

# ESP-Brookesia Programming Guide



Release master  
Espressif Systems  
Jun 03, 2026



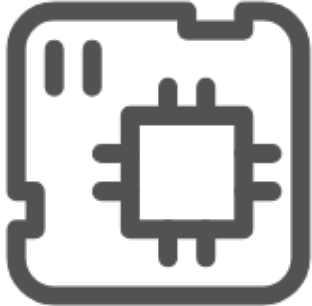



# Table of contents

<b>Table of contents</b>	<b>i</b>
<b>1 Getting Started</b>	<b>5</b>
1.1 ESP-Brookesia Versioning	5
1.2 Development Environment Setup	5
1.3 Hardware Preparation	5
1.4 How to Obtain and Use Components	6
1.5 How to Use Example Projects	6
<b>2 Utils</b>	<b>9</b>
2.1 General Utility Classes	9
2.1.1 Overview	9
2.1.2 Features	9
2.1.3 Core Utilities	9
2.1.4 Helpers	28
2.1.5 Debug Tools	38
<b>3 Hardware Abstraction Components</b>	<b>57</b>
3.1 HAL Interface	59
3.1.1 Overview	59
3.1.2 Features	59
3.1.3 API Reference	61
3.2 HAL Adaptor	81
3.2.1 Overview	81
3.2.2 Features	81
3.2.3 API Reference	82
3.3 HAL Board Support	85
3.3.1 Overview	85
3.3.2 Supported Boards	85
3.3.3 Directory Structure	87
3.3.4 Usage	88
<b>4 Service Components</b>	<b>91</b>
4.1 Service Framework	92
4.1.1 Service Manager	92
4.1.2 Service Helper	116
4.2 General Services	153
4.2.1 NVS	153
4.2.2 SNTP	156
4.2.3 Wi-Fi	160
4.2.4 Audio	171
4.2.5 Device Control	182
4.2.6 Video	194
4.2.7 Custom	201
4.3 Development Guide	202
4.3.1 Application Development Guide	202

<b>5</b>	<b>AI Agent Components</b>	<b>213</b>
<b>6</b>	<b>AI Agent Framework</b>	<b>215</b>
6.1	Agent Manager . . . . .	215
6.1.1	Overview . . . . .	215
6.1.2	State Machine Architecture . . . . .	215
6.1.3	Standard Include / Helper Class . . . . .	217
6.1.4	Service Interfaces . . . . .	217
6.1.5	Related Types and Configs . . . . .	227
6.1.6	API Reference . . . . .	228
6.2	AI Agent Helper . . . . .	229
6.2.1	Overview . . . . .	229
6.2.2	Features . . . . .	229
6.3	Agents . . . . .	237
6.3.1	Coze . . . . .	237
6.3.2	OpenAI . . . . .	239
6.3.3	XiaoZhi . . . . .	240
<b>7</b>	<b>AI Expression Components</b>	<b>245</b>
7.1	Emote . . . . .	245
7.1.1	Overview . . . . .	245
7.1.2	Standard Include / Helper Class . . . . .	245
7.1.3	Service Interfaces . . . . .	245
7.1.4	Related Types and Configs . . . . .	251
	<b>Index</b>	<b>253</b>
	<b>Index</b>	<b>253</b>



# ESP-BROOKESIA

		
<a href="#">Getting Started</a>	<a href="#">Utils Components</a>	<a href="#">HAL Components</a>
		
<a href="#">Service Components</a>	<a href="#">AI Agent Components</a>	<a href="#">AI Expression Components</a>

## Overview

ESP-Brookesia is a human-machine interaction development framework for AIoT devices. It is designed to simplify application development and AI capability integration. Built on ESP-IDF and a component-based architecture, it provides full-stack support from hardware abstraction and system services to AI agents, helping developers accelerate the development and time-to-market of HMI and AI products.

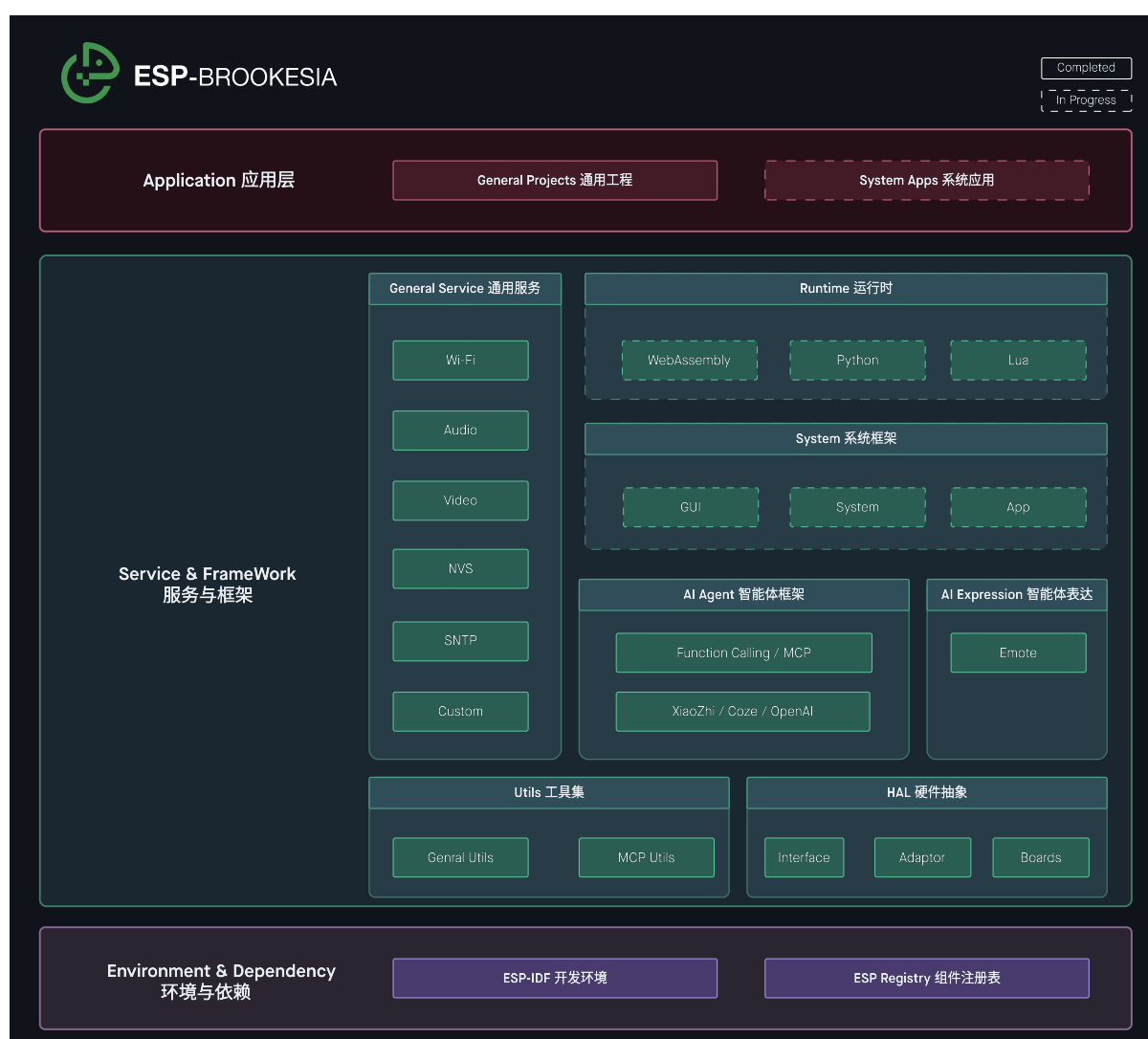
**Note:** "Brookesia" is a genus of chameleons known for camouflage and adaptability, which closely aligns with the goals of ESP-Brookesia. The framework is designed to provide a flexible and scalable solution that can adapt to different hardware devices and application requirements, much like the Brookesia chameleon itself.

The main features of ESP-Brookesia include:

- **Native ESP-IDF support:** Developed in C/C++ with deep integration into the ESP-IDF development workflow and the ESP Registry component catalog, fully leveraging Espressif's open-source component ecosystem.
- **Extensible hardware abstraction:** Defines unified hardware interfaces for audio, display, touch, storage, and more, and provides a board adaptation layer for fast porting across hardware platforms.
- **Rich system services:** Offers ready-to-use system-level services such as Wi-Fi connectivity and audio/video processing. A Manager + Helper architecture is used for decoupling and extension, and also provides support for Agent CLI.
- **Multi-LLM backend integration:** Includes built-in adapters for mainstream AI platforms such as OpenAI, Coze, and XiaoZhi, with unified agent management and lifecycle control.
- **MCP protocol support:** Exposes device service capabilities to large language models through Function Calling / MCP, enabling unified communication between LLMs and system services.
- **AI expression capabilities:** Supports emote sets, animation sets, and other visual AI expressions to provide rich visual feedback for anthropomorphic interaction.

## Functional Architecture

ESP-Brookesia adopts a layered architecture. From bottom to top, it consists of **Environment & Dependencies**, **Service & Framework**, and **Application Layer**, as shown below:



## Environment & Dependencies

The runtime foundation of the framework. `ESP-IDF` provides the build toolchain, real-time operating system, and peripheral drivers, while `ESP Registry` centrally manages the distribution and version evolution of framework components and their third-party dependencies.

## Service & Framework

The core layer of the framework. It connects downward to the environment and dependencies and provides standardized service interfaces upward to applications and AI agents, covering utilities, hardware abstraction, system services, AI agents, and expression modules.

- **Utils:** Provides common foundational capabilities for upper-layer modules. `General Utils` includes the logging system, error checking, state machine, task scheduler, plugin manager, and memory/thread/time profilers. `MCP Utils` acts as the bridge between ESP-Brookesia services and the MCP engine, exposing registered service functions as standard MCP tools so large language models can call device capabilities.
- **HAL:** Defines unified hardware access interfaces and provides board-level adaptation. `Interface` defines standardized hardware APIs for audio playback/recording, display panels and touch, status LEDs, and storage file systems. `Adaptor` provides implementations for specific development boards and completes hardware resource initialization and mapping. `Boards` provides board-level YAML configuration that describes the peripheral topology, pin assignments, and driver parameters of each board.
- **General Service:** Provides system-level foundational services, including Wi-Fi connection management, Audio capture and playback, Video codec processing, NVS non-volatile storage, SNTP network time synchronization, and a `Custom` service extension mechanism. All services use the `Manager + Helper` architecture and support both local calls and RPC-based remote communication.
- **AI Agent Framework:** Provides a unified management framework for AI agents, with built-in adapters for mainstream AI platforms such as Coze, OpenAI, and XiaoZhi. Through the `Function Calling / MCP` protocol, it enables bidirectional communication between large language models and system services, allowing LLMs to perceive and invoke device capabilities.
- **AI Expression:** Provides visual expression capabilities for AI interaction scenarios, including `Emote` sets and animation control, delivering rich visual feedback for anthropomorphic interaction.
- **System (planned):** Provides GUI, system management, and application framework support for different product forms such as mobile devices, speakers, and robots.
- **Runtime (planned):** Provides runtime support for WebAssembly, Python, Lua, and more, enabling dynamic application loading and execution.

## Application Layer

The final products and projects built on top of the layers above:

- `General Projects:` General project templates for product development that integrate framework components and can be used directly as the basis for product development.
- `System Apps (planned):` A collection of system-level applications for products, including settings, AI assistants, app stores, and more, which can be selectively integrated as needed.



# Chapter 1

## Getting Started

This guide explains how to obtain and use ESP-Brookesia components and how to build and run example projects.

### 1.1 ESP-Brookesia Versioning

From **v0.7**, ESP-Brookesia is componentized. Obtain components via the component registry as follows:

1. Components evolve independently but share the same **major.minor** version and depend on the same ESP-IDF release.
2. The **release** branch maintains historical major versions; **master** integrates new features.

Version support:

Table 1: ESP-Brookesia version support

ESP-Brookesia	ESP-IDF	Main changes	Status
master (v0.7)	$\geq v5.5$	Component manager support	Active development
release/v0.6	$\geq v5.3, \leq 5.5$	Preview system framework; ESP-VoCat firmware project	End of maintenance

### 1.2 Development Environment Setup

ESP-IDF is Espressif's framework for ESP series chips:

- Libraries and headers provide core building blocks for ESP SoC software.
- Tools for build, flash, debug, and measurement are included for development and production.

---

**Note:**

- Follow the [ESP-IDF Programming Guide](#) to set up the ESP-IDF environment.
  - It is not recommended to install the ESP-IDF environment using the VSCode extension, as this may cause build failures for some examples that depend on the `esp_board_manager` component.
- 

### 1.3 Hardware Preparation

ESP SoCs typically provide:

- Wi-Fi (2.4 GHz / 5 GHz dual band where applicable)

- Bluetooth 5.x (BLE / Mesh)
- High-performance multi-core CPUs (up to ~400 MHz)
- Ultra-low-power coprocessor and deep sleep
- **Rich peripherals:**
  - General-purpose: GPIO, UART, I2C, I2S, SPI, SDIO, USB OTG, etc.
  - Dedicated: LCD, camera, Ethernet, CAN, touch, LED PWM, temperature sensors, and more
- **Memory:**
  - Up to ~768 KB internal RAM
  - Optional external PSRAM
  - Optional external Flash
- **Security:**
  - Hardware crypto engine
  - Secure boot
  - Flash encryption
  - Digital signature

ESP SoCs use advanced process technology and offer leading RF performance, low power, and reliability for IoT, industrial, smart home, wearables, and similar applications.

---

**Note:**

- Refer to the [ESP Product Selector](#) for per-series details.
  - Refer to [ESP-Brookesia HAL Boards](#) for supported boards.
  - ESP-Brookesia requires a Flash capacity of at least 8 MB and a PSRAM capacity of at least 4 MB.
- 

## 1.4 How to Obtain and Use Components

Use the [ESP Component Registry](#) to add ESP-Brookesia components.

Example: add `brookesia_service_wifi`:

### 1. Command line

From your project directory:

```
idf.py add-dependency "espressif/brookesia_service_wifi"
```

### 2. Manifest

Create or edit `idf_component.yml`:

```
dependencies:  
  espressif/brookesia_service_wifi: "*" 
```

See [Espressif IDF Component Manager Docs](#) for more.

## 1.5 How to Use Example Projects

ESP-Brookesia provides multiple example projects. Some of them support online flashing through [ESP Launchpad](#), allowing you to flash prebuilt firmware and view serial output directly in the browser without setting up a local development environment first.

The typical build and flash steps for an example project are as follows:

1. Select the target chip or development board. The choice depends on the peripherals required by the example and usually falls into one of the following cases:
  - **Select a target chip:**  
For `examples/service/wifi`, the project only relies on the chip's built-in Wi-Fi peripheral, so you only need to select a target chip such as `esp32s3`:

```
idf.py set-target <target>
```

- **Select a target development board:**

For `examples/service/console`, the project depends on audio peripherals provided by the board, so you need to select a target development board such as `esp_vocat_board_v1_2`:

```
idf.py gen-bmgr-config -b <board>
```

---

**Note:** These projects include an `idf_ext.py` script in the project directory. Compared with regular example projects that simply depend on `esp_board_manager`, this script provides a few extra conveniences:

- No manual `IDF_EXTRA_ACTIONS_PATH` setup is required.
  - When a board is selected, `esp_board_manager` and `brookesia_hal_boards` are downloaded automatically based on the dependencies declared in `idf_component.yml`.
  - When `idf.py gen-bmgr-config` is executed, the script automatically adds the `-c` option to point to a `boards/` directory and searches in the following order: the project's local `boards/` directory first, then the `boards/` directory inside the `brookesia_hal_boards` component.
  - If needed, you can still provide a custom directory manually with the `-c` option.
- 

**Note:** Refer to [HAL Board Support](#) for the full list of supported boards.

---

2. Optional configuration:

```
idf.py menuconfig
```

3. Build and flash:

```
idf.py build
idf.py -p <PORT> flash
```

4. Monitor serial output:

```
idf.py -p <PORT> monitor
```

More examples are available under [examples/](#). For detailed usage instructions, refer to the README file in each example directory.



# Chapter 2

## Utils

This section documents ESP-Brookesia utility components.

### 2.1 General Utility Classes

- Component registry: [espressif/brookesia\\_lib\\_utils](#)
- Public header: `#include "brookesia/lib_utils.hpp"`

#### 2.1.1 Overview

*brookesia\_lib\_utils* is the general-purpose utility library for ESP-Brookesia: task scheduling, thread configuration, profiling, logging, state machines, plugins, and helpers.

#### 2.1.2 Features

- Thread configuration: RAII-style thread settings (name, priority, stack, core affinity)
- Task scheduler: Boost.Asio-based async scheduling (immediate, delayed, periodic)
- Logging: ESP\_LOG and printf-style output with formatting
- State machine and plugins: complex flows and modular extension
- Helpers: checks, function guards, describe/serialization
- Profilers: memory, thread, and time analysis

#### 2.1.3 Core Utilities

##### Thread Configuration

Public header: `#include "brookesia/lib_utils/thread_config.hpp"`

**Overview** *thread\_config* centralizes thread runtime parameters and uses RAII guards to apply and restore settings in a scope.

## Features

- *ThreadConfig* for name, priority, stack, core affinity
- Read defaults or current thread settings
- *ThreadConfigGuard* restores previous config on scope exit
- Convenience macros for configuration and queries

## API Reference

### Header File

- [utils/brookesia\\_lib\\_utils/include/brookesia/lib\\_utils/thread\\_config.hpp](#)

### Classes

class **ThreadConfigGuard**

RAII guard for thread configuration.

This class provides automatic restoration of thread configuration. When constructed, it applies the new configuration and saves the original. When destroyed, it restores the original configuration.

### Public Functions

**ThreadConfigGuard** (const ThreadConfig &config)

Apply a new thread configuration and remember the previous one.

**Parameters** **config** -- [in] Thread configuration to apply for the lifetime of this guard.

**~ThreadConfigGuard** ()

Restore the thread configuration that was active before construction.

### Macros

**\_BROOKESIA\_THREAD\_CONFIG\_CONCAT** (a, b)

**BROOKESIA\_THREAD\_CONFIG\_CONCAT** (a, b)

**BROOKESIA\_THREAD\_CONFIG\_GUARD** (...)

Apply a temporary thread configuration for the current scope.

This macro creates a *ThreadConfigGuard* instance that restores the previous configuration automatically when the surrounding scope exits.

```
{
    BROOKESIA_THREAD_CONFIG_GUARD ({
        .stack_size = 10 * 1024,
    });
    boost::thread([&]() {
        // Thread will be created with 10KB stack size
    });
} // Original configuration is restored here
```

### Parameters

- ... -- Arguments forwarded to the *ThreadConfigGuard* constructor.

**BROOKESIA\_THREAD\_GET\_CURRENT\_CONFIG()**

Query the runtime configuration of the current task.

```
BROOKESIA_LOGI("Current task: %1%", BROOKESIA_THREAD_GET_CURRENT_CONFIG());
```

---

**Note:** This returns the actual runtime configuration of the current FreeRTOS task, which may differ from the applied pthread configuration if the task was created directly via FreeRTOS APIs rather than pthread APIs.

---

**Returns** `ThreadConfig` containing the current task's runtime configuration.

**BROOKESIA\_THREAD\_GET\_APPLIED\_CONFIG()**

Query the pthread configuration currently applied to newly created threads.

```
BROOKESIA_LOGI("Applied task: %1%", BROOKESIA_THREAD_GET_APPLIED_CONFIG());
```

---

**Note:** This returns the applied pthread configuration, which may differ from the actual runtime configuration of existing tasks. Use `BROOKESIA_THREAD_GET_CURRENT_CONFIG()` to get the actual runtime state of the current task.

---

**Returns** `ThreadConfig` containing the active pthread defaults. When no override was applied, the system default configuration is returned.

## Task Scheduler

Public header: `#include "brookesia/lib_utils/task_scheduler.hpp"`

**Overview** `task_scheduler` is a Boost.Asio-based scheduler supporting immediate, delayed, and periodic tasks, with grouped serial/parallel execution.

### Features

- One-shot, delayed, periodic, and batch scheduling
- Task groups and serial execution control
- Pause, resume, cancel, and wait for completion
- Statistics and pre/post callbacks

### API Reference

#### Header File

- `utils/brookesia_lib_utils/include/brookesia/lib_utils/task_scheduler.hpp`

## Classes

### class **TaskScheduler**

Asynchronous task scheduler built on top of `boost::asio::io_context`.

The scheduler supports immediate, delayed, and periodic tasks, optional task grouping, serial execution within groups, suspension and resumption of timer-driven tasks, and task lifecycle statistics.

## Public Types

### enum class **TaskType**

Kind of scheduled task.

*Values:*

#### enumerator **Immediate**

Task scheduled with `post()` or `dispatch()`.

#### enumerator **Delayed**

One-shot timer task scheduled with `post_delayed()`.

#### enumerator **Periodic**

Repeating timer task scheduled with `post_periodic()`.

### enum class **TaskState**

Runtime state reported for a scheduled task.

*Values:*

#### enumerator **Running**

Task is active and eligible to execute.

#### enumerator **Suspended**

Task timer is suspended.

#### enumerator **Canceled**

Task was canceled before completion.

#### enumerator **Finished**

Task finished execution or is no longer tracked.

### using **TaskId** = uint64\_t

Identifier type assigned to each scheduled task.

### using **OnceTask** = std::function<void()>

Callable type for one-shot tasks.

### using **PeriodicTask** = std::function<bool()>

Callable type for periodic tasks.

Returning `false` stops future executions.

using **Group** = std::string

Task group name type.

using **Executor** = boost::asio::io\_context::executor\_type

Executor type exposed by the underlying `boost::asio::io_context`.

using **PreExecuteCallback** = std::function<void(const *Group*&, *TaskId*, *TaskType*)>

Callback invoked when a task is selected by `io_context` and about to execute.

---

**Note:** This callback is invoked after the task is selected by `io_context` and just before the actual task logic executes, guaranteeing that the task will execute (unless an exception occurs in the callback itself)

---

**Param group** Group name of the task

**Param task\_id** ID of the task about to execute

**Param task\_type** Type of the task

using **PostExecuteCallback** = std::function<void(const *Group*&, *TaskId*, *TaskType*, bool success)>

Callback invoked after a task completes execution.

---

**Note:** This callback is invoked after the task logic completes, regardless of success or failure, providing a hook for cleanup, logging, or statistics

---

**Param group** Group name of the completed task

**Param task\_id** ID of the completed task

**Param task\_type** Type of the task

**Param success** Whether the task executed successfully

## Public Functions

**TaskScheduler** () = default

Construct an idle task scheduler.

**~TaskScheduler** ()

Stop the scheduler and release its resources.

**TaskScheduler** (const *TaskScheduler*&) = delete

Copy construction is not supported.

*TaskScheduler* &**operator=** (const *TaskScheduler*&) = delete

Copy assignment is not supported.

**TaskScheduler** (*TaskScheduler*&&) = delete

Move construction is not supported.

*TaskScheduler* &**operator=** (*TaskScheduler*&&) = delete

Move assignment is not supported.

bool **start** (const *StartConfig* &config)

Start the scheduler with a custom worker configuration.

**Parameters** **config** -- [in] Worker-thread and callback configuration.

**Returns** `true` on success, or `false` when startup fails.

inline bool **start** ()

Start the scheduler with the default configuration.

**Returns** `true` on success, or `false` when startup fails.

void **stop** ()

Stop the scheduler and cancel all pending tasks.

Running tasks are allowed to finish, worker threads are joined, and internal state is reset.

inline bool **is\_running** () const

Check whether the scheduler is running.

**Returns** `true` when worker threads and the `io_context` are active, or `false` otherwise.

bool **configure\_group** (const *Group* &group, const *GroupConfig* &config)

Configure execution behavior for a task group.

**Parameters**

- **group** -- [in] Group name.
- **config** -- [in] Group behavior configuration.

**Returns** `true` if the group was configured successfully, or `false` otherwise.

bool **dispatch** (*OnceTask* task, *TaskId* \*id = nullptr, const *Group* &group = "")

Dispatch a task for immediate execution when possible.

When called from a scheduler worker thread, the task may run inline instead of being enqueued.

**Parameters**

- **task** -- [in] Task to execute.
- **id** -- [out] Optional pointer that receives the assigned task ID.
- **group** -- [in] Optional task group name.

**Returns** `true` if the task was scheduled successfully, or `false` otherwise.

bool **post** (*OnceTask* task, *TaskId* \*id = nullptr, const *Group* &group = "")

Post a task to the scheduler queue.

**Parameters**

- **task** -- [in] Task to execute.
- **id** -- [out] Optional pointer that receives the assigned task ID.
- **group** -- [in] Optional task group name.

**Returns** `true` if the task was scheduled successfully, or `false` otherwise.

bool **post\_delayed** (*OnceTask* task, int delay\_ms, *TaskId* \*id = nullptr, const *Group* &group = "")

Schedule a one-shot task to run after a delay.

**Parameters**

- **task** -- [in] Task to execute.
- **delay\_ms** -- [in] Delay before execution, in milliseconds.
- **id** -- [out] Optional pointer that receives the assigned task ID.
- **group** -- [in] Optional task group name.

**Returns** `true` if the task was scheduled successfully, or `false` otherwise.

bool **post\_periodic** (*PeriodicTask* task, int interval\_ms, *TaskId* \*id = nullptr, const *Group* &group = "")

Schedule a periodic task.

The task will be executed repeatedly at the specified interval. Return `false` from the task to stop the periodic execution.

**Parameters**

- **task** -- [in] Periodic task to execute. Returning `false` stops future runs.
- **interval\_ms** -- [in] Interval between executions, in milliseconds.
- **id** -- [out] Optional pointer that receives the assigned task ID.
- **group** -- [in] Optional task group name.

**Returns** `true` if the task was scheduled successfully, or `false` otherwise.

bool **post\_batch** (std::vector<OnceTask> tasks, std::vector<TaskId> \*ids = nullptr, const Group &group = "")

Post multiple one-shot tasks as a batch.

**Parameters**

- **tasks** -- [in] Vector of tasks to execute.
- **ids** -- [out] Optional pointer that receives the assigned task IDs.
- **group** -- [in] Optional task group name applied to every task.

**Returns** `true` if all tasks were scheduled successfully, or `false` otherwise.

void **cancel** (TaskId id)

Cancel a task by ID.

**Parameters** **id** -- [in] Task ID to cancel.

void **cancel\_group** (const Group &group)

Cancel every task in a group.

**Parameters** **group** -- [in] Group name.

void **cancel\_all** ()

Cancel all tracked tasks.

bool **suspend** (TaskId id)

Suspend a delayed or periodic task.

Only delayed and periodic tasks can be suspended.

**Parameters** **id** -- [in] Task ID to suspend.

**Returns** `true` if the task was suspended successfully, or `false` otherwise.

size\_t **suspend\_group** (const Group &group)

Suspend all suspendable tasks in a group.

**Parameters** **group** -- [in] Group name.

**Returns** Number of tasks suspended.

size\_t **suspend\_all** ()

Suspend all suspendable tasks.

**Returns** Number of tasks suspended.

bool **resume** (TaskId id)

Resume a suspended delayed or periodic task.

**Parameters** **id** -- [in] Task ID to resume.

**Returns** `true` if the task was resumed successfully, or `false` otherwise.

size\_t **resume\_group** (const Group &group)

Resume all suspended tasks in a group.

**Parameters** **group** -- [in] Group name.

**Returns** Number of tasks resumed.

size\_t **resume\_all** ()

Resume all suspended tasks.

**Returns** Number of tasks resumed.

bool **wait** (TaskId id, int timeout\_ms = -1)

Wait for a task to complete.

**Parameters**

- **id** -- [in] Task ID to wait for.
- **timeout\_ms** -- [in] Timeout in milliseconds. -1 waits indefinitely.

**Returns** `true` if the task completed within the timeout, or `false` otherwise.

bool **wait\_group** (const *Group* &group, int timeout\_ms = -1)

Wait for all tasks in a group to complete.

**Parameters**

- **group** -- **[in]** Group name.
- **timeout\_ms** -- **[in]** Timeout in milliseconds. -1 waits indefinitely.

**Returns** `true` if all tasks in the group completed within the timeout, or `false` otherwise.

bool **wait\_all** (int timeout\_ms = -1)

Wait for all tasks to complete.

**Parameters** **timeout\_ms** -- **[in]** Timeout in milliseconds. -1 waits indefinitely.

**Returns** `true` if all tasks completed within the timeout, or `false` otherwise.

bool **restart\_timer** (*TaskId* id)

Restart the timer for a delayed or periodic task.

This resets the timer countdown to its original interval. Useful for implementing debounce or watchdog-like behavior where you want to postpone execution.

**Parameters** **id** -- **[in]** Task ID whose timer should restart.

**Returns** `true` if the timer restarted successfully, or `false` when the task is missing, has the wrong type, or is not running.

*TaskType* **get\_type** (*TaskId* id) const

Query the type of a task.

**Parameters** **id** -- **[in]** Task ID.

**Returns** Task type for the ID, or *TaskType::Immediate* when the task is unknown.

*TaskState* **get\_state** (*TaskId* id) const

Query the state of a task.

**Parameters** **id** -- **[in]** Task ID.

**Returns** Task state for the ID, or *TaskState::Finished* when the task is unknown.

*Group* **get\_group** (*TaskId* id) const

Query the group of a task.

**Parameters** **id** -- **[in]** Task ID.

**Returns** Group name, or an empty string when the task is unknown or ungrouped.

size\_t **get\_group\_task\_count** (const *Group* &group) const

Query the number of tracked tasks in a group.

**Parameters** **group** -- **[in]** Group name.

**Returns** Number of tasks currently associated with the group.

std::vector<*Group*> **get\_active\_groups** () const

Get all groups that currently contain tasks.

**Returns** Vector of active group names.

*Statistics* **get\_statistics** () const

Get execution statistics accumulated by the scheduler.

**Returns** *Statistics* structure with task counters.

inline std::shared\_ptr<*Executor*> **get\_executor** ()

Get a shared executor handle for the underlying `io_context`.

**Returns** Shared pointer to the executor, or `nullptr` when the scheduler is not running.

inline size\_t **get\_worker\_count** () const

Get the number of worker threads.

**Returns** Number of worker threads currently owned by the scheduler.

void **reset\_statistics** ()

Reset all accumulated statistics counters to zero.

struct **GroupConfig**

Configuration for a task group.

### Public Members

bool **enable\_serial\_execution** = false

Serialize all tasks in this group through a strand when set to `true`.

Periodic tasks skip overlapping executions for the same task instance, while one-shot tasks are queued and executed sequentially.

*Group* **parent\_group** = ""

Optional parent group name whose execution context should be reused.

When the parent group is serial, this group also runs serially through the parent strand.

*PreExecuteCallback* **pre\_execute\_callback** = nullptr

Optional group pre-execute callback applied to all tasks in the group.

*PostExecuteCallback* **post\_execute\_callback** = nullptr

Optional group post-execute callback applied to all tasks in the group.

struct **StartConfig**

Startup configuration for the scheduler worker threads.

### Public Members

std::vector<ThreadConfig> **worker\_configs**

Worker thread configurations used when the scheduler starts.

The default configuration creates a single worker named `Worker` with a 6 KiB stack.

size\_t **worker\_poll\_interval\_ms** = 10

Worker polling interval in milliseconds.

*PreExecuteCallback* **pre\_execute\_callback** = nullptr

Optional global pre-execute callback applied to every task.

The callback fires just before any task executes, regardless of group membership.

*PostExecuteCallback* **post\_execute\_callback** = nullptr

Optional global post-execute callback applied to every task.

The callback fires after any task completes, regardless of group membership.

**struct `Statistics`**

Aggregate counters for task execution.

**Public Members**

`size_t total_tasks = {0}`

Total number of tasks ever scheduled.

`size_t completed_tasks = {0}`

Number of tasks that completed successfully.

`size_t failed_tasks = {0}`

Number of tasks that finished unsuccessfully.

`size_t canceled_tasks = {0}`

Number of tasks canceled before completion.

`size_t suspended_tasks = {0}`

Number of tasks currently suspended.

**State Base Class**

Public header: `#include "brookesia/lib_utils/state_base.hpp"`

**Overview** `state_base` defines the base state abstraction for state machines with enter, exit, and update callbacks, plus timeouts and update intervals.

**Features**

- Standard lifecycle: `on_enter` / `on_exit` / `on_update`
- Timeout actions and update intervals
- Easy to subclass for custom states

**API Reference****Header File**

- [utils/brookesia\\_lib\\_utils/include/brookesia/lib\\_utils/state\\_base.hpp](#)

**Classes****class `StateBase`**

Base class for state machine states.

Provides lifecycle hooks (`on_enter`, `on_exit`, `on_update`) and configuration for timeout and periodic update intervals.

## Public Functions

inline explicit **StateBase** (const std::string &name)

Constructor.

**Parameters** **name** -- State name.

virtual **~StateBase** () = default

Virtual destructor.

inline virtual bool **on\_enter** (const std::string &from\_state = "", const std::string &action = "")

Hook invoked before the state becomes active.

### Parameters

- **from\_state** -- Name of the previous state, or an empty string for the initial state.
- **action** -- Transition action name, or an empty string when not specified.

**Returns** `true` to allow the transition, or `false` to reject entry.

inline virtual bool **on\_exit** (const std::string &to\_state = "", const std::string &action = "")

Hook invoked before the state is left.

### Parameters

- **to\_state** -- Name of the next state, or an empty string when unspecified.
- **action** -- Transition action name, or an empty string when not specified.

**Returns** `true` to allow the transition, or `false` to reject exit.

inline virtual void **on\_update** ()

Hook invoked periodically while the state remains active.

---

**Note:** This callback is scheduled only when `set_update_interval()` configures a non-zero interval.

---

inline void **set\_timeout** (uint32\_t ms, const std::string &action)

Configure an automatic timeout transition for this state.

### Parameters

- **ms** -- Timeout duration in milliseconds. A value of 0 disables the timeout.
- **action** -- Action name triggered when the timeout expires.

inline void **set\_update\_interval** (uint32\_t interval\_ms)

Configure the periodic update interval for this state.

**Parameters** **interval\_ms** -- Update period in milliseconds. A value of 0 disables `on_update()` scheduling.

inline const std::string &**get\_name** () const

Get the state name.

**Returns** State name.

inline uint32\_t **get\_timeout\_ms** () const

Get the configured timeout duration.

**Returns** Timeout duration in milliseconds, or 0 when no timeout is configured.

inline const std::string &**get\_timeout\_action** () const

Get the action triggered when the timeout expires.

**Returns** Configured timeout action name.

inline uint32\_t **get\_update\_interval** () const

Get the configured periodic update interval.

**Returns** Update interval in milliseconds, or 0 when periodic updates are disabled.

## State Machine

Public header: `#include "brookesia/lib_utils/state_machine.hpp"`

**Overview** *state\_machine* implements registration, actions, transition callbacks, and runtime control for multi-state workflows.

### Features

- Add states and transition rules
- Initial state and forced switches
- Transition callbacks and runtime queries
- Optional integration with *TaskScheduler* for async work

### API Reference

#### Header File

- [utils/brookesia\\_lib\\_utils/include/brookesia/lib\\_utils/state\\_machine.hpp](#)

#### Classes

class **StateMachine**

Thread-safe Finite State Machine implementation.

Manages states, transitions, and state lifecycle with support for:

- State entry/exit guards (*on\_enter/on\_exit*)
- Periodic state updates (*on\_update*)
- State timeouts with automatic action triggering
- Asynchronous state transitions via action queue
- Serial execution guarantee (no concurrent state transitions)
- Transition rollback on entry failure

---

**Note:** All state transitions are executed serially through the task scheduler's group mechanism, ensuring thread-safe operations even when actions are triggered from multiple threads. State callbacks (*on\_enter*, *on\_exit*, *on\_update*) are executed without holding internal locks, allowing them to safely trigger new actions or perform blocking operations.

---

#### Public Types

using **StatePtr** = `std::shared_ptr<StateBase>`

Shared pointer type used for registered states.

using **TransitionFinishCallback** = `std::function<void(const std::string &from, const std::string &action, const std::string &to)>`

Callback type invoked after a transition completes successfully.

## Public Functions

**StateMachine** () = default

Constructor.

**~StateMachine** ()

Stop the state machine and release owned resources.

bool **add\_state** (*StatePtr* state)

Register a named state object.

**Parameters** **state** -- Shared pointer to the state implementation.

**Returns** `true` if the state was added, or `false` when a state with the same name already exists.

bool **add\_transition** (const std::string &from, const std::string &action, const std::string &to)

Register a transition between two states.

**Parameters**

- **from** -- Source state name.
- **action** -- Action name that triggers the transition.
- **to** -- Target state name.

**Returns** `true` if the transition was added, or `false` when the same (`from`, `action`) mapping already exists.

bool **start** (*Config* config)

Start the state machine and enter the initial state.

---

**Note:** If the state machine is already running, this returns `true` immediately. If the scheduler is not running, this will attempt to start it. The task group is automatically configured for serial execution to ensure thread-safe transitions.

---

**Parameters** **config** -- Configuration for the state machine.

**Returns** `true` on success, or `false` if the initial state cannot be entered or startup fails.

void **stop** ()

Stop the state machine and wait for pending scheduled work to finish.

---

**Note:** This method is thread-safe and can be called multiple times safely. It will wait for all pending state tasks to complete before returning.

---

bool **trigger\_action** (const std::string &action, bool use\_dispatch = false)

Queue or dispatch a transition action.

---

**Note:** The actual transition happens asynchronously in the task scheduler's serial queue. This ensures thread-safe state transitions even when called from multiple threads.

---

**Parameters**

- **action** -- Transition action name.
- **use\_dispatch** -- Set to `true` to call `TaskScheduler::dispatch()` instead of `post()`.

**Returns** `true` if the action was scheduled successfully, or `false` otherwise.

bool **wait\_all\_transitions** (uint32\_t timeout\_ms)

Wait until no transitions are queued or running.

**Parameters** `timeout_ms` -- Timeout in milliseconds. The special value `static_cast<uint32_t>(-1)` waits indefinitely.

**Returns** `true` if all transitions completed before the timeout, or `false` otherwise.

bool **force\_transition\_to** (const std::string &target\_state)

Cancel scheduled state tasks and overwrite the current state name immediately.

---

**Note:** This bypasses `on_exit()`, `on_enter()`, timeout scheduling, periodic updates, and transition callbacks.

---

**Parameters** `target_state` -- State name assigned as the new current state.

**Returns** `true` once the internal state name has been updated.

inline void **register\_transition\_finish\_callback** (*TransitionFinishCallback* callback)

Register a callback invoked after a successful transition.

---

**Note:** The callback is invoked with (from\_state, action, to\_state) parameters. The callback will be executed without holding internal locks, so it's safe to perform any operations including triggering new actions.

---

**Parameters** `callback` -- Callback invoked with (from\_state, action, to\_state).

inline bool **is\_running** () const

Check whether the state machine has been started.

---

**Note:** This method is thread-safe.

---

**Returns** `true` when the machine owns a scheduler and is running, or `false` otherwise.

inline bool **has\_transition\_running** () const

Check whether any transitions are queued or currently executing.

---

**Note:** This method is thread-safe.

---

**Returns** `true` when one or more transitions are pending or in progress, or `false` otherwise.

inline bool **has\_state\_updating** () const

Check whether the current state has an active periodic update task.

---

**Note:** This method is thread-safe.

---

**Returns** `true` when a periodic update task is installed, or `false` otherwise.

```
inline std::string get_current_state () const
```

Get the name of the current state.

---

**Note:** This method is thread-safe and returns a copy of the state name.

---

**Returns** Current state name, or an empty string when the machine has not started.

```
inline StatePtr get_state_ptr (const std::string &name) const
```

Look up a registered state by name.

---

**Note:** This method is thread-safe.

---

**Parameters** **name** -- State name to look up.

**Returns** Shared pointer to the registered state, or `nullptr` if not found.

### Public Static Attributes

```
static constexpr const char *DEFAULT_TASK_GROUP_NAME = "state_machine"
```

Default scheduler group name used by the state machine.

```
struct Config
```

Configuration for the state machine.

### Public Members

```
std::shared_ptr<TaskScheduler> task_scheduler
```

Task scheduler used to serialize transitions, timeouts, and periodic updates.

```
std::string task_group_name = DEFAULT_TASK_GROUP_NAME
```

Task group name used for serialized state-machine tasks.

```
TaskScheduler::GroupConfig task_group_config
```

Task group configuration.

```
std::string initial_state
```

Initial state name.

### Plugin System

Public header: `#include "brookesia/lib_utils/plugin.hpp"`

**Overview** *plugin* provides a generic plugin registry and instance management: register, discover, and create plugins by name.

## Features

- Template registration and lookup by name
- Lazy instantiation and singleton registration
- Enumerate, release, and cleanup
- Macros for registration symbols and link visibility

## API Reference

### Header File

- [utils/brookesia\\_lib\\_utils/include/brookesia/lib\\_utils/plugin.hpp](#)

### Classes

template<typename **T**>

class **PluginRegistry**

Thread-safe registry for named plugin factories and cached instances.

**Template Parameters** **T** -- Base class type exposed by the registry.

### Public Static Functions

static inline std::shared\_ptr<**T**> **get\_instance** (const std::string &name)

Get a plugin instance by name.

Returns the cached instance when available, otherwise creates it through the registered factory.

---

**Note:** The returned shared\_ptr shares ownership with the Registry. The instance remains valid as long as either the Registry or any returned shared\_ptr holds a reference, ensuring thread-safe access even if *release\_instance()* or *remove\_plugin()* is called.

---

**Parameters** **name** -- [in] Registered plugin name.

**Returns** Shared pointer to the plugin instance, or `nullptr` when the plugin is not registered or the factory is empty.

static inline std::map<std::string, std::shared\_ptr<**T**>> **get\_all\_instances** ()

Get instances for all registered plugins.

Missing cached instances are created on demand before they are returned.

---

**Note:** Each returned shared\_ptr shares ownership with the Registry. The instances remain valid as long as either the Registry or any returned shared\_ptr holds a reference, ensuring thread-safe access.

---

**Returns** Map from plugin name to shared plugin instance.

static inline size\_t **get\_plugin\_count** ()

Get the number of registered plugin names.

**Returns** Total number of registered plugins.

static inline bool **has\_plugin** (const std::string &name)

Check whether a plugin name is registered.

**Parameters** **name** -- [in] Plugin name to test.

**Returns** `true` when the plugin exists in the registry, or `false` otherwise.

static inline void **release\_instance** (const std::string &name)

Release the cached instance for a registered plugin.

This keeps the factory registration intact and only drops the registry-held shared pointer.

**Parameters** **name** -- [in] Plugin name.

static inline void **release\_all\_instances** ()

Release all cached plugin instances without removing registrations.

static inline void **remove\_plugin** (const std::string &name)

Remove a plugin registration and its cached instance.

**Parameters** **name** -- [in] Plugin name to remove.

static inline void **remove\_all\_plugins** ()

Remove all registered plugins and cached instances.

template<typename **PluginType**>

static inline void **register\_plugin** (const std::string &name, FactoryFunc factory)

Register a plugin factory under a unique name.

**Template Parameters** **PluginType** -- Concrete plugin type being registered.

**Parameters**

- **name** -- [in] Plugin name. Existing registrations with the same name are kept unchanged.
- **factory** -- [in] Factory used to lazily create instances.

## Macros

**\_BROOKESIA\_PLUGIN\_CONCAT** (a, b)

**BROOKESIA\_PLUGIN\_CONCAT** (a, b)

**BROOKESIA\_PLUGIN\_CREATE\_SYMBOL** (symbol\_name, static\_var\_name)

Create a linker-visible symbol that keeps a registrar object alive.

---

**Note:** The function is defined outside any namespace to ensure proper linking. The static variable reference ensures the registrar instance is not optimized away.

---

**BROOKESIA\_PLUGIN\_REGISTER\_PRE\_MAIN\_FUNCTION** (symbol\_name, function)

Auto-register a function to be invoked before `main()`.

Generates a file-local static registrar whose constructor calls `function` exactly once during C++ static initialization - i.e. before the program enters `main()`. On ESP-IDF this also guarantees the function runs before `app_main()` is entered.

Typical use cases: plugin auto-registration, installing HAL hook callbacks, seeding global singletons, or any one-shot boot-time setup. This macro is the building block behind every other `BROOKESIA_PLUGIN_REGISTER_*` form.

Relative ordering between translation units that use this (or any other static initialization) is unspecified. If ordering matters, embed it inside `function` itself.

The generated static instance is also exported as an external symbol via `BROOKESIA_PLUGIN_CREATE_SYMBOL` so the linker keeps it alive when this TU lives in a static library. Remember to add `-u <symbol_name>` to the final link.

```
static void boot_time_setup() {
    // Runs before main / app_main.
}
BROOKESIA_PLUGIN_REGISTER_PRE_MAIN_FUNCTION(my_boot_setup_sym, boot_time_
↪setup);

BROOKESIA_PLUGIN_REGISTER_PRE_MAIN_FUNCTION(my_inline_sym, [] () {
    // ... inline pre-main work ...
});
```

### Parameters

- **symbol\_name** -- Linker-visible identifier used with `-u` to keep the registrar alive.
- **function** -- Callable expression (return value is ignored); typically a function pointer or a capture-less lambda.

**BROOKESIA\_PLUGIN\_REGISTER\_WITH\_CONSTRUCTOR** (BaseType, PluginType, name, creator, symbol\_name)

Register a plugin using a custom creator expression.

---

**Note:** Automatically creates a fixed symbol name based on `PluginType` for linker `-u` option. Internally delegates to `BROOKESIA_PLUGIN_REGISTER_PRE_MAIN_FUNCTION` for the static-init dispatch; the only extra machinery here is the pointer-type-coercion helper that adapts shared/unique pointers returned by the creator into the registry's `shared_ptr<BaseType>` factory signature.

---

### Parameters

- **BaseType** -- Registry base type.
- **PluginType** -- Concrete plugin type to register.
- **name** -- Plugin name used by `PluginRegistry`.
- **creator** -- Creator expression returning `shared_ptr` or `unique_ptr` compatible with the registry.
- **symbol\_name** -- Linker symbol exported for `-u`.

**BROOKESIA\_PLUGIN\_REGISTER** (BaseType, PluginType, name, ...)

Register a plugin constructed with `std::make_shared`.

```
// Example usage:
BROOKESIA_PLUGIN_REGISTER(BaseService, MyPlugin, "my_plugin");
BROOKESIA_PLUGIN_REGISTER(BaseService, MyPlugin, "my_plugin", arg1, arg2);
```

### Parameters

- **BaseType** -- Registry base type.
- **PluginType** -- Concrete plugin type to register.
- **name** -- Plugin name used by `PluginRegistry`.
- **...** -- Optional constructor arguments forwarded to `PluginType`.

**BROOKESIA\_PLUGIN\_REGISTER\_SINGLETON** (BaseType, PluginType, name, instance\_expr)

Register a singleton object in the plugin registry.

This macro allows registering a singleton instance to the plugin registry. It uses a custom no-op deleter to prevent the `shared_ptr` from destroying the singleton. Automatically generates a fixed symbol name based on `PluginType` for linker `-u` option.

```
// Example usage:
BROOKESIA_PLUGIN_REGISTER_SINGLETON(
    BaseService,
    MySingleton,
    "my_singleton",
    MySingleton::get_instance()
);
// Creates symbol: _MySingleton_symbol_<line_number>
// Use: target_link_libraries(${COMPONENT_LIB} INTERFACE "-u _MySingleton_
↳symbol_<line_number>")
```

### Parameters

- **BaseType** -- Registry base type.
- **PluginType** -- Singleton type to register.
- **name** -- Plugin name used by PluginRegistry.
- **instance\_expr** -- Expression that yields a singleton instance reference, for example `Type::get_instance()`.

**BROOKESIA\_PLUGIN\_REGISTER\_WITH\_SYMBOL** (BaseType, PluginType, name, symbol\_name, ...)

Register a plugin constructed with `std::make_shared` and a custom linker symbol.

```
// Example usage:
BROOKESIA_PLUGIN_REGISTER_WITH_SYMBOL(BaseService, MyPlugin, "my_plugin",
↳"custom_symbol_name");
BROOKESIA_PLUGIN_REGISTER_WITH_SYMBOL(BaseService, MyPlugin, "my_plugin",
↳"custom_symbol_name", arg1, arg2);
// Use: target_link_libraries(${COMPONENT_LIB} INTERFACE "-u custom_symbol_name
↳")
```

### Parameters

- **BaseType** -- Registry base type.
- **PluginType** -- Concrete plugin type to register.
- **name** -- Plugin name used by PluginRegistry.
- **symbol\_name** -- Custom linker symbol exported for `-u`.
- **...** -- Optional constructor arguments forwarded to PluginType.

**BROOKESIA\_PLUGIN\_REGISTER\_SINGLETON\_WITH\_SYMBOL** (BaseType, PluginType, name, instance\_expr, symbol\_name)

Register a singleton object with a custom linker symbol.

This macro allows registering a singleton instance to the plugin registry. It uses a custom no-op deleter to prevent the `shared_ptr` from destroying the singleton. Uses the provided custom symbol name for linker `-u` option.

```
// Example usage:
BROOKESIA_PLUGIN_REGISTER_SINGLETON_WITH_SYMBOL(
    BaseService,
    MySingleton,
    "my_singleton",
    MySingleton::get_instance(),
    "custom_symbol_name"
);
// Use: target_link_libraries(${COMPONENT_LIB} INTERFACE "-u custom_symbol_name
↳")
```

### Parameters

- **BaseType** -- Registry base type.
- **PluginType** -- Singleton type to register.
- **name** -- Plugin name used by `PluginRegistry`.
- **instance\_expr** -- Expression that yields a singleton instance reference.
- **symbol\_name** -- Custom linker symbol exported for `-u`.

## 2.1.4 Helpers

### Logging System

Public header: `#include "brookesia/lib_utils/log.hpp"`

**Overview** `log` provides a unified logging API with formatting, switching between `ESP_LOG` and standard output, and `std::source_location` context.

### Features

- Levels: Trace / Debug / Info / Warn / Error
- Unified formatting compatible with `boost::format`-style placeholders
- Automatic function/file context
- Trace Guard for enter/exit logging

### API Reference

#### Header File

- `utils/brookesia_lib_utils/include/brookesia/lib_utils/log.hpp`

#### Classes

class **Log**

Logging backend facade used by the public logging macros.

#### Public Functions

inline void **print** (int level, const std::source\_location &loc, const char \*tag, const char \*format)

Print a preformatted message.

##### Parameters

- **level** -- *Log* level defined by `BROOKESIA_UTILS_LOG_LEVEL_*`.
- **loc** -- Source location used for metadata extraction.
- **tag** -- *Log* tag string.
- **format** -- Message string passed through without argument formatting.

template<typename ...**Args**>

inline void **print** (int level, const std::source\_location &loc, const char \*tag, const char \*format, *Args*&&... args)

Format a message and print it.

**Template Parameters** **Args** -- Format argument types.

##### Parameters

- **level** -- *Log* level defined by `BROOKESIA_UTILS_LOG_LEVEL_*`.
- **loc** -- Source location used for metadata extraction.

- **tag** -- *Log* tag string.
- **format** -- Boost-format-style message format.
- **args** -- Format arguments forwarded through the type-erased formatter.

### Public Static Functions

static inline *Log* &**getInstance** ()

Get the singleton logging facade.

**Returns** Reference to the shared *Log* instance.

static void **write** (int level, const std::source\_location &loc, const char \*tag, const std::string &message)

Emit a fully formatted message through the selected backend.

#### Parameters

- **level** -- *Log* level defined by BROOKESIA\_UTILS\_LOG\_LEVEL\_\*
- **loc** -- Source location used for metadata extraction.
- **tag** -- *Log* tag string.
- **message** -- Final message body.

static std::string **format\_message** (const char \*format, std::initializer\_list<FormatArg> args)

Format a message using boost::format-style placeholders.

#### Parameters

- **format** -- Format string.
- **args** -- Type-erased format arguments.

**Returns** Formatted message string.

static std::string\_view **extract\_function\_name** (const char \*func\_name)

Extract the display-friendly function name from a source location string.

**Parameters** **func\_name** -- Raw function signature string.

**Returns** Trimmed function name view.

static std::string\_view **extract\_file\_name** (const char \*file\_path)

Extract the leaf file name from a source path.

**Parameters** **file\_path** -- Raw source file path.

**Returns** File name view without directory components.

template<bool **Enabled**>

class **LogTraceGuard**

RAII helper that logs function entry and exit at trace level.

**Template Parameters** **Enabled** -- Compile-time flag that removes all work when false.

### Public Functions

inline **LogTraceGuard** (const void \*this\_ptr = nullptr, const std::source\_location &loc =  
std::source\_location::current(), const char \*tag = esp\_brookesia::lib\_utils::TAG)

Construct a trace guard for the current scope.

#### Parameters

- **this\_ptr** -- Optional object pointer logged for member functions.
- **loc** -- Source location captured for the scope.
- **tag** -- *Log* tag string.

inline ~**LogTraceGuard** ()

*Log* scope exit when trace logging is enabled.

## Macros

### **\_\_BROOKESIA\_LOG\_FORMAT\_THREAD\_NAME**

Helper macros to assemble format string and arguments based on enabled macros. These macros use string literal concatenation (adjacent string literals are automatically concatenated).

### **\_\_BROOKESIA\_LOG\_FORMAT\_FILE\_LINE**

### **\_\_BROOKESIA\_LOG\_FORMAT\_FUNCTION**

### **\_\_BROOKESIA\_LOG\_FORMAT\_MESSAGE**

### **\_\_BROOKESIA\_LOG\_FORMAT\_STRING**

### **\_\_BROOKESIA\_LOG\_ARGS\_THREAD\_NAME** (thread\_name)

### **\_\_BROOKESIA\_LOG\_ARGS\_FILE\_LINE** (file\_name, line)

### **\_\_BROOKESIA\_LOG\_ARGS\_FUNCTION** (func\_name)

### **\_\_BROOKESIA\_LOG\_ARGS\_MESSAGE** (format\_str)

### **\_\_BROOKESIA\_LOG\_ARGS** (thread\_name, file\_name, line, func\_name, format\_str)

### **BROOKESIA\_LOGT\_IMPL** (tag, format, ...)

Macros to simplify logging calls with a fixed tag.

### **BROOKESIA\_LOGD\_IMPL** (tag, format, ...)

### **BROOKESIA\_LOGI\_IMPL** (tag, format, ...)

### **BROOKESIA\_LOGW\_IMPL** (tag, format, ...)

### **BROOKESIA\_LOGE\_IMPL** (tag, format, ...)

### **BROOKESIA\_LOG\_DISABLE\_DEBUG\_TRACE**

Per-file switch that disables `BROOKESIA_LOGT`, `BROOKESIA_LOGD`, and trace-guard helpers.

Users can define this macro before including this header to override the default behavior.

```
#define BROOKESIA_LOG_DISABLE_DEBUG_TRACE 1 // Disable all debug & trace_  
↳ features for this file  
#include "brookesia/lib_utils/log.hpp"
```

### **BROOKESIA\_LOGT** (format, ...)

Compile-time log level filtering macros.

Calls below the configured log level expand to no-ops.

Emit a trace-level log message for the current translation unit tag.

### **BROOKESIA\_LOGD** (format, ...)

Emit a debug-level log message for the current translation unit tag.

### **BROOKESIA\_LOGI** (format, ...)

Emit an info-level log message for the current translation unit tag.

### **BROOKESIA\_LOGW** (format, ...)

Emit a warning-level log message for the current translation unit tag.

**BROOKESIA\_LOGE** (format, ...)

Emit an error-level log message for the current translation unit tag.

**\_BROOKESIA\_LOG\_TRACE\_FORMAT\_ENTER\_WITH\_PTR**

**\_BROOKESIA\_LOG\_TRACE\_FORMAT\_ENTER**

**\_BROOKESIA\_LOG\_TRACE\_FORMAT\_EXIT\_WITH\_PTR**

**\_BROOKESIA\_LOG\_TRACE\_FORMAT\_EXIT**

**\_BROOKESIA\_LOG\_TRACE\_FORMAT\_ENTER\_WITH\_PTR\_STRING**

**\_BROOKESIA\_LOG\_TRACE\_FORMAT\_ENTER\_STRING**

**\_BROOKESIA\_LOG\_TRACE\_FORMAT\_EXIT\_WITH\_PTR\_STRING**

**\_BROOKESIA\_LOG\_TRACE\_FORMAT\_EXIT\_STRING**

**\_BROOKESIA\_LOG\_TRACE\_ARGS\_WITH\_PTR** (thread\_name, file\_name, line, func\_name, this\_ptr)

**\_BROOKESIA\_LOG\_CONCAT** (a, b)

Create a scope guard that logs entry and exit for the current function.

**BROOKESIA\_LOG\_CONCAT** (a, b)

**BROOKESIA\_LOG\_TRACE\_GUARD** ()

**BROOKESIA\_LOG\_TRACE\_GUARD\_WITH\_THIS** ()

Create a scope guard that logs entry and exit and includes `this`.

## Error Checking

Public header: `#include "brookesia/lib_utils/check.hpp"`

**Overview** `check` provides unified check macros for invalid parameters, failed expressions, exceptions, error codes, and out-of-range values.

## Features

- Covers common cases: `nullptr`, `booleans`, `std::exception`, `esp_err_t`, `ranges`
- Multiple failure modes: `code blocks`, `return`, `exit`, `goto`
- Integrates with logging for context
- Configurable failure policy (no-op, log, assert)

## API Reference

### Header File

- `utils/brookesia_lib_utils/include/brookesia/lib_utils/check.hpp`

**Macros****BROOKESIA\_CHECK\_NULL\_EXECUTE** (*ptr*, *process\_code*, ...)

Execute a fallback code block when a pointer is null.

**Parameters**

- **ptr** -- Pointer expression to test.
- **process\_code** -- Code block executed when *ptr* is `nullptr`.
- ... -- Compatibility arguments kept for legacy call sites and ignored in this mode.

**BROOKESIA\_CHECK\_FALSE\_EXECUTE** (*value*, *process\_code*, ...)Execute a fallback code block when an expression evaluates to `false`.**Parameters**

- **value** -- Expression to test.
- **process\_code** -- Code block executed when *value* converts to `false`.
- ... -- Compatibility arguments kept for legacy call sites and ignored in this mode.

**BROOKESIA\_CHECK\_EXCEPTION\_EXECUTE** (*expression*, *process\_code*, ...)Execute a fallback code block when an expression throws `std::exception`.**Parameters**

- **expression** -- Expression to evaluate inside a `try` block.
- **process\_code** -- Code block executed when *expression* throws `std::exception`.
- ... -- Compatibility arguments kept for legacy call sites and ignored in this mode.

**BROOKESIA\_CHECK\_ESP\_ERR\_EXECUTE** (*esp\_result*, *process\_code*, ...)Execute a fallback code block when an ESP-IDF error code is not `ESP_OK`.**Parameters**

- **esp\_result** -- Expression that yields an ESP-IDF error code.
- **process\_code** -- Code block executed when *esp\_result* is not `ESP_OK`.
- ... -- Compatibility arguments kept for legacy call sites and ignored in this mode.

**BROOKESIA\_CHECK\_OUT\_RANGE\_EXECUTE** (*value*, *min*, *max*, *process\_code*, ...)Execute a fallback code block when a value is outside the inclusive range [*min*, *max*].**Parameters**

- **value** -- Expression to test.
- **min** -- Inclusive lower bound.
- **max** -- Inclusive upper bound.
- **process\_code** -- Code block executed when *value* is outside the range.
- ... -- Compatibility arguments kept for legacy call sites and ignored in this mode.

**BROOKESIA\_CHECK\_NULL\_RETURN** (*value*, *ret*, *fmt*, ...)

Return a value when a pointer is null.

**Parameters**

- **value** -- Pointer expression to test.
- **ret** -- Value returned when *value* is `nullptr`.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for *fmt*.

**BROOKESIA\_CHECK\_NULL\_EXIT** (*value*, *fmt*, ...)Return from the current `void` function when a pointer is null.**Parameters**

- **value** -- Pointer expression to test.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for *fmt*.

**BROOKESIA\_CHECK\_NULL\_GOTO** (*value*, *goto\_tag*, *fmt*, ...)

Jump to a label when a pointer is null.

**Parameters**

- **value** -- Pointer expression to test.
- **goto\_tag** -- Label jumped to when `value` is `nullptr`.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_FALSE\_RETURN** (`value`, `ret`, `fmt`, ...)

Return a value when an expression evaluates to `false`.

**Parameters**

- **value** -- Expression to test.
- **ret** -- Value returned when `value` converts to `false`.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_FALSE\_EXIT** (`value`, `fmt`, ...)

Return from the current `void` function when an expression evaluates to `false`.

**Parameters**

- **value** -- Expression to test.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_FALSE\_GOTO** (`value`, `goto_tag`, `fmt`, ...)

Jump to a label when an expression evaluates to `false`.

**Parameters**

- **value** -- Expression to test.
- **goto\_tag** -- Label jumped to when `value` converts to `false`.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_ESP\_ERR\_RETURN** (`value`, `ret`, `fmt`, ...)

Return a value when an ESP-IDF error code is not `ESP_OK`.

**Parameters**

- **value** -- Expression that yields an ESP-IDF error code.
- **ret** -- Value returned when `value` is not `ESP_OK`.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_ESP\_ERR\_EXIT** (`value`, `fmt`, ...)

Return from the current `void` function when an ESP-IDF error code is not `ESP_OK`.

**Parameters**

- **value** -- Expression that yields an ESP-IDF error code.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_ESP\_ERR\_GOTO** (`value`, `goto_tag`, `fmt`, ...)

Jump to a label when an ESP-IDF error code is not `ESP_OK`.

**Parameters**

- **value** -- Expression that yields an ESP-IDF error code.
- **goto\_tag** -- Label jumped to when `value` is not `ESP_OK`.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_EXCEPTION\_RETURN** (`expression`, `ret`, `fmt`, ...)

Return a value when an expression throws `std::exception`.

**Parameters**

- **expression** -- Expression to evaluate inside a `try` block.

- **ret** -- Value returned when `expression` throws `std::exception`.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_EXCEPTION\_EXIT** (`expression`, `fmt`, ...)

Return from the current `void` function when an expression throws `std::exception`.

**Parameters**

- **expression** -- Expression to evaluate inside a `try` block.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_EXCEPTION\_GOTO** (`expression`, `goto_tag`, `fmt`, ...)

Jump to a label when an expression throws `std::exception`.

**Parameters**

- **expression** -- Expression to evaluate inside a `try` block.
- **goto\_tag** -- Label jumped to when `expression` throws `std::exception`.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_OUT\_RANGE** (`value`, `min`, `max`, `fmt`, ...)

Log when a value is outside the inclusive range [`min`, `max`].

**Parameters**

- **value** -- Expression to test.
- **min** -- Inclusive lower bound.
- **max** -- Inclusive upper bound.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_OUT\_RANGE\_RETURN** (`value`, `min`, `max`, `ret`, `fmt`, ...)

Return a value when a value is outside the inclusive range [`min`, `max`].

**Parameters**

- **value** -- Expression to test.
- **min** -- Inclusive lower bound.
- **max** -- Inclusive upper bound.
- **ret** -- Value returned when `value` is outside the range.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_OUT\_RANGE\_EXIT** (`value`, `min`, `max`, `fmt`, ...)

Return from the current `void` function when a value is outside the inclusive range [`min`, `max`].

**Parameters**

- **value** -- Expression to test.
- **min** -- Inclusive lower bound.
- **max** -- Inclusive upper bound.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

**BROOKESIA\_CHECK\_OUT\_RANGE\_GOTO** (`value`, `min`, `max`, `goto_tag`, `fmt`, ...)

Jump to a label when a value is outside the inclusive range [`min`, `max`].

**Parameters**

- **value** -- Expression to test.
- **min** -- Inclusive lower bound.
- **max** -- Inclusive upper bound.
- **goto\_tag** -- Label jumped to when `value` is outside the range.
- **fmt** -- User log format string forwarded to `BROOKESIA_LOGE`.
- ... -- Optional format arguments for `fmt`.

## Function Guard

Public header: `#include "brookesia/lib_utils/function_guard.hpp"`

**Overview** *FunctionGuard* is an RAII guard that runs cleanup at scope exit to simplify resource release and rollback.

### Features

- Automatic callback on scope exit; optional *release* to cancel
- Move semantics for transferring guards
- Catches *std::exception* during cleanup to avoid breaking teardown

### API Reference

#### Header File

- [utils/brookesia\\_lib\\_utils/include/brookesia/lib\\_utils/function\\_guard.hpp](#)

#### Classes

template<typename **T**, typename ...**Args**>

class **FunctionGuard**

RAII helper that invokes a callable on destruction unless released.

---

**Note:** The destructor suppresses `boost::thread_interrupted` and prints other `std::exception` failures to `stdout`, so cleanup code should still be kept lightweight.

---

#### Template Parameters

- **T** -- Callable type stored by the guard.
- **Args** -- Argument types forwarded to the callable when the guard is destroyed.

#### Public Functions

inline **FunctionGuard** (*T* func, *Args*&&... args)

Construct a guard for a deferred callable invocation.

##### Parameters

- **func** -- Callable executed in the destructor unless *release()* is called.
- **args** -- Arguments stored and forwarded to *func* during destruction.

inline void **release** ()

Disable the deferred invocation.

After calling this method, destroying the guard no longer executes the stored callable.

#### Describe Helpers

Public header: `#include "brookesia/lib_utils/describe_helpers.hpp"`

**Overview** *describe\_helpers* uses Boost.Describe and Boost.JSON for object description plus serialization and de-serialization, reducing boilerplate for structs, enums, and complex types.

## Features

- Reflection for enums and struct members
- JSON encode/decode for strings, numbers, bools, containers, optionals, etc.
- Handles *variant*, *optional*, *map*, *vector*, and related composites
- Debug-friendly description output

## API Reference

### Header File

- [utils/brookesia\\_lib\\_utils/include/brookesia/lib\\_utils/describe\\_helpers.hpp](#)

### Classes

class **DescribeFormatManager**

Singleton that stores the global default DescribeOutputFormat.

#### Public Functions

inline void **set\_format** (const DescribeOutputFormat &fmt)

Set the process-wide default output format.

**Parameters** **fmt** -- New global format.

inline const DescribeOutputFormat &**get\_format** () const

Get the current process-wide default output format.

**Returns** Reference to the active global format.

inline void **reset\_to\_default** ()

Restore the built-in default output format.

#### Public Static Functions

static inline *DescribeFormatManager* &**instance** ()

Get the singleton instance.

**Returns** Reference to the global format manager.

### Macros

**BROOKESIA\_DESCRIBE\_STRUCT** (C, Bases, Members)

Register a struct with Boost.Describe reflection.

**BROOKESIA\_DESCRIBE\_ENUM** (C, ...)

Register an enum with Boost.Describe reflection.

**BROOKESIA\_DESCRIBE\_ENUM\_TO\_STR** (value)

Convert a described enum value to a string.

**BROOKESIA\_DESCRIBE\_ENUM\_TO\_NUM** (value)

Convert an enum value to its underlying integer value.

**BROOKESIA\_DESCRIBE\_NUM\_TO\_ENUM** (number, ret\_value)

Convert an integer value to a described enum value.

**BROOKESIA\_DESCRIBE\_STR\_TO\_ENUM** (str, ret\_value)

Convert an enumerator name to a described enum value.

**BROOKESIA\_DESCRIBE\_STR\_TO\_ENUM\_FLEXIBLE** (str, ret\_value)

Convert an enumerator name or snake\_case alias to a described enum value.

**BROOKESIA\_DESCRIBE\_TO\_JSON** (value)

Convert a supported value to `boost::json::value`.

**BROOKESIA\_DESCRIBE\_FROM\_JSON** (json\_value, ret\_value)

Convert a `boost::json::value` to a supported C++ value.

**BROOKESIA\_DESCRIBE\_JSON\_SERIALIZE** (value)

Serialize a supported value to JSON text.

**BROOKESIA\_DESCRIBE\_JSON\_DESERIALIZE** (str, ret\_value)

Deserialize JSON text into a supported C++ value.

**BROOKESIA\_DESCRIBE\_FORMAT\_VERBOSE**

Verbose multi-line formatting preset.

**BROOKESIA\_DESCRIBE\_FORMAT\_JSON**

JSON-like formatting preset.

**BROOKESIA\_DESCRIBE\_FORMAT\_COMPACT**

Compact single-line formatting preset.

**BROOKESIA\_DESCRIBE\_FORMAT\_DEFAULT**

Default human-readable formatting preset.

**BROOKESIA\_DESCRIBE\_FORMAT\_PYTHON**

Python-dict-inspired formatting preset.

**BROOKESIA\_DESCRIBE\_FORMAT\_CPP**

C++ designated-initializer-inspired formatting preset.

**BROOKESIA\_DESCRIBE\_SET\_GLOBAL\_FORMAT** (fmt)

Set the global default string formatting preset.

**BROOKESIA\_DESCRIBE\_GET\_GLOBAL\_FORMAT** ()

Get the global default string formatting preset.

**BROOKESIA\_DESCRIBE\_RESET\_GLOBAL\_FORMAT** ()

Reset the global default string formatting preset.

**BROOKESIA\_DESCRIBE\_TO\_STR** (value)

Convert a supported value to a string with the global default format.

**BROOKESIA\_DESCRIBE\_TO\_STR\_WITH\_FMT** (value, fmt)

Convert a supported value to a string with an explicit format preset.

## 2.1.5 Debug Tools

### Memory Profiler

Public header: `#include "brookesia/lib_utils/memory_profiler.hpp"`

**Overview** *memory\_profiler* samples heap usage, tracks peaks, and supports threshold alerts for runtime memory monitoring.

#### Features

- Current heap, historical peak, and statistics
- Periodic profiling and manual snapshots
- Threshold callbacks when limits are exceeded
- Optional integration with *TaskScheduler* for periodic sampling

#### API Reference

##### Header File

- [utils/brookesia\\_lib\\_utils/include/brookesia/lib\\_utils/memory\\_profiler.hpp](#)

##### Classes

class **MemoryProfiler**

Heap memory profiler for internal RAM and PSRAM usage.

This class provides a C++ interface to monitor ESP32 heap memory usage, including internal SRAM and external PSRAM. It integrates naturally with *TaskScheduler* for periodic profiling.

##### Public Types

enum class **ThresholdType**

Metric used when registering threshold callbacks.

*Values:*

enumerator **TotalFree**

Total free memory in bytes.

enumerator **InternalFree**

Internal SRAM free memory in bytes.

enumerator **ExternalFree**

External PSRAM free memory in bytes.

enumerator **TotalFreePercent**

Total free memory as a percentage.

enumerator **InternalFreePercent**

Internal SRAM free memory as a percentage.

enumerator **ExternalFreePercent**

External PSRAM free memory as a percentage.

enumerator **TotalLargestFreeBlock**

Largest free block across all heaps, in bytes.

enumerator **InternalLargestFreeBlock**

Largest internal SRAM free block, in bytes.

enumerator **ExternalLargestFreeBlock**

Largest external PSRAM free block, in bytes.

enumerator **MinTotalFree**

Minimum total free memory in bytes.

enumerator **MinInternalFree**

Minimum internal SRAM free memory in bytes.

enumerator **MinExternalFree**

Minimum external PSRAM free memory in bytes.

enumerator **MinTotalFreePercent**

Minimum total free memory as a percentage.

enumerator **MinInternalFreePercent**

Minimum internal SRAM free memory as a percentage.

enumerator **MinExternalFreePercent**

Minimum external PSRAM free memory as a percentage.

enumerator **MinTotalLargestFreeBlock**

Minimum largest free block across all heaps, in bytes.

enumerator **MinInternalLargestFreeBlock**

Minimum largest internal SRAM free block, in bytes.

enumerator **MinExternalLargestFreeBlock**

Minimum largest external PSRAM free block, in bytes.

using **ProfilingSignal** = boost::signals2::signal<void(const *ProfileSnapshot*&)>

Signal type emitted for each profiling snapshot.

using **ProfilingSignalSlot** = *ProfilingSignal*::slot\_type

Slot type accepted by *connect\_profiling\_signal()*.

using **ThresholdSignal** = boost::signals2::signal<void(const *ProfileSnapshot*&)>

Signal type emitted when a threshold condition is met.

using **ThresholdSignalSlot** = *ThresholdSignal*::slot\_type

Slot type accepted by *connect\_threshold\_signal()*.

using **SignalConnection** = boost::signals2::scoped\_connection

Signal connection type.

---

**Note:** This is an RAII smart handle for managing the lifetime of callbacks. When this object is destroyed, the corresponding callback is automatically disconnected. It is recommended to use `std::move()` to transfer ownership for manual management of the connection lifetime.

---

### Public Functions

**MemoryProfiler**() = default

Construct an idle memory profiler.

**MemoryProfiler**(const *MemoryProfiler*&) = delete

Copy construction is not supported.

*MemoryProfiler* &operator=(const *MemoryProfiler*&) = delete

Copy assignment is not supported.

**MemoryProfiler**(*MemoryProfiler*&&) = delete

Move construction is not supported.

*MemoryProfiler* &operator=(*MemoryProfiler*&&) = delete

Move assignment is not supported.

bool **configure\_profiling**(const *ProfilingConfig* &config)

Update the profiling configuration.

**Parameters** **config** -- New profiler configuration.

**Returns** `true` after the configuration is stored.

inline *ProfilingConfig* **get\_profiling\_config**() const

Get the current profiling configuration.

**Returns** Active *ProfilingConfig*.

bool **start\_profiling**(std::shared\_ptr<*TaskScheduler*> scheduler, uint32\_t period\_ms = 0)

Start periodic profiling with a task scheduler.

**Parameters**

- **scheduler** -- Scheduler used to execute periodic sampling.
- **period\_ms** -- Sampling period in milliseconds. Pass 0 to use `config.sample_interval_ms`.

**Returns** `true` on success, or `false` if startup fails.

void **stop\_profiling**()

Stop periodic profiling.

void **reset\_profiling**()

Reset captured snapshots and registered callbacks without changing configuration.

inline bool **is\_profiling** () const

Check whether periodic profiling is active.

**Returns** `true` when a profiling task is active, or `false` otherwise.

inline std::shared\_ptr<*ProfileSnapshot*> **get\_profiling\_latest\_snapshot** () const

Get the latest captured snapshot.

**Returns** Shared pointer to the latest snapshot, or `nullptr` if no snapshot is available.

*SignalConnection* **connect\_profiling\_signal** (*ProfilingSignalSlot* slot)

Subscribe to every profiling snapshot.

**Parameters** `slot` -- Callback invoked whenever a new snapshot is produced.

**Returns** RAII connection handle for the registered callback.

*SignalConnection* **connect\_threshold\_signal** (*ThresholdType* type, uint32\_t threshold\_value, *ThresholdSignalSlot* slot)

Subscribe to threshold events for a specific metric.

**Parameters**

- **type** -- Metric to monitor.
- **threshold\_value** -- Threshold value interpreted according to `type`.
- **slot** -- Callback invoked when the threshold is met by a snapshot.

**Returns** RAII connection handle for the registered callback.

## Public Static Functions

static inline *MemoryProfiler* &**get\_instance** ()

Get the process-wide singleton instance.

**Returns** Reference to the singleton profiler.

static std::shared\_ptr<*ProfileSnapshot*> **take\_snapshot** (*ProfileSnapshot* \*last\_snapshot = nullptr)

Capture the current memory state.

**Parameters** `last_snapshot` -- Previous snapshot used to update running statistics, or `nullptr` for the first sample.

**Returns** Shared pointer to the new snapshot, or `nullptr` if allocation fails.

static void **print\_snapshot** (const *ProfileSnapshot* &snapshot)

Print a formatted snapshot summary to the log.

**Parameters** `snapshot` -- Snapshot to print.

struct **HeapInfo**

Snapshot of a single heap region.

## Public Members

size\_t **total\_size** = 0

Total heap size in bytes.

size\_t **free\_size** = 0

Free heap size in bytes.

size\_t **largest\_free\_block** = 0

Size of the largest free block in bytes.

size\_t **free\_percent** = 0

Free memory ratio expressed as a percentage.

size\_t **used\_percent** = 0

Used memory ratio expressed as a percentage.

struct **MemoryInfo**

Aggregated memory information for all monitored heap regions.

### **Public Members**

*HeapInfo* **internal**

Internal SRAM heap information (MALLOC\_CAP\_INTERNAL).

*HeapInfo* **external**

External PSRAM heap information (MALLOC\_CAP\_SPIRAM).

size\_t **total\_size** = 0

Total heap size in bytes across all monitored heaps.

size\_t **total\_free** = 0

Total free memory in bytes across all monitored heaps.

size\_t **total\_free\_percent** = 0

Total free memory ratio expressed as a percentage.

size\_t **total\_largest\_free\_block** = 0

Largest free block in bytes across all monitored heaps.

struct **ProfileSnapshot**

Memory profile snapshot captured at one point in time.

### **Public Members**

std::chrono::system\_clock::time\_point **timestamp**

Snapshot capture time.

*MemoryInfo* **memory**

Instantaneous memory information.

*Statistics* **stats**

Running statistics after this snapshot.

struct **ProfilingConfig**

Configuration for periodic memory profiling.

### Public Members

uint32\_t **sample\_interval\_ms** = 5000

Sampling interval in milliseconds.

bool **enable\_auto\_logging** = true

Whether to log each generated snapshot automatically.

struct **Statistics**

Running minimum statistics accumulated across snapshots.

### Public Members

size\_t **sample\_count** = 0

Number of snapshots processed.

size\_t **min\_total\_free** = 0

Minimum total free memory observed, in bytes.

size\_t **min\_internal\_free** = 0

Minimum internal free memory observed, in bytes.

size\_t **min\_external\_free** = 0

Minimum external free memory observed, in bytes.

size\_t **min\_total\_free\_percent** = 0

Minimum total free percentage observed.

size\_t **min\_internal\_free\_percent** = 0

Minimum internal free percentage observed.

size\_t **min\_external\_free\_percent** = 0

Minimum external free percentage observed.

size\_t **min\_total\_largest\_free\_block** = 0

Minimum total largest free block observed, in bytes.

size\_t **min\_internal\_largest\_free\_block** = 0

Minimum internal largest free block observed, in bytes.

size\_t **min\_external\_largest\_free\_block** = 0

Minimum external largest free block observed, in bytes.

### Thread Profiler

Public header: #include "brookesia/lib\_utils/thread\_profiler.hpp"

**Overview** *thread\_profiler* samples task and thread state and CPU usage, helping analyze system load and hot tasks.

## Features

- Task status, CPU usage, stack info
- Periodic profiling and manual snapshots
- Sorting, filtering, and thresholds
- Optional *TaskScheduler* integration

## API Reference

### Header File

- `utils/brookesia_lib_utils/include/brookesia/lib_utils/thread_profiler.hpp`

### Classes

class **ThreadProfiler**

FreeRTOS task profiler for CPU and stack usage monitoring.

This class provides a C++ interface to monitor FreeRTOS task CPU usage, stack usage, and other runtime statistics. It integrates naturally with *TaskScheduler* for periodic profiling.

### Public Types

enum class **TaskState**

Task state mirrored from FreeRTOS `eTaskState`.

*Values:*

enumerator **Running**

`eRunning`.

enumerator **Ready**

`eReady`.

enumerator **Blocked**

`eBlocked`.

enumerator **Suspended**

`eSuspended`.

enumerator **Deleted**

`eDeleted`.

enumerator **Invalid**

`eInvalid`.

enum class **TaskStatus**

Presence status of a task across two samples.

*Values:*

enumerator **Normal**

Task exists in both samples.

enumerator **Created**

Task appeared during the measurement window.

enumerator **Deleted**

Task disappeared during the measurement window.

enum class **PrimarySortBy**

Optional primary sort criterion.

*Values:*

enumerator **None**

Disable primary sorting.

enumerator **CoreId**

Sort by CPU core ID first.

enum class **SecondarySortBy**

Secondary sort criterion applied within the primary ordering.

*Values:*

enumerator **CpuPercent**

Sort by CPU usage percentage in descending order.

enumerator **Priority**

Sort by task priority in descending order.

enumerator **StackUsage**

Sort by stack high-water mark in ascending order.

enumerator **Name**

Sort by task name alphabetically.

enum class **ThresholdType**

Metric used for threshold filtering and callbacks.

*Values:*

enumerator **CpuPercent**

Filter by CPU usage percentage.

enumerator **Priority**

Filter by task priority.

enumerator **StackUsage**

Filter by stack high-water mark.

using **ProfilingSignalSlot** = std::function<void(const *ProfileSnapshot*&)>

Callback type accepted by `connect_profiling_signal()`.

using **ThresholdSignalSlot** = std::function<void(const std::vector<*TaskInfo*>&)>

Callback type accepted by `connect_threshold_signal()`.

using **ProfilingSignal** = boost::signals2::signal<void(const *ProfileSnapshot*&)>

Signal type emitted for each profiling snapshot.

using **ThresholdSignal** = boost::signals2::signal<void(const std::vector<*TaskInfo*>&)>

Signal type emitted when threshold matches are found.

using **SignalConnection** = boost::signals2::scoped\_connection

Signal connection type.

---

**Note:** This is an RAII smart handle for managing the lifetime of callbacks. When this object is destroyed, the corresponding callback is automatically disconnected. It is recommended to use `std::move()` to transfer ownership for manual management of the connection lifetime.

---

## Public Functions

**ThreadProfiler** (const *ThreadProfiler*&) = delete

Copy construction is not supported.

*ThreadProfiler* &**operator=**(const *ThreadProfiler*&) = delete

Copy assignment is not supported.

**ThreadProfiler** (*ThreadProfiler*&&) = delete

Move construction is not supported.

*ThreadProfiler* &**operator=**(*ThreadProfiler*&&) = delete

Move assignment is not supported.

bool **configure\_profiling** (const *ProfilingConfig* &config)

Update the profiling configuration.

**Parameters** **config** -- New profiler configuration.

**Returns** `true` after the configuration is stored.

*ProfilingConfig* **get\_profiling\_config** () const

Get the current profiling configuration.

**Returns** Active *ProfilingConfig*.

bool **start\_profiling** (std::shared\_ptr<*TaskScheduler*> scheduler, uint32\_t sampling\_duration\_ms = 0, uint32\_t profiling\_interval\_ms = 0)

Start periodic thread profiling using a task scheduler.

### Parameters

- **scheduler** -- Scheduler used to run the two-stage sampling jobs. The caller must keep it alive until `stop_profiling()` returns.
- **sampling\_duration\_ms** -- Time between the first and second sample in milliseconds. Pass 0 to use `config.sampling_duration_ms`.
- **profiling\_interval\_ms** -- Period between profiling rounds in milliseconds. Pass 0 to use `config.profiling_interval_ms`.

**Returns** `true` on success, or `false` if profiling cannot be started.

void **stop\_profiling** ()

Stop periodic profiling.

void **reset\_profiling** ()

Reset captured data and registered callbacks without changing configuration.

inline bool **is\_profiling** () const

Check whether periodic profiling is active.

**Returns** `true` when profiling tasks are active, or `false` otherwise.

inline std::shared\_ptr<*ProfileSnapshot*> **get\_profiling\_latest\_snapshot** ()

Get the latest captured profiling snapshot.

**Returns** Shared pointer to the latest snapshot, or `nullptr` if none is available.

*SignalConnection* **connect\_profiling\_signal** (*ProfilingSignalSlot* slot)

Subscribe to every generated profiling snapshot.

**Parameters** `slot` -- Callback invoked whenever a new snapshot is produced.

**Returns** RAI connection handle for the registered callback.

*SignalConnection* **connect\_threshold\_signal** (*ThresholdType* type, uint32\_t threshold\_value, *ThresholdSignalSlot* slot)

Subscribe to threshold matches for a specific metric.

**Parameters**

- `type` -- Metric to monitor.
- `threshold_value` -- Threshold value interpreted according to `type`.
- `slot` -- Callback invoked with the tasks that matched the threshold.

**Returns** RAI connection handle for the registered callback.

## Public Static Functions

static inline *ThreadProfiler* &**get\_instance** ()

Get the process-wide singleton instance.

**Returns** Reference to the singleton profiler.

static std::shared\_ptr<*SampleResult*> **sample\_tasks** ()

Capture a single raw scheduler sample.

**Returns** Shared pointer to the captured sample, or `nullptr` on failure.

static std::shared\_ptr<*ProfileSnapshot*> **take\_snapshot** (const *SampleResult* &start\_result, const *SampleResult* &end\_result)

Build a profiling snapshot from two raw samples.

**Parameters**

- `start_result` -- Sample taken at the beginning of the measurement window.
- `end_result` -- Sample taken at the end of the measurement window.

**Returns** Shared pointer to the computed snapshot, or `nullptr` on failure.

static void **sort\_tasks** (std::vector<*TaskInfo*> &tasks, *PrimarySortBy* primary\_sort = *PrimarySortBy::CoreId*, *SecondarySortBy* secondary\_sort = *SecondarySortBy::CpuPercent*)

Sort task entries in place.

---

**Note:** The tasks will be sorted by the primary sort criteria first, then by the secondary sort criteria.

---

**Parameters**

- **tasks** -- Task vector to sort in place.
- **primary\_sort** -- Optional primary sort criterion.
- **secondary\_sort** -- Secondary sort criterion.

```
static void print_snapshot (const ProfileSnapshot &snapshot, PrimarySortBy primary_sort =  
                             PrimarySortBy::CoreId, SecondarySortBy secondary_sort =  
                             SecondarySortBy::CpuPercent)
```

Print a formatted snapshot table to the log.

**Parameters**

- **snapshot** -- Snapshot to print.
- **primary\_sort** -- Primary sort criterion shown in the first sort column.
- **secondary\_sort** -- Secondary sort criterion shown in the second sort column.

```
static bool get_task_by_name (const ProfileSnapshot &snapshot, const std::string &name, TaskInfo  
                             &task)
```

Find a task by name inside a snapshot.

**Parameters**

- **snapshot** -- Snapshot to search.
- **name** -- Task name to search for.
- **task** -- Output object that receives the matching task info.

**Returns** `true` if a matching task is found, or `false` otherwise.

```
static std::vector<TaskInfo> get_tasks_above_threshold (const ProfileSnapshot &snapshot,  
                                                       ThresholdType type, uint32_t  
                                                       threshold_value)
```

Collect tasks whose metric meets a threshold.

---

**Note:** For `CpuPercent`: returns tasks with CPU  $\geq$  `threshold_value` For `Priority`: returns tasks with priority  $\geq$  `threshold_value` For `StackUsage`: returns tasks with stack HWM  $\leq$  `threshold_value` (lower = more used)

---

**Parameters**

- **snapshot** -- Snapshot to inspect.
- **type** -- Threshold metric to apply.
- **threshold\_value** -- Threshold value interpreted according to `type`.

**Returns** Vector containing every task that satisfies the threshold.

```
struct ProfileSnapshot
```

Snapshot of all sampled task information.

**Public Members**

```
std::chrono::system_clock::time_point timestamp = {}
```

Snapshot capture time.

```
std::vector<TaskInfo> tasks
```

Task data for every sampled task.

```
Statistics stats = {}
```

Aggregate snapshot statistics.

uint32\_t **total\_runtime** = 0

Total runtime counter captured in the ending sample.

struct **ProfilingConfig**

Configuration for periodic thread profiling.

### Public Members

uint32\_t **sampling\_duration\_ms** = 1000

Delay between the start and end sample, in milliseconds.

uint32\_t **profiling\_interval\_ms** = 5000

Interval between profiling rounds, in milliseconds.

*PrimarySortBy* **primary\_sort** = *PrimarySortBy::CoreId*

Primary sort criterion.

*SecondarySortBy* **secondary\_sort** = *SecondarySortBy::CpuPercent*

Secondary sort criterion.

bool **enable\_auto\_logging** = true

Whether to log each generated snapshot automatically.

struct **SampleResult**

Raw sample captured at a single instant.

### Public Members

std::chrono::system\_clock::time\_point **timestamp**

Sample capture time.

std::vector<TaskStatus\_t> **task\_status**

Raw FreeRTOS task status array.

uint32\_t **runtime**

Raw total runtime counter.

struct **Statistics**

Aggregate statistics for a profiling snapshot.

### Public Members

size\_t **total\_tasks** = 0

Total number of tasks in the snapshot.

size\_t **running\_tasks** = 0

Number of running tasks.

size\_t **blocked\_tasks** = 0

Number of blocked tasks.

size\_t **suspended\_tasks** = 0

Number of suspended tasks.

uint32\_t **total\_cpu\_percent** = 0

Aggregate CPU usage percentage.

uint32\_t **sample\_duration\_ms** = 0

Measurement window duration in milliseconds.

struct **TaskInfo**

Profile data for a single FreeRTOS task.

### Public Members

std::string **name**

Task name.

TaskHandle\_t **handle** = nullptr

FreeRTOS task handle.

*TaskState* **state** = *TaskState::Invalid*

Current task state.

uint32\_t **priority** = 0

Current task priority.

BaseType\_t **core\_id** = -1

Bound CPU core ID, or -1 if unbound.

uint32\_t **stack\_high\_water\_mark** = 0

Stack high-water mark in bytes.

bool **is\_stack\_external** = false

Whether the stack is allocated in external RAM.

uint32\_t **runtime\_counter** = 0

Raw runtime counter captured in the ending sample.

uint32\_t **elapsed\_time** = 0

Runtime counter delta during the measurement window.

```
uint32_t cpu_percent = 0
```

CPU usage percentage over the measurement window.

```
TaskStatus status = TaskStatus::Normal
```

Presence status relative to the two samples.

## Time Profiler

Public header: `#include "brookesia/lib_utils/time_profiler.hpp"`

**Overview** *time\_profiler* provides hierarchical timing for scopes and events to locate bottlenecks and hot paths.

## Features

- Scope and event timing models
- Hierarchical reports with customizable output
- Per-thread sampling
- Convenience macros for instrumentation

**Warning:** On **ESP32-P4** with `CONFIG_SPIRAM_XIP_FROM_PSRAM=y`, *time\_profiler* may crash the system.

## API Reference

### Header File

- `utils/brookesia_lib_utils/include/brookesia/lib_utils/time_profiler.hpp`

### Classes

class **TimeProfiler**

Hierarchical time profiler for scoped code regions and named events.

This class provides hierarchical time profiling capabilities with support for nested scopes, cross-thread events, and detailed statistics reporting.

### Public Functions

void **set\_format\_options** (const *FormatOptions* &options)

Set formatting options used by *report()*.

**Parameters** *options* -- [in] Format options to apply.

void **enter\_scope** (const std::string &name)

Enter a named profiling scope.

Should be paired with *leave\_scope()*. Prefer using `BROOKESIA_TIME_PROFILER_SCOPE` macro for automatic scope management.

**Parameters** *name* -- [in] Scope name to record.

void **leave\_scope** ()

Leave the current profiling scope.

Records the elapsed time since the matching *enter\_scope()* call.

void **start\_event** (const std::string &name)

Start timing a named event.

Used for timing events that may span across function boundaries or threads. Must be paired with *end\_event()* with the same name.

**Parameters** **name** -- **[in]** Event name to start.

void **end\_event** (const std::string &name)

End timing a named event.

Records the elapsed time since the corresponding *start\_event()* call.

**Parameters** **name** -- **[in]** Event name to stop.

*Statistics* **get\_statistics** () const

Build a structured snapshot of the collected profiling data.

Returns a structured representation of all profiling data.

**Returns** *Statistics* structure containing all profiling information.

void **report** ()

Generate and log a formatted profiling report.

Prints a hierarchical report of all recorded timings to the log. This method internally calls *get\_statistics()* and formats the output.

void **clear** ()

Clear all recorded profiling data.

Resets all timing statistics and clears the profiling tree.

## Public Static Functions

static inline *TimeProfiler* &**get\_instance** ()

Get the singleton instance of *TimeProfiler*.

**Returns** Reference to the singleton *TimeProfiler* instance

struct **FormatOptions**

Formatting options used by *report()*.

## Public Types

enum class **SortBy**

Sort order for sibling profiling nodes.

*Values:*

enumerator **TotalDesc**

Sort by total time descending.

enumerator **NameAsc**

Sort by name ascending.

enumerator **None**

No sorting.

enum class **TimeUnit**

Time unit used for formatted output and exported statistics.

*Values:*

enumerator **Microseconds**

Display in microseconds.

enumerator **Milliseconds**

Display in milliseconds.

enumerator **Seconds**

Display in seconds.

### **Public Members**

int **name\_width** = 32

Width of name column.

int **calls\_width** = 6

Width of calls column.

int **num\_width** = 10

Width of numeric columns.

int **percent\_width** = 7

Width of percentage column.

int **precision** = 2

Decimal precision for numbers.

bool **use\_unicode** = false

Use ASCII characters to ensure alignment.

bool **show\_percentages** = true

Show percentage columns.

bool **use\_color** = false

Use ANSI color codes in output.

*SortBy* **sort\_by** = *SortBy::TotalDesc*

Sort order for output.

*TimeUnit* **time\_unit** = *TimeUnit::Milliseconds*

Time unit for display.

**struct Node**

Internal tree node representing one profiled scope.

**Public Members**

std::string **name**

Name of this profiling scope.

double **total** = 0.0

Total time spent in this scope.

size\_t **count** = 0

Number of times this scope was entered.

double **min** = std::numeric\_limits<double>::infinity()

Minimum time recorded.

double **max** = 0.0

Maximum time recorded.

std::map<std::string, std::unique\_ptr<Node>> **children**

Child scopes.

*Node* \***parent** = nullptr

Parent scope.

**struct NodeStatistics**

Aggregated statistics for a single profiling node.

**Public Members**

std::string **name**

Name of the profiling scope.

size\_t **count** = 0

Number of times this scope was entered.

double **total** = 0.0

Total time spent in this scope.

double **self\_time** = 0.0

Time spent in this scope excluding children.

double **avg** = 0.0

Average time per call.

double **min** = 0.0

Minimum time recorded.

double **max** = 0.0

Maximum time recorded.

double **pct\_parent** = 0.0

Percentage of parent's total time.

double **pct\_total** = 0.0

Percentage of overall total time.

std::vector<*NodeStatistics*> **children**

Child node statistics.

struct **Statistics**

Structured report returned by *get\_statistics()*.

### Public Members

std::string **unit\_name**

Time unit name (e.g., "ms", "us", "s")

double **overall\_total** = 0.0

Overall total time.

std::vector<*NodeStatistics*> **root\_children**

Root level node statistics.

class **TimeProfilerScope**

RAII wrapper that profiles the lifetime of a C++ scope.

Automatically calls *enter\_scope()* on construction and *leave\_scope()* on destruction. Use the `BROOKESIA_TIME_PROFILER_SCOPE` macro for convenient usage.

### Public Functions

explicit **TimeProfilerScope** (const std::string &name)

Enter a named profiling scope on construction.

**Parameters** `name` -- [in] Scope name to record.

**~TimeProfilerScope** ()

Leave the profiling scope on destruction.

### Macros

`_BROOKESIA_TIME_PROFILER_CONCAT` (a, b)

`BROOKESIA_TIME_PROFILER_CONCAT` (a, b)

`BROOKESIA_TIME_PROFILER_SCOPE` (name)

**BROOKESIA\_TIME\_PROFILER\_START\_EVENT** (name)

**BROOKESIA\_TIME\_PROFILER\_END\_EVENT** (name)

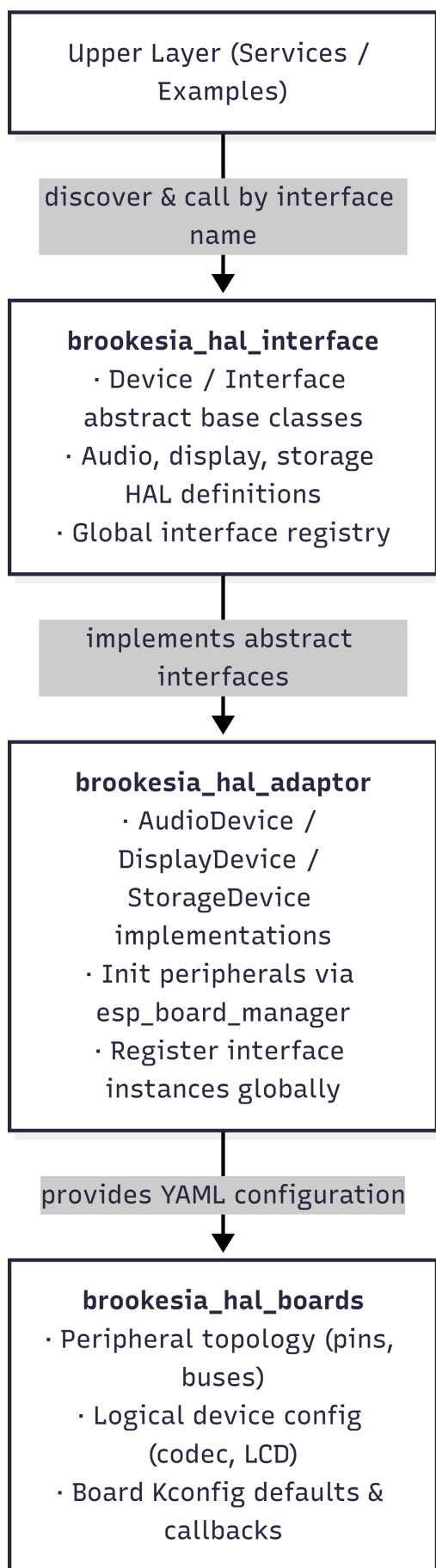
**BROOKESIA\_TIME\_PROFILER\_REPORT** ()

**BROOKESIA\_TIME\_PROFILER\_CLEAR** ()

## **Chapter 3**

# **Hardware Abstraction Components**

ESP-Brookesia HAL consists of three components that work together in layers, bridging the gap between board-level hardware and upper-layer business logic:



- `brookesia_hal_interface`: **Defines abstract interfaces**; upper-layer code depends only on this layer and remains decoupled from hardware details.
- `brookesia_hal_adaptor`: **Implements the abstract interfaces** by reading board configuration via `esp_board_manager` and initializing real peripherals.
- `brookesia_hal_boards`: **Provides board-level YAML configuration** describing the peripheral topology, pin assignments, and driver parameters for each supported board.

---

**Note:** Custom boards can be integrated into the ESP-Brookesia HAL framework in two ways:

- **Option 1 (Recommended):** Create a new board subdirectory under `brookesia_hal_boards/boards/<vendor>/` following the `esp_board_manager` specification and add the required configuration files. No changes to the adaptor layer are needed. See [Add a Custom Board](#).
  - **Option 2 (Fully Custom):** Remove the dependencies on `brookesia_hal_adaptor` and `brookesia_hal_boards`, and implement board-level initialization directly against the abstract interfaces in `brookesia_hal_interface`. This is suitable for cases where `esp_board_manager` cannot be used, but requires maintaining compatibility with the interface specification manually.
- 

## 3.1 HAL Interface

- Component registry: [espressif/brookesia\\_hal\\_interface](#)
- Public header: `#include "brookesia/hal_interface.hpp"`

### 3.1.1 Overview

`brookesia_hal_interface` is the hardware abstraction foundation of ESP-Brookesia, providing a unified abstraction between board-level implementations and upper-level subsystems. Its main capabilities include:

- **Device and interface model:** devices aggregate hardware units; interfaces express reusable capabilities (board information, battery, codec playback/recording, display panel/touch/backlight, storage volumes, etc.), with clear separation of concerns
- **Plugin-based registration:** device and interface implementations are registered in a registry and resolved by name at runtime, avoiding hard-coded implementation types in application code
- **Probing and lifecycle:** each device is first probed for availability before initialisation; batch and per-name single-device init/deinit are supported symmetrically
- **Global discovery:** devices can be resolved by plugin name or device logical name; interfaces can be enumerated globally by type or retrieved by name from within a device
- **Built-in HAL declarations:** abstract definitions for common audio, display, and storage interfaces; concrete behaviour is provided by the adaptor layer

### 3.1.2 Features

#### Devices and Interfaces

A **device** represents an independently managed hardware unit (e.g. an audio codec path, a display subsystem, a storage subsystem). Before a device operates it must be **probed**: only when the hardware is available in the current environment does it proceed to initialisation. During initialisation the device collects the **interface instances** it will expose.

An **interface** represents a stable capability boundary (e.g. board information, battery state and charger control, codec playback, recording, panel rendering, touch sampling, backlight control, storage medium and filesystem discovery). Multiple instances of the same abstract type can coexist, distinguished by the name used at registration time; that name uniquely identifies one instance in the global interface registry.

Device and interface implementation classes are registered through the project's plugin mechanism; the framework creates or retrieves instances by name at runtime, and upper layers depend only on the abstract types and conventions defined in this component.

### Registration and Lifecycle

During batch initialisation, the framework iterates the registry: for each device it executes probe and device-side init in order; on success the framework simultaneously registers that device's declared interfaces into the global interface table. A probe or init failure for one device typically only affects that device; the rest continue to be processed.

Per **plugin registration name** initialisation and deinitialisation are also supported. During deinit the device's interfaces are first removed from the global table, then any device-defined cleanup runs, and finally the device's internal references to interfaces are released.

Typical flow:

1. Registered devices enter the **probe** phase;
2. On success, the device is **initialised** and its interfaces are registered into the global registry; on failure, that **device is skipped**.

### Discovery and Naming

Two distinct name types are associated with devices:

Name type	Meaning
Plugin name	Key in the plugin registry, used to look up a device instance directly by registration entry
Device name	The logical name carried by the device object itself; may match or differ from the plugin name

Either path can be used at runtime to resolve the corresponding device.

Interface access is oriented around the **global interface registry**: all currently convertible instances of a given interface type can be listed, or the first matching instance of a type can be retrieved (returned as a name–instance pair). If starting from a device object, a specific interface can also be retrieved from that device's published interface set using the full registration name.

When multiple devices co-exist, it is recommended to add a device-distinguishing prefix or other namespace to interface registration names to avoid global conflicts.

### Built-In Capability Scope

The headers provide abstract definitions of common HAL interfaces, covering board information, battery and charger control, audio codec playback and recording, display panel and touch and backlight, and storage filesystem discovery. They describe static information, capability parameters, and virtual interface contracts; register operations, bus configuration, and timing are handled by the board-level adaptor or other components.

The following interface headers can be included at once via `brookesia/hal_interface/interfaces.hpp`, or together with the device base class via the aggregation entry `brookesia/hal_interface.hpp`:

Header	Primary types
<code>audio/codec_player.hpp</code>	<code>AudioCodecPlayerIface</code>
<code>audio/codec_recorder.hpp</code>	<code>AudioCodecRecorderIface</code>
<code>display/backlight.hpp</code>	<code>DisplayBacklightIface</code>
<code>display/panel.hpp</code>	<code>DisplayPanelIface</code>
<code>display/touch.hpp</code>	<code>DisplayTouchIface</code>
<code>general/board_info.hpp</code>	<code>BoardInfoIface</code>
<code>power/battery.hpp</code>	<code>PowerBatteryIface</code>
<code>storage/fs.hpp</code>	<code>StorageFsIface</code>

**Device Interface Classes** The component provides the following interface classes:

- `AudioCodecPlayerIface`
- `AudioCodecRecorderIface`
- `DisplayBacklightIface`
- `DisplayPanelIface`
- `DisplayTouchIface`
- `BoardInfoIface`
- `PowerBatteryIface`
- `StorageFsIface`

### 3.1.3 API Reference

#### Device & Interface Base

##### Device Base Class

Public header: `#include "brookesia/hal_interface/device.hpp"`

##### API Reference

##### Header File

- [hal/brookesia\\_hal\\_interface/include/brookesia/hal\\_interface/device.hpp](#)

##### Classes

class **Device**

Base class for HAL devices.

A concrete device can publish one or more interface instances through `interfaces_` during initialization, then those interfaces can be queried from the global registry.

Subclassed by `esp_brookesia::hal::AudioDevice`, `esp_brookesia::hal::DisplayDevice`, `esp_brookesia::hal::GeneralDevice`, `esp_brookesia::hal::PowerDevice`, `esp_brookesia::hal::StorageDevice`

##### Public Functions

virtual bool **probe** () = 0

Check whether the device is supported on current runtime conditions.

**Returns** `true` if the device can be initialized; otherwise `false`.

bool **register\_pre\_init\_callback** (PreInitCallback callback)

Register a callback that runs before this device is initialized.

The callback is invoked once just before `on_init()` inside `init()`. If the callback returns `false`, initialization is aborted. Registering a new callback overwrites any previously registered one.

**Parameters** `callback` -- [in] Callback to invoke; must be non-empty.

**Returns** `true` on success; `false` if the callback is empty.

bool **register\_post\_deinit\_callback** (PostDeinitCallback callback)

Register a callback that runs after this device is deinitialized.

The callback is invoked once right after `on_deinit()` returns. Its return value is only used for logging; deinitialization always proceeds to completion. Registering a new callback overwrites any previously registered one.

**Parameters** `callback` -- [in] Callback to invoke; must be non-empty.

**Returns** `true` on success; `false` if the callback is empty.

```
inline const std::string &get_name () const
```

Get the registry name of this device.

**Returns** *Device* name.

```
template<typename T>
```

```
inline std::shared_ptr<T> get_interface (const std::string &name) const
```

Retrieve a typed interface owned by this device by interface registry name.

**Template Parameters** `T` -- *Interface* type that derives from *Interface*.

**Parameters** `name` -- [in] Fully-qualified interface name in the registry.

**Returns** Matching typed interface pointer, or `nullptr` if the name is missing or the type does not match.

## Macros

**ESP\_BROOKESIA\_HAL\_DEVICE\_REGISTER\_ALL\_PRE\_INIT\_CALLBACK** (`symbol_name`, `callback`)

Auto-register a global pre-init callback used by `init_all_devices()`.

Generates a file-local registrar whose constructor forwards to `esp_brookesia::hal::register_pre_init_callback()`.

### Parameters

- **symbol\_name** -- Linker-visible identifier used with `-u` to keep the registrar alive.
- **callback** -- Callback expression convertible to `Device::PreInitCallback`.

**ESP\_BROOKESIA\_HAL\_DEVICE\_REGISTER\_ALL\_POST\_DEINIT\_CALLBACK** (`symbol_name`, `callback`)

Auto-register a global post-deinit callback used by `deinit_all_devices()`.

Generates a file-local registrar whose constructor forwards to `esp_brookesia::hal::register_post_deinit_callback()`.

### Parameters

- **symbol\_name** -- Linker-visible identifier used with `-u` to keep the registrar alive.
- **callback** -- Callback expression convertible to `Device::PostDeinitCallback`.

**ESP\_BROOKESIA\_HAL\_DEVICE\_REGISTER\_PRE\_INIT\_CALLBACKS** (`symbol_name`, ...)

Auto-register pre-init callbacks for multiple HAL devices at static init time.

Each variadic entry must be a brace-enclosed `{plugin_name, callback}` pair whose types match `std::pair<std::string, Device::PreInitCallback>`. Because the target devices and these callbacks are both registered through C++ static constructors in different translation units, their relative ordering is unspecified; therefore the macro does not attempt to resolve the device immediately. Instead each entry is queued through `detail::enqueue_pending_pre_init_callback()` and attached via `Device::register_pre_init_callback()` inside `init_device()` when the matching device is known to exist. Entries whose plugin name has no matching device are reported by `init_all_devices()` after the init pass finishes.

```
ESP_BROOKESIA_HAL_DEVICE_REGISTER_PRE_INIT_CALLBACKS (
    my_pre_init_callbacks_symbol,
    {"dev_a", []() { return true; }},
    {std::string(hal::AudioDevice::DEVICE_NAME), []() { return true; }}
);
```

### Parameters

- **symbol\_name** -- Linker-visible identifier used with `-u` to keep the registrar alive.
- ... -- One or more `{plugin_name, PreInitCallback}` entries separated by commas.

**ESP\_BROOKESIA\_HAL\_DEVICE\_REGISTER\_POST\_DEINIT\_CALLBACKS** (symbol\_name, ...)

Auto-register post-deinit callbacks for multiple HAL devices at static init time.

Behaves like `ESP_BROOKESIA_HAL_DEVICE_REGISTER_PRE_INIT_CALLBACKS`: each entry is queued through `detail::enqueue_pending_post_deinit_callback()` and attached via `Device::register_post_deinit_callback()` when the device is initialized, so the callback is in place before `Device::deinit()` runs.

**Parameters**

- **symbol\_name** -- Linker-visible identifier used with `-u` to keep the registrar alive.
- ... -- One or more {plugin\_name, PostDeinitCallback} entries separated by commas.

**Type Aliases**

```
using esp_brookesia::hal::DeviceRegistry = lib_utils::PluginRegistry<Device>
```

Registry alias for HAL devices.

**Concepts**

```
template<typename T>
```

```
concept IsDevice
```

`#include <device.hpp>` Type constraint for HAL device classes.

T must inherit from `Device`.

**Interface Base**

Public header: `#include "brookesia/hal_interface/interface.hpp"`

**API Reference****Header File**

- `hal/brookesia_hal_interface/include/brookesia/hal_interface/interface.hpp`

**Classes**

class **Interface**

Base class for all HAL interfaces exposed by a device.

Each concrete interface derives from this type and provides a stable interface name for runtime lookup via the plugin registry.

Subclassed by `esp_brookesia::hal::AudioCodecPlayerIface`, `esp_brookesia::hal::AudioCodecRecorderIface`,  
`esp_brookesia::hal::BoardInfoIface`, `esp_brookesia::hal::DisplayBacklightIface`,  
`esp_brookesia::hal::DisplayPanelIface`, `esp_brookesia::hal::DisplayTouchIface`,  
`esp_brookesia::hal::PowerBatteryIface`, `esp_brookesia::hal::StorageFsIface`

**Public Functions**

inline explicit **Interface** (std::string\_view name)

Construct an interface object with a runtime name.

**Parameters** **name** -- [in] *Interface* name used by the registry.

`virtual ~Interface () = default`  
Virtual destructor for polymorphic interface usage.

`inline std::string_view get_name () const noexcept`  
Get the registered interface name.

**Returns** *Interface* name.

## Audio Interface Classes

### Codec Player Interface

Public header: `#include "brookesia/hal_interface/interfaces/audio/codec_player.hpp"`

Class: `AudioCodecPlayerIface`

### API Reference

#### Header File

- [hal/brookesia\\_hal\\_interface/include/brookesia/hal\\_interface/interfaces/audio/codec\\_player.hpp](#)

#### Classes

class `AudioCodecPlayerIface` : public `esp_brookesia::hal::Interface`  
Player interface exposed by audio-capable devices.

#### Public Functions

`inline AudioCodecPlayerIface ()`  
Construct an audio interface.

`virtual ~AudioCodecPlayerIface () = default`  
Virtual destructor for polymorphic interfaces.

`virtual bool open (const Config &config) = 0`  
Open the backend.

**Parameters** `config` -- [in] Dynamic configuration.  
**Returns** `true` on success; otherwise `false`.

`virtual void close () = 0`  
Close the backend.

`virtual bool set_volume (uint8_t volume) = 0`  
Set volume.

**Parameters** `volume` -- [in] Requested output volume percentage.  
**Returns** `true` on success; otherwise `false`.

`virtual bool write_data (const uint8_t *data, size_t size) = 0`  
Write PCM/encoded payload to the backend.

**Parameters**

- `data` -- [in] Buffer pointer containing audio payload.
- `size` -- [in] Buffer size in bytes.

**Returns** `true` on success; otherwise `false`.

virtual bool **is\_pa\_on\_off\_supported** () = 0

Check if the PA control is supported.

**Returns** `true` if the PA on/off control is supported; otherwise `false`.

virtual bool **set\_pa\_on\_off** (bool on) = 0

Set the PA on/off state.

**Parameters** `on` -- **[in]** `true` to set the PA to be on; `false` to set the PA to be off.

**Returns** `true` if the PA on/off state is set successfully; otherwise `false`.

virtual bool **is\_pa\_on** () const = 0

Check if the PA is set on.

**Returns** `true` if the PA is set on; otherwise `false`.

### Public Static Attributes

static constexpr const char \***NAME** = "AudioCodecPlayer"

*Interface* registry name.

struct **Config**

Dynamic configuration.

### Public Members

uint8\_t **bits**

Sample bit width.

uint8\_t **channels**

Number of output channels.

uint32\_t **sample\_rate**

Output sample rate in Hz.

### Codec Recorder Interface

Public header: `#include "brookesia/hal_interface/interfaces/audio/codec_recorder.hpp"`

Class: `AudioCodecRecorderIface`

### API Reference

#### Header File

- [hal/brookesia\\_hal\\_interface/include/brookesia/hal\\_interface/interfaces/audio/codec\\_recorder.hpp](hal/brookesia_hal_interface/include/brookesia/hal_interface/interfaces/audio/codec_recorder.hpp)

#### Classes

class **AudioCodecRecorderIface** : public `esp_brookesia::hal::Interface`

Recording interface exposed by audio-capable devices.

### Public Functions

inline **AudioCodecRecorderInterface** (*Info* info)

Construct an audio recording interface.

**Parameters** **info** -- [in] Static recording capability information.

virtual **~AudioCodecRecorderInterface** () = default

Virtual destructor for polymorphic recording interfaces.

virtual bool **open** () = 0

Open the recording backend.

**Returns** *true* on success; otherwise *false*.

virtual void **close** () = 0

Close the recording backend.

virtual bool **read\_data** (uint8\_t \*data, size\_t size) = 0

Read captured audio payload.

**Parameters**

- **data** -- [out] Destination buffer for captured bytes.
- **size** -- [in] Requested byte count.

**Returns** *true* on success; otherwise *false*.

virtual bool **set\_general\_gain** (float gain) = 0

Set the general gain.

**Parameters** **gain** -- [in] The gain to set.

**Returns** *true* on success; otherwise *false*.

virtual bool **set\_channel\_gains** (const std::map<uint8\_t, float> &gains) = 0

Set the channel gains.

**Parameters** **gains** -- [in] The channel gains to set.

**Returns** *true* on success; otherwise *false*.

inline const *Info* &**get\_info** () const

Get static recording capability information.

**Returns** Recording information.

### Public Static Attributes

static constexpr const char \***NAME** = "AudioCodecRecorder"

*Interface* registry name.

struct **Info**

Static recording capability information.

### Public Members

uint8\_t **bits**

Sample bit width.

uint8\_t **channels**

Number of capture channels.

uint32\_t **sample\_rate**  
Capture sample rate in Hz.

std::string **mic\_layout**  
Microphone layout descriptor.

float **general\_gain**  
Global input gain.

std::map<uint8\_t, float> **channel\_gains**  
Per-channel gain overrides.

### Display Interface Classes

#### Display Panel Interface

Public header: #include "brookesia/hal\_interface/interfaces/display/panel.hpp"

#### API Reference

##### Header File

- [hal/brookesia\\_hal\\_interface/include/brookesia/hal\\_interface/interfaces/display/panel.hpp](#)

##### Classes

class **DisplayPanelInterface** : public esp\_brookesia::hal::Interface  
Display panel interface for rendering pixel data.

##### Public Types

enum class **PixelFormat** : uint8\_t

Pixel format enum.

*Values:*

enumerator **RGB565**

enumerator **RGB888**

enumerator **Max**

enum class **BusType** : uint8\_t

Driver-specific bus type enum.

*Values:*

enumerator **Generic**

enumerator **RGB**

< Generic bus type, such as SPI, I80, QSPI, etc.

enumerator **MIPI**

< RGB bus type.

enumerator **Max**

< MIPI bus type.

### Public Functions

inline **DisplayPanelInterface** (*Info* info)

Construct a display panel interface.

**Parameters** *info* -- [**in**] Static panel capability information.

virtual **~DisplayPanelInterface** () = default

Virtual destructor for polymorphic panel interfaces.

virtual bool **draw\_bitmap** (uint32\_t x1, uint32\_t y1, uint32\_t x2, uint32\_t y2, const uint8\_t \*data) = 0

Draw a bitmap into a rectangular panel region.

The region follows half-open bounds semantics: [x1, x2) and [y1, y2).

**Parameters**

- **x1** -- [**in**] Left coordinate.
- **y1** -- [**in**] Top coordinate.
- **x2** -- [**in**] Right coordinate (exclusive).
- **y2** -- [**in**] Bottom coordinate (exclusive).
- **data** -- [**in**] Pixel buffer pointer.

**Returns** *true* on success; otherwise *false*.

inline virtual bool **get\_driver\_specific** (*DriverSpecific* &specific)

Get the driver-specific data.

**Parameters** *specific* -- [**out**] The driver-specific data.

**Returns** *true* on success; otherwise *false*.

inline const *Info* &**get\_info** () const

Get static panel capability information.

**Returns** Panel information.

### Public Static Attributes

static constexpr const char \***NAME** = "DisplayPanel"

*Interface* registry name.

struct **DriverSpecific**

Driver-specific data.

### Public Members

```
void *io_handle = nullptr
```

Handle of the IO.

```
void *panel_handle = nullptr
```

Handle of the panel.

```
BusType bus_type = BusType::Max
```

Bus type.

```
uint8_t draw_x_align_bytes = 1
```

X-axis coordinate alignment bytes.

```
uint8_t draw_y_align_bytes = 1
```

Y-axis coordinate alignment bytes.

```
struct Info
```

Static panel capability information.

### Public Functions

```
inline uint8_t get_pixel_bits () const
```

Get the number of bits per pixel.

**Returns** Number of bits per pixel.

```
inline bool is_valid () const
```

Check if the panel information is valid.

**Returns** `true` if the panel information is valid; otherwise `false`.

### Public Members

```
uint16_t h_res = 0
```

Horizontal resolution in pixels.

```
uint16_t v_res = 0
```

Vertical resolution in pixels.

```
PixelFormat pixel_format = PixelFormat::Max
```

Pixel format.

### Touch Interface

Public header: `#include "brookesia/hal_interface/interfaces/display/touch.hpp"`

Class: `DisplayTouchIface`

### API Reference

#### Header File

- [hal/brookesia\\_hal\\_interface/include/brookesia/hal\\_interface/interfaces/display/touch.hpp](hal/brookesia_hal_interface/include/brookesia/hal_interface/interfaces/display/touch.hpp)

## Classes

class **DisplayTouchIface** : public esp\_brookesia::hal::Interface

Touch-input interface paired with a display.

## Public Types

enum class **OperationMode** : uint8\_t

Supported event acquisition mode.

*Values:*

enumerator **Polling**

Read touch points by polling.

enumerator **Interrupt**

Read touch points by interrupt.

enumerator **Max**

## Public Functions

inline **DisplayTouchIface** (*Info* info)

Construct a touch-input interface.

**Parameters** **info** -- [in] Static touch capability information.

virtual **~DisplayTouchIface** () = default

Virtual destructor for polymorphic touch interfaces.

virtual bool **read\_points** (std::vector<Point> &points) = 0

Read current touch points.

**Parameters** **points** -- [out] Output touch points.

**Returns** `true` on success; otherwise `false`.

virtual bool **register\_interrupt\_handler** (InterruptHandler handler) = 0

Register an interrupt handler.

**Parameters** **handler** -- [in] Interrupt handler. If `nullptr`, the interrupt handler will be unregistered.

**Returns** `true` on success; otherwise `false`.

inline virtual bool **get\_driver\_specific** (*DriverSpecific* &specific)

Get the driver-specific data.

**Parameters** **specific** -- [out] The driver-specific data.

**Returns** `true` on success; otherwise `false`.

inline const *Info* &**get\_info** () const

Get static touch capability information.

**Returns** Touch information.

### Public Static Attributes

static constexpr const char \***NAME** = "DisplayTouch"  
*Interface* registry name.

struct **DriverSpecific**  
Driver-specific data.

### Public Members

void \***io\_handle** = nullptr  
Handle of the IO.

void \***touch\_handle** = nullptr  
Handle of the touch.

struct **Info**  
Static touch capability information.

### Public Members

uint16\_t **x\_max** = 0  
Maximum X coordinate value.

uint16\_t **y\_max** = 0  
Maximum Y coordinate value.

*OperationMode* **operation\_mode** = *OperationMode::Max*  
Supported acquisition mode.

struct **Point**  
A single touch point sample.

### Public Members

int16\_t **x**  
X coordinate.

int16\_t **y**  
Y coordinate.

uint16\_t **pressure**  
Pressure or strength value from controller.

### Backlight Interface

Public header: #include "brookesia/hal\_interface/interfaces/display/backlight.hpp"

## API Reference

### Header File

- [hal/brookesia\\_hal\\_interface/include/brookesia\\_hal\\_interface/interfaces/display/backlight.hpp](#)

### Classes

class **DisplayBacklightIface** : public esp\_brookesia::hal::Interface

Display backlight control interface.

#### Public Functions

inline **DisplayBacklightIface** ()

Construct a display backlight interface.

virtual **~DisplayBacklightIface** () = default

Virtual destructor for polymorphic backlight interfaces.

virtual bool **set\_brightness** (uint8\_t percent) = 0

Set backlight brightness.

**Parameters** *percent* -- [in] Brightness percentage.

**Returns** *true* on success; otherwise *false*.

virtual bool **get\_brightness** (uint8\_t &percent) = 0

Get current backlight brightness.

**Parameters** *percent* -- [out] Current brightness percentage.

**Returns** *true* on success; otherwise *false*.

virtual bool **is\_light\_on\_off\_supported** () = 0

Check if the light on/off control is supported.

**Returns** *true* if the light on/off control is supported; otherwise *false*.

virtual bool **set\_light\_on\_off** (bool on) = 0

Set the backlight on/off state.

**Parameters** *on* -- [in] *true* to set the backlight on; *false* to set the backlight off.

**Returns** *true* if the backlight on/off state is set successfully; otherwise *false*.

virtual bool **is\_light\_on** () const = 0

Check if the backlight is set on.

**Returns** *true* if the backlight is set on; otherwise *false*.

#### Public Static Attributes

static constexpr const char \***NAME** = "DisplayBacklight"

*Interface* registry name.

## General Interface Classes

### Board Information Interface

Public header: `#include "brookesia/hal_interface/interfaces/general/board_info.hpp"`

Class: `BoardInfoIface`

`BoardInfoIface` exposes static board metadata, such as the board name, main chip, hardware version, description, and manufacturer. Upper layers can use it to identify the current board at runtime and adapt displayed device information or board-specific logic.

## API Reference

### Header File

- [hal/brookesia\\_hal\\_interface/include/brookesia/hal\\_interface/interfaces/general/board\\_info.hpp](hal/brookesia_hal_interface/include/brookesia/hal_interface/interfaces/general/board_info.hpp)

### Classes

class **BoardInfoIface** : public esp\_brookesia::hal::Interface  
Board information interface for querying static board metadata.

#### Public Functions

inline explicit **BoardInfoIface** (*Info* info)

Construct a board information interface.

**Parameters** *info* -- [in] Static board information.

virtual **~BoardInfoIface** () = default

Virtual destructor for polymorphic board information interfaces.

inline const *Info* &**get\_info** () const

Get static board information.

**Returns** Board information.

#### Public Static Attributes

static constexpr const char \***NAME** = "BoardInfo"

*Interface* registry name.

struct **Info**

Static board information.

#### Public Functions

inline bool **is\_valid** () const

Check whether the board information contains useful identity fields.

**Returns** `true` if either board name or chip is non-empty; otherwise `false`.

### Public Members

std::string **name**  
Board name.

std::string **chip**  
Main chip model.

std::string **version**  
Board hardware or BOM version.

std::string **description**  
Human-readable board description.

std::string **manufacturer**  
Board manufacturer.

### Power Interface Classes

#### Battery Interface

Public header: `#include "brookesia/hal_interface/interfaces/power/battery.hpp"`

Class: `PowerBatteryIface`

`PowerBatteryIface` describes battery and charger capabilities. It can query battery presence, level percentage, voltage, power source, and charge state, and can also read or update charger configuration and enable or disable charging when supported by the underlying hardware.

Not every board exposes the full set of battery and charger controls. Callers should first inspect `Info::abilities` or `Info::has_ability()` before reading capability-specific state fields or invoking control APIs.

### API Reference

#### Header File

- [hal/brookesia\\_hal\\_interface/include/brookesia/hal\\_interface/interfaces/power/battery.hpp](hal/brookesia_hal_interface/include/brookesia/hal_interface/interfaces/power/battery.hpp)

#### Classes

class **PowerBatteryIface** : public `esp_brookesia::hal::Interface`

Battery interface for querying battery state and controlling charger settings.

#### Public Types

enum class **Ability**

Battery backend capability.

*Values:*

enumerator **Voltage**

Battery voltage reading.

enumerator **Percentage**

Battery level percentage.

enumerator **PowerSource**

Battery/external power source state.

enumerator **ChargeState**

Charger state.

enumerator **VbusVoltage**

VBUS voltage reading.

enumerator **SystemVoltage**

System voltage reading.

enumerator **ChargerControl**

Charger enable/disable control.

enumerator **ChargeConfig**

Charger parameter configuration.

enum class **PowerSource**

Current power source.

*Values:*

enumerator **Unknown**

Power source is unknown.

enumerator **Battery**

Running from battery.

enumerator **External**

Running from external power.

enum class **ChargeState**

Current charge state.

*Values:*

enumerator **Unknown**

Charge state is unknown.

enumerator **NotCharging**

Battery is not charging.

enumerator **Charging**

Battery is charging, but the phase is unknown.

enumerator **Trickle**

Trickle charging.

enumerator **PreCharge**

Pre-charge phase.

enumerator **ConstantCurrent**

Constant-current phase.

enumerator **ConstantVoltage**

Constant-voltage phase.

enumerator **Full**

Battery is full.

enumerator **Fault**

Charger fault.

enum class **LevelSource**

Source used to calculate the battery level.

*Values:*

enumerator **Unknown**

Level source is unknown.

enumerator **FuelGauge**

Level comes from a fuel gauge.

enumerator **VoltageCurve**

Level is estimated from a voltage curve.

### Public Functions

inline explicit **PowerBatteryIface** (*Info* info)

Construct a battery interface.

**Parameters** **info** -- **[in]** Static battery capability information.

virtual **~PowerBatteryIface** () = default

Virtual destructor for polymorphic battery interfaces.

inline const *Info* &**get\_info** () const

Get static battery capability information.

**Returns** Battery information.

virtual bool **get\_state** (*State* &state) = 0

Get the current battery state.

**Parameters** **state** -- **[out]** Runtime battery state snapshot.

**Returns** `true` on success; otherwise `false`.

```
virtual bool get_charge_config (ChargeConfig &config) = 0
```

Get charger configuration.

**Parameters** **config** -- [out] Charger configuration.

**Returns** `true` on success; otherwise `false`.

```
virtual bool set_charge_config (const ChargeConfig &config) = 0
```

Set charger configuration.

**Parameters** **config** -- [in] Charger configuration.

**Returns** `true` on success; otherwise `false`.

```
virtual bool set_charging_enabled (bool enabled) = 0
```

Enable or disable charging.

**Parameters** **enabled** -- [in] `true` to enable charging; `false` to disable charging.

**Returns** `true` on success; otherwise `false`.

### Public Static Attributes

```
static constexpr const char *NAME = "Power:Battery"
```

*Interface* registry name.

```
struct ChargeConfig
```

Charger configuration.

### Public Members

```
bool enabled = false
```

Charger enable state.

```
uint32_t target_voltage_mv = 0
```

Target charge voltage in mV.

```
uint32_t charge_current_ma = 0
```

Main charge current in mA.

```
uint32_t precharge_current_ma = 0
```

Pre-charge current in mA.

```
uint32_t termination_current_ma = 0
```

Termination current in mA.

```
struct Info
```

Static battery capability information.

### Public Functions

```
inline bool has_ability (Ability ability) const
```

Check whether an ability is supported.

**Parameters** **ability** -- [in] Ability to query.

**Returns** `true` if the ability is present; otherwise `false`.

### Public Members

std::string **name**

Battery or backend name.

std::string **chemistry**

Battery chemistry, such as Li-ion.

std::vector<*Ability*> **abilities**

Supported backend abilities.

struct **State**

Runtime battery state snapshot.

### Public Members

bool **is\_present** = false

Whether a battery is detected.

*PowerSource* **power\_source** = *PowerSource::Unknown*

Current power source.

*ChargeState* **charge\_state** = *ChargeState::Unknown*

Current charge state.

*LevelSource* **level\_source** = *LevelSource::Unknown*

Battery percentage source.

std::optional<uint32\_t> **voltage\_mv** = std::nullopt

Battery voltage in mV.

std::optional<uint8\_t> **percentage** = std::nullopt

Battery percentage in [0, 100].

std::optional<uint32\_t> **vbus\_voltage\_mv** = std::nullopt

VBUS voltage in mV.

std::optional<uint32\_t> **system\_voltage\_mv** = std::nullopt

System voltage in mV.

bool **is\_low** = false

Battery is below the low threshold.

bool **is\_critical** = false

Battery is below the critical threshold.

## Storage Interface Classes

### Filesystem Interface

Public header: `#include "brookesia/hal_interface/interfaces/storage/fs.hpp"`

Class: `StorageFsIface`

### API Reference

#### Header File

- [hal/brookesia\\_hal\\_interface/include/brookesia/hal\\_interface/interfaces/storage/fs.hpp](#)

#### Classes

class **StorageFsIface** : public esp\_brookesia::hal::Interface  
File-system discovery interface for storage-capable devices.

#### Public Types

enum class **MediumType**  
Supported storage medium type.

*Values:*

enumerator **Flash**

Flash medium.

enumerator **SDCard**

SD / TF card medium.

enum class **FileSystemType**

Supported file-system type.

*Values:*

enumerator **SPIFFS**

SPIFFS file system.

enumerator **FATFS**

FAT file system.

enumerator **LittleFS**

LittleFS file system.

#### Public Functions

inline **StorageFsIface** ()  
Construct a storage file-system interface.

virtual **~StorageFsInterface** () = default

Virtual destructor for polymorphic storage interfaces.

inline const std::vector<*Info*> &**get\_all\_info** () const

Enumerate all mounted file-system entries.

**Returns** Collection of file-system metadata entries.

virtual bool **get\_capacity** (const char \*mount\_point, *Capacity* &capacity) = 0

Query the capacity of a mounted file system.

**Parameters**

- **mount\_point** -- Mount point returned by *get\_all\_info* ().
- **capacity** -- Output capacity snapshot.

**Returns** true if the capacity was queried successfully, otherwise false.

### Public Static Attributes

static constexpr const char \***NAME** = "StorageFs"

*Interface* registry name.

struct **Capacity**

Dynamic capacity snapshot for one mounted file-system entry.

### Public Members

uint64\_t **total\_bytes** = 0

Total size in bytes.

uint64\_t **used\_bytes** = 0

Used size in bytes.

uint64\_t **free\_bytes** = 0

Free size in bytes.

struct **Info**

Metadata for one mounted file-system entry.

### Public Members

*FileSystemType* **fs\_type**

File-system type.

*MediumType* **medium\_type**

Storage medium type.

const char \***mount\_point**

Mount point.

bool **supports\_directories** = false

Whether this file system supports directories.

## 3.2 HAL Adaptor

- Component registry: [espressif/brookesia\\_hal\\_adaptor](#)
- Public header: `#include "brookesia/hal_adaptor.hpp"`

### 3.2.1 Overview

`brookesia_hal_adaptor` is the board-level HAL adaptor of ESP-Brookesia. Based on the device/interface model in *HAL Interface*, it initialises real peripherals via `esp_board_manager` and registers **general information, audio, display, storage, and power** capabilities into the global HAL table for upper layers to discover by name.

### 3.2.2 Features

#### Built-In Devices

The component ships multiple board-level devices, each registered as a singleton; after initialisation they publish their interfaces into the global table:

Device class (logical name)	Registered interface implementations	Notes
GeneralDevice ("General")	BoardInfoIface (BOARD_INFO_IMPL_NAME)	Reads static board metadata such as board name, chip, version, description, and manufacturer from board-level configuration for device identification and information display.
AudioDevice ("Audio")	AudioCodecPlayerIface (CODEC_PLAYER_IMPL_NAME), AudioCodecRecorderIface (CODEC_RECORDER_IMPL_NAME)	Playback via board Audio DAC; recording via Audio ADC. Each sub-implementation can be disabled in Kconfig; requires board capability <code>ESP_BOARD_DEV_AUDIO_CODEC_SUPPORT</code> .
DisplayDevice ("Display")	DisplayBacklightIface (LEDC_BACKLIGHT_IMPL_NAME), DisplayPanelIface (LCD_PANEL_IMPL_NAME), DisplayTouchIface (LCD_TOUCH_IMPL_NAME)	LEDC backlight, LCD panel, and I2C touch can each be disabled; require <code>ESP_BOARD_DEV_LEDC_CTRL_SUPPORT</code> , <code>ESP_BOARD_DEV_DISPLAY_LCD_SUPPORT</code> , <code>ESP_BOARD_DEV_LCD_TOUCH_I2C_SUPPORT</code> respectively.
StorageDevice ("Storage")	StorageFsIface (GENERAL_FS_IMPL_NAME)	General filesystem implementation; supports SPIFFS ( <code>ESP_BOARD_DEV_FS_SPIFFS_SUPPORT</code> ) and SD card / FATFS ( <code>ESP_BOARD_DEV_FS_FAT_SUPPORT</code> ), enabled per Kconfig.
PowerDevice ("Power")	PowerBatteryIface (BATTERY_IMPL_NAME)	Battery and charger capability implementation. Supports ADC voltage estimation or AXP2101 power-management-chip backends; can query level, voltage, power source, and charge state, and control charger configuration when supported by the underlying hardware.

## Configuration

Each device and sub-interface can be enabled or disabled individually under **ESP-Brookesia: Hal Adaptor Configurations** in `menuconfig`; default capability parameters (volume range, recording format, backlight range, battery low-level thresholds, ADC voltage conversion parameters, etc.) are also adjustable there, and are mapped to compile-time macros by `macro_configs.h`.

To override default capability parameters before initialisation, call `set_codec_player_info`, `set_codec_recorder_info`, or `set_ledc_backlight_info` on the corresponding device singleton. Calls after initialisation typically have no effect.

### 3.2.3 API Reference

#### Header File

- [hal/brookesia\\_hal\\_adaptor/include/brookesia/hal\\_adaptor/general/device.hpp](#)

#### Classes

class **GeneralDevice** : public esp\_brookesia::hal::Device

Board-backed metadata device: publishes a board information HAL interface.

Obtained via `get_instance()`. Not copyable or movable.

#### Public Static Functions

static inline *GeneralDevice* &`get_instance` ()

Returns the process-wide singleton board device.

**Returns** Reference to the unique *GeneralDevice* instance.

#### Public Static Attributes

static constexpr const char \***DEVICE\_NAME** = "General"

Logical device name passed to the base *Device* constructor.

static constexpr const char \***BOARD\_INFO\_IMPL\_NAME** = "General:BoardInfo"

Registry key for the board information HAL interface ("General:BoardInfo").

#### Header File

- [hal/brookesia\\_hal\\_adaptor/include/brookesia/hal\\_adaptor/display/device.hpp](#)

#### Classes

class **DisplayDevice** : public esp\_brookesia::hal::Device

Board-backed display device: registers panel, touch, and backlight HAL interfaces after board bring-up.

Obtained via `get_instance()`. Not copyable or movable.

### Public Static Functions

static inline *DisplayDevice* &get\_instance ()

Returns the process-wide singleton display device.

**Returns** Reference to the unique *DisplayDevice* instance.

### Public Static Attributes

static constexpr const char \*DEVICE\_NAME = "Display"

Logical device name passed to the base *Device* constructor.

static constexpr const char \*LEDC\_BACKLIGHT\_IMPL\_NAME = "Display:LedcBacklight"

Registry key for the LEDC-based backlight HAL implementation ("Display:LedcBacklight").

static constexpr const char \*LCD\_PANEL\_IMPL\_NAME = "Display:LcdPanel"

Registry key for the LCD panel HAL implementation ("Display:LcdPanel").

static constexpr const char \*LCD\_TOUCH\_IMPL\_NAME = "Display:LcdTouch"

Registry key for the LCD touch HAL implementation ("Display:LcdTouch").

### Header File

- [hal/brookesia\\_hal\\_adaptor/include/brookesia/hal\\_adaptor/audio/device.hpp](#)

### Classes

class **AudioDevice** : public esp\_brookesia::hal::Device

Board-backed audio device: registers codec player and recorder HAL interfaces after board bring-up.

Obtained via *get\_instance()*. Not copyable or movable.

### Public Functions

bool set\_codec\_recorder\_info (*AudioCodecRecorderIfc::Info* info)

Overrides default static recording capability information used when constructing the codec recorder implementation.

**Parameters** *info* -- [in] Codec recorder capability descriptor (format, channels, gains, etc.).

**Returns** *true* if the value was stored; *false* on invalid input or if the recorder is already initialized.

### Public Static Functions

static inline *AudioDevice* &get\_instance ()

Returns the process-wide singleton audio device.

**Returns** Reference to the unique *AudioDevice* instance.

### Public Static Attributes

static constexpr const char \***DEVICE\_NAME** = "Audio"

Logical device name passed to the base *Device* constructor.

static constexpr const char \***CODEC\_PLAYER\_IMPL\_NAME** = "Audio:CodecPlayer"

Registry key for the codec player HAL implementation ("Audio:CodecPlayer").

static constexpr const char \***CODEC\_RECORDER\_IMPL\_NAME** = "Audio:CodecRecorder"

Registry key for the codec recorder HAL implementation ("Audio:CodecRecorder").

### Header File

- [hal/brookesia\\_hal\\_adaptor/include/brookesia/hal\\_adaptor/storage/device.hpp](#)

### Classes

class **StorageDevice** : public esp\_brookesia::hal::Device

Board-backed storage device: publishes a general filesystem HAL interface after bring-up.

Obtained via *get\_instance()*. Not copyable or movable.

### Public Static Functions

static inline *StorageDevice* &**get\_instance** ()

Returns the process-wide singleton storage device.

**Returns** Reference to the unique *StorageDevice* instance.

### Public Static Attributes

static constexpr const char \***DEVICE\_NAME** = "Storage"

Logical device name passed to the base *Device* constructor.

static constexpr const char \***GENERAL\_FS\_IMPL\_NAME** = "Storage:GenralFS"

Registry key for the general filesystem HAL interface ("Storage:GenralFS").

### Header File

- [hal/brookesia\\_hal\\_adaptor/include/brookesia/hal\\_adaptor/power/device.hpp](#)

### Classes

class **PowerDevice** : public esp\_brookesia::hal::Device

Power-backed metadata device: publishes a power HAL interface.

Obtained via *get\_instance()*. Not copyable or movable.

### Public Static Functions

static inline *PowerDevice* &get\_instance ()

Returns the process-wide singleton power device.

**Returns** Reference to the unique *PowerDevice* instance.

### Public Static Attributes

static constexpr const char \*DEVICE\_NAME = "Power"

Logical device name passed to the base *Device* constructor.

static constexpr const char \*BATTERY\_IMPL\_NAME = "Power:Battery"

Registry key for the power HAL interface ("Power:Battery").

## 3.3 HAL Board Support

- Component Registry: [espressif/brookesia\\_hal\\_boards](#)

### 3.3.1 Overview

`brookesia_hal_boards` is the board configuration collection for ESP-Brookesia. It uses YAML files to describe the peripheral topology and device parameters for each supported board, allowing *HAL Adaptor* to initialize hardware at runtime without any board-specific hardcoding.




### 3.3.2 Supported Boards

- *Espressif Boards*
- *Waveshare Boards*

#### Espressif Boards

Board Name	Parameters / Features	Product Link	Supported Interfaces
esp32_s3_korvo2_v3	<ul style="list-style-type: none"> <li>• <i>ESP32-S3</i></li> <li>• <i>Wi-Fi</i></li> <li>• <i>Flash 16MB</i></li> <li>• <i>PSRAM 8MB</i></li> </ul>	<a href="#">ESP32-S3-Korvo-2 V3</a>	<i>AudioCodecPlayer</i>  <i>AudioCodecRecorder</i> <i>DisplayPanel</i> <i>DisplayTouch</i> <i>DisplayBacklight</i> <i>StorageFs</i>
esp_box_3	<ul style="list-style-type: none"> <li>• <i>ESP32-S3</i></li> <li>• <i>Wi-Fi</i></li> <li>• <i>Flash 16MB</i></li> <li>• <i>PSRAM 16MB</i></li> </ul>	<a href="#">ESP-BOX-3</a>	<i>AudioCodecPlayer</i>  <i>AudioCodecRecorder</i> <i>DisplayPanel</i> <i>DisplayTouch</i> <i>DisplayBacklight</i> <i>StorageFs</i>
esp_vocat_board_v1_0	<ul style="list-style-type: none"> <li>• <i>ESP32-S3</i></li> <li>• <i>Wi-Fi</i></li> <li>• <i>Flash 32MB</i></li> <li>• <i>PSRAM 16MB</i></li> </ul>	<a href="#">ESP-VoCat V1.0</a>	<i>AudioCodecPlayer</i>  <i>AudioCodecRecorder</i> <i>DisplayPanel</i> <i>DisplayTouch</i> <i>DisplayBacklight</i> <i>StorageFs</i>
esp_vocat_board_v1_2	<ul style="list-style-type: none"> <li>• <i>ESP32-S3</i></li> <li>• <i>Wi-Fi</i></li> <li>• <i>Flash 32MB</i></li> <li>• <i>PSRAM 16MB</i></li> </ul>	<a href="#">ESP-VoCat V1.2</a>	<i>AudioCodecPlayer</i>  <i>AudioCodecRecorder</i> <i>DisplayPanel</i> <i>DisplayTouch</i> <i>DisplayBacklight</i> <i>StorageFs</i>
esp32_p4_function_ev	<ul style="list-style-type: none"> <li>• <i>ESP32-P4</i></li> <li>• <i>Wi-Fi</i></li> <li>• <i>Flash 16MB</i></li> <li>• <i>PSRAM 32MB</i></li> </ul>	<a href="#">ESP32-P4-Function-EV-Board</a>	<i>AudioCodecPlayer</i>  <i>AudioCodecRecorder</i> <i>DisplayPanel</i> <i>DisplayTouch</i> <i>DisplayBacklight</i> <i>StorageFs</i>
esp32_p4x_function_ev	<ul style="list-style-type: none"> <li>• <i>ESP32-P4</i></li> <li>• <i>Wi-Fi</i></li> <li>• <i>Flash 16MB</i></li> <li>• <i>PSRAM 32MB</i></li> </ul>	<a href="#">ESP32-P4X Function-EV Board</a>	<i>AudioCodecPlayer</i>  <i>AudioCodecRecorder</i> <i>DisplayPanel</i> <i>DisplayTouch</i> <i>DisplayBacklight</i> <i>StorageFs</i>
Espressif Systems		<b>86</b> <a href="#">Submit Document Feedback</a>	Release master <i>StorageFs</i>
esp32_s31_korvo1		<a href="#">ESP32-S31 Korvo1</a>	

## Waveshare Boards

Board Name	Parameters / Features	Product Link	Supported Interfaces
esp32_s3_touch_amoled_1_8 	<ul style="list-style-type: none"> <li>• ESP32-S3</li> <li>• Wi-Fi</li> <li>• Flash 16MB</li> <li>• PSRAM 8MB</li> </ul>	<a href="#">ESP32-S3-Touch-AMOLED-1.8</a>	<i>AudioCodecPlayer</i>  <i>AudioCodecRecorder</i> <i>DisplayPanel</i> <i>DisplayTouch</i> <i>DisplayBacklight</i> <i>PowerBattery</i> <i>StorageFs</i>
esp32_s3_touch_amoled_2_16 	<ul style="list-style-type: none"> <li>• ESP32-S3</li> <li>• Wi-Fi</li> <li>• Flash 16MB</li> <li>• PSRAM 8MB</li> </ul>	<a href="#">ESP32-S3-Touch-AMOLED-2.16</a>	<i>AudioCodecPlayer</i>  <i>AudioCodecRecorder</i> <i>DisplayPanel</i> <i>DisplayTouch</i> <i>DisplayBacklight</i> <i>PowerBattery</i> <i>StorageFs</i>
esp32_s3_touch_amoled_1_75c 	<ul style="list-style-type: none"> <li>• ESP32-S3</li> <li>• Wi-Fi</li> <li>• Flash 32MB</li> <li>• PSRAM 8MB</li> </ul>	<a href="#">ESP32-S3-Touch-AMOLED-1.75C</a>	<i>AudioCodecPlayer</i>  <i>AudioCodecRecorder</i> <i>DisplayPanel</i> <i>DisplayTouch</i> <i>DisplayBacklight</i> <i>PowerBattery</i> <i>StorageFs</i>

### 3.3.3 Directory Structure

Each board has its own subdirectory under `boards/<vendor>/<board-name>/` containing the following files:

```
boards/
├── <vendor>/
│   └── <board>/
│       ├── components/...           # Special component dependencies (optional, ↪
│       ↪for non-standard peripherals or special features)
│       ├── board_info.yaml         # Board metadata (name, chip, version, ↪
│       ↪manufacturer, etc.)
│       └── board_devices.yaml      # Logical device configurations (audio codec, ↪
│       ↪LCD, touch, storage, etc.)
```

(continues on next page)

(continued from previous page)

```

├── board_peripherals.yaml # Low-level peripheral configurations (I2C/I2S/
↪SPI buses, GPIO, LEDC, etc.)
├── sdkconfig.defaults.board # Board-specific Kconfig defaults (Flash,
↪PSRAM, etc.)
├── setup_device.c # Board-specific device factory callbacks
↪(optional, for custom driver initialization)

```

**Note:** For the complete board configuration format reference, see the [esp\\_board\\_manager component documentation](#).

## Device Types

`board_devices.yaml` describes the logical functional modules on the board. Common device types include:

Device Type	Description
audio_codec	Audio codec chip (DAC playback / ADC recording); supports ES8311, ES7210, internal ADC, and more
display_lcd	LCD display; supports SPI (ST77916, ILI9341), DSI (EK79007), and other interfaces
lcd_touch	Touch panel; supports CST816S, GT911, and other I2C touch controllers
ledc_ctrl	PWM backlight control via LEDC
fs_fat / fs_spiffs	File system storage; supports SD cards (SDMMC/SPI) and SPIFFS
camera	Camera (CSI interface)
power_ctrl	GPIO-based power control (audio power, LCD/SD card power, and more)
gpio_ctrl	General-purpose GPIO control (LEDs, buttons, and more)
gpio_expander	GPIO expander chip (e.g. TCA9554)
custom	Custom device type; for implementing non-standard peripherals or special features

## Peripheral Configuration

`board_peripherals.yaml` describes the pin assignments and parameters for low-level hardware resources:

- **I2C:** SDA/SCL pins and port number
- **I2S:** MCLK/BCLK/WS/DOUT/DIN pins, sample rate, and bit depth
- **SPI:** MOSI/MISO/CLK/CS pins, SPI host number, and transfer size
- **LEDC:** Backlight GPIO, PWM frequency, and resolution
- **GPIO:** Standalone pin configurations such as power control, amplifier enable, and LEDs

`sdkconfig.defaults.board` contains Kconfig defaults tightly coupled to the board hardware, such as Flash size, PSRAM mode and frequency, CPU clock frequency, and audio recording format parameters for `brooke-sia_hal_adaptor`.

If a driver requires a custom initialization flow, such as passing a vendor-specific register sequence to an LCD driver, it is implemented through factory callbacks in `setup_device.c`.

### 3.3.4 Usage

#### Select a Board

See [How to Use Example Projects](#).

### Add a Custom Board

Create a new board subdirectory under `boards/<vendor>/` and add the following files in order:

1. **board\_info.yaml**: Fill in the board name, chip model, version, and description.
2. **board\_peripherals.yaml**: Configure peripheral parameters based on the actual pins and buses.
3. **board\_devices.yaml**: Describe on-board devices with their types and configurations.
4. **sdkconfig.defaults.board**: Add board-specific Kconfig defaults.
5. **setup\_device.c** (*optional*): Implement factory functions if the driver requires extra initialization steps.
6. **components/...** (*optional*): Add special component dependencies for non-standard peripherals or special features.

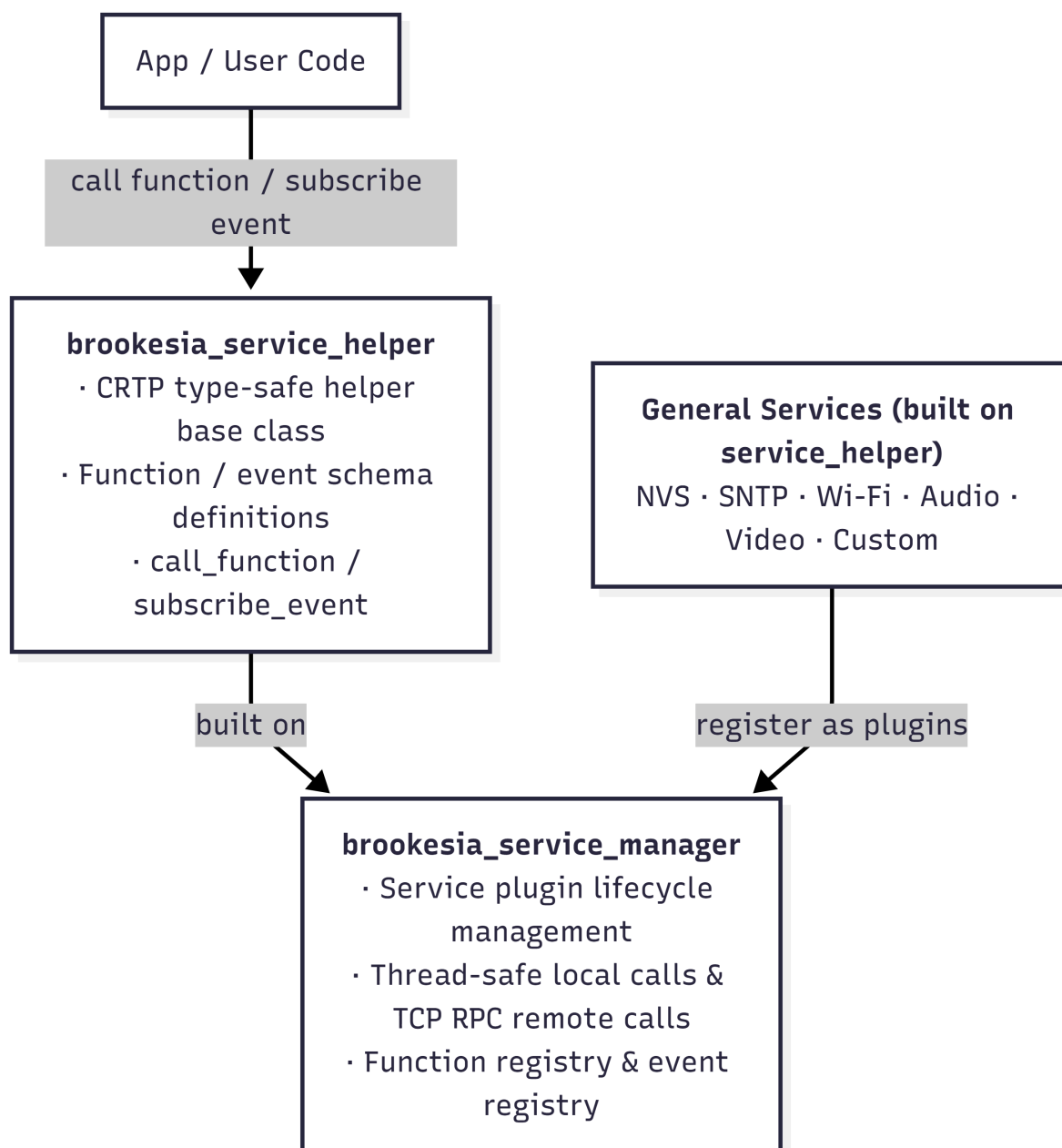
Once done, run `idf.py gen-bmgr-config -b <new_board>` to use the new board.



## **Chapter 4**

# **Service Components**

This section documents the ESP-Brookesia service framework components. The framework consists of a service framework layer and a general-services layer. Their component hierarchy is shown below:



- `brookesia_service_manager`: The service framework core, responsible for plugin registration, function routing, and event dispatching in both local and RPC communication modes.
- `brookesia_service_helper`: A CRTP-based type-safe helper layer that simplifies service function/event definition and invocation.
- **General Services**: Concrete services implemented on top of `service_helper`, registered into `service_manager` and discoverable by name from upper layers.

## 4.1 Service Framework

### 4.1.1 Service Manager

- Component registry:  [espressif/brookesia\\_service\\_manager](#)
- Public header: `#include "brookesia/service_manager.hpp"`

## Overview

*brookesia\_service\_manager* is the core service framework: lifecycle management, function and event registration, local calls, and remote RPC.

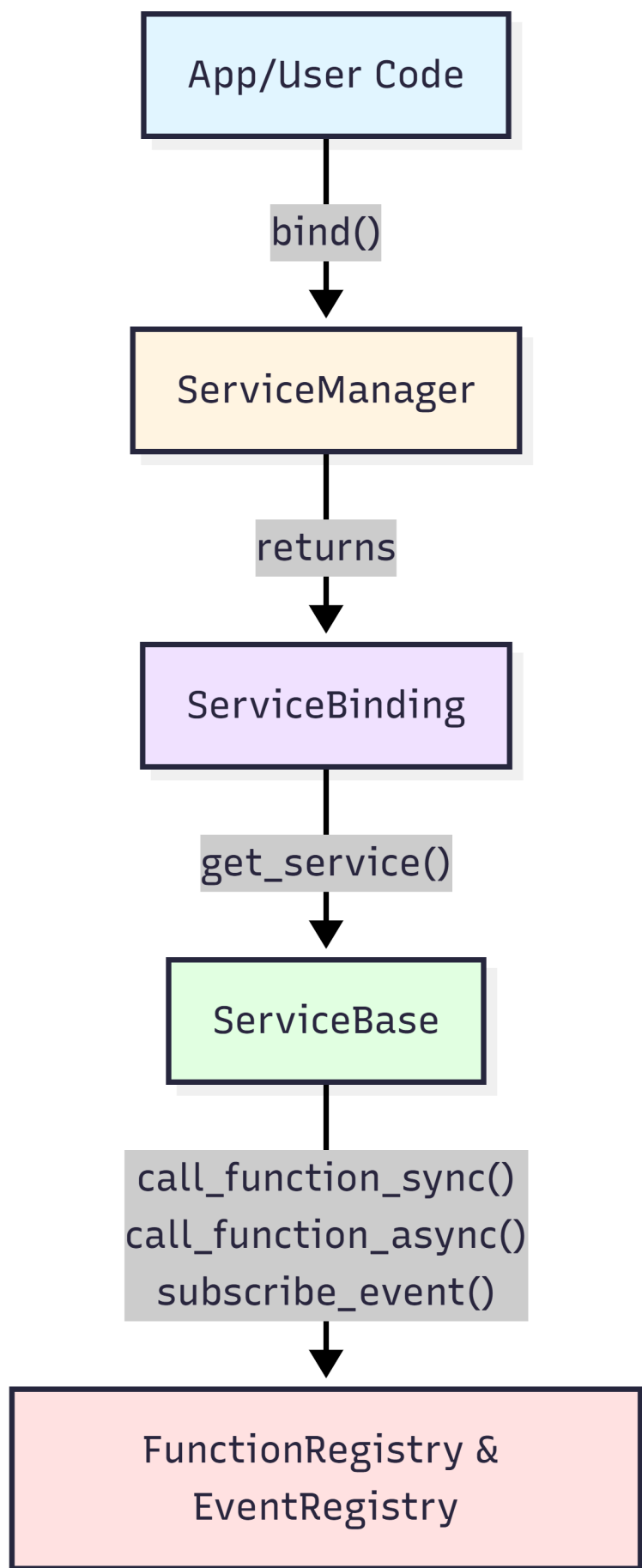
## Features

- **Lifecycle:** Centralized init, start, stop, and deinit.
- **Dual mode:** High-performance local calls and TCP/JSON RPC.
- **Unified model:** Function definitions, registries, events, and subscriptions.
- **Thread safety:** Local runners and schedulers for safe execution.
- **Decoupling:** Apps call through a stable API without depending on provider details.

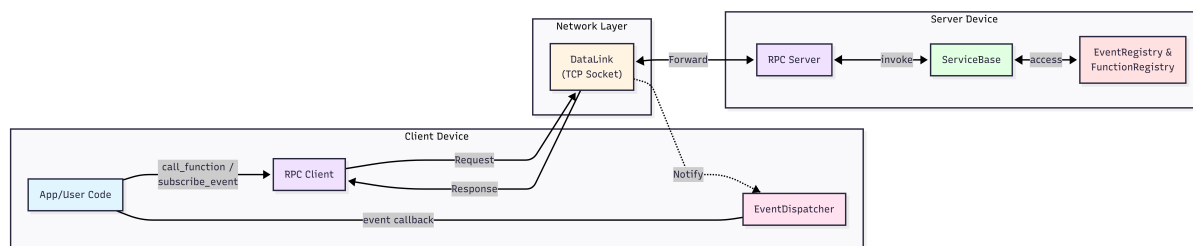
## Communication Architecture

*brookesia\_service\_manager* supports **local** and **remote RPC** modes.

**Local Mode** After *ServiceManager* binds a service, the app uses *ServiceBase* to access function and event registries with minimal overhead.



**Remote RPC Mode** Clients use TCP sockets and JSON to reach remote services—suitable for cross-device or cross-language use.



### Local vs Remote RPC

Item	Local ( <i>ServiceBase</i> )	Remote RPC ( <i>rpc::Client</i> )
Deployment	Same device	Cross-device
Transport	Direct calls	TCP + JSON
Latency	Low (ms)	Higher (network)
Performance	No serialization	Serialize/deserialize
Call rate	High frequency	Medium/low frequency
Thread safety	Async scheduling & guards	Network isolation
Languages	C++	Language-agnostic
Network	Not required	LAN or routable network
Typical use	In-device collaboration	Cross-device / cross-language calls

### Modules

**Service Runtime** Lifecycle, dispatch, binding, and RPC integration—built on `ServiceBase`, `ServiceManager`, and `LocalTestRunner`.

[API reference](#)

**Function System** Defines callable interfaces, validates parameters, and dispatches to handlers—function model and registry.

[API reference](#)

**Event System** Event definitions, validation, and dispatch to local subscribers or RPC subscribers—definitions, registry, and dispatcher.

[API reference](#)

**RPC** TCP + JSON exposure of functions and events: protocol, data link, bridging, client, and server.

[API reference](#)

**Common** Shared types and macros for the service manager.

[API reference](#)

### Service Runtime Layer

### API Reference

**Service Base** Public header: `#include "brookesia/service_manager/service/base.hpp"`

## Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/service/base.hpp](#)

## Classes

class **ServiceBase**

Base class for bindable services managed by *ServiceManager*.

Subclasses expose callable functions and publishable events through the function and event registries owned by the service.

Subclassed by *esp\_brookesia::agent::Base*, *esp\_brookesia::agent::Manager*, *esp\_brookesia::expression::Emote*, *esp\_brookesia::service::Audio*, *esp\_brookesia::service::CustomService*, *esp\_brookesia::service::Device*, *esp\_brookesia::service::NVS*, *esp\_brookesia::service::SNTP*, *esp\_brookesia::service::VideoDecoder*, *esp\_brookesia::service::VideoEncoder*, *esp\_brookesia::service::wifi::Wifi*

## Public Types

using **FunctionHandlerMap** = `std::map<std::string, FunctionHandler>`

Map from function names to service-side handlers.

using **FunctionResultHandler** = `std::function<void(FunctionResult&&)>`

Callback invoked with the result of an asynchronous function call.

using **FunctionBatchResultHandler** = `std::function<void(FunctionBatchResult&&)>`

Callback invoked with the result of an asynchronous batched function call.

## Public Functions

inline **ServiceBase** (const *Attributes* &attributes)

Construct a service with immutable attributes.

**Parameters** *attributes* -- [in] Public metadata and scheduler preferences for the service.

virtual **~ServiceBase** ()

Virtual destructor.

inline virtual `std::vector<FunctionSchema>` **get\_function\_schemas** ()

Get the function schemas list.

Subclasses should override this method to return an array of function schemas

```
std::vector<FunctionSchema> get_function_schemas() override {
    return {
        {
            "add", "Add numbers", {
                {"a", "First", FunctionValueType::Number},
                {"b", "Second", FunctionValueType::Number}
            }
        },
        {
```

(continues on next page)

(continued from previous page)

```

        "sub", "Subtract", {
            {"a", "First", FunctionValueType::Number},
            {"b", "Second", FunctionValueType::Number}
        }
    };
}

```

**Returns** `std::vector<FunctionSchema>` List of function schemas

inline virtual `std::vector<EventSchema>` **get\_event\_schemas** ()

Get the event schemas list.

Subclasses should override this method to return an array of event schemas

```

std::vector<EventSchema> get_event_schemas() override {
    return {
        {
            "value_change", "Value changed", {
                {"value", "New value", EventItemType::Number}
            }
        }
    };
}

```

**Returns** `std::vector<EventSchema>` List of event schemas

bool **call\_function\_async** (const `std::string` &name, `FunctionParameterMap` parameters\_map, *FunctionResultHandler* handler = nullptr)

Call a function asynchronously with parameters map (non-blocking)

#### Parameters

- **name** -- [in] Function name to call
- **parameters\_map** -- [in] `FunctionParameterMap` map (key-value pairs)
- **handler** -- [in] `FunctionResultHandler` to handle the result, if not provided, the result will be ignored

**Returns** true if called successfully, false otherwise

bool **call\_function\_async** (const `std::string` &name, `std::vector<FunctionValue>` parameters\_values, *FunctionResultHandler* handler = nullptr)

Call a function asynchronously with parameters values (non-blocking)

#### Parameters

- **name** -- [in] Function name to call
- **parameters\_values** -- [in] `FunctionParameterMap` values (ordered array)
- **handler** -- [in] `FunctionResultHandler` to handle the result, if not provided, the result will be ignored

**Returns** true if called successfully, false otherwise

bool **call\_function\_async** (const `std::string` &name, const `boost::json::object` &parameters\_json, *FunctionResultHandler* handler = nullptr)

Call a function asynchronously with JSON parameters (non-blocking)

#### Parameters

- **name** -- [in] Function name to call
- **parameters\_json** -- [in] `FunctionParameterMap` in JSON object format
- **handler** -- [in] `FunctionResultHandler` to handle the result, if not provided, the result will be ignored

**Returns** true if called successfully, false otherwise

bool **call\_functions\_async** (std::vector<FunctionCall> calls, *FunctionBatchResultHandler* handler = nullptr)

Call multiple functions asynchronously on this service in order.

The batch is fail-fast: once one function fails, later functions are skipped.

**Parameters**

- **calls** -- **[in]** Function calls to execute.
- **handler** -- **[in]** *FunctionBatchResultHandler* to handle the batch result.

**Returns** true if the batch was accepted, false otherwise

FunctionResult **call\_function\_sync** (const std::string &name, FunctionParameterMap parameters\_map, uint32\_t timeout\_ms = BROOKE-SIA\_SERVICE\_MANAGER\_DEFAULT\_CALL\_FUNCTION\_TIMEOUT\_MS)

Call a function synchronously with parameters map (blocking with timeout)

**Parameters**

- **name** -- **[in]** Function name to call
- **parameters\_map** -- **[in]** FunctionParameterMap map (key-value pairs)
- **timeout\_ms** -- **[in]** Timeout in milliseconds (default: 100ms)

**Returns** FunctionResult Result of the function call

FunctionResult **call\_function\_sync** (const std::string &name, std::vector<FunctionValue> parameters\_values, uint32\_t timeout\_ms = BROOKE-SIA\_SERVICE\_MANAGER\_DEFAULT\_CALL\_FUNCTION\_TIMEOUT\_MS)

Call a function synchronously with parameters values (blocking with timeout)

**Parameters**

- **name** -- **[in]** Function name to call
- **parameters\_values** -- **[in]** FunctionParameterMap values (ordered array)
- **timeout\_ms** -- **[in]** Timeout in milliseconds (default: 100ms)

**Returns** FunctionResult Result of the function call

FunctionResult **call\_function\_sync** (const std::string &name, const boost::json::object &parameters\_json, uint32\_t timeout\_ms = BROOKE-SIA\_SERVICE\_MANAGER\_DEFAULT\_CALL\_FUNCTION\_TIMEOUT\_MS)

Call a function synchronously with JSON parameters (blocking with timeout)

**Parameters**

- **name** -- **[in]** Function name to call
- **parameters\_json** -- **[in]** FunctionParameterMap in JSON object format
- **timeout\_ms** -- **[in]** Timeout in milliseconds (default: 100ms)

**Returns** FunctionResult Result of the function call

FunctionBatchResult **call\_functions\_sync** (std::vector<FunctionCall> calls, uint32\_t timeout\_ms = BROOKE-SIA\_SERVICE\_MANAGER\_DEFAULT\_CALL\_FUNCTION\_TIMEOUT\_MS)

Call multiple functions synchronously on this service in order.

**Parameters**

- **calls** -- **[in]** Function calls to execute.
- **timeout\_ms** -- **[in]** Timeout in milliseconds.

**Returns** FunctionBatchResult Result of the batch call

*EventRegistry::SignalConnection* **subscribe\_event** (const std::string &event\_name, const *EventRegistry::SignalSlot* &slot)

Subscribe to an event.

**Parameters**

- **event\_name** -- **[in]** Event name to subscribe
- **slot** -- **[in]** Callback slot to be invoked when event is published

**Returns** *EventRegistry::SignalConnection* RAII scoped connection object for managing the subscription, automatically disconnects the subscription when the connection object is destroyed.

inline bool **is\_initialized** () const

Check if the service is initialized.

**Returns** true if initialized, false otherwise

inline bool **is\_running** () const

Check if the service is running.

**Returns** true if running, false otherwise

inline bool **is\_server\_connected** () const

Check if the service is connected to server.

**Returns** true if connected, false otherwise

inline const *Attributes* &**get\_attributes** () const

Get the service attributes.

**Returns** const *Attributes*& Reference to service attributes

inline virtual std::string **get\_call\_task\_group** () const

Get the call task group name.

**Returns** std::string Call task group name

inline virtual std::string **get\_event\_task\_group** () const

Get the event task group name.

**Returns** std::string Event task group name

inline virtual std::string **get\_request\_task\_group** () const

Get the request task group name.

**Returns** std::string Request task group name

### Public Static Functions

template<typename T>

static inline FunctionResult **to\_function\_result** (std::expected<T, std::string> result)

Helper function to convert std::expected to FunctionResult.

**Template Parameters** **T** -- Return value type, can be void or any type convertible to FunctionValue

**Parameters** **result** -- [in] std::expected object

**Returns** FunctionResult Converted result

struct **Attributes**

Service attributes configuration.

### Public Functions

inline bool **has\_scheduler** () const

Check whether a dedicated task scheduler configuration is present.

**Returns** true if the service should create its own scheduler.

```
inline const lib_utils::TaskScheduler::StartConfig &get_scheduler_config () const
```

Get the dedicated task scheduler configuration.

---

**Note:** Call this only when `has_scheduler()` returns `true`.

---

**Returns** const `lib_utils::TaskScheduler::StartConfig&` Config stored in `task_scheduler_config`.

### Public Members

std::string **name**

Service name.

std::vector<std::string> **dependencies** = {}

Optional list of dependent service names, started in order.

std::optional<lib\_utils::TaskScheduler::StartConfig> **task\_scheduler\_config** = std::nullopt

Optional task scheduler configuration.

If configured, service request tasks will be scheduled to this scheduler; otherwise, *ServiceManager*'s scheduler will be used.

bool **bindable** = true

Optional: Whether the service can be bound.

### Macros

**BROOKESIA\_SERVICE\_FUNC\_HANDLER\_0** (func\_name, func\_call)

**BROOKESIA\_SERVICE\_FUNC\_HANDLER\_1** (func\_name, param\_name, param\_type, func\_call)

**BROOKESIA\_SERVICE\_FUNC\_HANDLER\_2** (func\_name, param1\_name, param1\_type, param2\_name, param2\_type, func\_call)

**BROOKESIA\_SERVICE\_FUNC\_HANDLER\_3** (func\_name, p1\_name, p1\_type, p2\_name, p2\_type, p3\_name, p3\_type, func\_call)

**Service Manager** Public header: `#include "brookesia/service_manager/service/manager.hpp"`

### Header File

- `service/brookesia_service_manager/include/brookesia/service_manager/service/manager.hpp`

### Classes

class **ServiceBinding**

Service binding handle.

RAII wrapper for service binding that automatically releases the service and its dependencies when the binding goes out of scope.

**Public Functions**

inline bool **is\_valid** () const

Check if the binding is valid.

**Returns** true if valid and service is running, false otherwise

inline explicit **operator bool** () const

Explicit conversion to bool.

**Returns** true if valid, false otherwise

inline std::shared\_ptr<*ServiceBase*> **get\_service** () const

Get the service object.

**Returns** std::shared\_ptr<ServiceBase> Pointer to the service

inline std::shared\_ptr<*ServiceBase*> **get\_dependency\_service** (const std::string &name) const

Get a dependency service by name.

**Parameters name** -- [in] Dependency service name

**Returns** std::shared\_ptr<ServiceBase> Pointer to the dependency service, or nullptr if not found

void **release** ()

Release the service binding.

class **ServiceManager**

Service manager singleton.

Manages service lifecycle, dependencies, and RPC communication.

**Public Functions**

bool **init** ()

Initialize the service manager.

**Returns** true if initialized successfully, false otherwise

void **deinit** ()

Deinitialize the service manager.

bool **start** (const lib\_utils::TaskScheduler::StartConfig &config =  
DEFAULT\_TASK\_SCHEDULER\_START\_CONFIG)

Start the service manager.

**Parameters config** -- [in] Task scheduler start configuration

**Returns** true if started successfully, false otherwise

void **stop** ()

Stop the service manager.

bool **add\_service** (std::shared\_ptr<*ServiceBase*> service)

Add a service to the service manager.

**Parameters service** -- [in] Service to add

**Returns** true if added successfully, false otherwise

bool **remove\_service** (const std::string &name)

Remove a service by name.

**Parameters name** -- [in] Service name

**Returns** true if removed successfully, false otherwise

*ServiceBinding* **bind** (const std::string &name)

Bind a service by name.

**Parameters** **name** -- [in] Service name

**Returns** *ServiceBinding* Service binding handle

bool **start\_rpc\_server** (const rpc::Server::Config &config = rpc::Server::Config(), uint32\_t timeout\_ms = 100)

Start the RPC server.

**Parameters**

- **config** -- [in] RPC server configuration
- **timeout\_ms** -- [in] Timeout in milliseconds (default: 100ms)

**Returns** true if started successfully, false otherwise

void **stop\_rpc\_server** ()

Stop the RPC server.

bool **connect\_rpc\_server\_to\_services** (std::vector<std::string> names = {})

Connect RPC server to services.

**Parameters** **names** -- [in] Service names to connect (empty means all services)

**Returns** true if connected successfully, false otherwise

bool **disconnect\_rpc\_server\_from\_services** (std::vector<std::string> names = {})

Disconnect RPC server from services.

**Parameters** **names** -- [in] Service names to disconnect (empty means all services)

**Returns** true if disconnected successfully, false otherwise

std::shared\_ptr<rpc::Client> **new\_rpc\_client** (const *RPC\_ClientConfig* &config = *RPC\_ClientConfig*())

Create a new RPC client.

**Parameters** **config** -- [in] RPC client configuration

**Returns** std::shared\_ptr<rpc::Client> Shared pointer to the RPC client

FunctionResult **call\_rpc\_function\_sync** (std::string host, const std::string &service\_name, const std::string &function\_name, boost::json::object params, uint32\_t timeout\_ms = BROOKE-SIA\_SERVICE\_MANAGER\_RPC\_CLIENT\_CALL\_FUNCTION\_TIMEOUT, uint16\_t port = BROOKE-SIA\_SERVICE\_MANAGER\_RPC\_SERVER\_LISTEN\_PORT)

Call an RPC function synchronously.

**Parameters**

- **host** -- [in] Host address
- **service\_name** -- [in] Service name
- **function\_name** -- [in] Function name
- **params** -- [in] Function parameters in JSON format
- **timeout\_ms** -- [in] Timeout in milliseconds (default: configured timeout)
- **port** -- [in] Server port (default: configured port)

**Returns** FunctionResult Result of the function call

inline bool **is\_initialized** () const

Check if the service manager is initialized.

**Returns** true if initialized, false otherwise

inline bool **is\_running** () const

Check if the service manager is running.

**Returns** true if running, false otherwise

```
inline bool is_rpc_server_running() const
```

Check if the RPC server is running.

**Returns** true if running, false otherwise

```
inline std::shared_ptr<ServiceBase> get_service(const std::string &name)
```

Get a service by name.

**Parameters** **name** -- [in] Service name

**Returns** std::shared\_ptr<ServiceBase> Pointer to the service, or nullptr if not found

### Public Static Functions

```
static inline lib_utils::TaskScheduler::StartConfig make_default_task_scheduler_start_config()
```

Default worker configuration used by `start()`.

The configuration creates two worker threads for service dispatching and uses the module-level scheduling defaults defined in `macro_configs.h`.

```
static inline ServiceManager &get_instance()
```

Get the singleton instance.

**Returns** `ServiceManager&` Reference to the singleton instance

```
struct RPC_ClientConfig
```

Optional callbacks applied to RPC clients created by the manager.

### Public Members

```
rpc::Client::DisconnectCallback on_disconnect_callback
```

Called after the RPC transport disconnects.

```
rpc::Client::DeinitCallback on_deinit_callback
```

Called when the client is deinitialized.

**Local Test Runner** Public header: `#include "brookesia/service_manager/service/local_runner.hpp"`

### Header File

- `service/brookesia_service_manager/include/brookesia/service_manager/service/local_runner.hpp`

### Classes

```
class LocalTestRunner
```

Local service test runner.

A test framework based on `TaskScheduler` that supports executing test sequences in order, where each test item can specify a start delay and run duration.

### Public Functions

bool **run\_tests** (const *RunTestsConfig* &config, const std::vector<LocalTestItem> &test\_items)

Run test sequence.

#### Parameters

- **config** -- [in] Run tests configuration
- **test\_items** -- [in] List of test items

**Returns** true if all tests passed, false otherwise

inline bool **run\_tests** (const std::string &service\_name, const std::vector<LocalTestItem> &test\_items)

Run test sequence with default configuration.

#### Parameters

- **service\_name** -- [in] Service name to test
- **test\_items** -- [in] List of test items

**Returns** true if all tests passed, false otherwise

const std::vector<bool> &**get\_results** () const

Get test results.

**Returns** const std::vector<bool>& Result of each test item

struct **RunTestsConfig**

### Function System

#### API Reference

**Function Definition** Public header: #include "brookesia/service\_manager/function/definition.hpp"

#### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/function/definition.hpp](#)

**Function Registry** Public header: #include "brookesia/service\_manager/function/registry.hpp"

#### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/function/registry.hpp](#)

### Classes

class **FunctionRegistry**

Registry of callable service functions and their schemas.

### Public Functions

bool **add** (FunctionSchema func\_schema, FunctionHandler func\_handler)

Register a function schema and its handler.

**Parameters**

- **func\_schema** -- [in] Function metadata.
- **func\_handler** -- [in] Handler that executes the function.

**Returns** true on success, false if the function name already exists.

bool **remove** (const std::string &func\_name)

Remove a registered function.

**Parameters** **func\_name** -- [in] Function name to remove.

**Returns** true if the function was removed.

bool **remove\_all** ()

Remove every registered function.

**Returns** true if the registry was cleared successfully.

FunctionResult **call** (const std::string &func\_name, FunctionParameterMap parameters)

Validate parameters and invoke a registered function.

**Parameters**

- **func\_name** -- [in] Function name to call.
- **parameters** -- [in] Parameter map passed to the handler.

**Returns** FunctionResult Execution result or validation failure information.

inline const FunctionSchema \***get\_schema** (const std::string &func\_name)

Look up the schema for a registered function.

**Parameters** **func\_name** -- [in] Function name to query.

**Returns** const FunctionSchema\* Pointer to the schema, or `nullptr` if not found.

std::vector<FunctionSchema> **get\_schemas** () const

Get a snapshot of all registered function schemas.

**Returns** std::vector<FunctionSchema> Copy of the registered schemas.

boost::json::array **get\_schemas\_json** ()

Export all registered function schemas as JSON.

**Returns** boost::json::array JSON representation of every schema.

inline bool **has** (const std::string &func\_name)

Check whether a function is registered.

**Parameters** **func\_name** -- [in] Function name to check.

**Returns** true if the function exists.

inline size\_t **get\_count** ()

Get the number of registered functions.

**Returns** size\_t Count of registered functions.

## Event System

**API Reference** Public header: `#include "brookesia/service_manager/event/definition.hpp"`

## Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/event/definition.hpp](service/brookesia_service_manager/include/brookesia/service_manager/event/definition.hpp)

**Event Dispatcher** Public header: `#include "brookesia/service_manager/event/dispatcher.hpp"`

### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/event/dispatcher.hpp](#)

### Classes

class **EventDispatcher**

Dispatches RPC event notifications to local subscription callbacks.

### Public Types

using **NotifyCallback** = std::function<void(const EventItemMap&)>

Callback invoked when a subscribed event notification arrives.

### Public Functions

bool **subscribe** (const std::string &subscription\_id, *NotifyCallback* callback)

Register a callback for a subscription identifier.

#### Parameters

- **subscription\_id** -- **[in]** Subscription id returned by the RPC server.
- **callback** -- **[in]** Callback to invoke when matching notifications arrive.

**Returns** true on success, false if the subscription id is already registered.

void **unsubscribe** (const std::string &subscription\_id)

Remove a previously registered subscription callback.

**Parameters** **subscription\_id** -- **[in]** Subscription id to remove.

void **on\_notify** (const std::vector<std::string> &subscription\_ids, const EventItemMap &event\_items)

Dispatch an incoming notification to all matching local callbacks.

#### Parameters

- **subscription\_ids** -- **[in]** Subscription ids targeted by the notification.
- **event\_items** -- **[in]** Event payload associated with the notification.

**Event Registry** Public header: `#include "brookesia/service_manager/event/registry.hpp"`

### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/event/registry.hpp](#)

### Classes

class **EventRegistry**

Registry of event schemas and active RPC subscriptions for one service.

## Public Types

using **Subscriptions** = std::unordered\_set<std::string>

Set of subscription identifiers attached to an event.

using **Signal** = boost::signals2::signal<void(const std::string &event\_name, const EventItemMap &event\_items)>

Signal type used for local in-process event listeners.

using **SignalConnection** = boost::signals2::scoped\_connection

RAII connection handle returned by `Signal::connect()`.

using **SignalSlot** = *Signal*::slot\_type

Slot type accepted by `subscribe_event()`.

## Public Functions

bool **add** (EventSchema event\_schema)

Register an event schema.

**Parameters** `event_schema` -- **[in]** Schema to add.

**Returns** true on success, false if an event with the same name already exists.

void **remove** (const std::string &event\_name)

Remove a registered event schema.

**Parameters** `event_name` -- **[in]** Name of the event to remove.

void **remove\_all** ()

Remove all registered event schemas.

bool **validate\_items** (const std::string &event\_name, const EventItemMap &event\_items)

Validate an event payload against a registered schema.

**Parameters**

- `event_name` -- **[in]** Event name to validate against.

- `event_items` -- **[in]** Event payload to validate.

**Returns** true if the payload conforms to the schema, false otherwise.

bool **on\_rpc\_subscribe** (const std::string &event\_name, std::string &subscription\_id, std::string &error\_message)

Create a new RPC subscription for the given event.

**Parameters**

- `event_name` -- **[in]** Name of the event to subscribe to.

- `subscription_id` -- **[out]** Generated subscription id on success.

- `error_message` -- **[out]** Validation or registration failure details.

**Returns** true if the subscription was created, false otherwise.

void **on\_rpc\_unsubscribe\_by\_name** (const std::string &event\_name)

Remove all RPC subscriptions associated with an event name.

**Parameters** `event_name` -- **[in]** Name of the event whose subscriptions should be removed.

void **on\_rpc\_unsubscribe\_by\_subscriptions** (const *Subscriptions* &subscriptions)

Remove a set of RPC subscriptions from every registered event.

**Parameters** `subscriptions` -- **[in]** Subscription ids to remove.

inline const EventSchema \***get\_schema** (const std::string &event\_name)

Look up the schema for a registered event.

**Parameters** **event\_name** -- **[in]** Event name to query.

**Returns** const EventSchema\* Pointer to the schema, or `nullptr` if not found.

std::vector<EventSchema> **get\_schemas** () const

Get a snapshot of all registered event schemas.

**Returns** std::vector<EventSchema> Copy of the registered schemas.

boost::json::array **get\_schemas\_json** ()

Export all registered event schemas as JSON.

**Returns** boost::json::array JSON representation of every schema.

inline bool **has** (const std::string &event\_name)

Check whether an event schema exists.

**Parameters** **event\_name** -- **[in]** Event name to check.

**Returns** true if the event is registered.

inline size\_t **get\_count** ()

Get the number of registered event schemas.

**Returns** size\_t Count of registered events.

*Subscriptions* **get\_subscriptions** (const std::string &event\_name)

Get all RPC subscription ids associated with an event.

**Parameters** **event\_name** -- **[in]** Event name to query.

**Returns** Subscriptions Copy of the registered subscription ids.

*Signal* \***get\_signal** (const std::string &event\_name)

Get the in-process signal associated with an event.

**Parameters** **event\_name** -- **[in]** Event name to query.

**Returns** Signal\* Pointer to the signal, or `nullptr` if the event does not exist.

## RPC Communication

### API Reference

**RPC Protocol** Public header: `#include "brookesia/service_manager/rpc/protocol.hpp"`

#### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/rpc/protocol.hpp](#)

**RPC Connection** Public header: `#include "brookesia/service_manager/rpc/connection.hpp"`

#### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/rpc/connection.hpp](#)

### Classes

class **ServerConnection**

Bridges one service's registries to the RPC server transport.

## Public Types

using **Responder** = std::function<bool(size\_t, Response&&)>

Callback used to send an RPC response back to a client connection.

using **Notifier** = std::function<bool(std::size\_t, Notify&&)>

Callback used to push an RPC notification to a client connection.

using **RequestHandler** = std::function<bool(size\_t, std::string&&, std::string&&, FunctionParameterMap&&)>

Optional callback that overrides request handling for custom routing.

## Public Functions

inline **ServerConnection** (std::string name, *FunctionRegistry* &function\_registry, *EventRegistry* &event\_registry)

Construct a server-side connection wrapper for one service.

### Parameters

- **name** -- **[in]** Service name exposed to RPC clients.
- **function\_registry** -- **[in]** Registry used for function calls.
- **event\_registry** -- **[in]** Registry used for event subscriptions and notifications.

inline void **set\_responder** (*Responder* responder)

Set the responder used for sending RPC responses.

**Parameters** **responder** -- **[in]** Transport callback.

inline void **set\_notifier** (*Notifier* notifier)

Set the notifier used for sending RPC notifications.

**Parameters** **notifier** -- **[in]** Transport callback.

inline void **set\_request\_handler** (*RequestHandler* request\_handler)

Set a custom request handler.

**Parameters** **request\_handler** -- **[in]** Custom transport-aware request handler.

inline void **activate** (bool active)

Enable or disable request handling on this connection.

**Parameters** **active** -- **[in]** `true` to accept RPC traffic, `false` to reject it.

std::expected<std::shared\_ptr<FunctionResult>, std::string> **on\_request** (std::string &&request\_id, size\_t connection\_id, std::string &&method, FunctionParameterMap &&parameters)

Process an incoming RPC request for this service.

### Parameters

- **request\_id** -- **[in]** RPC request id.
- **connection\_id** -- **[in]** Transport connection id.
- **method** -- **[in]** Requested method name.
- **parameters** -- **[in]** Method parameters.

**Returns** std::expected<std::shared\_ptr<FunctionResult>, std::string> Function result on success, or an error.

void **on\_connection\_closed** (size\_t connection\_id)

Clear per-connection subscription state when a client disconnects.

**Parameters** **connection\_id** -- **[in]** Closed transport connection id.

bool **publish\_event** (const std::string &event\_name, const EventItemMap &event\_items)

Publish a service event to subscribed RPC clients.

**Parameters**

- **event\_name** -- **[in]** Event name to publish.
- **event\_items** -- **[in]** Event payload.

**Returns** true if notifications were dispatched successfully.

bool **respond\_request** (size\_t connection\_id, Response &&response)

Send an RPC response to one client connection.

**Parameters**

- **connection\_id** -- **[in]** Transport connection id.
- **response** -- **[in]** Response to send.

**Returns** true if the response was handed to the transport.

**RPC Data Link Base** Public header: #include "brookesia/service\_manager/rpc/data\_link\_base.hpp"

### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/rpc/data\\_link\\_base.hpp](#)

### Classes

class **DataLinkBase**

Abstract TCP data-link layer shared by RPC clients and servers.

Subclassed by *esp\_brookesia::service::rpc::DataLinkClient*, *esp\_brookesia::service::rpc::DataLinkServer*

### Public Types

using **OnDataReceived** = std::function<void(const std::string &data, size\_t connection\_id)>

Callback invoked when a complete payload string is received.

using **OnConnectionEstablished** = std::function<void(size\_t connection\_id)>

Callback invoked when a transport connection becomes active.

using **OnConnectionClosed** = std::function<void(size\_t connection\_id)>

Callback invoked when a transport connection closes.

### Public Functions

inline **DataLinkBase** (boost::asio::io\_context::executor\_type executor)

Construct the base transport with an executor.

**Parameters** **executor** -- **[in]** Executor that owns asynchronous I/O operations.

inline void **set\_on\_data\_received** (*OnDataReceived* callback)

Set the callback used for incoming payloads.

**Parameters** **callback** -- [in] Callback to install.

inline void **set\_on\_connection\_established** (*OnConnectionEstablished* callback)

Set the callback used for successful connection establishment.

**Parameters** **callback** -- [in] Callback to install.

inline void **set\_on\_connection\_closed** (*OnConnectionClosed* callback)

Set the callback used for connection teardown.

**Parameters** **callback** -- [in] Callback to install.

### Public Static Functions

static inline size\_t **get\_active\_global\_sockets\_count** ()

Get the number of active sockets across all data-link instances.

**Returns** size\_t Current global socket count.

static inline size\_t **get\_max\_global\_sockets\_count** ()

Get the configured global socket cap.

**Returns** size\_t Maximum number of sockets allowed globally.

static inline bool **is\_global\_sockets\_limit\_reached** ()

Check whether the global socket cap has been reached.

**Returns** true if no more sockets should be opened.

static inline size\_t **get\_max\_active\_global\_sockets\_count** ()

Get the maximum number of simultaneously active sockets observed.

**Returns** size\_t Historical peak socket count.

**RPC Data Link Client** Public header: #include "brookesia/service\_manager/rpc/data\_link\_client.hpp"

### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/rpc/data\\_link\\_client.hpp](#)

### Classes

class **DataLinkClient** : public esp\_brookesia::service::rpc::DataLinkBase

TCP transport used by RPC clients.

### Public Functions

inline **DataLinkClient** (boost::asio::io\_context::executor\_type executor)

Construct a client transport on the given executor.

**Parameters** **executor** -- [in] Executor used for socket operations.

**~DataLinkClient** () override

Destructor.

bool **connect** (const std::string &host, uint16\_t port, size\_t timeout\_ms)

Connect to a remote server.

**Parameters**

- **host** -- **[in]** Remote host name or address.
- **port** -- **[in]** Remote TCP port.
- **timeout\_ms** -- **[in]** Connection timeout in milliseconds.

**Returns** true if the connection succeeds.

bool **disconnect** ()

Disconnect the active client connection.

**Returns** true if a connection existed and cleanup succeeded.

bool **send\_data** (std::string &&data)

Send one payload over the active connection.

**Parameters** **data** -- **[in]** Framed payload to send.

**Returns** true if the send operation was queued successfully.

bool **is\_connected** ()

Check whether the client currently has an active connection.

**Returns** true if connected.

**RPC Data Link Server** Public header: #include "brookesia/service\_manager/rpc/data\_link\_server.hpp"

### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/rpc/data\\_link\\_server.hpp](service/brookesia_service_manager/include/brookesia/service_manager/rpc/data_link_server.hpp)

### Classes

class **DataLinkServer** : public esp\_brookesia::service::rpc::DataLinkBase

TCP transport used by the RPC server to accept multiple clients.

### Public Functions

inline explicit **DataLinkServer** (boost::asio::io\_context::executor\_type executor, size\_t max\_connections)

Construct a server transport.

**Parameters**

- **executor** -- **[in]** Executor used for socket operations.
- **max\_connections** -- **[in]** Maximum number of concurrent client connections.

**~DataLinkServer** () override

Destructor.

bool **start** (uint16\_t port, size\_t timeout\_ms)

Start listening for client connections.

**Parameters**

- **port** -- **[in]** Local TCP port.
- **timeout\_ms** -- **[in]** Startup timeout in milliseconds.

**Returns** true if the server starts successfully.

void **stop** ()

Stop accepting clients and close all active connections.

bool **send\_data** (size\_t connection\_id, std::string &&data)

Send one payload to a connected client.

**Parameters**

- **connection\_id** -- [in] *Client* connection id.
- **data** -- [in] Framed payload to send.

**Returns** true if the payload was queued successfully.

size\_t **get\_active\_connections\_count** ()

Get the number of currently active client connections.

**Returns** size\_t Active connection count.

std::vector<size\_t> **get\_active\_connection\_ids** ()

Get the ids of all active client connections.

**Returns** std::vector<size\_t> Snapshot of active connection ids.

inline bool **is\_running** ()

Check whether the server transport is currently accepting traffic.

**Returns** true if running.

inline size\_t **get\_max\_connections\_count** ()

Get the configured maximum number of client connections.

**Returns** size\_t Connection cap.

inline bool **is\_connection\_limit\_reached** ()

Check whether the local connection cap has been reached.

**Returns** true if no more clients should be accepted.

inline size\_t **get\_max\_active\_connections\_count** ()

Get the peak number of simultaneously active client connections.

**Returns** size\_t Historical maximum for this server instance.

**RPC Client** Public header: #include "brookesia/service\_manager/rpc/client.hpp"

### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/rpc/client.hpp](#)

### Classes

class **Client**

RPC client used to call service functions and subscribe to service events.

### Public Types

using **DeinitCallback** = std::function<void()>

Callback invoked when the client is deinitialized.

using **DisconnectCallback** = std::function<void()>

Callback invoked after the transport disconnects.

## Public Functions

inline **Client** (*DeinitCallback* on\_deinit\_callback = nullptr)

Construct a client with an optional deinitialization callback.

**Parameters** **on\_deinit\_callback** -- [in] Callback invoked by *deinit()*.

**~Client** ()

Destructor.

bool **init** (boost::asio::io\_context::executor\_type executor, *DisconnectCallback* on\_disconnect\_callback)

Initialize the client transport on an executor.

### Parameters

- **executor** -- [in] Executor used for async socket operations.
- **on\_disconnect\_callback** -- [in] Callback invoked if the transport disconnects.

**Returns** true if initialization succeeds.

void **deinit** ()

Deinitialize the client and release its transport resources.

bool **connect** (const std::string &host, uint16\_t port, uint32\_t timeout\_ms)

Connect to an RPC server.

### Parameters

- **host** -- [in] Remote host name or address.
- **port** -- [in] Remote port.
- **timeout\_ms** -- [in] Connection timeout in milliseconds.

**Returns** true if the connection succeeds.

void **disconnect** ()

Disconnect from the current RPC server.

std::future<FunctionResult> **call\_function\_async** (const std::string &target, const std::string &method, boost::json::object &&params)

Call a remote service function asynchronously.

### Parameters

- **target** -- [in] Target service name.
- **method** -- [in] Remote function name.
- **params** -- [in] JSON object containing function parameters.

**Returns** std::future<FunctionResult> Future resolved with the call result.

FunctionResult **call\_function\_sync** (const std::string &target, const std::string &method, boost::json::object &&params, size\_t timeout\_ms)

Call a remote service function synchronously.

### Parameters

- **target** -- [in] Target service name.
- **method** -- [in] Remote function name.
- **params** -- [in] JSON object containing function parameters.
- **timeout\_ms** -- [in] Wait timeout in milliseconds.

**Returns** FunctionResult Call result or timeout/error information.

std::string **subscribe\_event** (const std::string &target, const std::string &event\_name, *EventDispatcher::NotifyCallback* callback, size\_t timeout\_ms)

Subscribe to a remote service event.

### Parameters

- **target** -- [in] Target service name.
- **event\_name** -- [in] Event name to subscribe to.
- **callback** -- [in] Callback invoked when notifications arrive.
- **timeout\_ms** -- [in] RPC timeout in milliseconds.

**Returns** `std::string` Subscription id, or an empty string on failure.

bool **unsubscribe\_events** (const `std::string` &target, const `std::vector<std::string>` &subscription\_ids, `size_t` timeout\_ms)

Unsubscribe from multiple remote event subscriptions.

#### Parameters

- **target** -- **[in]** Target service name.
- **subscription\_ids** -- **[in]** Subscription ids to cancel.
- **timeout\_ms** -- **[in]** RPC timeout in milliseconds.

**Returns** true if the unsubscribe request succeeds.

**RPC Server** Public header: `#include "brookesia/service_manager/rpc/server.hpp"`

#### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/rpc/server.hpp](#)

#### Classes

class **Server**

RPC server that exposes registered services over TCP.

#### Public Functions

inline **Server** (boost::asio::io\_context::executor\_type executor, const *Config* &config = *Config*())

Construct an RPC server on the given executor.

#### Parameters

- **executor** -- **[in]** Executor used for socket operations.
- **config** -- **[in]** Transport configuration.

**~Server** ()

Destructor.

bool **init** ()

Initialize internal transport resources.

**Returns** true if initialization succeeds.

void **deinit** ()

Deinitialize the server and release transport resources.

bool **start** (uint32\_t timeout\_ms)

Start listening for RPC clients.

**Parameters** **timeout\_ms** -- **[in]** Startup timeout in milliseconds.

**Returns** true if the server starts successfully.

void **stop** ()

Stop the RPC server.

bool **add\_connection** (std::shared\_ptr<*ServerConnection*> connection)

Attach a service connection to the server.

**Parameters** **connection** -- **[in]** Service connection wrapper to expose.

**Returns** true if the connection is added successfully.

bool **remove\_connection** (const std::string &name)

Remove a service connection by service name.

**Parameters** `name` -- **[in]** Service name to remove.

**Returns** true if a matching connection was removed.

std::shared\_ptr<*ServerConnection*> **get\_connection** (const std::string &name)

Get a registered service connection.

**Parameters** `name` -- **[in]** Service name to query.

**Returns** std::shared\_ptr<*ServerConnection*> Matching connection, or nullptr.

struct **Config**

Runtime configuration for the RPC server transport.

### Public Members

uint16\_t **listen\_port**

TCP port used for listening.

size\_t **max\_connections**

Maximum number of simultaneous client connections.

## Common

**API Reference** Public header: `#include "brookesia/service_manager/common.hpp"`

### Header File

- [service/brookesia\\_service\\_manager/include/brookesia/service\\_manager/common.hpp](#)

## 4.1.2 Service Helper

- Component registry: [espressif/brookesia\\_service\\_helper](#)
- Public header: `#include "brookesia/service_helper.hpp"`

### Overview

*brookesia\_service\_helper* is the unified application-facing helper layer for Brookesia services: typed wrappers and schema entry points.

### Features

- **Single entry:** Helpers encapsulate each service to simplify integration.
- **Schema-friendly:** Works with function/event schemas for standardized calls and docs.
- **Decoupling:** Apps target helpers instead of concrete providers.
- **Extensible:** New services can add helpers and documentation over time.

### Module Overview

## Base

- Public header: `#include "brookesia/service_helper/base.hpp"`

**Overview** `esp_brookesia::service::helper::Base<Derived>` is the CRTP base for Service Helpers, providing unified function calls, event subscriptions, and schema validation.

## Features

- **CRTP contract:** *DerivedMeta* requires *FunctionId* / *EventId*, *get\_name*, *get\_\*\_schemas*, etc.
- **Calls:** *call\_function\_sync* / *call\_function\_async* with packing and parsing.
- **Events:** Type-safe *subscribe\_event* overloads.
- **Availability:** *is\_available*, *get\_function\_schema*, *get\_event\_schema*.
- **Timeouts:** Optional *Timeout(ms)* on sync calls.
- **Macros:** `BROOKESIA_SERVICE_HELPER_FUNC_HANDLER_*` for handler boilerplate.

**Usage** Derive a helper with function/event enums and schemas:

```
class MyHelper : public esp_brookesia::service::helper::Base<MyHelper> {
public:
    enum class FunctionId { Ping };
    enum class EventId { Ready };
    static std::string_view get_name();
    static std::vector<FunctionSchema> get_function_schemas();
    static std::vector<EventSchema> get_event_schemas();
};
```

## API Reference

### Header File

- `service/brookesia_service_helper/include/brookesia/service_helper/base.hpp`

### Classes

template<typename **Derived**>

class **Base**

*Base* class for all service helpers (CRTP)

Note: Concept check is removed from template parameter to allow CRTP pattern where *Derived* is incomplete. Type and method checks are performed when methods are actually used via `static_assert` or compiler errors.

**Template Parameters** **Derived** -- The derived class (must be a subclass of *Base*)

### Public Static Functions

template<typename **ReturnType**>

static inline std::expected<*ReturnType*, std::string> **process\_function\_result** (const FunctionResult &result)

Helper function to process function result and convert to expected return type.

**Parameters** **result** -- [in] Function result from service call

**Returns** std::expected containing *ReturnType* or error message

template<typename **ReturnType** = void, typename **FunctionIdType**, typename ...**Args**>

```
static inline std::expected<ReturnTpe, std::string> call_function_sync (FunctionIdType function_id,
                                                                    Args&&... args)
```

Call a function synchronously with variadic arguments (timeout at end)

```
// No arguments with default timeout
auto result = call_function_sync<int>(FunctionId::GetValue);

// With arguments and default timeout
auto result = call_function_sync<int>(FunctionId::Add, 1.0, 2.0);

// With custom timeout at the end
auto result = call_function_sync<int>(FunctionId::Add, 1.0, 2.0,
↳Timeout(500));
```

---

**Note:** Arguments are packed into std::vector<FunctionValue> in the order provided

---



---

**Note:** Last argument can be Timeout(milliseconds) to specify custom timeout

---



---

**Note:** All non-Timeout arguments must satisfy ConvertibleToFunctionValue concept

---

#### Template Parameters

- **ReturnTpe** -- Expected return type (default: void)
- **FunctionIdType** -- Type of the function identifier (enum)
- **Args** -- Types of function arguments (must be convertible to FunctionValue)

#### Parameters

- **function\_id** -- Function identifier
- **args** -- Function arguments in order, last argument can be Timeout(ms)

**Returns** std::expected<ReturnTpe, std::string> Function result or error

```
template<typename FunctionIdType, typename ...Args>
static inline bool call_function_async (FunctionIdType function_id, Args&&... args)
```

Call a function asynchronously with variadic arguments.

```
// No arguments
call_function_async(FunctionId::GetValue);

// With arguments
call_function_async(FunctionId::Add, 1.0, 2.0);

// With handler
call_function_async(FunctionId::Add, 1.0, 2.0, [] (FunctionResult &&result)
↳{
    // Handle result
});
```

---

**Note:** Arguments are packed into std::vector<FunctionValue> in the order provided

---



---

**Note:** All arguments (except optional FunctionResultHandler) must satisfy ConvertibleToFunctionVa-

lue concept

---

**Note:** This overload is not used when the first argument is `FunctionParameterMap` or `boost::json::object`

---

### Template Parameters

- **FunctionIdType** -- Type of the function identifier (enum)
- **Args** -- Types of function arguments (must be convertible to `FunctionValue`, optionally with `FunctionResultHandler` at the end)

### Parameters

- **function\_id** -- Function identifier
- **args** -- Function arguments in order, optionally with `FunctionResultHandler` at the end

**Returns** true if the call was successfully submitted, false otherwise

```
template<typename EventIdType>
static inline EventRegistry::SignalConnection subscribe_event (EventIdType event_id,
                                                             EventRegistry::SignalSlot slot)
```

Subscribe to an event with a raw `SignalSlot`.

```
auto conn = subscribe_event (EventId::ValueChanged,
                             [] (const std::string &event_name, const EventItemMap &items) {
                                 // Handle event with raw items directly
                             });
```

**Template Parameters** **EventIdType** -- Type of the event identifier (enum)

### Parameters

- **event\_id** -- Event identifier
- **slot** -- Signal slot function with signature: `void(const std::string &event_name, const EventItemMap &event_items)`

**Returns** *EventRegistry::SignalConnection* Connection object (scoped, automatically unsubscribes on destruction)

```
template<typename EventIdType, typename Callable>
static inline EventRegistry::SignalConnection subscribe_event (EventIdType event_id, Callable
                                                             &&callback)
```

Subscribe to an event with automatic parameter extraction.

```
// Event with two schema parameters: (string name, double value)
auto conn = subscribe_event (EventId::ValueChanged,
                             [] (const std::string &event_name, const std::string &name, double_
↪value) {
                             std::cout << event_name << ": " << name << " = " << value <<_
↪std::endl;
                             });

// Event with no schema parameters, only event_name
auto conn = subscribe_event (EventId::Ready,
                             [] (const std::string &event_name) {
                             std::cout << event_name << " triggered!" << std::endl;
                             });

// Using string_view for event_name
auto conn = subscribe_event (EventId::StatusChanged,
                             [] (std::string_view event_name, bool status) {
```

(continues on next page)

(continued from previous page)

```

        std::cout << event_name << ": " << (status ? "ON" : "OFF") <<␣
↪std::endl;
    });

```

**Note:** REQUIRED: First parameter must be `std::string` or `std::string_view` to receive `event_name`.

**Note:** Second parameter must not be `EventItemMap`; use the other overload for that case.

**Note:** Remaining parameters are extracted from `EventItemMap` based on the event-schema order.

**Note:** All non-`event_name` parameter types must satisfy the `ConvertibleToEventItem` concept.

**Note:** Parameter types must match the actual `EventItem` types in the map.

### Template Parameters

- **EventIdType** -- Type of the event identifier (enum)
- **Callable** -- Type of the callable object (lambda, function, `std::function`, etc.) Event identifier is passed as `event_id`. Callable object is passed as `callback`, with first parameter as `event_name` and remaining parameters matching the event schema order.

**Returns** `EventRegistry::SignalConnection` Connection object (scoped, automatically unsubscribes on destruction)

```

template<typename T>
static inline FunctionResult to_function_result (std::expected<T, std::string> &&result)

```

Helper function to convert `std::expected` to `FunctionResult`.

**Template Parameters** **T** -- Return value type, can be void or any type convertible to `FunctionValue`

**Parameters** **result** -- [in] `std::expected` object

**Returns** `FunctionResult` Converted result

```

template<auto EventIdValue>

```

```

class EventMonitor

```

Event monitor for monitoring and waiting for specific events.

```

// Create a monitor for WiFi GeneralEventHappened
WifiHelper::EventMonitor<WifiHelper::EventId::GeneralEventHappened>␣
↪monitor;
monitor.start();
// ... trigger some action ...
bool got_event = monitor.wait_for(std::vector<service::EventItem>{
↪"Connected", false}, 5000);
monitor.stop();

```

**Template Parameters** **EventIdValue** -- The specific `EventId` enum value to monitor

**Public Functions**

inline bool **start** ()

Start monitoring for events.

**Returns** true if successfully started monitoring, false if already monitoring or failed

inline void **stop** ()

Stop monitoring for events.

inline void **clear** ()

Clear all received events.

inline bool **wait\_for** (const ReceivedItmes &expected\_items, uint32\_t timeout\_ms)

Wait for an event containing specific items.

**Parameters**

- **expected\_items** -- The expected items to find
- **timeout\_ms** -- Maximum time to wait in milliseconds

**Returns** true if matching items were received, false on timeout

inline bool **wait\_for\_any** (uint32\_t timeout\_ms)

Wait for any event to be received within a timeout period.

**Parameters** **timeout\_ms** -- Maximum time to wait in milliseconds

**Returns** true if any event was received, false on timeout

inline size\_t **get\_count** () const

Get the number of received events.

**Returns** Number of events received since *start()* or last *clear()*

inline bool **has** (const ReceivedItmes &expected\_items) const

Check if specific items have been received.

**Parameters** **expected\_items** -- The expected items to find

**Returns** true if matching items exists in received items

inline const std::vector<ReceivedItmes> &**get\_all** () const

Get all received events (unfiltered)

**Returns** Copy of all received events

inline std::optional<ReceivedItmes> **get\_last** () const

Get the last received event.

**Returns** std::optional containing the last received items, std::nullopt if no items were received

template<typename ...**Args**>

inline std::vector<std::tuple<*Args...*>> **get\_all** () const

Get received events filtered by item types and extracted as tuples.

```
// Get events where first item is string and second is bool
auto events = monitor.get_all<std::string, bool>();
for (const auto& [str_val, bool_val] : events) {
    // use str_val and bool_val
}
```

**Template Parameters** **Args** -- Expected types for each position in the event (bool, double, std::string, boost::json::object, boost::json::array, RawBuffer)

**Returns** Vector of tuples containing extracted values from matching events

template<typename ...**Args**>

inline std::optional<std::tuple<Args...>> **get\_last** () const

Get the last received event filtered by item types and extracted as tuple.

```
// Get last event where first item is string and second is bool
auto last_event = monitor.get_last<std::string, bool>();
if (last_event.has_value()) {
    const auto &[event_str, is_unexpected] = last_event.value();
    // Or use std::get<0>/std::get<1>
    const auto &event_str = std::get<0>(last_event.value());
    bool is_unexpected = std::get<1>(last_event.value());
}

// Get last event with single boost::json::array item
auto last_scan = monitor.get_last<boost::json::array>();
if (last_scan.has_value()) {
    const auto &ap_infos_array = std::get<0>(last_scan.value());
}
```

**Template Parameters** **Args** -- Expected types for each position in the event (bool, double, std::string, boost::json::object, boost::json::array, RawBuffer)

**Returns** std::optional containing tuple of extracted values if the last event matches, std::nullopt if no events or type mismatch

inline bool **is\_running** () const

Check if currently running.

**Returns** true if actively running

## Macros

**BROOKESIA\_SERVICE\_HELPER\_FUNC\_HANDLER\_0** (Derived, function\_id, func\_call)

Create a zero-parameter function handler based on Derived and FunctionId.

Example: `BROOKESIA_SERVICE_HELPER_FUNC_HANDLER_0(MyService, MyService::FunctionId::GetVolume, function_get_volume())`

### Parameters

- **Derived** -- The derived class type
- **function\_id** -- FunctionId enum value
- **func\_call** -- Actual function call (e.g., function\_get\_volume())

**BROOKESIA\_SERVICE\_HELPER\_FUNC\_HANDLER\_1** (Derived, function\_id, param\_type, func\_call)

Create a single-parameter function handler based on Derived and FunctionId.

Example: `BROOKESIA_SERVICE_HELPER_FUNC_HANDLER_1(MyService, MyService::FunctionId::PlayUrl, std::string, function_play_url(PARAM))`

### Parameters

- **Derived** -- The derived class type
- **function\_id** -- FunctionId enum value
- **param\_type** -- Parameter C++ type (e.g., std::string, double)
- **func\_call** -- Function call, use PARAM as parameter placeholder

**BROOKESIA\_SERVICE\_HELPER\_FUNC\_HANDLER\_2** (Derived, function\_id, param1\_type, param2\_type, func\_call)

Create a two-parameter function handler based on Derived and FunctionId.

Example: `BROOKESIA_SERVICE_HELPER_FUNC_HANDLER_2(MyService, MyService::FunctionId::Add, double, double, function_add(PARAM1, PARAM2))`

#### Parameters

- **Derived** -- The derived class type
- **function\_id** -- FunctionId enum value
- **param1\_type** -- First parameter C++ type
- **param2\_type** -- Second parameter C++ type
- **func\_call** -- Function call, use PARAM1, PARAM2 as parameter placeholders

`BROOKESIA_SERVICE_HELPER_FUNC_HANDLER_3` (Derived, function\_id, param1\_type, param2\_type, param3\_type, func\_call)

Create a three-parameter function handler based on Derived and FunctionId.

Example: `BROOKESIA_SERVICE_HELPER_FUNC_HANDLER_3(MyService, MyService::FunctionId::SetConfig, std::string, int, bool, function_set_config(PARAM1, PARAM2, PARAM3))`

#### Parameters

- **Derived** -- The derived class type
- **function\_id** -- FunctionId enum value
- **param1\_type** -- First parameter C++ type
- **param2\_type** -- Second parameter C++ type
- **param3\_type** -- Third parameter C++ type
- **func\_call** -- Function call, use PARAM1, PARAM2, PARAM3 as parameter placeholders

### Audio Helper

- Public header: `#include "brookesia/service_helper/audio.hpp"`

**Overview** This page documents the Audio helper Doxygen API: types, enums, methods, and macros.

### API Reference

#### Header File

- `service/brookesia_service_helper/include/brookesia/service_helper/audio.hpp`

#### Classes

class **Audio** : public `esp_brookesia::service::helper::Base<Audio>`

Helper schema definitions for the audio service.

#### Public Types

enum class **PlayControlAction** : `uint8_t`

Playback control actions.

*Values:*

enumerator **Pause**

enumerator **Resume**

enumerator **Stop**

enum class **PlayState** : uint8\_t

Playback states.

*Values:*

enumerator **Idle**

enumerator **Playing**

enumerator **Paused**

enum class **CodecFormat** : uint8\_t

Codec related configurations.

*Values:*

enumerator **PCM**

enumerator **OPUS**

enumerator **G711A**

enumerator **Max**

enum class **FunctionId** : uint8\_t

*Audio* service function identifiers.

*Values:*

enumerator **SetPlaybackConfig**

enumerator **SetEncoderStaticConfig**

enumerator **SetDecoderStaticConfig**

enumerator **SetAFE\_Config**

enumerator **GetAFE\_WakeWords**

enumerator **PauseAFE\_WakeupEnd**

enumerator **ResumeAFE\_WakeupEnd**

enumerator **PlayUrl**

enumerator **PlayUrls**

enumerator **PlayControl**

enumerator **SetVolume**

enumerator **GetVolume**

enumerator **SetMute**

enumerator **StartEncoder**

enumerator **StopEncoder**

enumerator **PauseEncoder**

enumerator **ResumeEncoder**

enumerator **StartDecoder**

enumerator **StopDecoder**

enumerator **FeedDecoderData**

enumerator **ResetData**

enumerator **Max**

enum class **EventId** : uint8\_t

*Audio* service event identifiers.

*Values:*

enumerator **PlayStateChanged**

enumerator **AFE\_EventHappened**

enumerator **EncoderDataReady**

enumerator **RecorderDataReady**

enumerator **Max**

enum class **FunctionSetPlaybackConfigParam** : uint8\_t

Parameter keys for *FunctionId::SetPlaybackConfig*.

*Values:*

enumerator **Config**

enum class **FunctionSetEncoderStaticConfigParam** : uint8\_t  
Parameter keys for *FunctionId::SetEncoderStaticConfig*.  
*Values:*  
  
enumerator **Config**

enum class **FunctionSetDecoderStaticConfigParam** : uint8\_t  
Parameter keys for *FunctionId::SetDecoderStaticConfig*.  
*Values:*  
  
enumerator **Config**

enum class **FunctionSetAFE\_ConfigParam** : uint8\_t  
Parameter keys for *FunctionId::SetAFE\_Config*.  
*Values:*  
  
enumerator **Config**

enum class **FunctionPlayUrlParam** : uint8\_t  
Parameter keys for *FunctionId::PlayUrl*.  
*Values:*  
  
enumerator **Url**  
  
enumerator **Config**

enum class **FunctionPlayUrlsParam** : uint8\_t  
Parameter keys for *FunctionId::PlayUrls*.  
*Values:*  
  
enumerator **Urls**  
  
enumerator **Config**

enum class **FunctionPlayControlParam** : uint8\_t  
Parameter keys for *FunctionId::PlayControl*.  
*Values:*  
  
enumerator **Action**

enum class **FunctionSetVolumeParam** : uint8\_t  
Parameter keys for *FunctionId::SetVolume*.  
*Values:*  
  
enumerator **Volume**

enum class **FunctionSetMuteParam** : uint8\_t

Parameter keys for *FunctionId::SetMute*.

*Values:*

enumerator **Enable**

enum class **FunctionStartEncoderParam** : uint8\_t

Parameter keys for *FunctionId::StartEncoder*.

*Values:*

enumerator **Config**

enum class **FunctionStartDecoderParam** : uint8\_t

Parameter keys for *FunctionId::StartDecoder*.

*Values:*

enumerator **Config**

enum class **FunctionFeedDecoderDataParam** : uint8\_t

Parameter keys for *FunctionId::FeedDecoderData*.

*Values:*

enumerator **Data**

enum class **EventPlayStateChangedParam** : uint8\_t

Item keys for *EventId::PlayStateChanged*.

*Values:*

enumerator **State**

enum class **EventAFE\_EventHappenedParam** : uint8\_t

Item keys for *EventId::AFE\_EventHappened*.

*Values:*

enumerator **Event**

enum class **EventEncoderDataReadyParam** : uint8\_t

Item keys for *EventId::EncoderDataReady*.

*Values:*

enumerator **Data**

enum class **EventRecorderDataReadyParam** : uint8\_t

Item keys for *EventId::RecorderDataReady*.

*Values:*

enumerator **Data**

### Public Static Functions

static inline constexpr std::string\_view **get\_name** ()

Service name used by *ServiceManager*.

**Returns** std::string\_view Stable service name.

static inline std::span<const FunctionSchema> **get\_function\_schemas** ()

Get function schemas exported by audio service.

**Returns** std::span<const FunctionSchema> Static function schema span.

static inline std::span<const EventSchema> **get\_event\_schemas** ()

Get event schemas exported by audio service.

**Returns** std::span<const EventSchema> Static event schema span.

struct **AFE\_Config**

struct **AFE\_VAD\_Config**

AFE related configurations.

### Public Members

uint8\_t **mode** = 4

VAD mode

uint32\_t **min\_speech\_ms** = 64

Minimum speech duration

uint32\_t **min\_noise\_ms** = 1000

Minimum noise duration

struct **AFE\_WakeNetConfig**

### Public Members

std::string **model\_partition\_label** = "model"

Wake model partition label

std::string **mn\_language** = "cn"

Wake model language

uint32\_t **start\_timeout\_ms** = 30000

Timeout before wake start

uint32\_t **end\_timeout\_ms** = 10000

Timeout before wake end

struct **CodecGeneralConfig**

### Public Members

uint8\_t **channels**

Number of audio channels (1-4)

uint8\_t **sample\_bits**

Bit depth in bits (e.g., 8, 16, 24, 32)

uint32\_t **sample\_rate**

Sample rate in Hz (e.g., 8000, 16000, 24000, 32000, 44100, 48000)

uint8\_t **frame\_duration**

Frame duration in milliseconds

struct **DecoderDynamicConfig**

### Public Members

*CodecFormat* **type**

Decoder codec type

*CodecGeneralConfig* **general**

Decoder common codec settings

struct **DecoderStaticConfig**

Decoder related configurations.

struct **EncoderDynamicConfig**

### Public Members

*CodecFormat* **type**

Encoder codec type

*CodecGeneralConfig* **general**

Encoder common codec settings

std::variant<std::monostate, *EncoderExtraConfigOpus*> **extra** = std::monostate{}

Optional codec-specific settings

uint32\_t **fetch\_interval\_ms** = 10

Encoder fetch interval in milliseconds

uint32\_t **fetch\_data\_size** = 4096

Encoder fetch size in bytes

struct **EncoderExtraConfigOpus**

### Public Members

bool **enable\_vbr**

Enable Variable Bit Rate (VBR)

uint32\_t **bitrate**

Bitrate in bps

struct **EncoderStaticConfig**

Encoder related configurations.

struct **MixerGainConfig**

Mixer related configurations.

### Public Members

float **initial\_gain**

Initial gain value

float **target\_gain**

Target gain value

int **transition\_time**

Transition duration in milliseconds

struct **PlaybackConfig**

Playback related configurations.

struct **PlayUrlConfig**

Runtime options for `PlayUrl` and `PlayUrls`.

### Public Members

bool **interrupt** = true

Whether current playback can be interrupted

uint32\_t **delay\_ms** = 0

Delay before playback starts

uint32\_t **loop\_count** = 0

Number of extra loops

```
uint32_t loop_interval_ms = 0
```

Interval between loops

```
uint32_t timeout_ms = 0
```

Timeout for finishing playback

## Device Helper

- Public header: `#include "brookesia/service_helper/device.hpp"`

**Overview** This page documents the Device helper Doxygen API: types, enums, methods, and macros. The Device helper is the standard application-layer entry point for device control and status-query interfaces backed by HAL capabilities.

## API Reference

### Header File

- [service/brookesia\\_service\\_helper/include/brookesia/service\\_helper/device.hpp](#)

### Classes

```
class Device : public esp_brookesia::service::helper::Base<Device>
```

Helper schema definitions for the device control service.

### Public Types

```
using Capabilities = hal::Capabilities
```

Runtime capability snapshot for the device control service.

```
using BoardInfo = hal::BoardInfoIface::Info
```

Static board information.

```
using StorageFsInfo = hal::StorageFsIface::Info
```

Storage file system information.

```
using StorageFsCapacity = hal::StorageFsIface::Capacity
```

Dynamic storage file system capacity snapshot.

```
using PowerBatteryInfo = hal::PowerBatteryIface::Info
```

Static battery information.

```
using PowerBatteryState = hal::PowerBatteryIface::State
```

Runtime battery state snapshot.

```
using PowerBatteryChargeConfig = hal::PowerBatteryIface::ChargeConfig
```

Battery charge configuration.

## Emote Helper

- Public header: `#include "brookesia/service_helper/expression/emote.hpp"`

**Overview** This page documents the Expression/Emote helper Doxygen API: types, enums, methods, and macros.

## API Reference

### Header File

- `service/brookesia_service_helper/include/brookesia/service_helper/expression/emote.hpp`

### Classes

class **ExpressionEmote** : public esp\_brookesia::service::helper::Base<ExpressionEmote>

Helper schema definitions for the emote-expression service.

### Public Types

enum class **EventMessageType**

Supported message categories rendered by the expression service.

*Values:*

enumerator **Idle**

enumerator **Speak**

enumerator **Listen**

enumerator **System**

enumerator **User**

enumerator **Battery**

enumerator **Max**

enum class **AssetSourceType**

Supported asset-source backends used when loading expression assets.

*Values:*

enumerator **Path**

enumerator **PartitionLabel**

enumerator **Max**

**struct AssetSource**

Description of one asset source used by the expression service.

**Public Members**

std::string **source**

Source identifier such as a path or partition label.

*AssetSourceType* **type**

How `source` should be interpreted.

bool **flag\_enable\_mmap** = false

Whether mmap-backed loading should be enabled.

**struct Config**

Runtime display and task configuration for the expression service.

**Public Members**

uint32\_t **h\_res** = 0

Horizontal resolution in pixels.

uint32\_t **v\_res** = 0

Vertical resolution in pixels.

size\_t **buf\_pixels** = 0

Display buffer size in pixels.

uint32\_t **fps** = 0

Target render frame rate.

int **task\_priority** = 0

Render task priority.

int **task\_stack** = 0

Render task stack size in bytes.

int **task\_affinity** = 0

Core affinity for the render task.

bool **task\_stack\_in\_ext** = false

Whether the task stack should live in external memory.

bool **flag\_swap\_color\_bytes** = false

Whether output color bytes must be swapped.

bool **flag\_double\_buffer** = false  
Whether double buffering is enabled.

bool **flag\_buff\_dma** = false  
Whether display buffers must be DMA-capable.

bool **flag\_buff\_spiram** = false  
Whether display buffers may be allocated in SPIRAM.

struct **FlushReadyEventParam**  
Parameters delivered with the flush-ready event.

### Public Members

int **x\_start** = 0  
Left edge of the dirty region.

int **y\_start** = 0  
Top edge of the dirty region.

int **x\_end** = 0  
Right edge of the dirty region.

int **y\_end** = 0  
Bottom edge of the dirty region.

const void \***data** = nullptr  
Pixel buffer for the dirty region.

### NVS Helper

- Public header: `#include "brookesia/service_helper/nvs.hpp"`

**Overview** This page documents the NVS helper Doxygen API: types, enums, methods, and macros.

### API Reference

#### Header File

- [service/brookesia\\_service\\_helper/include/brookesia/service\\_helper/nvs.hpp](#)

#### Classes

class **NVS** : public esp\_brookesia::service::helper::Base<NVS>  
Helper schema definitions for the *NVS* service.

## Public Types

enum class **ValueType**

They are used as parameter and return types for functions and events. Users can access or modify these types via serialization and deserialization.

*Values:*

enumerator **Bool**

enumerator **Int**

enumerator **String**

enumerator **Max**

enum class **FunctionId**

*NVS* service function identifiers.

*Values:*

enumerator **List**

enumerator **Set**

enumerator **Get**

enumerator **Erase**

enumerator **Max**

enum class **EventId**

*NVS* service event identifiers.

*Values:*

enumerator **Max**

enum class **FunctionListParam**

Parameter keys for *FunctionId::List*.

*Values:*

enumerator **Nspace**

enum class **FunctionSetParam**

Parameter keys for *FunctionId::Set*.

*Values:*

enumerator **Nspace**

enumerator **KeyValuePairs**

enum class **FunctionGetParam**

Parameter keys for *FunctionId::Get*.

*Values:*

enumerator **Nspace**

enumerator **Keys**

enum class **FunctionEraseParam**

Parameter keys for *FunctionId::Erase*.

*Values:*

enumerator **Nspace**

enumerator **Keys**

using **Value** = std::variant<bool, int32\_t, std::string>

Value type stored in *NVS* entries.

using **KeyValueMap** = std::map<std::string, *Value*>

Key-value mapping payload for batch set/get operations.

### Public Static Functions

static inline constexpr std::string\_view **get\_name** ()

Service name used by *ServiceManager*.

**Returns** std::string\_view Stable service name.

static inline std::span<const FunctionSchema> **get\_function\_schemas** ()

Get function schemas exported by *NVS* service.

**Returns** std::span<const FunctionSchema> Static function schema span.

static inline std::span<const EventSchema> **get\_event\_schemas** ()

Get event schemas exported by *NVS* service.

**Returns** std::span<const EventSchema> Empty span because *NVS* has no events.

template<typename **T**>

static inline std::expected<void, std::string> **save\_key\_value** (const std::string &nospace, const std::string &key, const *T* &value, uint32\_t timeout\_ms = *DEFAULT\_TIMEOUT\_MS*)

Save key-value pairs to the *NVS* namespace.

#### Direct Storage (No Serialization):

- `bool`: Stored directly as JSON boolean value (true/false)
- `int32_t`: Stored directly as JSON number (int64\_t in JSON)
- Integer types with size <= 32 bits (`int8_t`, `uint8_t`, `int16_t`, `uint16_t`, `char`, `short`, etc.): Converted to `int32_t` and stored as JSON number for optimal performance

**Serialized Storage:**

- Integer types with size > 32 bits (int64\_t, uint64\_t, long long, etc.): Serialized to JSON string using BROOKESIA\_DESCRIBE\_JSON\_SERIALIZE
- Floating point types (float, double): Serialized to JSON string using BROOKESIA\_DESCRIBE\_JSON\_SERIALIZE
- String types (std::string, const char\*): Serialized to JSON string using BROOKESIA\_DESCRIBE\_JSON\_SERIALIZE
- Complex types (std::vector, std::map, custom structs, etc.): Serialized to JSON string using BROOKESIA\_DESCRIBE\_JSON\_SERIALIZE

---

**Note:** The storage method depends on the type T:

---

**Template Parameters** **T** -- The type of the value to save  
**Parameters**

- **nspace** -- The namespace of the key-value pairs to save
- **key** -- The key of the key-value pair to save
- **value** -- The value of the key-value pair to save
- **timeout\_ms** -- The timeout in milliseconds

**Returns** std::expected<void, std::string> The result of the operation

```
template<typename T>
static inline std::expected<T, std::string> get_key_value (const std::string &nspace, const std::string
&key, uint32_t timeout_ms =
DEFAULT_TIMEOUT_MS)
```

Get key-value pair from the *NVS* namespace.

**Direct Retrieval (No Deserialization):**

- `bool`: Retrieved directly from JSON boolean value
- `int32_t`: Retrieved directly from JSON number
- Integer types with size <= 32 bits (int8\_t, uint8\_t, int16\_t, uint16\_t, char, short, etc.): Retrieved directly from JSON number and converted to the target integer type

**Deserialized Retrieval:**

- Integer types with size > 32 bits (int64\_t, uint64\_t, long long, etc.): Retrieved directly from JSON string and deserialized to the target integer type
- Floating point types (float, double): Retrieved directly from JSON string and deserialized to the target floating point type
- String types (std::string): Retrieved directly from JSON string and deserialized to the target string type
- Complex types (std::vector, std::map, custom structs, etc.): Retrieved directly from JSON string and deserialized to the target complex type

---

**Note:** The retrieval method depends on the type T and matches the storage method used in *save\_key\_value()*:

---

**Template Parameters** **T** -- The type of the value to retrieve  
**Parameters**

- **nspace** -- The namespace of the key-value pair to retrieve
- **key** -- The key of the key-value pair to retrieve
- **timeout\_ms** -- The timeout in milliseconds

**Returns** std::expected<T, std::string> The retrieved value or error message

```
static inline std::expected<void, std::string> erase_keys (const std::string &nspace, const
std::vector<std::string> &keys = {}, uint32_t
timeout_ms = DEFAULT_TIMEOUT_MS)
```

Erase key-value pairs from the *NVS* namespace.

#### Parameters

- **namespace** -- The namespace of the key-value pairs to erase
- **keys** -- The keys of the key-value pairs to erase, optional. If not provided or empty, all key-value pairs in the namespace will be erased
- **timeout\_ms** -- The timeout in milliseconds

**Returns** `std::expected<void, std::string>` The result of the operation

#### Public Static Attributes

```
static constexpr uint32_t DEFAULT_TIMEOUT_MS =  
BROOKESIA_SERVICE_MANAGER_DEFAULT_CALL_FUNCTION_TIMEOUT_MS
```

Default timeout for synchronous *NVS* helper calls.

```
struct EntryInfo
```

Metadata for one entry in an *NVS* namespace.

#### Public Members

```
std::string namespace  
    Namespace name
```

```
std::string key  
    Entry key
```

```
ValueType type  
    Entry value type
```

#### SNTP Helper

- Public header: `#include "brookesia/service_helper/sntp.hpp"`

**Overview** This page documents the SNTP helper Doxygen API: types, enums, methods, and macros.

#### API Reference

##### Header File

- `service/brookesia_service_helper/include/brookesia/service_helper/sntp.hpp`

##### Classes

```
class SNTP : public esp_brookesia::service::helper::Base<SNTP>
```

Helper schema definitions for the *SNTP* service.

## Public Types

enum class **FunctionId**

*SNTP* service function identifiers.

*Values:*

enumerator **SetServers**

enumerator **SetTimezone**

enumerator **Start**

enumerator **Stop**

enumerator **GetServers**

enumerator **GetTimezone**

enumerator **IsTimeSynced**

enumerator **ResetData**

enumerator **Max**

enum class **EventId**

*SNTP* service event identifiers.

*Values:*

enumerator **Max**

enum class **FunctionSetServersParam**

Parameter keys for *FunctionId::SetServers*.

*Values:*

enumerator **Servers**

enum class **FunctionSetTimezoneParam**

Parameter keys for *FunctionId::SetTimezone*.

*Values:*

enumerator **Timezone**

## Public Static Functions

```
static inline constexpr std::string_view get_name ()
```

Name of the *SNTP* service.

**Returns** std::string\_view Stable service name.

```
static inline std::span<const FunctionSchema> get_function_schemas ()
```

Get the function schemas exported by the *SNTP* service.

**Returns** std::span<const FunctionSchema> Static schema span.

```
static inline std::span<const EventSchema> get_event_schemas ()
```

Get the event schemas exported by the *SNTP* service.

**Returns** std::span<const EventSchema> Empty span because *SNTP* exposes no events.

## Video Helper

- Public header: `#include "brookesia/service_helper/video.hpp"`

**Overview** This page documents the Video helper Doxygen API: types, enums, methods, and macros.

## API Reference

### Header File

- [service/brookesia\\_service\\_helper/include/brookesia/service\\_helper/video.hpp](#)

## Classes

class **Video**

Shared schema/type definitions for video encoder and decoder helper services.

## Public Types

```
enum class EncoderSinkFormat : uint8_t
```

Encoder related configurations.

*Values:*

enumerator **H264**

enumerator **MJPEG**

enumerator **RGB565**

enumerator **RGB888**

enumerator **BGR888**

enumerator **YUV420**

enumerator **YUV422**

enumerator **O\_UYY\_E\_VYY**

enumerator **Max**

enum class **DecoderSourceFormat** : uint8\_t

Decoder related configurations.

*Values:*

enumerator **H264**

enumerator **MJPEG**

enumerator **Max**

enum class **EncoderFunctionId** : uint8\_t

*Video* encoder function identifiers.

*Values:*

enumerator **Open**

enumerator **Close**

enumerator **Start**

enumerator **Stop**

enumerator **FetchFrame**

enumerator **Max**

enum class **EncoderEventId** : uint8\_t

*Video* encoder event identifiers.

*Values:*

enumerator **StreamSinkFrameReady**

enumerator **FetchSinkFrameReady**

enumerator **Max**

enum class **DecoderFunctionId** : uint8\_t

*Video* decoder function identifiers.

*Values:*

enumerator **Open**

enumerator **Close**

enumerator **Start**

enumerator **Stop**

enumerator **FeedFrame**

enumerator **Max**

enum class **DecoderEventId** : uint8\_t

*Video* decoder event identifiers.

*Values:*

enumerator **SinkFrameReady**

enumerator **Max**

enum class **EncoderFunctionOpenParam** : uint8\_t

Parameter keys for *EncoderFunctionId::Open*.

*Values:*

enumerator **Config**

enum class **EncoderFunctionFetchFrameParam** : uint8\_t

Parameter keys for *EncoderFunctionId::FetchFrame*.

*Values:*

enumerator **SinkIndex**

enum class **DecoderFunctionOpenParam** : uint8\_t

Parameter keys for *DecoderFunctionId::Open*.

*Values:*

enumerator **Config**

enum class **DecoderFunctionFeedFrameParam** : uint8\_t

Parameter keys for *DecoderFunctionId::FeedFrame*.

*Values:*

enumerator **Frame**

enum class **EncoderEventStreamSinkFrameReadyParam** : uint8\_t

Item keys for *EncoderEventId::StreamSinkFrameReady*.

*Values:*

enumerator **SinkIndex**

enumerator **SinkInfo**

enumerator **Frame**

enum class **EncoderEventFetchSinkFrameReadyParam** : uint8\_t

Item keys for *EncoderEventId::FetchSinkFrameReady*.

*Values:*

enumerator **SinkIndex**

enumerator **SinkInfo**

enumerator **Frame**

enum class **DecoderEventSinkFrameReadyParam** : uint8\_t

Item keys for *DecoderEventId::SinkFrameReady*.

*Values:*

enumerator **Width**

enumerator **Height**

enumerator **Frame**

### Public Static Functions

static inline std::span<const FunctionSchema> **get\_encoder\_function\_schemas** ()

Get all encoder function schemas.

**Returns** std::span<const FunctionSchema> Static schema span.

static inline std::span<const EventSchema> **get\_encoder\_event\_schemas** ()

Get all encoder event schemas.

**Returns** std::span<const EventSchema> Static schema span.

static inline std::span<const FunctionSchema> **get\_decoder\_function\_schemas** ()

Get all decoder function schemas.

**Returns** std::span<const FunctionSchema> Static schema span.

static inline std::span<const EventSchema> **get\_decoder\_event\_schemas** ()

Get all decoder event schemas.

**Returns** std::span<const EventSchema> Static schema span.

### Public Static Attributes

static constexpr std::string\_view **ENCODER\_NAME\_PREFIX** = "VideoEncoder"

Prefix used to build encoder helper service names.

static constexpr std::string\_view **DECODER\_NAME\_PREFIX** = "VideoDecoder"

Prefix used to build decoder helper service names.

struct **DecoderConfig**

Decoder open configuration.

### Public Members

uint16\_t **width**

Decode frame width

uint16\_t **height**

Decode frame height

*DecoderSourceFormat* **source\_format**

Decoder input format

DecoderSinkFormat **sink\_format**

Decoder output format

bool **enable\_stream\_mode**

Whether decoder works in stream mode

bool **enable\_hw\_acceleration**

Whether hardware acceleration is enabled

struct **EncoderConfig**

Encoder open configuration.

### Public Members

std::vector<*EncoderSinkInfo*> **sinks**

Output sink list

bool **enable\_stream\_mode**

Whether encoder works in stream push mode

struct **EncoderSinkInfo**

One encoder sink stream description.

### Public Members

*EncoderSinkFormat* **format**

Output sink format

uint16\_t **width**

Output width

uint16\_t **height**

Output height

uint8\_t **fps**

Output frames per second

template<int **Id**>

class **VideoEncoder** : public esp\_brookesia::service::helper::Base<VideoEncoder<Id>>

### Public Types

using **FunctionId** = Video::EncoderFunctionId

Re-exported function id enum for encoder service instance.

using **EventId** = Video::EncoderEventId

Re-exported event id enum for encoder service instance.

### Public Static Functions

static inline std::string\_view **get\_name** ()

Get service name of this encoder instance.

**Returns** std::string\_view Service name in format *VideoEncoder*<Id>.

static inline std::span<const FunctionSchema> **get\_function\_schemas** ()

Get function schemas for this encoder instance.

**Returns** std::span<const FunctionSchema> Static schema span.

static inline std::span<const EventSchema> **get\_event\_schemas** ()

Get event schemas for this encoder instance.

**Returns** std::span<const EventSchema> Static schema span.

template<int **Id**>

class **VideoDecoder** : public esp\_brookesia::service::helper::Base<VideoDecoder<Id>>

### Public Types

using **FunctionId** = Video::DecoderFunctionId

Re-exported function id enum for decoder service instance.

using **EventId** = Video::DecoderEventId

Re-exported event id enum for decoder service instance.

### Public Static Functions

static inline std::string\_view **get\_name** ()

Get service name of this decoder instance.

**Returns** std::string\_view Service name in format *VideoDecoder*<Id>.

static inline std::span<const FunctionSchema> **get\_function\_schemas** ()

Get function schemas for this decoder instance.

**Returns** std::span<const FunctionSchema> Static schema span.

static inline std::span<const EventSchema> **get\_event\_schemas** ()

Get event schemas for this decoder instance.

**Returns** std::span<const EventSchema> Static schema span.

### Wi-Fi Helper

- Public header: `#include "brookesia/service_helper/wifi.hpp"`

**Overview** This page documents the Wi-Fi helper Doxygen API: types, enums, methods, and macros.

### API Reference

#### Header File

- [service/brookesia\\_service\\_helper/include/brookesia/service\\_helper/wifi.hpp](#)

#### Classes

class **Wifi** : public esp\_brookesia::service::helper::Base<Wifi>

Helper schema definitions for the Wi-Fi service.

#### Public Types

enum class **GeneralAction**

General Wi-Fi control actions.

*Values:*

enumerator **Init**

Initialize Wi-Fi resources.

enumerator **Deinit**

Deinitialize Wi-Fi resources.

enumerator **Start**

Start Wi-Fi subsystem.

enumerator **Stop**

Stop Wi-Fi subsystem.

enumerator **Connect**

Connect to configured AP.

enumerator **Disconnect**

Disconnect from current AP.

enumerator **Max**

Sentinel value.

enum class **GeneralEvent**

General Wi-Fi lifecycle events.

*Values:*

enumerator **Deinited**

Wi-Fi has been deinitialized.

enumerator **Inited**

Wi-Fi has been initialized.

enumerator **Stopped**

Wi-Fi has been stopped.

enumerator **Started**

Wi-Fi has started.

enumerator **Disconnected**

Wi-Fi disconnected from AP.

enumerator **Connected**

Wi-Fi connected to AP.

enumerator **Max**

Sentinel value.

enum class **GeneralState**

General state for WiFi state machine.

Stable states: Idle, Inited, Started, Connected  
Transient states: Initing, Deiniting, Starting, Stopping,  
Connecting, Disconnecting

*Values:*

enumerator **Idle**

Stable: Wi-Fi is not initialized.

enumerator **Initing**

Transient: Wi-Fi is initializing.

enumerator **Inited**

Stable: Wi-Fi initialized but not started.

enumerator **Deiniting**

Transient: Wi-Fi is deinitializing.

enumerator **Starting**

Transient: Wi-Fi is starting.

enumerator **Started**

Stable: Wi-Fi started but not connected.

enumerator **Stopping**

Transient: Wi-Fi is stopping.

enumerator **Connecting**

Transient: Wi-Fi is connecting.

enumerator **Connected**

Stable: Wi-Fi is connected.

enumerator **Disconnecting**

Transient: Wi-Fi is disconnecting.

enumerator **Max**

Sentinel value.

enum class **ScanApSignalLevel**

Signal-strength bucket for a scanned AP.

*Values:*

enumerator **LEVEL\_0**

RSSI  $\leq$  -81 dBm.

enumerator **LEVEL\_1**

RSSI in [-80, -71] dBm.

enumerator **LEVEL\_2**

RSSI in [-70, -61] dBm.

enumerator **LEVEL\_3**

RSSI in [-60, -51] dBm.

enumerator **LEVEL\_4**

RSSI  $\geq$  -50 dBm.

enum class **SoftApEvent**

SoftAP lifecycle events.

*Values:*

enumerator **Started**

SoftAP started.

enumerator **Stopped**

SoftAP stopped.

enumerator **Max**

Sentinel value.

enum class **FunctionId**

Wi-Fi service function identifiers.

*Values:*

enumerator **TriggerGeneralAction**

enumerator **GetGeneralState**

enumerator **SetConnectAp**

enumerator **GetConnectAp**

enumerator **GetConnectedAps**

enumerator **SetScanParams**

enumerator **TriggerScanStart**

enumerator **TriggerScanStop**

enumerator **SetSoftApParams**

enumerator **GetSoftApParams**

enumerator **TriggerSoftApStart**

enumerator **TriggerSoftApStop**

enumerator **TriggerSoftApProvisionStart**

enumerator **TriggerSoftApProvisionStop**

enumerator **ResetData**

enumerator **Max**

enum class **EventId**

Wi-Fi service event identifiers.

*Values:*

enumerator **GeneralActionTriggered**

enumerator **GeneralEventHappened**

enumerator **ScanStateChanged**

enumerator **ScanApInfosUpdated**

enumerator **SoftApEventHappened**

enumerator **Max**

enum class **FunctionTriggerGeneralActionParam**

Parameter keys for *FunctionId::TriggerGeneralAction*.

*Values:*

enumerator **Action**

enum class **FunctionSetConnectApParam**

Parameter keys for *FunctionId::SetConnectAp*.

*Values:*

enumerator **SSID**

enumerator **Password**

enum class **FunctionSetScanParamsParam**

Parameter keys for *FunctionId::SetScanParams*.

*Values:*

enumerator **Param**

enum class **FunctionSetSoftApParamsParam**

Parameter keys for *FunctionId::SetSoftApParams*.

*Values:*

enumerator **Param**

enum class **EventGeneralActionTriggeredParam**

Item keys for *EventId::GeneralActionTriggered*.

*Values:*

enumerator **Action**

enum class **EventGeneralEventHappenedParam**

Item keys for *EventId::GeneralEventHappened*.

*Values:*

enumerator **Event**

enumerator **IsUnexpected**

enum class **EventScanStateChangedParam**

Item keys for *EventId::ScanStateChanged*.

*Values:*

enumerator **IsRunning**

enum class **EventScanApInfosUpdatedParam**

Item keys for *EventId::ScanApInfosUpdated*.

*Values:*

enumerator **ApInfos**

enum class **EventSoftApEventHappenedParam**

Item keys for *EventId::SoftApEventHappened*.

*Values:*

enumerator **Event**

### **Public Static Functions**

static inline constexpr std::string\_view **get\_name** ()

Get helper contract name.

**Returns** Constant service name string.

static inline std::span<const FunctionSchema> **get\_function\_schemas** ()

Get all function schemas exposed by this helper.

**Returns** Read-only span of function schemas.

static inline std::span<const EventSchema> **get\_event\_schemas** ()

Get all event schemas exposed by this helper.

**Returns** Read-only span of event schemas.

struct **ConnectApInfo**

Target AP connection info.

### Public Members

std::string **ssid**

AP SSID.

std::string **password**

AP password.

bool **is\_connectable** = true

Whether this AP is considered connectable.

struct **ScanApInfo**

One scanned AP entry.

### Public Members

std::string **ssid**

AP SSID.

bool **is\_locked**

Whether AP authentication is required.

int **rss\_i**

Signal strength in dBm.

*ScanApSignalLevel* **signal\_level**

Bucketed signal level derived from RSSI.

uint8\_t **channel**

RF channel.

### Public Static Functions

static inline *ScanApSignalLevel* **get\_signal\_level** (int rssi)

Convert RSSI to a signal-level bucket.

**Parameters** *rssi* -- [in] RSSI value in dBm.

**Returns** Corresponding *ScanApSignalLevel*.

struct **ScanParams**

Parameters for periodic AP scanning.

### Public Members

size\_t **ap\_count** = 20

Maximum number of AP entries to keep.

uint32\_t **interval\_ms** = 10000

Scan interval in milliseconds.

```
uint32_t timeout_ms = 60000  
    Total scan timeout in milliseconds.
```

```
struct SoftApParams  
    SoftAP runtime configuration.
```

### Public Members

```
std::string ssid = ""  
    SoftAP SSID.
```

```
std::string password = ""  
    SoftAP password.
```

```
uint8_t max_connection = 4  
    Maximum station connections.
```

```
std::optional<uint8_t> channel = std::nullopt  
    Optional fixed channel.
```

## 4.2 General Services

### 4.2.1 NVS

- Component registry: [espressif/brookesia\\_service\\_nvs](#)
- Helper header: `#include "brookesia/service_helper/nvs.hpp"`
- Helper class: `esp_brookesia::service::helper::NVS`

#### Overview

*brookesia\_service\_nvs* provides NVS (non-volatile storage) for the ESP-Brookesia ecosystem:

- **Namespaces:** Key-value storage partitioned by namespace.
- **Types:** Boolean, 32-bit signed integer, and UTF-8 string.
- **Persistence:** Data survives power cycles in the NVS partition.
- **Thread safety:** Optional *TaskScheduler*-based async execution.
- **Queries:** List keys in a namespace or read specific keys.

#### Features

##### Namespaces

- **Default:** If omitted, namespace "storage" is used.
- **Multiple:** Create several namespaces for different data.
- **Isolation:** Namespaces are independent.

### Supported Types

- **Bool:** true / false.
- **Int:** 32-bit signed integer.
- **String:** UTF-8 string.

---

**Note:** Beyond these primitives, NVS Helper templates `esp_brookesia::service::helper::NVS::save_key_value()` and `esp_brookesia::service::helper::NVS::get_key_value()` can store and retrieve arbitrary types.

---

### Core Capabilities

- List key metadata in a namespace.
- Set multiple key-value pairs in batch.
- Read one key or all pairs in a namespace.
- Delete a key or clear a namespace.

### Standard Include / Helper Class

- Standard include: `#include "brookesia/service_helper/nvs.hpp"`
- Helper class: `esp_brookesia::service::helper::NVS`

### Service Interfaces

#### Functions

#### List

**Description** List key-value entries in an NVS namespace. Return type: JSON array<object>. Example: `[{"nspace":"storage","key":"key1","type":"String"}, {"nspace":"storage","key":"key2","type":"Int"}]`

#### Execution

- Requires scheduler: Required

#### Parameters

- Nspace
  - **Type:** String
  - **Required:** optional
  - **Default:** "storage"
  - **Description:** Namespace to list (optional). Uses the default namespace when omitted.

### Schema JSON

#### CLI Command

```
svc_call NVS List {"Nspace":null}
```

#### Set

**Description** Set key-value pairs in an NVS namespace.

### Execution

- Requires scheduler: Required

### Parameters

- Nspace
  - **Type:** String
  - **Required:** optional
  - **Default:** "storage"
  - **Description:** Namespace to write (optional). Uses the default namespace when omitted or empty.
- KeyValuePairs
  - **Type:** Object
  - **Required:** required
  - **Description:** Key-value pairs as a JSON object. Allowed value types: ["Bool", "Int", "String"]. Example: {"key1": "value1", "key2": 2, "key3": true}

### Schema JSON

#### CLI Command

```
svc_call NVS Set {"Nspace":null,"KeyValuePairs":null}
```

### Get

**Description** Get key-value pairs by keys from an NVS namespace. Return type: JSON object. Example: {"key1": "value1", "key2": 2, "key3": true}

### Execution

- Requires scheduler: Required

### Parameters

- Nspace
  - **Type:** String
  - **Required:** optional
  - **Default:** "storage"
  - **Description:** Namespace to read (optional). Uses the default namespace when omitted.
- Keys
  - **Type:** Array
  - **Required:** optional
  - **Default:** []
  - **Description:** Keys to read as JSON array<string> (optional). Returns all pairs when omitted. Example: ["key1", "key2", "key3"]

### Schema JSON

#### CLI Command

```
svc_call NVS Get {"Nspace":null,"Keys":null}
```

## Erase

**Description** Erase key-value pairs from an NVS namespace.

### Execution

- Requires scheduler: Required

### Parameters

- Nspace
  - **Type:** String
  - **Required:** optional
  - **Default:** "storage"
  - **Description:** Namespace to erase (optional). Uses the default namespace when omitted.
- Keys
  - **Type:** Array
  - **Required:** optional
  - **Default:** []
  - **Description:** Keys to erase as JSON array<string> (optional). Erases all pairs when omitted or empty. Example: ["key1","key2","key3"]

### Schema JSON

#### CLI Command

```
svc_call NVS Erase {"Nspace":null,"Keys":null}
```

**Events** This contract does not publish standard event schemas.

### Related Types and Configs

For structs, enums, helper methods, and the full Doxygen-rendered API surface, see [Raw helper API reference](#).

## 4.2.2 Sntp

- Component registry: [espressif/brookesia\\_service\\_sntp](#)
- Helper header: `#include "brookesia/service_helper/sntp.hpp"`
- Helper class: `esp_brookesia::service::helper::Sntp`

### Overview

*brookesia\_service\_sntp* provides Sntp (Simple Network Time Protocol) for the ESP-Brookesia ecosystem:

- **NTP servers:** Multiple servers; picks an available one.
- **Time zone:** System time zone with offset applied.
- **Auto sync:** Starts when the network is up.
- **Status:** Query sync state, server list, and zone.
- **Persistence:** Optional *brookesia\_service\_nvs* for servers and zone.

## Features

### NTP Servers

- Default: "pool.ntp.org".
- Multiple servers with automatic selection.
- Query the configured list.

### Time Zone

- Default: CST-8 (China, UTC+8).
- Standard zone strings (UTC, CST-8, EST-5, ...).
- Applied to system time when set.

### Core Operations

- **Set NTP servers:** Configure one or more NTP servers.
- **Set time zone:** Configure the system time zone.
- **Start / stop service:** Start or stop SNTP and time synchronization.
- **Get server list:** Read the configured NTP server list.
- **Get time zone:** Read the configured time zone.
- **Check sync status:** Check whether system time is synchronized with NTP.
- **Reset data:** Reset all configuration to defaults.

### Auto Management

- Load from NVS on start.
- Save after changes.
- Detect network and start sync when available.
- Monitor sync status and flags.

### Standard Include / Helper Class

- Standard include: #include \"brookesia/service\_helper/sntp.hpp\"
- Helper class: esp\_brookesia::service::helper::SNTP

### Service Interfaces

### Functions

### SetServers

**Description** Set NTP servers.

### Execution

- Requires scheduler: Not required

### Parameters

- Servers
  - **Type:** Array
  - **Required:** required
  - **Description:** NTP servers as JSON array<string>. Example: ["pool.ntp.org","cn.pool.ntp.org"]

### Schema JSON

#### CLI Command

```
svc_call SNTP SetServers {"Servers":null}
```

#### SetTimezone

**Description** Set timezone.

#### Execution

- Requires scheduler: Not required

#### Parameters

- Timezone
  - **Type:** String
  - **Required:** required
  - **Description:** Timezone string.

### Schema JSON

#### CLI Command

```
svc_call SNTP SetTimezone {"Timezone":null}
```

#### Start

**Description** Start SNTP service.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call SNTP Start
```

#### Stop

**Description** Stop SNTP service.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call SNTP Stop
```

### GetServers

**Description** Get NTP servers. Return type: JSON array<string>. Example: ["pool.ntp.org","cn.pool.ntp.org"]

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call SNTP GetServers
```

### GetTimezone

**Description** Get timezone. Return type: string. Example: "CST-8"

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call SNTP GetTimezone
```

### IsTimeSynced

**Description** Check whether time is synced. Return type: boolean. Example: true

**Execution**

- Requires scheduler: Required

**Parameters**

- No parameters.

**Schema JSON**

**CLI Command**

```
svc_call SNTP IsTimeSynced
```

**ResetData**

**Description** Reset NTP servers, timezone, and sync status.

**Execution**

- Requires scheduler: Required

**Parameters**

- No parameters.

**Schema JSON**

**CLI Command**

```
svc_call SNTP ResetData
```

**Events** This contract does not publish standard event schemas.

**Related Types and Configs**

For structs, enums, helper methods, and the full Doxygen-rendered API surface, see [Raw helper API reference](#).

### 4.2.3 Wi-Fi

- Component registry: [espressif/brookesia\\_service\\_wifi](#)
- Helper header: #include "brookesia/service\_helper/wifi.hpp"
- Helper class: `esp_brookesia::service::helper::Wifi`

## Overview

*brookesia\_service\_wifi* manages Wi-Fi connectivity:

- **State machine:** Init, start, connect, and related lifecycle states.
- **Auto reconnect:** Reconnect to known APs after drops.
- **Scanning:** Periodic scans and discovery of connectable APs.
- **Connection management:** Target AP and connected-AP lists with history.
- **Events:** Rich notifications for state changes.
- **Persistence:** Optional *brookesia\_service\_nvs* for credentials and parameters.

## Features

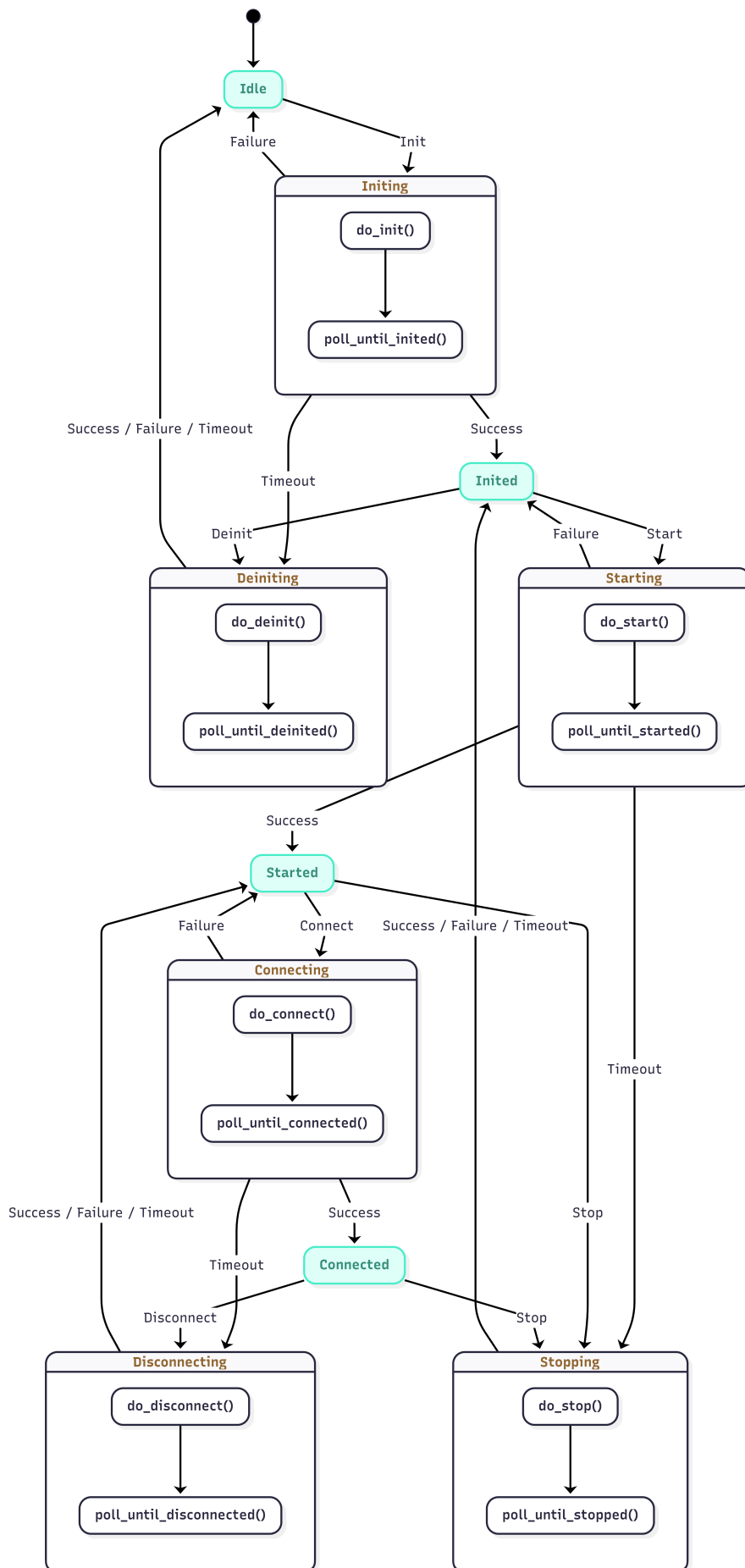
**State Machine** The Wi-Fi service manages lifecycle states in a state machine for safe, consistent transitions. The state machine has four core states:

State	Description
Idle	Wi-Fi not initialized (initial state).
Inited	Initialized but not started; parameters can be configured.
Started	Started; scanning or waiting to connect.
Connected	Associated with an AP and ready for traffic.

Transitions use **actions**:

- **Forward:** Idle → Inited (Init) → Started (Start) → Connected (Connect)
- **Disconnect:** Connected → Started (Disconnect)
- **Stop:** Started / Connected → Inited (Stop)
- **Deinit:** Inited → Idle (Deinit)

State transition diagram:



### Auto Reconnect

- Connect to historical APs after start.
- Reconnect after unexpected disconnects.
- Auto-connect when a target or historical AP appears during scan.

### Wi-Fi Scanning

- Configurable interval and timeout.
- Events with SSID, signal level, security, etc.

### SoftAP

- SSID, password, max clients, channel.
- Optional automatic best-channel selection.
- SoftAP provisioning mode.

### Standard Include / Helper Class

- Standard include: `#include \"brookesia/service_helper/wifi.hpp\"`
- Helper class: `esp_brookesia::service::helper::Wifi`

### Service Interfaces

#### Functions

#### TriggerGeneralAction

**Description** Trigger a Wi-Fi general action.

#### Execution

- Requires scheduler: Required

#### Parameters

- Action
  - **Type:** String
  - **Required:** required
  - **Description:** General action. Allowed values: [Init, Deinit, Start, Stop, Connect, Disconnect]

#### Schema JSON

#### CLI Command

```
svc_call Wifi TriggerGeneralAction {"Action":null}
```

#### GetGeneralState

**Description** Get current general state. Return type: string. Allowed values: [Idle, Initing, Inited, Deiniting, Starting, Started, Stopping, Connecting, Connected, Disconnecting]. Example: "Connected"

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Wifi GetGeneralState
```

### SetConnectAp

**Description** Set target AP credentials.

### Execution

- Requires scheduler: Required

### Parameters

- SSID
  - **Type:** String
  - **Required:** required
  - **Description:** AP SSID.
- Password
  - **Type:** String
  - **Required:** optional
  - **Default:** ""
  - **Description:** AP password (optional).

### Schema JSON

#### CLI Command

```
svc_call Wifi SetConnectAp {"SSID":null,"Password":null}
```

### GetConnectAp

**Description** Get target AP. Return type: JSON object. Example: {"ssid":"ssid1","password":"password1","is\_connectable":true}

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

## Schema JSON

### CLI Command

```
svc_call Wifi GetConnectAp
```

### GetConnectedAps

**Description** Get connected AP list. Return type: JSON array<object>. Example: [{"ssid":"ssid1","password":"password1","is\_connectable":true},{"ssid":"ssid2","password":"password2","is\_connectable":true}]

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

## Schema JSON

### CLI Command

```
svc_call Wifi GetConnectedAps
```

### SetScanParams

**Description** Set scan parameters.

### Execution

- Requires scheduler: Required

### Parameters

- Param
  - **Type:** Object
  - **Required:** required
  - **Description:** Scan parameters as a JSON object. Example: {"ap\_count":20,"interval\_ms":10000,"timeout\_ms":60000}

## Schema JSON

### CLI Command

```
svc_call Wifi SetScanParams {"Param":null}
```

### TriggerScanStart

**Description** Start Wi-Fi scan.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Wifi TriggerScanStart
```

### TriggerScanStop

**Description** Stop Wi-Fi scan.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Wifi TriggerScanStop
```

### SetSoftApParams

**Description** Set SoftAP parameters.

### Execution

- Requires scheduler: Required

### Parameters

- Param
  - **Type:** Object
  - **Required:** required
  - **Description:** SoftAP parameters as a JSON object. Example:  
{"ssid":"ssid","password":"password","max\_connection":4,"channel":1}

### Schema JSON

### CLI Command

```
svc_call Wifi SetSoftApParams {"Param":null}
```

### GetSoftApParams

**Description** Get SoftAP parameters. Return type: JSON object. Example: {"ssid":"","password":"","max\_connection":4,"channel":null}

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call Wifi GetSoftApParams
```

### TriggerSoftApStart

**Description** Start SoftAP.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call Wifi TriggerSoftApStart
```

### TriggerSoftApStop

**Description** Stop SoftAP.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call Wifi TriggerSoftApStop
```

### TriggerSoftApProvisionStart

**Description** Start SoftAP provisioning.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call Wifi TriggerSoftApProvisionStart
```

### TriggerSoftApProvisionStop

**Description** Stop SoftAP provisioning.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call Wifi TriggerSoftApProvisionStop
```

### ResetData

**Description** Reset Wi-Fi data, including target AP, scan parameters, and connected APs. Also clears NVS data.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Wifi ResetData
```

### Events

#### GeneralActionTriggered

**Description** Emitted when a general action is triggered.

#### Execution

- Requires scheduler: Required

#### Items

- Action
  - **Type:** String
  - **Description:** General action. Allowed values: [Init, Deinit, Start, Stop, Connect, Disconnect]

### Schema JSON

#### CLI Command

```
svc_subscribe Wifi GeneralActionTriggered
```

#### GeneralEventHappened

**Description** Emitted when a general event occurs.

#### Execution

- Requires scheduler: Required

#### Items

- Event
  - **Type:** String
  - **Description:** General event. Allowed values: [Deinit, Init, Stopped, Started, Disconnected, Connected]
- IsUnexpected
  - **Type:** Boolean
  - **Description:** Whether the event was unexpected.

### Schema JSON

#### CLI Command

```
svc_subscribe Wifi GeneralEventHappened
```

### ScanStateChanged

**Description** Emitted when scan state changes.

#### Execution

- Requires scheduler: Required

#### Items

- IsRunning
  - **Type:** Boolean
  - **Description:** Whether scanning is running.

### Schema JSON

#### CLI Command

```
svc_subscribe Wifi ScanStateChanged
```

### ScanApInfosUpdated

**Description** Emitted when scan AP list is updated.

#### Execution

- Requires scheduler: Required

#### Items

- ApInfos
  - **Type:** Array
  - **Description:** Scanned AP list as a JSON array<object>. Example:  
[{"ssid":"ssid1","password":"password1","is\_connectable":true}, {"ssid":"ssid2","password":"password2","is\_connectable":true}]

### Schema JSON

#### CLI Command

```
svc_subscribe Wifi ScanApInfosUpdated
```

### SoftApEventHappened

**Description** Emitted when a SoftAP event occurs.

### Execution

- Requires scheduler: Required

### Items

- Event
  - **Type:** String
  - **Description:** SoftAP event. Allowed values: [Started, Stopped]

### Schema JSON

### CLI Command

```
svc_subscribe Wifi SoftApEventHappened
```

### Related Types and Configs

For structs, enums, helper methods, and the full Doxygen-rendered API surface, see [Raw helper API reference](#).

## 4.2.4 Audio

- Component registry: [espressif/brookesia\\_service\\_audio](#)
- Helper header: `#include "brookesia/service_helper/audio.hpp"`
- Helper class: `esp_brookesia::service::helper::Audio`

### Overview

*brookesia\_service\_audio* provides:

- **Playback:** Stream from URL; pause, resume, stop.
- **Codecs:** PCM, OPUS, G711A encode/decode.
- **Playback state:** Idle / playing / paused with events.
- **Encoder:** Start/stop/configure; configurable read size.
- **Decoder:** Start/stop and feed compressed data; streaming decode.

### Features

**Codec Formats** The audio service supports the following codec formats:

Format	Encode	De-code	Notes
<b>PCM</b>	Yes	Yes	Lossless
<b>OPUS</b>	Yes	Yes	VBR and fixed bitrate
<b>G711A</b>	Yes	Yes	Telephony quality

### Playback

- URL playback.
- Pause, resume, stop.
- State events for UI sync.

### Encoder Configuration

- Codecs: PCM, OPUS, G711A.
- Channels: 1–4.
- Bits per sample: 8, 16, 24, 32.
- Sample rates: 8000, 16000, 24000, 32000, 44100, 48000 Hz.
- Frame duration (ms).
- OPUS: VBR and bitrate.

### Decoder Configuration

- **Codecs:** PCM, OPUS, G711A.
- **Channels:** 1–4.
- **Bits per sample:** 8, 16, 24, 32.
- **Sample rates:** 8000, 16000, 24000, 32000, 44100, 48000 Hz.
- **Frame duration (ms).**

### Events

- Playback state changes (Idle, Playing, Paused).
- Encoder events.
- Encoded data ready.

### Standard Include / Helper Class

- Standard include: `#include \"brookesia/service_helper/audio.hpp\"`
- Helper class: `esp_brookesia::service::helper::Audio`

### Service Interfaces

#### Functions

#### SetPlaybackConfig

**Description** Set playback config. Call before starting the service.

#### Execution

- Requires scheduler: Not required

#### Parameters

- Config
  - **Type:** Object
  - **Required:** required
  - **Description:** Playback config. Example: `{\"player_task\":{\"name\":\"Thread\",\"core_id\":0,\"priority\":5,\"stack_size\":4096,\"stack_min_size\":1,\"target_gain\":1E0,\"transition_time\":1500}}`

#### Schema JSON

#### CLI Command

```
svc_call Audio SetPlaybackConfig {\"Config\":null}
```

### SetEncoderStaticConfig

**Description** Set encoder static config. Call before starting the service.

#### Execution

- Requires scheduler: Not required

#### Parameters

- Config
  - **Type:** Object
  - **Required:** required
  - **Description:** Encoder static config. Example: {"recorder\_task":{"name":"Thread","core\_id":0,"priority":10,"stack\_size":4096}}

#### Schema JSON

#### CLI Command

```
svc_call Audio SetEncoderStaticConfig {"Config":null}
```

### SetDecoderStaticConfig

**Description** Set decoder static config. Call before starting the service.

#### Execution

- Requires scheduler: Not required

#### Parameters

- Config
  - **Type:** Object
  - **Required:** required
  - **Description:** Decoder static config. Example: {"feeder\_task":{"name":"Thread","core\_id":1,"priority":5,"stack\_size":4096,"target\_gain":1E0,"transition\_time":1500}}

#### Schema JSON

#### CLI Command

```
svc_call Audio SetDecoderStaticConfig {"Config":null}
```

### SetAFE\_Config

**Description** Set AFE config. Call before starting the service.

#### Execution

- Requires scheduler: Not required

### Parameters

- Config
  - **Type:** Object
  - **Required:** required
  - **Description:** AFE config. Example: {"feeder\_task":{"name":"Thread","core\_id":1,"priority":5,"stack\_size":40960,"stack\_

### Schema JSON

#### CLI Command

```
svc_call Audio SetAFE_Config {"Config":null}
```

### GetAFE\_WakeWords

**Description** Get AFE wake words. Return type: JSON array<string>. Example: ["ni hao xiao zhi","hello brooke-sia"]

#### Execution

- Requires scheduler: Not required

#### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Audio GetAFE_WakeWords
```

### PauseAFE\_WakeupEnd

**Description** Pause AFE wakeup-end task.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Audio PauseAFE_WakeupEnd
```

### ResumeAFE\_WakeupEnd

**Description** Resume AFE wakeup-end task.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

#### Schema JSON

#### CLI Command

```
svc_call Audio ResumeAFE_WakeupEnd
```

#### PlayUrl

**Description** Play audio from a URL. Supports loop and interrupt playback.

#### Execution

- Requires scheduler: Required

#### Parameters

- Url
  - **Type:** String
  - **Required:** required
  - **Description:** Audio URL, for example: "file://littlefs/example.mp3".
- Config
  - **Type:** Object
  - **Required:** optional
  - **Default:** {"interrupt":true,"delay\_ms":0,"loop\_count":0,"loop\_interval\_ms":0,"timeout\_ms":0}
  - **Description:** Playback config. Example: {"interrupt":true,"delay\_ms":0,"loop\_count":0,"loop\_interval\_ms":0,"timeout\_ms":0}

#### Schema JSON

#### CLI Command

```
svc_call Audio PlayUrl {"Url":null,"Config":null}
```

#### PlayUrls

**Description** Play audio from multiple URLs. Supports loop and interrupt playback.

#### Execution

- Requires scheduler: Required

### Parameters

- `Urls`
  - **Type:** `Array`
  - **Required:** required
  - **Description:** Audio URL list. Example: `["file://littlefs/example1.mp3","file://littlefs/example2.mp3"]`
- `Config`
  - **Type:** `Object`
  - **Required:** optional
  - **Default:** `{"interrupt":true,"delay_ms":0,"loop_count":0,"loop_interval_ms":0,"timeout_ms":0}`
  - **Description:** Playback config. Example: `{"interrupt":true,"delay_ms":0,"loop_count":0,"loop_interval_ms":0,"timeout_ms":0}`

### Schema JSON

#### CLI Command

```
svc_call Audio PlayUrls {"Urls":null,"Config":null}
```

### PlayControl

**Description** Control audio playback.

#### Execution

- Requires scheduler: Required

#### Parameters

- `Action`
  - **Type:** `String`
  - **Required:** required
  - **Description:** Playback action. Allowed values: [Pause, Resume, Stop]

### Schema JSON

#### CLI Command

```
svc_call Audio PlayControl {"Action":null}
```

### SetVolume

**Description** Set playback volume. [Deprecated] Use `Device::SetAudioPlayerVolume` instead.

#### Execution

- Requires scheduler: Required

### Parameters

- Volume
  - **Type:** Number
  - **Required:** required
  - **Description:** Volume in range [0, 100].

### Schema JSON

#### CLI Command

```
svc_call Audio SetVolume {"Volume":null}
```

### GetVolume

**Description** Get playback volume. Return type: number. Example: 70. [Deprecated] Use *Device::GetAudioPlayerVolume* instead.

#### Execution

- Requires scheduler: Not required

#### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Audio GetVolume
```

### SetMute

**Description** Set playback mute. [Deprecated] Use *Device::SetAudioPlayerMute* instead.

#### Execution

- Requires scheduler: Required

#### Parameters

- Enable
  - **Type:** Boolean
  - **Required:** required
  - **Description:** Enable mute.

### Schema JSON

#### CLI Command

```
svc_call Audio SetMute {"Enable":null}
```

### StartEncoder

**Description** Start audio encoder.

### Execution

- Requires scheduler: Required

### Parameters

- Config
  - **Type:** Object
  - **Required:** required
  - **Description:** Audio encoder config. Example: {"type":"PCM","general":{"channels":0,"sample\_bits":0,"sample\_rate":0,"f

### Schema JSON

### CLI Command

```
svc_call Audio StartEncoder {"Config":null}
```

### StopEncoder

**Description** Stop audio encoder.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call Audio StopEncoder
```

### PauseEncoder

**Description** Pause audio encoder.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call Audio PauseEncoder
```

### ResumeEncoder

**Description** Resume audio encoder.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call Audio ResumeEncoder
```

### StartDecoder

**Description** Start audio decoder.

### Execution

- Requires scheduler: Required

### Parameters

- Config
  - **Type:** Object
  - **Required:** required
  - **Description:** Audio decoder config. Example: {"type":"PCM","general":{"channels":0,"sample\_bits":0,"sample\_rate":0,"f

### Schema JSON

### CLI Command

```
svc_call Audio StartDecoder {"Config":null}
```

### StopDecoder

**Description** Stop audio decoder.

**Execution**

- Requires scheduler: Required

**Parameters**

- No parameters.

**Schema JSON**

**CLI Command**

```
svc_call Audio StopDecoder
```

**FeedDecoderData**

**Description** Feed audio data to the decoder.

**Execution**

- Requires scheduler: Not required

**Parameters**

- Data
  - **Type:** RawBuffer
  - **Required:** required
  - **Description:** Audio data to decode.

**Schema JSON**

**CLI Command**

```
svc_call Audio FeedDecoderData {"Data":null}
```

**ResetData**

**Description** Reset audio data. Includes player volume. [Deprecated] Use *Device::ResetData* instead.

**Execution**

- Requires scheduler: Required

**Parameters**

- No parameters.

**Schema JSON**

### CLI Command

```
svc_call Audio ResetData
```

### Events

#### PlayStateChanged

**Description** Emitted when playback state changes.

#### Execution

- Requires scheduler: Required

#### Items

- State
  - **Type:** String
  - **Description:** Playback state. Allowed values: [Idle, Playing, Paused]

### Schema JSON

#### CLI Command

```
svc_subscribe Audio PlayStateChanged
```

#### AFE\_EventHappened

**Description** Emitted when an AFE event occurs.

#### Execution

- Requires scheduler: Required

#### Items

- Event
  - **Type:** String
  - **Description:** AFE event. Allowed values: [VAD\_Start, VAD\_End, WakeStart, WakeEnd]

### Schema JSON

#### CLI Command

```
svc_subscribe Audio AFE_EventHappened
```

#### EncoderDataReady

**Description** Emitted when encoder data is ready.

### Execution

- Requires scheduler: Not required

### Items

- Data
  - **Type:** `RawBuffer`
  - **Description:** Encoded audio data.

### Schema JSON

#### CLI Command

```
svc_subscribe Audio EncoderDataReady
```

### RecorderDataReady

**Description** Emitted when recorder data is ready.

### Execution

- Requires scheduler: Not required

### Items

- Data
  - **Type:** `RawBuffer`
  - **Description:** Recorded raw audio data.

### Schema JSON

#### CLI Command

```
svc_subscribe Audio RecorderDataReady
```

### Related Types and Configs

For structs, enums, helper methods, and the full Doxygen-rendered API surface, see [Raw helper API reference](#).

## 4.2.5 Device Control

- Component registry: [espressif/brookesia\\_service\\_device](#)
- Helper header: `#include "brookesia/service_helper/device.hpp"`
- Helper class: `esp_brookesia::service::helper::Device`

## Overview

`brookesia_service_device` is an application-facing device control service. It is not a new hardware driver. Instead, it provides a unified service interface for application code to access HAL capabilities. When the service starts, it discovers initialized HAL interfaces and exposes common control and status-query operations as Service functions and events.

Typical usage:

- Application code calls functions or subscribes to events through `service::helper::Device`.
- HAL adaptor and board components provide low-level interfaces such as `AudioCodecPlayerIface`, `DisplayBacklightIface`, `StorageFsIface`, and `PowerBatteryIface`.
- `brookesia_service_device` sits in between to validate parameters, cache state, publish events, and persist selected application-level state.

Therefore, applications usually do not need to hold HAL interface pointers directly unless they need low-level or board-specific behavior.

## Features

**Capability Discovery** The service provides `GetCapabilities` to return a snapshot of registered HAL interfaces. Applications can query capabilities first, then decide whether to show related UI or invoke related control functions.

Common capabilities:

HAL Interface	Typical Usage
<code>BoardInfoIface</code>	Get static board name, chip, version, vendor, and description.
<code>DisplayBacklightIface</code>	Set/query backlight brightness and on/off state.
<code>AudioCodecPlayerIface</code>	Set/query player volume and mute state.
<code>StorageFsIface</code>	Get mounted file-system information.
<code>PowerBatteryIface</code>	Get battery information, state, and charge configuration; control charging when supported.

**Control Functions** Control functions update device state and are intended for UI, Agent tool calls, or remote-control workflows:

- **Display control:** set backlight brightness percentage and backlight on/off state.
- **Audio control:** set player volume percentage and mute state.
- **Power control:** set battery charge configuration or enable/disable charging when supported by HAL.
- **Data reset:** clear persisted volume, mute, and brightness state, then restore defaults.

Brightness and volume use application-level percentages in  $[0, 100]$ . The service maps them to the hardware min/max ranges configured through `Kconfig` before calling HAL.

**Status Query Functions** Status-query functions read static information or runtime state:

- **Capabilities and board information:** get the current HAL capability snapshot and static board information.
- **Display state:** get current backlight brightness percentage and on/off state.
- **Audio state:** get target player volume percentage and mute state.
- **Storage state:** get mounted file systems and mount points.
- **Battery state:** get battery capabilities, voltage, percentage, charge state, low/critical flags, and charge configuration when supported.

**Events** The service publishes events when cached state changes. Applications can subscribe through `service::helper::Device::subscribe_event`:

- `DisplayBacklightBrightnessChanged`
- `DisplayBacklightOnOffChanged`
- `AudioPlayerVolumeChanged`
- `AudioPlayerMuteChanged`
- `PowerBatteryStateChanged`
- `PowerBatteryChargeConfigChanged`

Battery state is polled from HAL at the configured interval, and an event is published when the snapshot changes.

**Persisted State** When `brookesia_service_nvs` is enabled and running, Device service attempts to save and restore the following application-level state:

- Player volume
- Player mute
- Backlight brightness

If the NVS service is unavailable, Device service still works, but these values are not persisted across reboot.

### Usage Recommendations

- Initialize required HAL devices before starting `ServiceManager` and the Device service.
- Query `GetCapabilities` before invoking optional control functions.
- For UI and Agent tool calls, prefer Device service as the application-layer entry point to HAL instead of depending on board-specific HAL implementations directly.
- For high-throughput data paths such as audio streams or display frame drawing, use dedicated services or HAL interfaces instead of routing bulk data through Device service.

### Standard Include / Helper Class

- Standard include: `#include \"brookesia/service_helper/device.hpp\"`
- Helper class: `esp_brookesia::service::helper::Device`

### Service Interfaces

#### Functions

#### GetCapabilities

**Description** Get device control service capabilities. Return type: JSON object. Example: `{\"Board-Info\":[\"General:BoardInfo\"],\"DisplayPanel\":[\"Display:PanelA\",\"Display:PanelB\"]}`

#### Execution

- Requires scheduler: Not required

#### Parameters

- No parameters.

#### Schema JSON

### CLI Command

```
svc_call Device GetCapabilities
```

### GetBoardInfo

**Description** Get static board information. Return type: JSON object. Example: {"name":"esp32\_s3\_touch\_amoled\_1\_8","chip":"ESP32-S3","version":"v1.0","description":"Example board information","manufacturer":"Espressif"}

### Execution

- Requires scheduler: Not required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call Device GetBoardInfo
```

### SetDisplayBacklightBrightness

**Description** Set display backlight brightness percentage.

### Execution

- Requires scheduler: Required

### Parameters

- Brightness
  - **Type:** Number
  - **Required:** required
  - **Description:** Backlight brightness percentage in range [0, 100]. The actual brightness will be mapped to the hardware range [BROOKE-SIA\_SERVICE\_DEVICE\_DISPLAY\_BACKLIGHT\_BRIGHTNESS\_MIN, BROOKE-SIA\_SERVICE\_DEVICE\_DISPLAY\_BACKLIGHT\_BRIGHTNESS\_MAX].

### Schema JSON

### CLI Command

```
svc_call Device SetDisplayBacklightBrightness {"Brightness":null}
```

### GetDisplayBacklightBrightness

**Description** Get current display backlight brightness percentage [0, 100]. Return type: number.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Device GetDisplayBacklightBrightness
```

### SetDisplayBacklightOnOff

**Description** Turn display backlight on or off.

### Execution

- Requires scheduler: Required

### Parameters

- On
  - **Type:** Boolean
  - **Required:** required
  - **Description:** True to turn on the backlight, false to turn it off.

### Schema JSON

#### CLI Command

```
svc_call Device SetDisplayBacklightOnOff {"On":null}
```

### GetDisplayBacklightOnOff

**Description** Get whether display backlight is enabled. Return type: boolean.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Device GetDisplayBacklightOnOff
```

### SetAudioPlayerVolume

**Description** Set target audio player volume percentage.

#### Execution

- Requires scheduler: Required

#### Parameters

- Volume
  - **Type:** Number
  - **Required:** required
  - **Description:** Player volume percentage in range [0, 100]. The actual volume will be mapped to the hardware range [BROOKESIA\_SERVICE\_DEVICE\_AUDIO\_PLAYER\_VOLUME\_MIN, BROOKESIA\_SERVICE\_DEVICE\_AUDIO\_PLAYER\_VOLUME\_MAX].

#### Schema JSON

#### CLI Command

```
svc_call Device SetAudioPlayerVolume {"Volume":null}
```

### GetAudioPlayerVolume

**Description** Get target audio player volume percentage. Return type: number.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

#### Schema JSON

#### CLI Command

```
svc_call Device GetAudioPlayerVolume
```

### SetAudioPlayerMute

**Description** Set whether audio player is muted.

#### Execution

- Requires scheduler: Required

### Parameters

- Enable
  - **Type:** Boolean
  - **Required:** required
  - **Description:** True to mute audio player, false to unmute it.

### Schema JSON

#### CLI Command

```
svc_call Device SetAudioPlayerMute {"Enable":null}
```

### GetAudioPlayerMute

**Description** Get whether audio player is muted. Return type: boolean.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Device GetAudioPlayerMute
```

### GetStorageFileSystems

**Description** Get mounted storage file systems. Return type: JSON array<object>. Example: [{"fs\_type":"LittleFS","medium\_type":"Flash","mount\_point":"/littlefs","supports\_directories":true},{ "fs\_type":"FATFS","medium\_ty

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Device GetStorageFileSystems
```

### GetStorageFileSystemCapacity

**Description** Get one mounted storage file system capacity. Return type: JSON object. Example: {"total\_bytes":1048576,"used\_bytes":262144,"free\_bytes":786432}

### Execution

- Requires scheduler: Required

### Parameters

- MountPoint
  - **Type:** String
  - **Required:** required
  - **Description:** Mount point returned by GetStorageFileSystems.

### Schema JSON

#### CLI Command

```
svc_call Device GetStorageFileSystemCapacity {"MountPoint":null}
```

### GetPowerBatteryInfo

**Description** Get static power battery information. Return type: JSON object. Example: {"name":"MainBattery","chemistry":"Li-ion","abilities":["Voltage","Percentage","ChargeState"]}

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Device GetPowerBatteryInfo
```

### GetPowerBatteryState

**Description** Get current power battery state. Return type: JSON object. Example: {"is\_present":true,"power\_source":"Battery","charge\_state":"NotCharging","level\_source":"VoltageCurve","voltage\_mv":3920,"percentage":100}

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

## Schema JSON

### CLI Command

```
svc_call Device GetPowerBatteryState
```

### GetPowerBatteryChargeConfig

**Description** Get current power battery charge configuration. Return type: JSON object. Example: {"enabled":true,"target\_voltage\_mv":4200,"charge\_current\_ma":500,"precharge\_current\_ma":100,"termination\_current\_ma":100}

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

## Schema JSON

### CLI Command

```
svc_call Device GetPowerBatteryChargeConfig
```

### SetPowerBatteryChargeConfig

**Description** Set power battery charge configuration.

### Execution

- Requires scheduler: Required

### Parameters

- Config
  - **Type:** Object
  - **Required:** required
  - **Description:** Battery charge configuration object.

## Schema JSON

### CLI Command

```
svc_call Device SetPowerBatteryChargeConfig {"Config":null}
```

### SetPowerBatteryChargingEnabled

**Description** Enable or disable battery charging.

### Execution

- Requires scheduler: Required

### Parameters

- Enabled
  - **Type:** Boolean
  - **Required:** required
  - **Description:** True to enable charging, false to disable charging.

### Schema JSON

#### CLI Command

```
svc_call Device SetPowerBatteryChargingEnabled {"Enabled":null}
```

### ResetData

**Description** Reset persisted device control state, including backlight brightness, player volume, and player mute.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Device ResetData
```

### Events

#### DisplayBacklightBrightnessChanged

**Description** Emitted when the target display backlight brightness changes.

### Execution

- Requires scheduler: Required

### Items

- Brightness
  - **Type:** Number
  - **Description:** Current target backlight brightness percentage [0, 100].

### Schema JSON

#### CLI Command

```
svc_subscribe Device DisplayBacklightBrightnessChanged
```

### DisplayBacklightOnOffChanged

**Description** Emitted when the target display backlight on/off state changes.

#### Execution

- Requires scheduler: Required

#### Items

- IsOn
  - **Type:** Boolean
  - **Description:** Whether display backlight is currently on. True if on, false if off.

### Schema JSON

#### CLI Command

```
svc_subscribe Device DisplayBacklightOnOffChanged
```

### AudioPlayerVolumeChanged

**Description** Emitted when the target audio player volume changes.

#### Execution

- Requires scheduler: Required

#### Items

- Volume
  - **Type:** Number
  - **Description:** Current target player volume percentage [0, 100].

### Schema JSON

#### CLI Command

```
svc_subscribe Device AudioPlayerVolumeChanged
```

### AudioPlayerMuteChanged

**Description** Emitted when the target audio player mute state changes.

### Execution

- Requires scheduler: Required

### Items

- IsMuted
  - **Type:** Boolean
  - **Description:** Whether audio player is currently muted. True if muted, false if unmuted.

### Schema JSON

#### CLI Command

```
svc_subscribe Device AudioPlayerMuteChanged
```

### PowerBatteryStateChanged

**Description** Emitted when the power battery state snapshot changes.

### Execution

- Requires scheduler: Required

### Items

- State
  - **Type:** Object
  - **Description:** Current power battery state snapshot.

### Schema JSON

#### CLI Command

```
svc_subscribe Device PowerBatteryStateChanged
```

### PowerBatteryChargeConfigChanged

**Description** Emitted when the power battery charge configuration changes.

### Execution

- Requires scheduler: Required

### Items

- Config
  - **Type:** Object
  - **Description:** Current power battery charge configuration.

### Schema JSON

### CLI Command

```
svc_subscribe Device PowerBatteryChargeConfigChanged
```

### Related Types and Configs

For structs, enums, helper methods, and the full Doxygen-rendered API surface, see [Raw helper API reference](#).

## 4.2.6 Video

- Component registry: [espressif/brookesia\\_service\\_video](#)
- Helper header: `#include "brookesia/service_helper/video.hpp"`
- Helper class: `esp_brookesia::service::helper::Video`

### Overview

*brookesia\_service\_video* provides:

- **Encoding:** Capture to compressed or raw formats (resolution, FPS, format per stream).
- **Decoding:** H.264, MJPEG, and others to RGB/YUV for display or processing.

### Features

**Encoder** Input comes from a **local video device** (default `/dev/video0`; Kconfig can set path prefix and count).

Each output stream can be one of (per-stream resolution and FPS):

Type	Description
<b>H.264</b>	Common network/storage codec
<b>MJPEG</b>	Frame-wise JPEG-like compression
<b>RGB565 / RGB888 / BGR888</b>	RGB formats
<b>YUV420 / YUV422 / O_UYY_E_VYY</b>	YUV formats

**Warning:** Multiple simultaneous outputs are not supported yet.

**Decoder** Compressed input and pixel output; configure per use case.

### Input (Compressed)

Format	Description
<b>H.264</b>	Common video codec
<b>MJPEG</b>	Frame-wise JPEG stream

**Output (Pixel)**

Format	Description
<b>RGB565 (BE/LE)</b>	16-bit RGB for some panels/framebuffers
<b>RGB888 / BGR888</b>	24-bit RGB/BGR
<b>YUV420P / YUV422P</b>	Planar YUV for pipelines
<b>YUV422 / UYVY422</b>	Packed YUV for capture/display
<b>O_UYY_E_VYY</b>	Packed layout (hardware/pipeline dependent)

**Standard Include / Helper Class**

- Standard include: `#include \"brookesia/service_helper/video.hpp\"`
- Helper class: `esp_brookesia::service::helper::Video`

**Service Interfaces****Functions****Encoder****Open**

**Description** Open the encoder with config.

**Execution**

- Requires scheduler: Required

**Parameters**

- Config
  - **Type:** Object
  - **Required:** required
  - **Description:** Encoder config. Example: `{\"sinks\": [{\"format\": \"H264\", \"width\": 320, \"height\": 240, \"fps\": 30}, {\"format\": \"MJPE`

**Schema JSON****CLI Command**

```
svc_call VideoEncoder0 Open {\"Config\":null}
```

**Close**

**Description** Close encoder.

**Execution**

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call VideoEncoder0 Close
```

### Start

**Description** Start encoder.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call VideoEncoder0 Start
```

### Stop

**Description** Stop encoder.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call VideoEncoder0 Stop
```

### FetchFrame

**Description** Fetch an encoder output frame and emit *FetchSinkFrameReady*. Only available in non-stream mode.

### Execution

- Requires scheduler: Required

### Parameters

- SinkIndex
  - **Type:** Number
  - **Required:** optional
  - **Default:** 0x0
  - **Description:** Sink index.

### Schema JSON

#### CLI Command

```
svc_call VideoEncoder0 FetchFrame {"SinkIndex":null}
```

### Decoder

#### Open

**Description** Open the decoder with config.

### Execution

- Requires scheduler: Required

### Parameters

- Config
  - **Type:** Object
  - **Required:** required
  - **Description:** Decoder config. Example: {"width":320,"height":240,"source\_format":"H264","sink\_format":"RGB565\_LE"}

### Schema JSON

#### CLI Command

```
svc_call VideoDecoder0 Open {"Config":null}
```

### Close

**Description** Close decoder.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call VideoDecoder0 Close
```

#### Start

**Description** Start decoder.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call VideoDecoder0 Start
```

#### Stop

**Description** Stop decoder.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call VideoDecoder0 Stop
```

### FeedFrame

**Description** Feed a decoder input frame.

#### Execution

- Requires scheduler: Required

### Parameters

- Frame
  - **Type:** RawBuffer
  - **Required:** required
  - **Description:** Frame data.

### Schema JSON

#### CLI Command

```
svc_call VideoDecoder0 FeedFrame {"Frame":null}
```

### Events

#### Encoder

#### StreamSinkFrameReady

**Description** Emitted when an encoder stream frame is ready. Stream mode only.

#### Execution

- Requires scheduler: Not required

#### Items

- SinkIndex
  - **Type:** Number
  - **Description:** Sink index.
- SinkInfo
  - **Type:** Object
  - **Description:** Sink info. Example: {"format":"H264","width":320,"height":240,"fps":30}
- Frame
  - **Type:** RawBuffer
  - **Description:** Encoded frame data.

### Schema JSON

#### CLI Command

```
svc_subscribe VideoEncoder0 StreamSinkFrameReady
```

#### FetchSinkFrameReady

**Description** Emitted when an encoder fetched frame is ready. Non-stream mode only.

#### Execution

- Requires scheduler: Not required

### Items

- SinkIndex
  - **Type:** Number
  - **Description:** Sink index.
- SinkInfo
  - **Type:** Object
  - **Description:** Sink info. Example: {"format":"MJPEG","width":320,"height":240,"fps":15}
- Frame
  - **Type:** RawBuffer
  - **Description:** Encoded frame data.

### Schema JSON

#### CLI Command

```
svc_subscribe VideoEncoder0 FetchSinkFrameReady
```

### Decoder

#### SinkFrameReady

**Description** Emitted when a decoder output frame is ready.

#### Execution

- Requires scheduler: Not required

### Items

- Width
  - **Type:** Number
  - **Description:** Decoded frame width.
- Height
  - **Type:** Number
  - **Description:** Decoded frame height.
- Frame
  - **Type:** RawBuffer
  - **Description:** Decoded frame data.

### Schema JSON

#### CLI Command

```
svc_subscribe VideoDecoder0 SinkFrameReady
```

### Related Types and Configs

For structs, enums, helper methods, and the full Doxygen-rendered API surface, see [Raw helper API reference](#).

## 4.2.7 Custom

- Component registry: [espressif/brookesia\\_service\\_custom](#)
- Public header: `#include "brookesia/service_custom.hpp"`

### Overview

`brookesia_service_custom` ships a ready-to-use **CustomService** so you can register custom functions and events without a separate Brookesia component.

### Typical Uses

- **Lightweight features:** LEDs, PWM, GPIO toggles, simple sensors—wrapped as `CustomService` calls.
- **Prototypes:** Expose logic as functions or events for local or RPC access.
- **Extensibility:** Add capabilities without changing the core service framework.

### Features

- **Dynamic registration:** `register_function()` / `register_event()` at runtime.
- **Fixed handler shape:** `FunctionParameterMap` in, `FunctionResult` out; lambdas, `std::function`, free functions, functors, `std::bind`.
- **Events:** Full publish/subscribe lifecycle.
- **ServiceManager:** Local calls and remote RPC.
- **Optional worker:** Task scheduler for thread-safe execution.

### API Reference

#### Header File

- [service/brookesia\\_service\\_custom/include/brookesia/service\\_custom/service\\_custom.hpp](#)

#### Classes

class **CustomService** : public `esp_brookesia::service::ServiceBase`

Dynamic service that lets callers register functions and events at runtime.

#### Public Functions

bool **register\_function** (FunctionSchema schema, FunctionHandler handler)

Register a function for custom service.

##### Parameters

- **schema** -- [in] Function schema
- **handler** -- [in] Function handler

**Returns** true if registered successfully, false otherwise

bool **unregister\_function** (const std::string &function\_name)

Unregister a function.

**Parameters** **function\_name** -- [in] Function name

**Returns** true if unregistered successfully, false otherwise

virtual std::vector<FunctionSchema> **get\_function\_schemas** () override  
Get currently registered function schemas for custom service.  
**Returns** std::vector<FunctionSchema> Function schemas

bool **register\_event** (EventSchema event\_schema)  
Register an event for custom service.  
**Parameters** **event\_schema** -- [in] Event schema to register.  
**Returns** true if registered successfully, false otherwise

bool **unregister\_event** (const std::string &event\_name)  
Unregister an event.  
**Parameters** **event\_name** -- [in] Event name  
**Returns** true if unregistered successfully, false otherwise

bool **publish\_event** (const std::string &event\_name, EventItemMap event\_items)  
Publish an event for custom service.  
**Parameters**

- **event\_name** -- [in] Event name
- **event\_items** -- [in] Event items

**Returns** true if published successfully, false otherwise

virtual std::vector<EventSchema> **get\_event\_schemas** () override  
Get currently registered event schemas for custom service.  
**Returns** std::vector<EventSchema> Event schemas

### Public Static Functions

static inline *CustomService* &**get\_instance** ()  
Get the process-wide singleton instance.  
**Returns** Reference to the singleton custom service.

## 4.3 Development Guide

### 4.3.1 Application Development Guide

This guide explains how to integrate the ESP-Brookesia **service framework** in an ESP-IDF project, using the **Wi-Fi service** as an example. A full, buildable sample is in `examples/service/wifi`.

In a typical project, using a component directly means adding a dependency, including headers, and calling its APIs. As features grow, coupling between modules and components increases; removing a component then requires cleaning up headers and calls as well as the dependency, which raises maintenance cost.

ESP-Brookesia abstracts this behind two unified surfaces: **functions** and **events**. In many cases you only need `brookesia_service_manager` and `brookesia_service_helper` to reach service features through the Helper; you can also check at runtime whether a service is available. If the concrete service component is not linked, Helper usage usually still compiles, which helps with trimming and maintenance.

The framework also relies on mechanisms such as the *Task Scheduler* for asynchronous execution on the service side, reducing blocking on the caller. Exposed APIs are designed with thread safety in mind so application code does less ad-hoc synchronization.

The sections below follow this order: **dependencies** → **initialization and binding** → **calling functions** → **subscribing to events** → **event monitors**.

## Adding Component Dependencies

Declare dependencies in `idf_component.yml` at the project root or in a component directory. For the Wi-Fi service:

```
dependencies:
  espressif/brookesia_service_wifi: "*"
  # espressif/brookesia_service_nvs: "*" # optional, if you need the NVS service
```

If your project does not link a concrete service implementation and you only need to satisfy “Helper code compiles”, you can depend on `brookesia_service_helper` alone:

```
dependencies:
  espressif/brookesia_service_helper: "*"
```

For how to obtain components and version constraints, see *Obtaining and using components*.

## Headers, Namespace, and Type Aliases

```
#include "brookesia/lib_utils.hpp"
#include "brookesia/service_manager.hpp"
// Include all general service helper headers
#include "brookesia/service_helper.hpp"
// Or include the Helper header for a specific service component, e.g. Wi-Fi
// #include "brookesia/service_helper/wifi.hpp"

// Brookesia data types live under the esp_brookesia namespace
using namespace esp_brookesia;
// Type aliases keep examples short
// Service-related types are under the service namespace
using WifiHelper = service::helper::Wifi;
```

## Starting the Service Manager

Before calling any service API, initialize the global **singleton** `ServiceManager` by calling `start()`.

```
auto &service_manager = service::ServiceManager::get_instance();
service_manager.start();
```

## Starting / Stopping a Service

```
// Before binding, you can check whether the Helper's service is linked into the_
↳build
if (!WifiHelper::is_available()) {
  // Service unavailable: omitting the service component does not break_
↳compilation, but this check fails
  return;
}

// Bind to the service: starts the service and its dependencies while `binding` is_
↳alive;
// when `binding` goes out of scope it is destroyed and the service stops.
// To keep the service running longer, store `binding` outside the stack (e.g._
↳static or heap).
auto binding = service_manager.bind(WifiHelper::get_name().data());
if (!binding.is_valid()) {
  // Failed to start the service
```

(continues on next page)

```

return;
}

```

### Calling Service Functions

Before calling a function, confirm **parameter types, order, and return value** from the Helper contract or headers, for example:

- [Wi-Fi service function interface reference](#)
- [Wi-Fi Helper header source](#)

#### Note:

- **Description:** Function behavior, return information, etc. If there is no **return value** section, the function has no return value.
- **Parameter list: Parameter names, types, descriptions, and defaults.**
  - **Types: Six kinds are supported: String, Number, Boolean, Object, Array, RawBuffer. Mapping to c**
    - \* String: `std::string`
    - \* Number: `double / int / any arithmetic type`
    - \* Boolean: `bool`
    - \* Object: `boost::json::object` (often produced by serializing a struct)
    - \* Array: `boost::json::array` (often produced by serializing `std::vector`)
    - \* RawBuffer: `service::RawBuffer`
  - **Description:**
    - \* For String with enumerated values in the description, you pass enums via serialization.
    - \* For Object / Array, the description may include examples.
  - **Default:** Shown when the parameter is **optional**.
- **Execution requirement: Whether the function requires the Task Scheduler.**
  - **Required:** Must run **after the service has started**; `sync/async` calls go through the scheduler.
  - **Not required: May run before the service starts; only sync calls are allowed. Typical cases:**
    - \* The function must run before startup.
    - \* The parameter list contains `RawBuffer`.
    - \* The implementation is already thread-safe (locks, etc.), so the scheduler is not required.

Use static Helper methods for **synchronous** or **asynchronous** calls. **Synchronous** calls are described first.

**Synchronous Calls** The call blocks until the function finishes or times out.

```

auto result = WifiHelper::call_function_sync<ReturnType>
↳(WifiHelper::FunctionId::FunctionName [, parameters...] [, timeout]);
if (!result) {
    // Call failed, log error
    BROOKESIA_LOGE("Failed: %1%", result.error());
} else {
    // Call succeeded
    // If the function returns a value, use `result.value()`
    auto &value = result.value(); // `value` has type `ReturnType`
    // ...
}

```

#### Note:

- `parameters` must match the schema in type, count, and order; omit them if the function has no parameters.
- If the return type is `void`, omit the `<ReturnType>` template argument.
- `timeout` is optional; if omitted, the default is `BROOKESIA_SERVICE_MANAGER_DEFAULT_CALL_FUNCTION_TIMEOUT`.

- See `esp_brookesia::service::helper::Base::call_function_sync()`.

**Example: Multiple Parameters** SetConnectAp —interface summary and sample code:

- Function: SetConnectAp
- Parameters:
  - SSID: String
  - Password: String
- Return type: void

```
// Pass parameters in schema order; do not skip required parameters
auto set_connect_ap_result = WifiHelper::call_function_sync(
    WifiHelper::FunctionId::SetConnectAp, "ssid1", "password1",
    ↪service::helper::Timeout(100));
if (!set_connect_ap_result) {
    // Call failed, log error
    BROOKESIA_LOGE("Failed: %1%", set_connect_ap_result.error());
}
```

**Example: Serialized Parameters** TriggerGeneralAction —interface summary and sample code:

- Function: TriggerGeneralAction
- Parameters:
  - Action: String (serialize from `WifiHelper::GeneralAction`)
- Return type: void

```
auto start_result = WifiHelper::call_function_sync(
    WifiHelper::FunctionId::TriggerGeneralAction,
    BROOKESIA_DESCRIBE_TO_STR(WifiHelper::GeneralAction::Start));
if (!start_result) {
    // Call failed, log error
    BROOKESIA_LOGE("Failed: %1%", start_result.error());
}
```

SetScanParams —interface summary and sample code:

- Function: SetScanParams
- Parameters:
  - Param: Object (serialize from `WifiHelper::ScanParams`)
- Return type: void

```
WifiHelper::ScanParams scan_params{
    .ap_count = 10,
    .interval_ms = 10000,
    .timeout_ms = 60000,
};
auto set_scan_params_result = WifiHelper::call_function_sync(
    WifiHelper::FunctionId::SetScanParams, BROOKESIA_DESCRIBE_TO_JSON(scan_params).
    ↪as_object());
if (!set_scan_params_result) {
    // Call failed, log error
    BROOKESIA_LOGE("Failed: %1%", set_scan_params_result.error());
}
```

**Example: Parsing Return Values** GetConnectedAps —interface summary and sample code:

- Function: GetConnectedAps
- Parameters: none
- Return type: `boost::json::array` (deserialize to `std::vector<WifiHelper::ConnectApInfo>`)

```

auto get_connected_aps_result =
    WifiHelper::call_function_sync<boost::json::array>
    ↪(WifiHelper::FunctionId::GetConnectedAps);
if (!get_connected_aps_result) {
    // Call failed, log error
    BROOKESIA_LOGE("Failed to get connected APs: %1%", get_connected_aps_result.
    ↪error());
}

// Parse return value
std::vector<WifiHelper::ConnectApInfo> infos;
auto parse_result = BROOKESIA_DESCRIBE_FROM_JSON(get_connected_aps_result.value(), ↪
    ↪infos);
if (!parse_result) {
    // Parse failed, log error
    BROOKESIA_LOGE("Failed to parse connected APs: %1%", get_connected_aps_result.
    ↪error());
} else {
    // Parse succeeded, log result
    BROOKESIA_LOGI("Connected APs: %1%", infos);
}

```

**Asynchronous Calls** The call **submits** work and returns immediately without blocking the caller. If you pass a result handler, the framework invokes it **asynchronously** when the function completes.

```

auto on_function_handler = [] (service::FunctionResult &&result) {
    // Process return value
    if (!result.success) {
        // Execution failed, log error
        BROOKESIA_LOGE("Failed: %1%", result.error_message);
    } else {
        // Execution succeeded
        // If there is a return value:
        // auto &value = result.get_data<ReturnType>();
        // ...
    }
};
auto result = WifiHelper::call_function_async(WifiHelper::FunctionId::FunctionName_
    ↪[, parameters...] [, on_function_handler]);
if (!result) {
    // Call failed
}

```

---

**Note:**

- parameters must match the schema; omit if the function has no parameters.
- on\_function\_handler is optional; if omitted, results are ignored.
- Serial async calls to the same service run **in submission order** inside the service:

```

Helper::call_function_async(Helper::FunctionId::Function1);
Helper::call_function_async(Helper::FunctionId::Function2);
Helper::call_function_async(Helper::FunctionId::Function3);
// Internal execution order: Function1 -> Function2 -> Function3

```

- See `esp_brookesia::service::helper::Base::call_function_async()`.
- 

**Example** GetConnectedAps —interface summary and sample code:

- Function: GetConnectedAps

- Parameters: none
- Return type: `boost::json::array` (same as `std::vector<WifiHelper::ConnectApInfo>` when serialized)

```

auto on_get_connected_aps_handler = [](service::FunctionResult &&result) {
    if (!result.success) {
        // Execution failed, log error
        BROOKESIA_LOGE("Failed to get connected APs: %1%", result.error_message);
        return;
    }
    // Execution succeeded, parse return value
    std::vector<WifiHelper::ConnectApInfo> infos;
    auto &data = result.get_data<boost::json::array>();
    auto parse_result = BROOKESIA_DESCRIBE_FROM_JSON(data, infos);
    if (!parse_result) {
        // Parse failed, log error
        BROOKESIA_LOGE("Failed to parse connected APs: %1%", result);
        return;
    }
    // Parse succeeded, log result
    BROOKESIA_LOGI("Connected APs: %1%", infos);
};
auto get_connected_aps_result =
    WifiHelper::call_function_async(WifiHelper::FunctionId::GetConnectedAps, on_get_
    ↪connected_aps_handler);
if (!get_connected_aps_result) {
    // Call failed
    return;
}

```

### Subscribing to Service Events

Confirm **item names, types, and order** from the Helper contract or headers, for example:

- [Wi-Fi service event interface reference](#)
- [Wi-Fi Helper header source](#)

#### Note:

- **Description:** Event behavior and return information. If there is no **return value** section, the event carries no return payload in that sense.
- **Item list: Item names, types, descriptions, and defaults.**
  - **Types: Same six kinds as for functions; mapping is the same:**
    - \* String: `std::string`
    - \* Number: `double / int / any arithmetic type`
    - \* Boolean: `bool`
    - \* Object: `boost::json::object` (often produced by serializing a struct)
    - \* Array: `boost::json::array` (often produced by serializing `std::vector`)
    - \* RawBuffer: `service::RawBuffer`
  - **Description:**
    - \* String with enumerated values deserializes to enum-like data.
    - \* Object / Array may include examples.
- **Execution requirement: Whether the event requires the Task Scheduler.**
  - **Required:** Must be published **after** the service has started.
  - **Not required: May be published before startup. Typical cases:**
    - \* The event must fire before startup.
    - \* The item list contains `RawBuffer`.

**Subscribing to Events** Use `subscribe_event()` to register a callback; when the event fires, callbacks run **serially** in subscription order.

```
auto on_event_handler = [] (const std::string &event_name[, items...]) {
    // Handle the event
};
// `conn` is the subscription; destruction **unsubscribes** (RAII).
// To keep the subscription alive, store `conn` outside the stack (e.g. static or_
↳heap).
auto conn = WifiHelper::subscribe_event(WifiHelper::EventId::EventName, on_event_
↳handler);
if (!conn.connected()) {
    // Subscribe failed
}
// You can also unsubscribe manually
conn.disconnect();
```

**Note:**

- Call `subscribe_event()` only **after** `ServiceManager::start()`.
- `items` must match the event schema; omit if the event has no items.
- There is no hard limit on subscriptions; all callbacks for an event run in subscription order.
- See `esp_brookesia::service::helper::Base::subscribe_event()`.

**Example** `GeneralEventHappened` —interface summary and sample code:

- Event: `GeneralEventHappened`
- **Items:**
  - Event: `String` (deserialize to `WifiHelper::GeneralEvent`)
  - `IsUnexpected`: `Boolean`

```
auto on_general_event_happened_handler = [] (const std::string &event_name, const_
↳std::string &event,
                                         bool is_unexpected) {
    // Event fired, parse items
    WifiHelper::GeneralEvent general_event;
    auto parse_result = BROOKESIA_DESCRIBE_STR_TO_ENUM(event, general_event);
    if (!parse_result) {
        // Parse failed, log error
        BROOKESIA_LOGE("Failed to parse general event: %1%", event);
        return;
    }
    // Parse succeeded, log result
    BROOKESIA_LOGI("General event: %1%", general_event);
};
// Unsubscribe automatically when the connection is destroyed (RAII)
auto general_event_happened_connection = WifiHelper::subscribe_event(
    WifiHelper::EventId::GeneralEventHappened, on_general_event_happened_handler);
if (!general_event_happened_connection.connected()) {
    // Subscribe failed
    return;
}

// Trigger Start action
WifiHelper::call_function_async(WifiHelper::FunctionId::TriggerGeneralAction,
    BROOKESIA_DESCRIBE_TO_
↳STR(WifiHelper::GeneralAction::Start));

// Wait for the event (example uses sleep; use a proper sync primitive in_
↳production)
```

(continues on next page)

(continued from previous page)

```
boost::this_thread::sleep_for(boost::chrono::seconds(5));
```

ScanApInfosUpdated —interface summary and sample code:

- **Event:** ScanApInfosUpdated
- **Items:**
  - ApInfos: Array (deserialize to `std::vector<WifiHelper::ScanApInfo>`)

```
auto on_scan_ap_infos_updated_handler = [] (const std::string &event_name, const_
↳boost::json::array &ap_infos) {
    // Event fired, parse items
    std::vector<WifiHelper::ScanApInfo> scanned_aps;
    auto parse_result = BROOKESIA_DESCRIBE_FROM_JSON(ap_infos, scanned_aps);
    if (!parse_result) {
        // Parse failed, log error
        BROOKESIA_LOGE("Failed to parse scan AP infos: %1%", ap_infos);
        return;
    }
    // Parse succeeded, log result
    BROOKESIA_LOGI("Scanned APs: %1%", scanned_aps);
};
auto scan_ap_infos_updated_connection =
    WifiHelper::subscribe_event(WifiHelper::EventId::ScanApInfosUpdated, on_scan_ap_
↳infos_updated_handler);
if (!scan_ap_infos_updated_connection.connected()) {
    // Subscribe failed
    return;
}

WifiHelper::call_function_async(WifiHelper::FunctionId::TriggerScanStart);

boost::this_thread::sleep_for(boost::chrono::seconds(10));
```

**Event Monitor** EventMonitor adds **blocking wait** on top of **asynchronous events**: after triggering an API, you can wait for a matching event or any occurrence to drive control flow.

```
// Create and start the monitor
WifiHelper::EventMonitor<WifiHelper::EventId::EventName> event_monitor;
event_monitor.start();

// Call the API that raises this event (omitted)

// Wait for any occurrence
auto got_any_event = event_monitor.wait_for_any(timeout_ms);
if (!got_any_event) {
    // Timeout, event did not fire
    return;
}

// Or wait for an event whose items match
auto got_event = event_monitor.wait_for(std::vector<service::EventItem>{items...},
↳timeout_ms);
if (!got_event) {
    // Timeout, items did not match
    return;
}

event_monitor.stop();
```

**Note:**

- items must match the event definition in type, count, and order.
- See `esp_brookesia::service::helper::EventManager`.

**Example: Wait for Specific Event Items** `GeneralEventHappened` —monitor usage:

- Event: `GeneralEventHappened`
- Items:
  - Event: `String` (serialize from `WifiHelper::GeneralEvent`)
  - `IsUnexpected`: `Boolean`

```
WifiHelper::EventManager<WifiHelper::EventId::GeneralEventHappened> general_event_
↳monitor;
if (!general_event_monitor.start()) {
    // Failed to start general event monitor
    return;
}

WifiHelper::call_function_async(WifiHelper::FunctionId::TriggerGeneralAction,
                              BROOKESIA_DESCRIBE_TO_
↳STR(WifiHelper::GeneralAction::Start));

auto got_event = general_event_monitor.wait_for(
    std::vector<service::EventItem>{BROOKESIA_DESCRIBE_TO_
↳STR(WifiHelper::GeneralEvent::Started), false}, 5000);
if (!got_event) {
    // Timeout, event did not fire
    return;
}
```

**Example: Latest Received Event Items** `ScanApInfosUpdated` —monitor usage:

- Event: `ScanApInfosUpdated`
- Items:
  - `ApInfos`: `Array` (deserialize to `std::vector<WifiHelper::ScanApInfo>`)

```
WifiHelper::EventManager<WifiHelper::EventId::ScanApInfosUpdated> scan_ap_infos_
↳updated_monitor;
if (!scan_ap_infos_updated_monitor.start()) {
    // Failed to start scan AP infos updated monitor
    return;
}

WifiHelper::call_function_async(WifiHelper::FunctionId::TriggerScanStart);

auto got_event = scan_ap_infos_updated_monitor.wait_for_any(10000);
if (!got_event) {
    // Timeout, event did not fire
    return;
}

auto last_items = scan_ap_infos_updated_monitor.get_last<boost::json::array>();
if (!last_items.has_value()) {
    return;
}

const auto &ap_infos = std::get<0>(last_items.value());
std::vector<WifiHelper::ScanApInfo> scanned_aps;
auto parse_result = BROOKESIA_DESCRIBE_FROM_JSON(ap_infos, scanned_aps);
if (!parse_result) {
    BROOKESIA_LOGE("Failed to parse scan AP infos: %1%", ap_infos);
}
```

(continues on next page)

(continued from previous page)

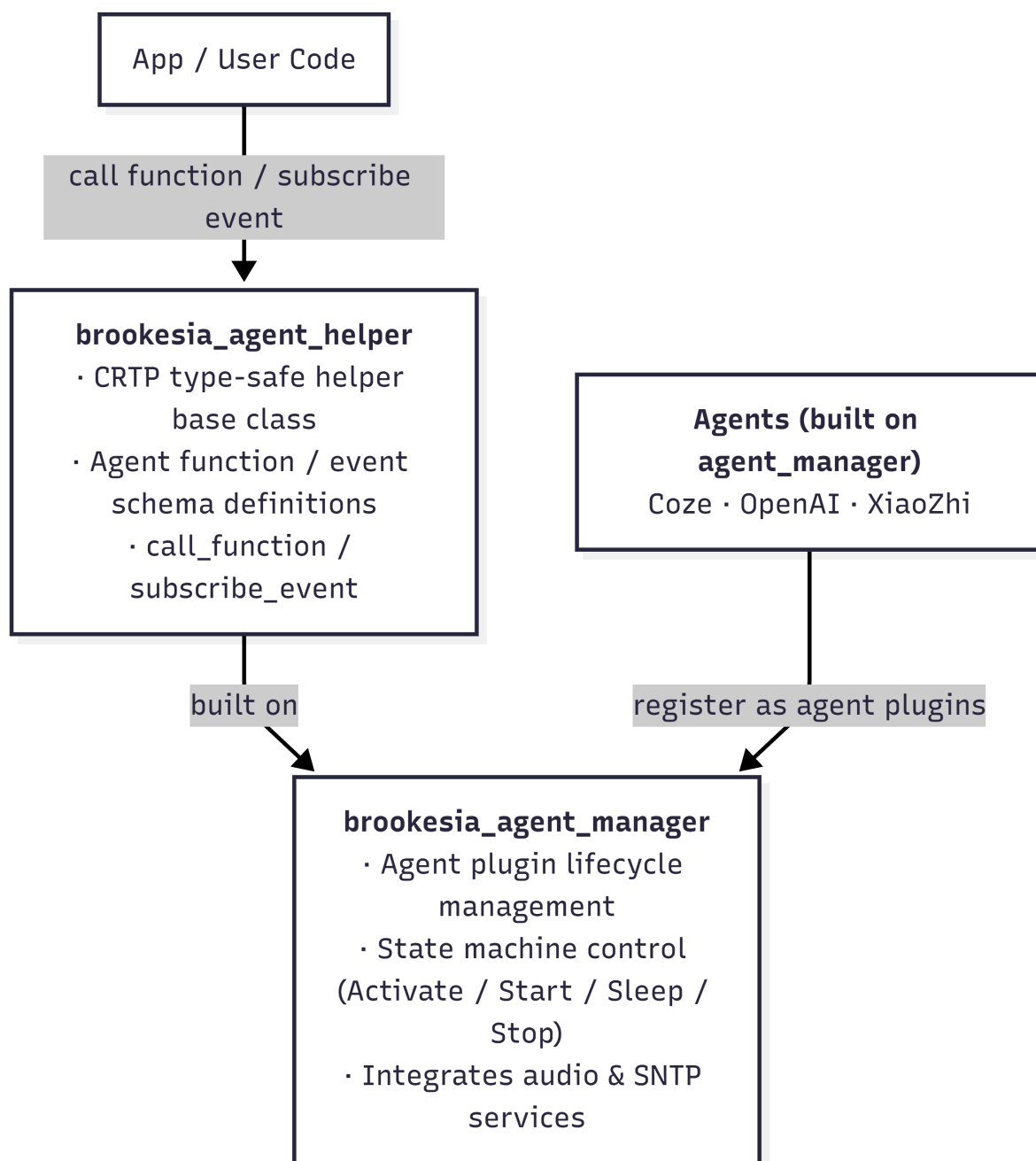
```
    return;  
}  
BROOKESIA_LOGI("Scanned APs: %1%", scanned_aps);
```



## **Chapter 5**

# **AI Agent Components**

This section documents ESP-Brookesia agent framework components. The framework consists of an agent framework layer and a concrete-agents layer. Their component hierarchy is shown below:



- **brookesia\_agent\_manager**: The agent framework core, responsible for plugin registration, state machine lifecycle control, and integration of audio and time-sync services.
- **brookesia\_agent\_helper**: A CRTP-based type-safe helper layer that simplifies agent function/event definition and invocation, built on top of `service_helper`.
- **Agents**: Concrete AI platform integrations implemented on top of `agent_manager`, registered into the framework and managed uniformly by upper layers.

## Chapter 6

# AI Agent Framework

### 6.1 Agent Manager

- Component registry: [espressif/brookesia\\_agent\\_manager](#)
- Helper header: `#include "brookesia/agent_helper/manager.hpp"`
- Helper class: `esp_brookesia::agent::helper::Manager`

#### 6.1.1 Overview

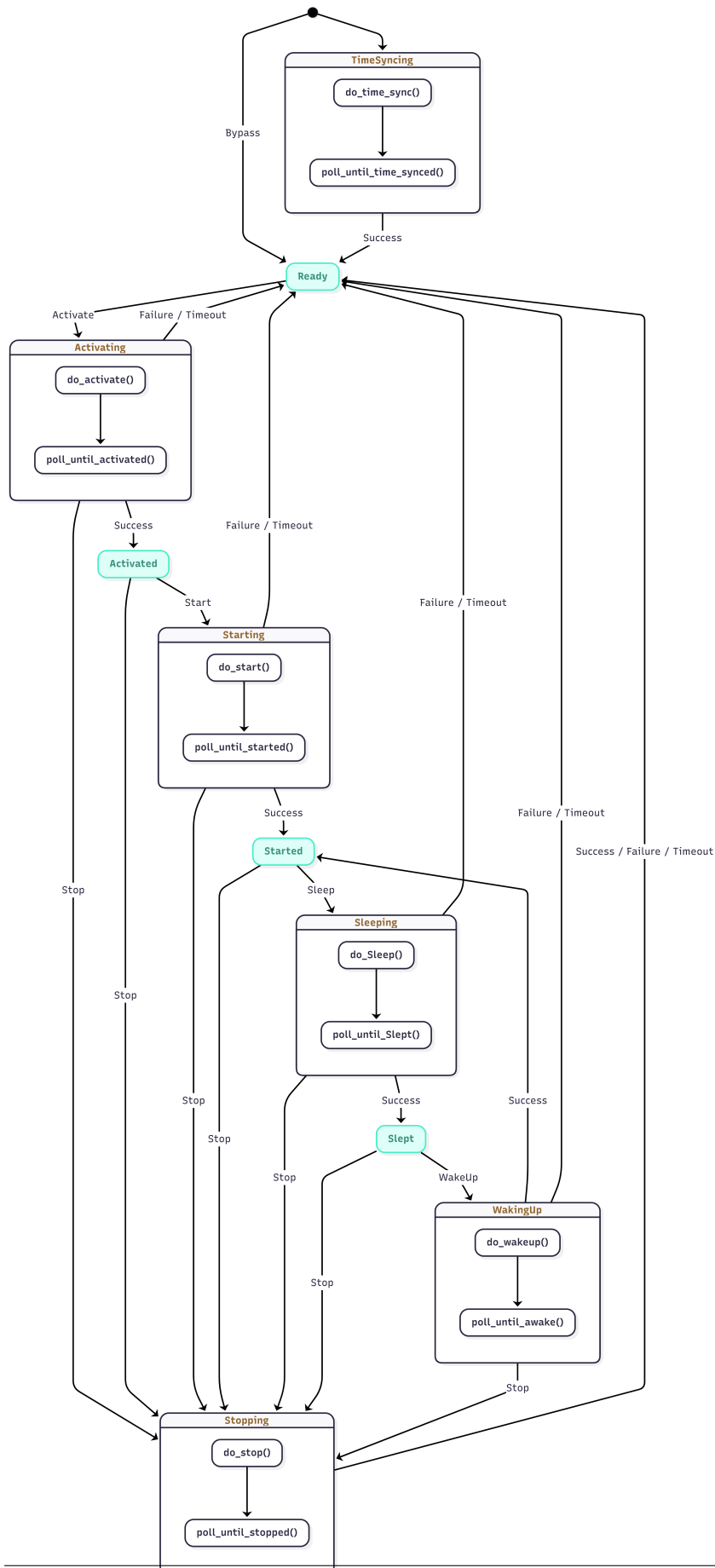
*brookesia\_agent\_manager* is the core agent framework:

- **Lifecycle:** Plugin-based init, activate, start, stop, sleep, wake, and dynamic switching.
- **State machine:** Correct, consistent transitions.
- **Control:** Pause/resume, interrupt, status queries.
- **Dialog modes:** RealTime and Manual (manual listen start/stop).
- **Events:** Operations, state, speech/listen, text, emotes—decoupled from apps.
- **Extensions:** Function calling, text, interrupt, emotes via agent properties.
- **AFE (optional):** Wake begin/end handling.
- **Services:** Audio and SNTP integration.
- **Persistence:** Optional *brookesia\_service\_nvs* for active agent and mode.

#### 6.1.2 State Machine Architecture

The manager uses a state machine for agent lifecycle.

## State Diagram



## State Descriptions

State	Type	Description
<b>TimeSyncing</b>	Transient	Waiting for time sync.
<b>Ready</b>	Stable	Time OK (or skipped); waiting for activate.
<b>Activating</b>	Transient	Activating agent.
<b>Activated</b>	Stable	Activated; waiting for start.
<b>Starting</b>	Transient	Starting agent.
<b>Started</b>	Stable	Running; audio in/out enabled.
<b>Sleeping</b>	Transient	Entering sleep.
<b>Slept</b>	Stable	Asleep; can wake or stop.
<b>WakingUp</b>	Transient	Waking from sleep.
<b>Stopping</b>	Transient	Stopping agent.

### 6.1.3 Standard Include / Helper Class

- Standard include: `#include \"brookesia/agent_helper/manager.hpp\"`
- Helper class: `esp_brookesia::agent::helper::Manager`

### 6.1.4 Service Interfaces

#### Functions

##### SetAgentInfo

**Description** Set agent info.

#### Execution

- Requires scheduler: Not required

#### Parameters

- Name
  - **Type:** String
  - **Required:** required
  - **Description:** Agent name.
- Info
  - **Type:** Object
  - **Required:** required
  - **Description:** Agent info as a JSON object. Example: `{"api_key":"api_key_value","model":"model_name"}`

#### Schema JSON

#### CLI Command

```
svc_call AgentManager SetAgentInfo {"Name":null,"Info":null}
```

#### SetChatMode

**Description** Set chat mode.

### Execution

- Requires scheduler: Required

### Parameters

- Mode
  - **Type:** String
  - **Required:** required
  - **Description:** Chat mode. Allowed values: [Manual, RealTime]

### Schema JSON

#### CLI Command

```
svc_call AgentManager SetChatMode {"Mode":null}
```

### SetTargetAgent

**Description** Set target agent. The agent will be activated by triggering *Activate* action.

### Execution

- Requires scheduler: Required

### Parameters

- Name
  - **Type:** String
  - **Required:** required
  - **Description:** Agent name to set as target.

### Schema JSON

#### CLI Command

```
svc_call AgentManager SetTargetAgent {"Name":null}
```

### GetTargetAgent

**Description** Get target agent name. Return type: string. Example: "AgentCoze"

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call AgentManager GetTargetAgent
```

### GetAgentNames

**Description** Get agent names. Return type: JSON array<string>. Example: ["Agent1", "Agent2"]

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call AgentManager GetAgentNames
```

### GetAgentAttributes

**Description** Get agent attributes. Return type: JSON array<object>. Example: [{"name": "Agent", "operation\_timeout": {"activate": 1000, "start": 1000, "sleep": 1000, "wake\_up": 1000, "stop": 1000}, "support\_general\_fu

### Execution

- Requires scheduler: Not required

### Parameters

- Name
  - **Type:** String
  - **Required:** optional
  - **Default:** ""
  - **Description:** Agent name (optional). Returns all agents when omitted.

### Schema JSON

### CLI Command

```
svc_call AgentManager GetAgentAttributes {"Name":null}
```

### GetChatMode

**Description** Get chat mode. Return type: string. Allowed values: [Manual, RealTime]. Example: "Manual"

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call AgentManager GetChatMode
```

### GetActiveAgent

**Description** Get active agent name. Return type: string. Example: "AgentCoze"

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call AgentManager GetActiveAgent
```

### TriggerGeneralAction

**Description** Trigger a general action.

### Execution

- Requires scheduler: Required

### Parameters

- Action
  - **Type:** String
  - **Required:** required
  - **Description:** General action. Allowed values: [TimeSync, Activate, Start, Stop, Sleep, WakeUp]

### Schema JSON

### CLI Command

```
svc_call AgentManager TriggerGeneralAction {"Action":null}
```

### Suspend

**Description** Suspend the agent.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call AgentManager Suspend
```

### Resume

**Description** Resume the agent.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call AgentManager Resume
```

### InterruptSpeaking

**Description** Interrupt speaking; the agent stops and keeps listening.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call AgentManager InterruptSpeaking
```

### ManualStartListening

**Description** Manually start listening. Only in *Manual* mode.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call AgentManager ManualStartListening
```

### ManualStopListening

**Description** Manually stop listening. Only in *Manual* mode.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call AgentManager ManualStopListening
```

### GetGeneralState

**Description** Get general state. Return type: string. Allowed values: [TimeSyncing, Ready, Activating, Activated, Starting, Stopping, Started, Sleeping, WakingUp, Slept]. Example: "Started"

#### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call AgentManager GetGeneralState
```

### GetSuspendStatus

**Description** Get suspend status. Return type: boolean. Example: false

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call AgentManager GetSuspendStatus
```

### GetSpeakingStatus

**Description** Get speaking status. Return type: boolean. Example: true

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call AgentManager GetSpeakingStatus
```

### GetListeningStatus

**Description** Get listening status. Return type: boolean. Example: true

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call AgentManager GetListeningStatus
```

### ResetData

**Description** Reset manager and agent data.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call AgentManager ResetData
```

### Events

#### GeneralActionTriggered

**Description** Emitted when a general action is triggered.

### Execution

- Requires scheduler: Required

### Items

- Action
  - **Type:** String
  - **Description:** General action. Allowed values: [TimeSync, Activate, Start, Stop, Sleep, WakeUp]

### Schema JSON

### CLI Command

```
svc_subscribe AgentManager GeneralActionTriggered
```

### GeneralEventHappened

**Description** Emitted when a general event occurs.

### Execution

- Requires scheduler: Required

### Items

- Event
  - **Type:** String
  - **Description:** General event. Allowed values: [TimeSynced, Activated, Started, Stopped, Slept, Awake]
- IsUnexpected
  - **Type:** Boolean
  - **Description:** Whether the event was unexpected.

### Schema JSON

### CLI Command

```
svc_subscribe AgentManager GeneralEventHappened
```

### SuspendStatusChanged

**Description** Emitted when suspend status changes.

### Execution

- Requires scheduler: Required

### Items

- IsSuspended
  - **Type:** Boolean
  - **Description:** Whether the agent is suspended.

### Schema JSON

### CLI Command

```
svc_subscribe AgentManager SuspendStatusChanged
```

### SpeakingStatusChanged

**Description** Emitted when speaking status changes.

### Execution

- Requires scheduler: Required

### Items

- IsSpeaking
  - **Type:** Boolean
  - **Description:** Whether the agent is speaking.

### Schema JSON

#### CLI Command

```
svc_subscribe AgentManager SpeakingStatusChanged
```

### ListeningStatusChanged

**Description** Emitted when listening status changes.

### Execution

- Requires scheduler: Required

### Items

- IsListening
  - **Type:** Boolean
  - **Description:** Whether the agent is listening.

### Schema JSON

#### CLI Command

```
svc_subscribe AgentManager ListeningStatusChanged
```

### AgentSpeakingTextGot

**Description** Emitted when the agent speaks text.

### Execution

- Requires scheduler: Required

### Items

- Text
  - **Type:** String
  - **Description:** Spoken text.

### Schema JSON

### CLI Command

```
svc_subscribe AgentManager AgentSpeakingTextGot
```

### UserSpeakingTextGot

**Description** Emitted when the user speaks text.

### Execution

- Requires scheduler: Required

### Items

- Text
  - **Type:** String
  - **Description:** User text.

### Schema JSON

### CLI Command

```
svc_subscribe AgentManager UserSpeakingTextGot
```

### EmoteGot

**Description** Emitted when the agent shows an emote.

### Execution

- Requires scheduler: Required

### Items

- Emote
  - **Type:** String
  - **Description:** Emote name.

### Schema JSON

### CLI Command

```
svc_subscribe AgentManager EmoteGot
```

## 6.1.5 Related Types and Configs

For structs, enums, helper methods, and the full Doxygen-rendered API surface, see [Raw helper API reference](#).

## 6.1.6 API Reference

### Agent Base Class

Public header: `#include "brookesia/agent_manager/base.hpp"`

#### Header File

- [agent/brookesia\\_agent\\_manager/include/brookesia/agent\\_manager/base.hpp](#)

#### Classes

class **Base** : public esp\_brookesia::service::ServiceBase

Common base class for all agent implementations.

The class extends `service::ServiceBase` with agent lifecycle hooks, audio pipeline coordination, and shared state tracked by `Manager` and `StateMachine`.

Subclassed by `esp_brookesia::agent::Coze`, `esp_brookesia::agent::Openai`, `esp_brookesia::agent::XiaoZhi`

#### Public Functions

inline const AgentAttributes &**get\_attributes** () const

Get immutable metadata describing the agent.

**Returns** const AgentAttributes& Agent attributes passed at construction time.

inline const AudioConfig &**get\_audio\_config** () const

Get the audio configuration used by the agent.

**Returns** const AudioConfig& Current encoder and decoder configuration.

### Agent Manager

Public header: `#include "brookesia/agent_manager/manager.hpp"`

#### Header File

- [agent/brookesia\\_agent\\_manager/include/brookesia/agent\\_manager/manager.hpp](#)

#### Classes

class **Manager** : public esp\_brookesia::service::ServiceBase

Service responsible for managing agent selection and shared agent state.

#### Public Types

enum class **DataType**

Persistent keys managed by the agent manager.

*Values:*

enumerator **TargetAgent**

enumerator **ChatMode**

enumerator **Max**

### Public Static Functions

static inline *Manager* &get\_instance ()

Get the global agent-manager singleton.

**Returns** *Manager*& Singleton instance.

## 6.2 AI Agent Helper

- Component registry: [espressif/brookesia\\_agent\\_helper](#)
- Public header: `#include "brookesia/agent_helper.hpp"`

### 6.2.1 Overview

*brookesia\_agent\_helper* uses C++20 concepts and CRTP to provide type-safe schemas and a unified calling surface for agents.

### 6.2.2 Features

- **Type-safe definitions:** Strong enums and structs with compile-time checks.
- **Schemas:** Standard function and event metadata (names, parameters, descriptions).
- **Calls:** Type-safe sync/async calls with conversion and error handling.
- **Events:** Type-safe subscriptions and callbacks.

### Module Overview

#### Manager Helper

- Public header: `#include "brookesia/agent_helper/manager.hpp"`

**Overview** Raw Doxygen API for the Manager helper: types, enums, methods, and macros.

### API Reference

#### Header File

- [agent/brookesia\\_agent\\_helper/include/brookesia/agent\\_helper/manager.hpp](#)

#### Classes

class **Manager** : public esp\_brookesia::service::helper::Base<Manager>

Helper schema definitions for the agent-manager service.

## Public Types

enum class **AgentGeneralFunction** : uint8\_t

Optional per-agent functions surfaced by the manager.

*Values:*

enumerator **InterruptSpeaking**

enumerator **Max**

enum class **AgentGeneralEvent** : uint8\_t

Optional per-agent events surfaced by the manager.

*Values:*

enumerator **SpeakingStatusChanged**

enumerator **ListeningStatusChanged**

enumerator **AgentSpeakingTextGot**

enumerator **UserSpeakingTextGot**

enumerator **EmoteGot**

enumerator **Max**

enum class **ChatMode** : uint8\_t

Conversation mode used by the active agent.

*Values:*

enumerator **RealTime**

Real-time conversation mode. Agent can listen while speaking.

enumerator **Manual**

Manual conversation mode. User can start and stop listening manually.

enumerator **HalfDuplex**

Half-duplex conversation mode. Agent can only speak or only listen.

enumerator **Max**

enum class **GeneralAction** : uint8\_t

High-level actions accepted by the agent manager.

*Values:*

enumerator **TimeSync**

enumerator **Activate**

enumerator **Start**

enumerator **Sleep**

enumerator **WakeUp**

enumerator **Stop**

enumerator **Max**

enum class **GeneralEvent** : uint8\_t

High-level completion events emitted by managed agents.

*Values:*

enumerator **TimeSynced**

enumerator **Activated**

enumerator **Started**

enumerator **Slept**

enumerator **Awake**

enumerator **Stopped**

enumerator **Max**

enum class **GeneralState** : uint8\_t

State-machine states used by the agent manager.

*Values:*

enumerator **TimeSyncing**

enumerator **Ready**

enumerator **Activating**

enumerator **Activated**

enumerator **Starting**

enumerator **Started**

enumerator **Sleeping**

enumerator **Slept**

enumerator **WakingUp**

enumerator **Stopping**

enumerator **Max**

### Public Static Functions

static inline constexpr std::string\_view **get\_name** ()

Name of the agent-manager service.

**Returns** std::string\_view Stable service name.

struct **AgentAttributes**

Public metadata describing one agent implementation.

### Public Functions

inline std::string **get\_name** () const

Get the agent name stored in the attributes.

**Returns** std::string Agent name.

inline bool **is\_general\_functions\_supported** (*AgentGeneralFunction* function) const

Check whether an optional general function is supported.

**Parameters** **function** -- **[in]** Function to check.

**Returns** true if the function is listed in `support_general_functions`.

inline bool **is\_general\_events\_supported** (*AgentGeneralEvent* event) const

Check whether an optional general event is supported.

**Parameters** **event** -- **[in]** Event to check.

**Returns** true if the event is listed in `support_general_events`.

### Public Members

std::string **name**

Agent name exposed to the manager.

*AgentOperationTimeout* **operation\_timeout** = {}

Timeout configuration for lifecycle actions.

std::vector<*AgentGeneralFunction*> **support\_general\_functions** = {}

Optional functions supported by the agent.

std::vector<*AgentGeneralEvent*> **support\_general\_events** = {}

Optional events emitted by the agent.

bool **require\_time\_sync** = false

Whether activation requires SNTP synchronization first.

struct **AgentOperationTimeout**

Timeout budget for the major lifecycle operations of an agent.

### Public Members

uint32\_t **activate** = 1000

Timeout for activation.

uint32\_t **start** = 1000

Timeout for startup.

uint32\_t **sleep** = 1000

Timeout for sleep.

uint32\_t **wake\_up** = 1000

Timeout for wake-up.

uint32\_t **stop** = 1000

Timeout for shutdown.

### Coze Helper

- Public header: `#include "brookesia/agent_helper/coze.hpp"`

**Overview** Raw Doxygen API for the Coze helper: types, enums, methods, and macros.

### API Reference

#### Header File

- [agent/brookesia\\_agent\\_helper/include/brookesia/agent\\_helper/coze.hpp](#)

#### Classes

class **Coze** : public esp\_brookesia::service::helper::Base<Coze>

Helper schema definitions for the *Coze* agent service.

#### Public Types

enum class **CozeEvent**

Coze-specific events surfaced by the agent.

*Values:*

enumerator **InsufficientCreditsBalance**

enumerator **Max**

### Public Static Functions

static inline constexpr std::string\_view **get\_name** ()

Name of the *Coze* agent service.

**Returns** std::string\_view Stable service name.

static inline std::span<const service::FunctionSchema> **get\_function\_schemas** ()

Get the function schemas exported by the *Coze* agent.

**Returns** std::span<const service::FunctionSchema> Static schema span.

static inline std::span<const service::EventSchema> **get\_event\_schemas** ()

Get the event schemas exported by the *Coze* agent.

**Returns** std::span<const service::EventSchema> Static schema span.

struct **AuthInfo**

Credentials required to authenticate with the *Coze* backend.

### Public Members

std::string **session\_name**

Session name used by the *Coze* SDK.

std::string **device\_id**

Unique device identifier.

std::string **custom\_consumer**

Consumer identifier passed to the backend.

std::string **app\_id**

Application id.

std::string **user\_id**

End-user id.

std::string **public\_key**

Public key used for authentication.

std::string **private\_key**

Private key used for authentication.

struct **Info**

Persistent *Coze* agent configuration.

### Public Members

#### *AuthInfo* authorization

Authentication material.

std::vector<*RobotInfo*> **robots**

Available robot definitions.

struct **RobotInfo**

Metadata for one available *Coze* robot.

### Public Members

std::string **name**

Human-readable robot name.

std::string **bot\_id**

*Coze* bot identifier.

std::string **voice\_id**

Voice profile identifier.

std::string **description**

Optional robot description.

### OpenAI Helper

- Public header: `#include "brookesia/agent_helper/openai.hpp"`

**Overview** Raw Doxygen API for the OpenAI helper: types, enums, methods, and macros.

### API Reference

#### Header File

- `agent/brookesia_agent_helper/include/brookesia/agent_helper/openai.hpp`

#### Classes

class **Openai** : public esp\_brookesia::service::helper::Base<*Openai*>

Helper schema definitions for the OpenAI agent service.

#### Public Static Functions

static inline constexpr std::string\_view **get\_name** ()

Name of the OpenAI agent service.

**Returns** std::string\_view Stable service name.

```
static inline std::span<const service::FunctionSchema> get_function_schemas ()
```

Get the function schemas exported by the OpenAI agent.

**Returns** std::span<const service::FunctionSchema> Static schema span.

```
static inline std::span<const service::EventSchema> get_event_schemas ()
```

Get the event schemas exported by the OpenAI agent.

**Returns** std::span<const service::EventSchema> Static schema span.

```
struct Info
```

Persistent configuration for the OpenAI agent backend.

### Public Members

```
std::string model
```

Model identifier used for requests.

```
std::string api_key
```

API key used for authentication.

```
std::string voice
```

Voice identifier used for requests. Default is "alloy".

### XiaoZhi Helper

- Public header: #include "brookesia/agent\_helper/xiaozhi.hpp"

**Overview** Raw Doxygen API for the XiaoZhi helper: types, enums, methods, and macros.

### API Reference

#### Header File

- agent/brookesia\_agent\_helper/include/brookesia/agent\_helper/xiaozhi.hpp

#### Classes

```
class XiaoZhi : public esp_brookesia::service::helper::Base<XiaoZhi>
```

Helper schema definitions for the *XiaoZhi* agent service.

#### Public Static Functions

```
static inline constexpr std::string_view get_name ()
```

Name of the *XiaoZhi* agent service.

**Returns** std::string\_view Stable service name.

```
static inline std::span<const service::FunctionSchema> get_function_schemas ()
```

Get the function schemas exported by the *XiaoZhi* agent.

**Returns** std::span<const service::FunctionSchema> Static schema span.

```
static inline std::span<const service::EventSchema> get_event_schemas ()
```

Get the event schemas exported by the *XiaoZhi* agent.

**Returns** `std::span<const service::EventSchema>` Static schema span.

## 6.3 Agents

### 6.3.1 Coze

- Component registry: [espressif/brookesia\\_agent\\_coze](#)
- Helper header: `#include "brookesia/agent_helper/coze.hpp"`
- Helper class: `esp_brookesia::agent::helper::Coze`

#### Overview

*brookesia\_agent\_coze* implements the Coze agent on top of ESP-Brookesia Agent Manager.

#### Features

- **Coze API:** WebSocket to Coze for real-time voice and text.
- **OAuth2:** Token acquisition and refresh for Coze.
- **Multi-bot:** Configure multiple bots and switch the active one.
- **Audio:** Built-in codecs; default G711A at 16 kHz.
- **Emotes:** Receive and display Coze expression events.
- **Platform events:** e.g. balance warnings.
- **Persistence:** Optional *brookesia\_service\_nvs* for auth and bot metadata.

#### Standard Include / Helper Class

- Standard include: `#include \"brookesia/agent_helper/coze.hpp\"`
- Helper class: `esp_brookesia::agent::helper::Coze`

#### Service Interfaces

#### Functions

##### **SetActiveRobotIndex**

**Description** Set active robot index.

#### Execution

- Requires scheduler: Not required

#### Parameters

- Index
  - **Type:** `Number`
  - **Required:** required
  - **Description:** Robot index to activate.

## Schema JSON

### CLI Command

```
svc_call AgentCoze SetActiveRobotIndex {"Index":null}
```

### GetActiveRobotIndex

**Description** Get active robot index. Return type: number. Example: 0

### Execution

- Requires scheduler: Not required

### Parameters

- No parameters.

## Schema JSON

### CLI Command

```
svc_call AgentCoze GetActiveRobotIndex
```

### GetRobotInfos

**Description** Get robot info list. Return type: JSON array<object>. Example: [{"name":"robot1","bot\_id":"bot\_id1","voice\_id":"voice\_id1","description":"description1"}, {"name":"robot2","bot\_id":"bot\_id2","voice\_id":"voice\_id2","description":"description2"}]

### Execution

- Requires scheduler: Not required

### Parameters

- No parameters.

## Schema JSON

### CLI Command

```
svc_call AgentCoze GetRobotInfos
```

## Events

### CozeEventHappened

**Description** Emitted when a Coze event occurs.

### Execution

- Requires scheduler: Required

### Items

- CozeEvent
  - **Type:** String
  - **Description:** Coze event. Allowed values: [InsufficientCreditsBalance]

### Schema JSON

### CLI Command

```
svc_subscribe AgentCoze CozeEventHappened
```

### Related Types and Configs

For structs, enums, helper methods, and the full Doxygen-rendered API surface, see [Raw helper API reference](#).

## 6.3.2 OpenAI

- Component registry: [espressif/brookesia\\_agent\\_openai](#)
- Helper header: `#include "brookesia/agent_helper/openai.hpp"`
- Helper class: `esp_brookesia::agent::helper::Openai`

### Overview

`brookesia_agent_openai` implements the OpenAI agent on ESP-Brookesia Agent Manager.

### Features

- **OpenAI Realtime:** WebRTC data channels for real-time voice and text.
- **P2P:** `esp_peer` for signaling and data channels.
- **Audio:** OPUS by default at 16 kHz, 24 kbps.
- **Data channel:** Audio, text, function calls, session control, etc.
- **Low latency:** Real-time duplex audio.
- **Persistence:** Optional `brookesia_service_nvs` for configuration.

### Standard Include / Helper Class

- Standard include: `#include \"brookesia/agent_helper/openai.hpp\"`
- Helper class: `esp_brookesia::agent::helper::Openai`

### Service Interfaces

**Functions** This contract does not expose standard function schemas.

**Events** This contract does not publish standard event schemas.

## Related Types and Configs

For structs, enums, helper methods, and the full Doxygen-rendered API surface, see [Raw helper API reference](#).

### 6.3.3 XiaoZhi

- Component registry: [espressif/brookesia\\_agent\\_xiaozhi](#)
- Helper header: `#include "brookesia/agent_helper/xiaozhi.hpp"`
- Helper class: `esp_brookesia::agent::helper::XiaoZhi`

#### Overview

`brookesia_agent_xiaozhi` integrates the XiaoZhi AI platform for ESP-Brookesia.

#### Features

- **Platform:** Built on `esp_xiaozhi` SDK.
- **Voice:** OPUS at 16 kHz, 24 kbps.
- **Events:** Speaking/listening, agent/user text, emotes, etc.
- **Manual listen:** Start/stop listening for Manual mode.
- **Barge-in:** Interrupt agent speech.
- **Lifecycle:** Unified management via `brookesia_agent_manager`.

#### Standard Include / Helper Class

- Standard include: `#include \"brookesia/agent_helper/xiaozhi.hpp\"`
- Helper class: `esp_brookesia::agent::helper::XiaoZhi`

#### Service Interfaces

#### Functions

##### **AddMCP\_ToolsWithServiceFunction**

**Description** Add MCP tools from service functions. Return type: JSON array<string> of added tool names. Example: `[\"Service.Audio.SetVolume\", \"Service.Audio.GetVolume\"]`

#### Execution

- Requires scheduler: Not required

#### Parameters

- `ServiceName`
  - **Type:** String
  - **Required:** required
  - **Description:** Service name.
- `FunctionNames`
  - **Type:** Array
  - **Required:** optional
  - **Default:** `[]`

- **Description:** Function names as JSON array<string>. Empty means all functions in the service. Example: ["SetVolume","GetVolume"]

### Schema JSON

#### CLI Command

```
svc_call AgentXiaoZhi AddMCP_ToolsWithServiceFunction {"ServiceName":null,  
↔"FunctionNames":null}
```

#### AddMCP\_ToolsWithCustomFunction

**Description** Add custom MCP tools. Return type: JSON array<string> of added tool names. Example: ["Custom.Display.GetBrightness","Custom.Display.SetBrightness"]

#### Execution

- Requires scheduler: Not required

#### Parameters

- Tools
  - **Type:** Array
  - **Required:** required
  - **Description:** Tools to add as JSON array<object>. Example: [{"description":"custom tool description 1","name":"Display.GetBrightness"},{"description":"custom tool description 2","name":"Display.SetBrightness"}]

### Schema JSON

#### CLI Command

```
svc_call AgentXiaoZhi AddMCP_ToolsWithCustomFunction {"Tools":null}
```

#### RemoveMCP\_Tools

**Description** Remove MCP tools.

#### Execution

- Requires scheduler: Not required

#### Parameters

- Tools
  - **Type:** Array
  - **Required:** required
  - **Description:** Tool names to remove. Example: ["Service.Audio.SetVolume","Custom.Display.GetBrightness"]

### Schema JSON

### CLI Command

```
svc_call AgentXiaoZhi RemoveMCP_Tools {"Tools":null}
```

### ExplainImage

**Description** Explain an image. Return type: string. Example: "This image contains a cup on a desk."

### Execution

- Requires scheduler: Required

### Parameters

- Image
  - **Type:** RawBuffer
  - **Required:** required
  - **Description:** Image data.
- Question
  - **Type:** String
  - **Required:** optional
  - **Default:** "What is in the image?"
  - **Description:** Question text.

### Schema JSON

#### CLI Command

```
svc_call AgentXiaoZhi ExplainImage {"Image":null,"Question":null}
```

### Events

#### ActivationCodeReceived

**Description** Emitted when an activation code is received.

### Execution

- Requires scheduler: Required

### Items

- Code
  - **Type:** String
  - **Description:** Activation code.

### Schema JSON

#### CLI Command

```
svc_subscribe AgentXiaoZhi ActivationCodeReceived
```

### **Related Types and Configs**

For structs, enums, helper methods, and the full Doxygen-rendered API surface, see [Raw helper API reference](#).



## Chapter 7

# AI Expression Components

This section documents ESP-Brookesia AI expression components.

## 7.1 Emote

- Component registry: [espressif/brookesia\\_expression\\_emote](#)
- Helper header: `#include "brookesia/service_helper/expression/emote.hpp"`
- Helper class: `esp_brookesia::service::helper::ExpressionEmote`

### 7.1.1 Overview

*brookesia\_expression\_emote* manages emotes on the Brookesia service framework:

- **Resources:** Load and configure emoji/animation assets.
- **Emotes:** Set and manage emoji for emotional states.
- **Animation:** Playback control; wait for frames; stop.
- **QR codes:** Show and hide QR overlays.
- **Event messages:** Show and hide event text.

### 7.1.2 Standard Include / Helper Class

- Standard include: `#include \"brookesia/service_helper/expression/emote.hpp\"`
- Helper class: `esp_brookesia::service::helper::ExpressionEmote`

### 7.1.3 Service Interfaces

#### Functions

#### SetConfig

**Description** Set emote config.

#### Execution

- Requires scheduler: Not required

### Parameters

- Config
  - **Type:** Object
  - **Required:** required
  - **Description:** Config. Example: {"h\_res":320,"v\_res":240,"buf\_pixels":7680,"fps":30,"task\_priority":5,"task\_stack":4096}

### Schema JSON

#### CLI Command

```
svc_call Emote SetConfig {"Config":null}
```

### LoadAssetsSource

**Description** Load assets from the specified source.

#### Execution

- Requires scheduler: Required

### Parameters

- Source
  - **Type:** Object
  - **Required:** required
  - **Description:** Asset source as a JSON object. Example: {"source":"anim\_icon","type":"PartitionLabel","flag\_enable\_mmap":true}

### Schema JSON

#### CLI Command

```
svc_call Emote LoadAssetsSource {"Source":null}
```

### SetEmoji

**Description** Set emoji and hide animation immediately.

#### Execution

- Requires scheduler: Required

### Parameters

- Emoji
  - **Type:** String
  - **Required:** required
  - **Description:** Emoji name.

### Schema JSON

### CLI Command

```
svc_call Emote SetEmoji {"Emoji":null}
```

### HideEmoji

**Description** Hide current emoji.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call Emote HideEmoji
```

### SetAnimation

**Description** Set animation and hide emoji immediately.

### Execution

- Requires scheduler: Required

### Parameters

- Animation
  - **Type:** String
  - **Required:** required
  - **Description:** Animation name.

### Schema JSON

### CLI Command

```
svc_call Emote SetAnimation {"Animation":null}
```

### InsertAnimation

**Description** Insert animation; it hides immediately and shows after the duration.

### Execution

- Requires scheduler: Required

### Parameters

- Animation
  - **Type:** String
  - **Required:** required
  - **Description:** Animation name.
- DurationMs
  - **Type:** Number
  - **Required:** required
  - **Description:** Animation duration in milliseconds. Stops automatically after this duration.

### Schema JSON

#### CLI Command

```
svc_call Emote InsertAnimation {"Animation":null,"DurationMs":null}
```

### StopAnimation

**Description** Stop current animation and hide it immediately.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Emote StopAnimation
```

### WaitAnimationFrameDone

**Description** Wait for each animation frame to finish.

#### Execution

- Requires scheduler: Required

#### Parameters

- TimeoutMs
  - **Type:** Number
  - **Required:** optional
  - **Default:** 0E0
  - **Description:** Timeout in milliseconds. 0 means wait forever.

### Schema JSON

### CLI Command

```
svc_call Emote WaitAnimationFrameDone {"TimeoutMs":null}
```

### SetEventMessage

**Description** Set message for a specified emote event.

### Execution

- Requires scheduler: Required

### Parameters

- Event
  - **Type:** String
  - **Required:** required
  - **Description:** Event type. Allowed values: [Idle, Speak, Listen, System, User, Battery]
- Message
  - **Type:** String
  - **Required:** optional
  - **Default:** " "
  - **Description:** Message text. For Battery event, the message format is "<charging\_status>,<percentage>", where <charging\_status> is 0 for not charging and 1 for charging, and <percentage> is the battery percentage in [0, 100].

### Schema JSON

### CLI Command

```
svc_call Emote SetEventMessage {"Event":null,"Message":null}
```

### HideEventMessage

**Description** Hide current event message.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

### CLI Command

```
svc_call Emote HideEventMessage
```

### SetQrcode

**Description** Set QR code and hide emoji and animation immediately.

#### Execution

- Requires scheduler: Required

#### Parameters

- Qrcode
  - **Type:** String
  - **Required:** required
  - **Description:** QR code content.

#### Schema JSON

#### CLI Command

```
svc_call Emote SetQrcode {"Qrcode":null}
```

#### HideQrcode

**Description** Hide current QR code and show emoji immediately.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

#### Schema JSON

#### CLI Command

```
svc_call Emote HideQrcode
```

#### NotifyFlushFinished

**Description** Notify emote flush finished.

#### Execution

- Requires scheduler: Required

#### Parameters

- No parameters.

#### Schema JSON

### CLI Command

```
svc_call Emote NotifyFlushFinished
```

### RefreshAll

**Description** Refresh the screen.

### Execution

- Requires scheduler: Required

### Parameters

- No parameters.

### Schema JSON

#### CLI Command

```
svc_call Emote RefreshAll
```

### Events

#### FlushReady

**Description** Emitted when emote flush is ready.

### Execution

- Requires scheduler: Not required

### Items

- Param
  - **Type:** Object
  - **Description:** Flush-ready parameter as a JSON object. Example:  
{"x\_start":0,"y\_start":0,"x\_end":100,"y\_end":100,"data":"@0x12345678"}

### Schema JSON

#### CLI Command

```
svc_subscribe Emote FlushReady
```

## 7.1.4 Related Types and Configs

For structs, enums, helper methods, and the full Doxygen-rendered API surface, see [Raw helper API reference](#).



# Index

## Symbols

- `_BROOKESIA_LOG_ARGS` (*C macro*), 30
- `_BROOKESIA_LOG_ARGS_FILE_LINE` (*C macro*), 30
- `_BROOKESIA_LOG_ARGS_FUNCTION` (*C macro*), 30
- `_BROOKESIA_LOG_ARGS_MESSAGE` (*C macro*), 30
- `_BROOKESIA_LOG_ARGS_THREAD_NAME` (*C macro*), 30
- `_BROOKESIA_LOG_CONCAT` (*C macro*), 31
- `_BROOKESIA_LOG_FORMAT_FILE_LINE` (*C macro*), 30
- `_BROOKESIA_LOG_FORMAT_FUNCTION` (*C macro*), 30
- `_BROOKESIA_LOG_FORMAT_MESSAGE` (*C macro*), 30
- `_BROOKESIA_LOG_FORMAT_STRING` (*C macro*), 30
- `_BROOKESIA_LOG_FORMAT_THREAD_NAME` (*C macro*), 30
- `_BROOKESIA_LOG_TRACE_ARGS_WITH_PTR` (*C macro*), 31
- `_BROOKESIA_LOG_TRACE_FORMAT_ENTER` (*C macro*), 31
- `_BROOKESIA_LOG_TRACE_FORMAT_ENTER_STRING` (*C macro*), 31
- `_BROOKESIA_LOG_TRACE_FORMAT_ENTER_WITH_PTR` (*C macro*), 31
- `_BROOKESIA_LOG_TRACE_FORMAT_ENTER_WITH_PTR_STRING` (*C macro*), 31
- `_BROOKESIA_LOG_TRACE_FORMAT_EXIT` (*C macro*), 31
- `_BROOKESIA_LOG_TRACE_FORMAT_EXIT_STRING` (*C macro*), 31
- `_BROOKESIA_LOG_TRACE_FORMAT_EXIT_WITH_PTR` (*C macro*), 31
- `_BROOKESIA_LOG_TRACE_FORMAT_EXIT_WITH_PTR_STRING` (*C macro*), 31
- `_BROOKESIA_PLUGIN_CONCAT` (*C macro*), 25
- `_BROOKESIA_THREAD_CONFIG_CONCAT` (*C macro*), 10
- `_BROOKESIA_TIME_PROFILER_CONCAT` (*C macro*), 55
- `BROOKESIA_CHECK_ESP_ERR_EXECUTE` (*C macro*), 32
- `BROOKESIA_CHECK_ESP_ERR_EXIT` (*C macro*), 33
- `BROOKESIA_CHECK_ESP_ERR_GOTO` (*C macro*), 33
- `BROOKESIA_CHECK_ESP_ERR_RETURN` (*C macro*), 33
- `BROOKESIA_CHECK_EXCEPTION_EXECUTE` (*C macro*), 32
- `BROOKESIA_CHECK_EXCEPTION_EXIT` (*C macro*), 34
- `BROOKESIA_CHECK_EXCEPTION_GOTO` (*C macro*), 34
- `BROOKESIA_CHECK_EXCEPTION_RETURN` (*C macro*), 33
- `BROOKESIA_CHECK_FALSE_EXECUTE` (*C macro*), 32
- `BROOKESIA_CHECK_FALSE_EXIT` (*C macro*), 33
- `BROOKESIA_CHECK_FALSE_GOTO` (*C macro*), 33
- `BROOKESIA_CHECK_FALSE_RETURN` (*C macro*), 33
- `BROOKESIA_CHECK_NULL_EXECUTE` (*C macro*), 32
- `BROOKESIA_CHECK_NULL_EXIT` (*C macro*), 32
- `BROOKESIA_CHECK_NULL_GOTO` (*C macro*), 32
- `BROOKESIA_CHECK_NULL_RETURN` (*C macro*), 32
- `BROOKESIA_CHECK_OUT_RANGE` (*C macro*), 34
- `BROOKESIA_CHECK_OUT_RANGE_EXECUTE` (*C macro*), 32
- `BROOKESIA_CHECK_OUT_RANGE_EXIT` (*C macro*), 34
- `BROOKESIA_CHECK_OUT_RANGE_GOTO` (*C macro*), 34
- `BROOKESIA_CHECK_OUT_RANGE_RETURN` (*C macro*), 34
- `BROOKESIA_DESCRIBE_ENUM` (*C macro*), 36
- `BROOKESIA_DESCRIBE_ENUM_TO_NUM` (*C macro*), 36
- `BROOKESIA_DESCRIBE_ENUM_TO_STR` (*C macro*), 36
- `BROOKESIA_DESCRIBE_FORMAT_COMPACT` (*C macro*), 37
- `BROOKESIA_DESCRIBE_FORMAT_CPP` (*C macro*), 37
- `BROOKESIA_DESCRIBE_FORMAT_DEFAULT` (*C macro*), 37
- `BROOKESIA_DESCRIBE_FORMAT_JSON` (*C macro*), 37

- BROOKESIA\_DESCRIBE\_FORMAT\_PYTHON (C macro), 37
- BROOKESIA\_DESCRIBE\_FORMAT\_VERBOSE (C macro), 37
- BROOKESIA\_DESCRIBE\_FROM\_JSON (C macro), 37
- BROOKESIA\_DESCRIBE\_GET\_GLOBAL\_FORMAT (C macro), 37
- BROOKESIA\_DESCRIBE\_JSON\_DESERIALIZE (C macro), 37
- BROOKESIA\_DESCRIBE\_JSON\_SERIALIZE (C macro), 37
- BROOKESIA\_DESCRIBE\_NUM\_TO\_ENUM (C macro), 36
- BROOKESIA\_DESCRIBE\_RESET\_GLOBAL\_FORMAT (C macro), 37
- BROOKESIA\_DESCRIBE\_SET\_GLOBAL\_FORMAT (C macro), 37
- BROOKESIA\_DESCRIBE\_STR\_TO\_ENUM (C macro), 37
- BROOKESIA\_DESCRIBE\_STR\_TO\_ENUM\_FLEXIBLE (C macro), 37
- BROOKESIA\_DESCRIBE\_STRUCT (C macro), 36
- BROOKESIA\_DESCRIBE\_TO\_JSON (C macro), 37
- BROOKESIA\_DESCRIBE\_TO\_STR (C macro), 37
- BROOKESIA\_DESCRIBE\_TO\_STR\_WITH\_FMT (C macro), 37
- BROOKESIA\_LOG\_CONCAT (C macro), 31
- BROOKESIA\_LOG\_DISABLE\_DEBUG\_TRACE (C macro), 30
- BROOKESIA\_LOG\_TRACE\_GUARD (C macro), 31
- BROOKESIA\_LOG\_TRACE\_GUARD\_WITH\_THIS (C macro), 31
- BROOKESIA\_LOGD (C macro), 30
- BROOKESIA\_LOGD\_IMPL (C macro), 30
- BROOKESIA\_LOGE (C macro), 30
- BROOKESIA\_LOGE\_IMPL (C macro), 30
- BROOKESIA\_LOGI (C macro), 30
- BROOKESIA\_LOGI\_IMPL (C macro), 30
- BROOKESIA\_LOGT (C macro), 30
- BROOKESIA\_LOGT\_IMPL (C macro), 30
- BROOKESIA\_LOGW (C macro), 30
- BROOKESIA\_LOGW\_IMPL (C macro), 30
- BROOKESIA\_PLUGIN\_CONCAT (C macro), 25
- BROOKESIA\_PLUGIN\_CREATE\_SYMBOL (C macro), 25
- BROOKESIA\_PLUGIN\_REGISTER (C macro), 26
- BROOKESIA\_PLUGIN\_REGISTER\_PRE\_MAIN\_FUNCTION (C macro), 25
- BROOKESIA\_PLUGIN\_REGISTER\_SINGLETON (C macro), 26
- BROOKESIA\_PLUGIN\_REGISTER\_SINGLETON\_WITH\_SYMBOL (C macro), 27
- BROOKESIA\_PLUGIN\_REGISTER\_WITH\_CONSTRUCTOR (C macro), 26
- BROOKESIA\_PLUGIN\_REGISTER\_WITH\_SYMBOL (C macro), 27
- BROOKESIA\_SERVICE\_FUNC\_HANDLER\_0 (C macro), 100
- BROOKESIA\_SERVICE\_FUNC\_HANDLER\_1 (C macro), 100
- BROOKESIA\_SERVICE\_FUNC\_HANDLER\_2 (C macro), 100
- BROOKESIA\_SERVICE\_FUNC\_HANDLER\_3 (C macro), 100
- BROOKESIA\_SERVICE\_HELPER\_FUNC\_HANDLER\_0 (C macro), 122
- BROOKESIA\_SERVICE\_HELPER\_FUNC\_HANDLER\_1 (C macro), 122
- BROOKESIA\_SERVICE\_HELPER\_FUNC\_HANDLER\_2 (C macro), 122
- BROOKESIA\_SERVICE\_HELPER\_FUNC\_HANDLER\_3 (C macro), 123
- BROOKESIA\_THREAD\_CONFIG\_CONCAT (C macro), 10
- BROOKESIA\_THREAD\_CONFIG\_GUARD (C macro), 10
- BROOKESIA\_THREAD\_GET\_APPLIED\_CONFIG (C macro), 11
- BROOKESIA\_THREAD\_GET\_CURRENT\_CONFIG (C macro), 10
- BROOKESIA\_TIME\_PROFILER\_CLEAR (C macro), 56
- BROOKESIA\_TIME\_PROFILER\_CONCAT (C macro), 55
- BROOKESIA\_TIME\_PROFILER\_END\_EVENT (C macro), 56
- BROOKESIA\_TIME\_PROFILER\_REPORT (C macro), 56
- BROOKESIA\_TIME\_PROFILER\_SCOPE (C macro), 55
- BROOKESIA\_TIME\_PROFILER\_START\_EVENT (C macro), 55
- ## E
- esp\_brookesia::agent::Base (C++ class), 228
- esp\_brookesia::agent::Base::get\_attributes (C++ function), 228
- esp\_brookesia::agent::Base::get\_audio\_config (C++ function), 228
- esp\_brookesia::agent::helper::Coze (C++ class), 233
- esp\_brookesia::agent::helper::Coze::AuthInfo (C++ struct), 234
- esp\_brookesia::agent::helper::Coze::AuthInfo::app (C++ member), 234
- esp\_brookesia::agent::helper::Coze::AuthInfo::cus (C++ member), 234
- esp\_brookesia::agent::helper::Coze::AuthInfo::dev (C++ member), 234
- esp\_brookesia::agent::helper::Coze::AuthInfo::pri (C++ member), 234
- esp\_brookesia::agent::helper::Coze::AuthInfo::pub (C++ member), 234

esp\_brookesia::agent::helper::Coze::Autesp\_brookesia::agent::helper::Manager::AgentGener  
 (C++ member), 234 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Coze::Autesp\_brookesia::agent::helper::Manager::AgentGener  
 (C++ member), 234 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Coze::Coze::Autesp\_brookesia::agent::helper::Manager::AgentGener  
 (C++ enum), 233 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Coze::Coze::Autesp\_brookesia::agent::helper::Manager::AgentGener  
 (C++ enumerator), 233 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Coze::Coze::Autesp\_brookesia::agent::helper::Manager::AgentGener  
 (C++ enumerator), 233 (C++ enum), 230  
 esp\_brookesia::agent::helper::Coze::getesp\_brookesia::agent::helper::Manager::AgentGener  
 (C++ function), 234 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Coze::getesp\_brookesia::agent::helper::Manager::AgentGener  
 (C++ function), 234 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Coze::getesp\_brookesia::agent::helper::Manager::AgentOpera  
 (C++ function), 234 (C++ struct), 233  
 esp\_brookesia::agent::helper::Coze::Infesp\_brookesia::agent::helper::Manager::AgentOpera  
 (C++ struct), 234 (C++ member), 233  
 esp\_brookesia::agent::helper::Coze::Infesp\_brookesia::agent::helper::Manager::AgentOpera  
 (C++ member), 235 (C++ member), 233  
 esp\_brookesia::agent::helper::Coze::Infesp\_brookesia::agent::helper::Manager::AgentOpera  
 (C++ member), 235 (C++ member), 233  
 esp\_brookesia::agent::helper::Coze::Robesp\_brookesia::agent::helper::Manager::AgentOpera  
 (C++ struct), 235 (C++ member), 233  
 esp\_brookesia::agent::helper::Coze::Robesp\_brookesia::agent::helper::Manager::AgentOpera  
 (C++ member), 235 (C++ member), 233  
 esp\_brookesia::agent::helper::Coze::Robesp\_brookesia::agent::helper::Manager::ChatMode  
 (C++ member), 235 (C++ enum), 230  
 esp\_brookesia::agent::helper::Coze::Robesp\_brookesia::agent::helper::Manager::ChatMode::  
 (C++ member), 235 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Coze::Robesp\_brookesia::agent::helper::Manager::ChatMode::  
 (C++ member), 235 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Manager esp\_brookesia::agent::helper::Manager::ChatMode::  
 (C++ class), 229 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::ChatMode::  
 (C++ struct), 232 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::GeneralAct  
 (C++ function), 232 (C++ enum), 230  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::GeneralAct  
 (C++ function), 232 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::GeneralAct  
 (C++ function), 232 (C++ enumerator), 231  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::GeneralAct  
 (C++ member), 232 (C++ enumerator), 231  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::GeneralAct  
 (C++ member), 232 (C++ enumerator), 231  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::GeneralAct  
 (C++ member), 232 (C++ enumerator), 231  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::GeneralAct  
 (C++ member), 232 (C++ enumerator), 230  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::GeneralAct  
 (C++ member), 232 (C++ enumerator), 231  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::GeneralEve  
 (C++ enum), 230 (C++ enum), 231  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::GeneralEve  
 (C++ enumerator), 230 (C++ enumerator), 231  
 esp\_brookesia::agent::helper::Manager: Aspes\_brookesia::agent::helper::Manager::GeneralEve  
 (C++ enumerator), 230 (C++ enumerator), 231

esp\_brookesia::agent::helper::Manager::GeneralKvesia::Mage::helper::XiaoZhi::get\_name  
 (C++ enumerator), 231 (C++ function), 236  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Sagept::Manager (C++  
 (C++ enumerator), 231 class), 228  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Sagept::Manager::DataType  
 (C++ enumerator), 231 (C++ enum), 228  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Sagept::Manager::DataType::ChatMode  
 (C++ enumerator), 231 (C++ enumerator), 229  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::TargetSynchManager::DataType::Max  
 (C++ enumerator), 231 (C++ enumerator), 229  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::agent::Manager::DataType::TargetAg  
 (C++ enum), 231 (C++ enumerator), 228  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Agent::Manager::get\_instance  
 (C++ enumerator), 231 (C++ function), 229  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::AudioAudioCodecPlayerIface  
 (C++ enumerator), 231 (C++ class), 64  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Max::AudioCodecPlayerIface::~~Audio  
 (C++ enumerator), 232 (C++ function), 64  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Ready::AudioCodecPlayerIface::~AudioC  
 (C++ enumerator), 231 (C++ function), 64  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Sleep::AudioCodecPlayerIface::~close  
 (C++ enumerator), 231 (C++ function), 64  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Sleep::AudioCodecPlayerIface::~Config  
 (C++ enumerator), 232 (C++ struct), 65  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Start::AudioCodecPlayerIface::~Config  
 (C++ enumerator), 231 (C++ member), 65  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Start::AudioCodecPlayerIface::~Config  
 (C++ enumerator), 231 (C++ member), 65  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Stopp::AudioCodecPlayerIface::~Config  
 (C++ enumerator), 232 (C++ member), 65  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::ThaneS::AudioCodecPlayerIface::~is\_pa  
 (C++ enumerator), 231 (C++ function), 65  
 esp\_brookesia::agent::helper::Manager::GeneralKvesia::Waking::AudioCodecPlayerIface::~is\_pa  
 (C++ enumerator), 232 (C++ function), 64  
 esp\_brookesia::agent::helper::Manager::esp\_brookesia::hal::AudioCodecPlayerIface::~NAME  
 (C++ function), 232 (C++ member), 65  
 esp\_brookesia::agent::helper::Openai esp\_brookesia::hal::AudioCodecPlayerIface::~open  
 (C++ class), 235 (C++ function), 64  
 esp\_brookesia::agent::helper::Openai::esp\_brookesia::hal::AudioCodecPlayerIface::~set\_pa  
 (C++ function), 236 (C++ function), 65  
 esp\_brookesia::agent::helper::Openai::esp\_brookesia::hal::AudioCodecPlayerIface::~set\_vo  
 (C++ function), 235 (C++ function), 64  
 esp\_brookesia::agent::helper::Openai::esp\_brookesia::hal::AudioCodecPlayerIface::~write  
 (C++ function), 235 (C++ function), 64  
 esp\_brookesia::agent::helper::Openai::Info esp\_brookesia::hal::AudioCodecRecorderIface  
 (C++ struct), 236 (C++ class), 65  
 esp\_brookesia::agent::helper::Openai::Info::map::kvesia::hal::AudioCodecRecorderIface::~~Aud  
 (C++ member), 236 (C++ function), 66  
 esp\_brookesia::agent::helper::Openai::Info::brookesia::hal::AudioCodecRecorderIface::~Audi  
 (C++ member), 236 (C++ function), 66  
 esp\_brookesia::agent::helper::Openai::Info::brookesia::hal::AudioCodecRecorderIface::~clos  
 (C++ member), 236 (C++ function), 66  
 esp\_brookesia::agent::helper::XiaoZhi esp\_brookesia::hal::AudioCodecRecorderIface::~get\_  
 (C++ class), 236 (C++ function), 66  
 esp\_brookesia::agent::helper::XiaoZhi::esp\_brookesia::hal::AudioCodecRecorderIface::~Info  
 (C++ function), 236 (C++ struct), 66  
 esp\_brookesia::agent::helper::XiaoZhi::esp\_brookesia::hal::AudioCodecRecorderIface::~Info  
 (C++ function), 236 (C++ member), 66

esp\_brookesia::hal::AudioCodecRecorderIface::(C++ function), 62  
     (C++ member), 67  
 esp\_brookesia::hal::AudioCodecRecorderIface::(C++ function), 62s  
     (C++ member), 66  
 esp\_brookesia::hal::AudioCodecRecorderIface::(C++ function), 61\_gain  
     (C++ member), 67  
 esp\_brookesia::hal::AudioCodecRecorderIface::(C++ function), 61\_out  
     (C++ member), 67  
 esp\_brookesia::hal::AudioCodecRecorderIface::(C++ function), 61\_rate  
     (C++ member), 66  
 esp\_brookesia::hal::AudioCodecRecorderIface::(C++ type), 63  
     (C++ member), 66  
 esp\_brookesia::hal::AudioCodecRecorderIface::(C++ class), 72  
     (C++ function), 66  
 esp\_brookesia::hal::AudioCodecRecorderIface::(C++ function), 72  
     (C++ function), 66  
 esp\_brookesia::hal::AudioCodecRecorderIface::(C++ function), 72s  
     (C++ function), 66  
 esp\_brookesia::hal::AudioCodecRecorderIface::(C++ function), 72s  
     (C++ function), 66  
 esp\_brookesia::hal::AudioDevice (C++ class), 83  
     (C++ function), 72  
 esp\_brookesia::hal::AudioDevice::CODEC\_PLAYER (C++ member), 72  
     (C++ member), 84  
 esp\_brookesia::hal::AudioDevice::CODEC\_RECORDER (C++ member), 72  
     (C++ member), 84  
 esp\_brookesia::hal::AudioDevice::DEVICE\_NAME (C++ function), 72  
     (C++ member), 84  
 esp\_brookesia::hal::AudioDevice::get\_instance (C++ function), 72  
     (C++ function), 83  
 esp\_brookesia::hal::AudioDevice::set\_codec\_rec (C++ function), 82  
     (C++ function), 83  
 esp\_brookesia::hal::BoardInfoIface (C++ class), 73  
     (C++ member), 83  
 esp\_brookesia::hal::BoardInfoIface::~BoardInfo (C++ function), 83  
     (C++ function), 73  
 esp\_brookesia::hal::BoardInfoIface::BoardInfo (C++ member), 83  
     (C++ function), 73  
 esp\_brookesia::hal::BoardInfoIface::get\_info (C++ member), 83  
     (C++ function), 73  
 esp\_brookesia::hal::BoardInfoIface::Info (C++ member), 83  
     (C++ struct), 73  
 esp\_brookesia::hal::BoardInfoIface::Info::chip (C++ class), 67  
     (C++ member), 74  
 esp\_brookesia::hal::BoardInfoIface::Info::des (C++ function), 68  
     (C++ member), 74  
 esp\_brookesia::hal::BoardInfoIface::Info::is\_v (C++ enum), 67  
     (C++ function), 73  
 esp\_brookesia::hal::BoardInfoIface::Info::man (C++ enumerator), 67  
     (C++ member), 74  
 esp\_brookesia::hal::BoardInfoIface::Info::name (C++ enumerator), 68  
     (C++ member), 74  
 esp\_brookesia::hal::BoardInfoIface::Info::ver (C++ enumerator), 68  
     (C++ member), 74  
 esp\_brookesia::hal::BoardInfoIface::NAME (C++ enumerator), 67  
     (C++ member), 73  
 esp\_brookesia::hal::Device (C++ class), 61  
     (C++ function), 68  
 esp\_brookesia::hal::Device::get\_interface  
     esp\_brookesia::hal::DisplayBacklightIface  
         (C++ function), 61\_gains  
         esp\_brookesia::hal::Device::get\_name  
         esp\_brookesia::hal::Device::probe  
         esp\_brookesia::hal::Device::register\_post\_deinit  
         esp\_brookesia::hal::Device::register\_pre\_init\_cal  
         esp\_brookesia::hal::DeviceRegistry  
         esp\_brookesia::hal::DisplayBacklightIface  
         esp\_brookesia::hal::DisplayBacklightIface::~Displ  
         esp\_brookesia::hal::DisplayBacklightIface::Displa  
         esp\_brookesia::hal::DisplayBacklightIface::get\_br  
         esp\_brookesia::hal::DisplayBacklightIface::is\_lig  
         esp\_brookesia::hal::DisplayBacklightIface::is\_lig  
         esp\_brookesia::hal::DisplayBacklightIface::NAME  
         esp\_brookesia::hal::DisplayBacklightIface::set\_br  
         esp\_brookesia::hal::DisplayBacklightIface::set\_li  
         esp\_brookesia::hal::DisplayDevice  
         esp\_brookesia::hal::DisplayDevice::DEVICE\_NAME  
         esp\_brookesia::hal::DisplayDevice::get\_instance  
         esp\_brookesia::hal::DisplayDevice::LCD\_PANEL\_IMPL  
         esp\_brookesia::hal::DisplayDevice::LCD\_TOUCH\_IMPL  
         esp\_brookesia::hal::DisplayDevice::LEDC\_BACKLIGHT  
         esp\_brookesia::hal::DisplayPanelIface  
         esp\_brookesia::hal::DisplayPanelIface::~DisplayPan  
         esp\_brookesia::hal::DisplayPanelIface::BusType  
         esp\_brookesia::hal::DisplayPanelIface::BusType::G  
         esp\_brookesia::hal::DisplayPanelIface::BusType::M  
         esp\_brookesia::hal::DisplayPanelIface::BusType::M  
         esp\_brookesia::hal::DisplayPanelIface::BusType::R  
         esp\_brookesia::hal::DisplayPanelIface::DisplayPan  
         esp\_brookesia::hal::DisplayPanelIface::draw\_bitma





- (C++ member), 84
- esp\_brookesia::hal::StorageDevice::get\_esp\_brookesia::lib\_utils::FunctionGuard::FunctionGuard (C++ function), 84
- esp\_brookesia::hal::StorageFsIface (C++ class), 79
- esp\_brookesia::hal::StorageFsIface::~~StorageFsIface (C++ function), 79
- esp\_brookesia::hal::StorageFsIface::Capabilities (C++ struct), 80
- esp\_brookesia::hal::StorageFsIface::Capabilities::free\_bytes (C++ member), 80
- esp\_brookesia::hal::StorageFsIface::Capabilities::total\_bytes (C++ member), 80
- esp\_brookesia::hal::StorageFsIface::Capabilities::used\_bytes (C++ member), 80
- esp\_brookesia::hal::StorageFsIface::FilesystemTypes (C++ enum), 79
- esp\_brookesia::hal::StorageFsIface::FilesystemTypes::FATFS (C++ enumerator), 79
- esp\_brookesia::hal::StorageFsIface::FilesystemTypes::LittleFS (C++ enumerator), 79
- esp\_brookesia::hal::StorageFsIface::FilesystemTypes::SPIFFS (C++ enumerator), 79
- esp\_brookesia::hal::StorageFsIface::get\_esp\_brookesia::lib\_utils::LogTraceGuard (C++ function), 80
- esp\_brookesia::hal::StorageFsIface::get\_esp\_brookesia::lib\_utils::MemoryProfiler (C++ function), 80
- esp\_brookesia::hal::StorageFsIface::Info (C++ struct), 80
- esp\_brookesia::hal::StorageFsIface::Info::fs\_type (C++ member), 80
- esp\_brookesia::hal::StorageFsIface::Info::media\_type (C++ member), 80
- esp\_brookesia::hal::StorageFsIface::Info::mount\_point (C++ member), 80
- esp\_brookesia::hal::StorageFsIface::Info::supported\_directories (C++ member), 80
- esp\_brookesia::hal::StorageFsIface::MediaType (C++ enum), 79
- esp\_brookesia::hal::StorageFsIface::MediaType::Flash (C++ enumerator), 79
- esp\_brookesia::hal::StorageFsIface::MediaType::SDCard (C++ enumerator), 79
- esp\_brookesia::hal::StorageFsIface::NAME (C++ member), 80
- esp\_brookesia::hal::StorageFsIface::StorageFsIface (C++ function), 79
- esp\_brookesia::lib\_utils::DescribeFormatManager (C++ class), 36
- esp\_brookesia::lib\_utils::DescribeFormatManager::format (C++ function), 36
- esp\_brookesia::lib\_utils::DescribeFormatManager::instance (C++ function), 36
- esp\_brookesia::lib\_utils::DescribeFormatManager::set\_to\_default (C++ function), 36
- esp\_brookesia::lib\_utils::DescribeFormatManager::set\_format (C++ function), 36
- esp\_brookesia::lib\_utils::FunctionGuard (C++ class), 35
- esp\_brookesia::lib\_utils::FunctionGuard::FunctionGuard (C++ function), 35
- esp\_brookesia::lib\_utils::FunctionGuard::release (C++ function), 35
- esp\_brookesia::lib\_utils::Log (C++ class), 28
- esp\_brookesia::lib\_utils::Log::extract\_file\_name (C++ function), 29
- esp\_brookesia::lib\_utils::Log::extract\_function\_name (C++ function), 29
- esp\_brookesia::lib\_utils::Log::format\_message (C++ function), 29
- esp\_brookesia::lib\_utils::Log::getInstance (C++ function), 29
- esp\_brookesia::lib\_utils::Log::print (C++ function), 28
- esp\_brookesia::lib\_utils::Log::write (C++ function), 29
- esp\_brookesia::lib\_utils::LittleFS::LogTraceGuard (C++ class), 29
- esp\_brookesia::lib\_utils::SPIFFS::LogTraceGuard::~~LogTraceGuard (C++ function), 29
- esp\_brookesia::lib\_utils::LogTraceGuard::LogTraceGuard (C++ function), 29
- esp\_brookesia::lib\_utils::MemoryProfiler (C++ class), 38
- esp\_brookesia::lib\_utils::MemoryProfiler::config (C++ function), 40
- esp\_brookesia::lib\_utils::MemoryProfiler::connect (C++ function), 41
- esp\_brookesia::lib\_utils::MemoryProfiler::connect (C++ function), 41
- esp\_brookesia::lib\_utils::MemoryProfiler::get\_instance (C++ function), 41
- esp\_brookesia::lib\_utils::MemoryProfiler::get\_profile (C++ function), 40
- esp\_brookesia::lib\_utils::MemoryProfiler::get\_profile (C++ function), 41
- esp\_brookesia::lib\_utils::MemoryProfiler::HeapInfo (C++ struct), 41
- esp\_brookesia::lib\_utils::MemoryProfiler::HeapInfo (C++ member), 41
- esp\_brookesia::lib\_utils::MemoryProfiler::HeapInfo (C++ member), 41
- esp\_brookesia::lib\_utils::MemoryProfiler::HeapInfo (C++ member), 41
- esp\_brookesia::lib\_utils::MemoryProfiler::HeapInfo (C++ member), 41
- esp\_brookesia::lib\_utils::MemoryProfiler::MemoryInfo (C++ struct), 42
- esp\_brookesia::lib\_utils::MemoryProfiler::MemoryInfo (C++ member), 42
- esp\_brookesia::lib\_utils::MemoryProfiler::MemoryInfo (C++ member), 42





























(C++ class), 111  
 esp\_brookesia::service::rpc::DataLinkClient::DataLinkClient::rpc::Server::start  
 (C++ function), 111  
 esp\_brookesia::service::rpc::DataLinkClient::DataLinkClient::rpc::Server::stop  
 (C++ function), 111  
 esp\_brookesia::service::rpc::DataLinkClient::DataLinkClient::rpc::ServerConnection  
 (C++ function), 111  
 (C++ class), 108  
 esp\_brookesia::service::rpc::DataLinkClient::DataLinkClient::rpc::ServerConnection::ac  
 (C++ function), 112  
 (C++ function), 109  
 esp\_brookesia::service::rpc::DataLinkClient::DataLinkClient::rpc::ServerConnection::No  
 (C++ function), 112  
 (C++ type), 109  
 esp\_brookesia::service::rpc::DataLinkClient::DataLinkClient::rpc::ServerConnection::on  
 (C++ function), 112  
 (C++ function), 109  
 esp\_brookesia::service::rpc::DataLinkService::DataLinkService::rpc::ServerConnection::on  
 (C++ class), 112  
 (C++ function), 109  
 esp\_brookesia::service::rpc::DataLinkService::DataLinkService::rpc::ServerConnection::pu  
 (C++ function), 112  
 (C++ function), 110  
 esp\_brookesia::service::rpc::DataLinkService::DataLinkService::rpc::ServerConnection::Re  
 (C++ function), 112  
 (C++ type), 109  
 esp\_brookesia::service::rpc::DataLinkService::DataLinkService::rpc::ServerConnection::re  
 (C++ function), 113  
 (C++ function), 110  
 esp\_brookesia::service::rpc::DataLinkService::DataLinkService::rpc::ServerConnection::Re  
 (C++ function), 113  
 (C++ type), 109  
 esp\_brookesia::service::rpc::DataLinkService::DataLinkService::rpc::ServerConnection::Se  
 (C++ function), 113  
 (C++ function), 109  
 esp\_brookesia::service::rpc::DataLinkService::DataLinkService::rpc::ServerConnection::se  
 (C++ function), 113  
 (C++ function), 109  
 esp\_brookesia::service::rpc::DataLinkService::DataLinkService::rpc::ServerConnection::se  
 (C++ function), 113  
 (C++ function), 109  
 esp\_brookesia::service::rpc::DataLinkService::DataLinkService::ServiceBase  
 (C++ function), 113  
 (C++ class), 96  
 esp\_brookesia::service::rpc::DataLinkService::DataLinkService::ServiceBase::~ServiceBase  
 (C++ function), 112  
 (C++ function), 96  
 esp\_brookesia::service::rpc::DataLinkService::DataLinkService::ServiceBase::Attributes  
 (C++ function), 112  
 (C++ struct), 99  
 esp\_brookesia::service::rpc::Server esp\_brookesia::service::ServiceBase::Attributes::  
 (C++ class), 115  
 (C++ member), 100  
 esp\_brookesia::service::rpc::Server::~Server esp\_brookesia::service::ServiceBase::Attributes::  
 (C++ function), 115  
 (C++ member), 100  
 esp\_brookesia::service::rpc::Server::add esp\_brookesia::service::ServiceBase::Attributes::  
 (C++ function), 115  
 (C++ function), 99  
 esp\_brookesia::service::rpc::Server::Config esp\_brookesia::service::ServiceBase::Attributes::  
 (C++ struct), 116  
 (C++ function), 99  
 esp\_brookesia::service::rpc::Server::Config esp\_brookesia::service::ServiceBase::Attributes::  
 (C++ member), 116  
 (C++ member), 100  
 esp\_brookesia::service::rpc::Server::Config esp\_brookesia::service::ServiceBase::Attributes::  
 (C++ member), 116  
 (C++ member), 100  
 esp\_brookesia::service::rpc::Server::disconnect esp\_brookesia::service::ServiceBase::call\_functio  
 (C++ function), 115  
 (C++ function), 97  
 esp\_brookesia::service::rpc::Server::get esp\_brookesia::service::ServiceBase::call\_functio  
 (C++ function), 116  
 (C++ function), 98  
 esp\_brookesia::service::rpc::Server::init esp\_brookesia::service::ServiceBase::call\_functio  
 (C++ function), 115  
 (C++ function), 98  
 esp\_brookesia::service::rpc::Server::remove esp\_brookesia::service::ServiceBase::call\_functio  
 (C++ function), 115  
 (C++ function), 98  
 esp\_brookesia::service::rpc::Server::Server esp\_brookesia::service::ServiceBase::FunctionBatc

(C++ type), 96  
 esp\_brookesia::service::ServiceBase::FuncEspIotDkKesiMaService::ServiceManager::init  
 (C++ type), 96 (C++ function), 101  
 esp\_brookesia::service::ServiceBase::FuncEspIotDkKesiMaService::ServiceManager::is\_initia  
 (C++ type), 96 (C++ function), 102  
 esp\_brookesia::service::ServiceBase::getEspIotDkKesiMaService::ServiceManager::is\_rpc\_se  
 (C++ function), 99 (C++ function), 102  
 esp\_brookesia::service::ServiceBase::getEspIotDkKesiMaService::ServiceManager::is\_runnin  
 (C++ function), 99 (C++ function), 102  
 esp\_brookesia::service::ServiceBase::getEspIotDkKesiMaService::ServiceManager::make\_defa  
 (C++ function), 97 (C++ function), 103  
 esp\_brookesia::service::ServiceBase::getEspIotDkKesiMaService::ServiceManager::new\_rpc\_c  
 (C++ function), 99 (C++ function), 102  
 esp\_brookesia::service::ServiceBase::getEspIotDkKesiMaService::ServiceManager::remove\_se  
 (C++ function), 96 (C++ function), 101  
 esp\_brookesia::service::ServiceBase::getEspIotDkKesiMaService::ServiceManager::RPC\_Clien  
 (C++ function), 99 (C++ struct), 103  
 esp\_brookesia::service::ServiceBase::isEspIotDkKesiMaService::ServiceManager::RPC\_Clien  
 (C++ function), 99 (C++ member), 103  
 esp\_brookesia::service::ServiceBase::isEspIotDkKesiMaService::ServiceManager::RPC\_Clien  
 (C++ function), 99 (C++ member), 103  
 esp\_brookesia::service::ServiceBase::isEspIotDkKesiMaService::ServiceManager::start  
 (C++ function), 99 (C++ function), 101  
 esp\_brookesia::service::ServiceBase::ServiceBase::ServiceManager::start\_rpc  
 (C++ function), 96 (C++ function), 102  
 esp\_brookesia::service::ServiceBase::subscribeEspIotDkKesiMaService::ServiceManager::stop  
 (C++ function), 98 (C++ function), 101  
 esp\_brookesia::service::ServiceBase::toEspIotDkKesiMaService::ServiceManager::stop\_rpc  
 (C++ function), 99 (C++ function), 102  
 esp\_brookesia::service::ServiceBinding ESP\_BROOKESIA\_HAL\_DEVICE\_REGISTER\_ALL\_POST\_DEINIT  
 (C++ class), 100 (C macro), 62  
 esp\_brookesia::service::ServiceBinding ESP\_BROOKESIA\_HAL\_DEVICE\_REGISTER\_ALL\_PRE\_INIT\_CA  
 (C++ function), 101 (C macro), 62  
 esp\_brookesia::service::ServiceBinding ESP\_BROOKESIA\_HAL\_DEVICE\_REGISTER\_POST\_DEINIT\_CAL  
 (C++ function), 101 (C macro), 62  
 esp\_brookesia::service::ServiceBinding ESP\_BROOKESIA\_HAL\_DEVICE\_REGISTER\_PRE\_INIT\_CALLBA  
 (C++ function), 101 (C macro), 62  
 esp\_brookesia::service::ServiceBinding::operator  
 bool (C++ function), 101  
 esp\_brookesia::service::ServiceBinding::release  
 (C++ function), 101  
 esp\_brookesia::service::ServiceManager  
 (C++ class), 101  
 esp\_brookesia::service::ServiceManager::add\_service  
 (C++ function), 101  
 esp\_brookesia::service::ServiceManager::bind  
 (C++ function), 101  
 esp\_brookesia::service::ServiceManager::call\_rpc\_function\_sync  
 (C++ function), 102  
 esp\_brookesia::service::ServiceManager::connect\_rpc\_server\_to\_services  
 (C++ function), 102  
 esp\_brookesia::service::ServiceManager::deinit  
 (C++ function), 101  
 esp\_brookesia::service::ServiceManager::disconnect\_rpc\_server\_from\_services  
 (C++ function), 102  
 esp\_brookesia::service::ServiceManager::get\_instance  
 (C++ function), 103  
 esp\_brookesia::service::ServiceManager::get\_service