ,
            "Producer 1"  )
```

9.12 examples/pc/simple/simple.cpp File Reference

```
#include <unistd.h>
#include <sys/time.h>
#include <cstring>
#include <cstdint>
#include <iostream>
#include <setjmp.h>
#include <string>
#include "atomicx.hpp"
```

Data Structures

- class SelfManagedThread
- class Thread

Functions

- void ListAllThreads ()
- atomicx_time Atomicx_GetTick (void)

  Implement the custom Tick acquisition.
- void Atomicx_SleepTick (atomicx_time nSleep)

  Implement a custom sleep, usually based in the same GetTick granularity.
- int main ()

9.12.1 Function Documentation

9.12.1.1 Atomicx_GetTick()

```
atomicx_time Atomicx_GetTick (
            void  )
```

Implement the custom Tick acquisition.

Returns

atomicx_time

9.12.1.2 Atomicx_SleepTick()

```
void Atomicx_SleepTick (
            atomicx_time nSleep )
```

Implement a custom sleep, usually based in the same GetTick granularity.

Parameters

nSleep	How long custom tick to wait

Note

This function is particularly special, since it give freedom to tweak the processor power consuption if necessary

9.12.1.3 ListAllThreads()

```
void ListAllThreads ( )
```

9.12.1.4 main()

```
int main ( )
```

9.13 main.cpp File Reference

```
#include <unistd.h>
#include <sys/time.h>
#include <cstring>
#include <cstdint>
#include <iostream>
#include <setjmp.h>
#include <string>
#include "atomicx.hpp"
```

Data Structures

- class ThreadConsummer
- class Thread

Functions

- void ListAllThreads ()
- atomicx_time Atomicx_GetTick (void)

  Implement the custom Tick acquisition.
- void Atomicx_SleepTick (atomicx_time nSleep)

  Implement a custom sleep, usually based in the same GetTick granularity.
- int main ()

Variables

- size_t nCounter = 0
- atomicx::queue< size_t > q (5)
- size_t nGlobalCount = 0
- atomicx::semaphore sem (2)
- Thread t1 (500, "Producer 1")
- ThreadConsummer e1 (500, "Consumer 1")

9.13.1 Function Documentation

9.13.1.1 Atomicx_GetTick()

```
atomicx_time Atomicx_GetTick (
            void  )
```

Implement the custom Tick acquisition.

Returns

atomicx_time

9.13.1.2 Atomicx_SleepTick()

```
void Atomicx_SleepTick (
            atomicx_time nSleep )
```

Implement a custom sleep, usually based in the same GetTick granularity.

Parameters

nSleep	How long custom tick to wait

Note

This function is particularly special, since it give freedom to tweak the processor power consuption if necessary

9.13.1.3 ListAllThreads()

```
void ListAllThreads ( )
```

9.13.1.4 main()

```
int main ( )
```

9.13.2 Variable Documentation

9.13.2.1 e1

```
ThreadConsummer e1(500, "Consumer 1") (
            500 ,
            "Consumer 1"  )
```

9.13.2.2 nCounter

```
size_t nCounter = 0
```

9.13.2.3 nGlobalCount

```
size_t nGlobalCount = 0
```

9.13.2.4 q

```
atomicx::queue< size_t > q(5) (
            5  )
```

9.13.2.5 sem

```
atomicx::semaphore sem(2) (
            2  )
```

9.13.2.6 t1

```
Thread t1(500, "Producer 1") (
            500 ,
            "Producer 1"  )
```

9.14 examples/Arduino/README.MD File Reference

9.15 README.md File Reference

Index