## 1 C API Reference

1.1 RainMaker Core ................................. 3
1.2 RainMaker Standard Types .................. 23
1.3 RainMaker MQTT ............................... 35
1.4 RainMaker OTA ............................... 37
1.5 RainMaker Console ........................... 42
1.6 RainMaker Common ........................... 42

## 2 Python API Reference

2.1 Library ........................................... 51
2.2 Commands ....................................... 57

Python Module Index .............................. 63

Index .............................................. 65
ESP RainMaker is a platform that allows developers to build connected devices with Espressif’s ESP32-S2 SoC without hassle of managing the infrastructure. It provides a device SDK, self-adapting phone apps, transparent cloud middleware and host utilities to reduce complexity in development of connected devices.

This is the C API (for firmware) and Python API (for host tools) documentation for ESP RainMaker. All other documentation can be found at http://rainmaker.espressif.com
1.1 RainMaker Core

1.1.1 Core

Header File

- esp_rainmaker/include/esp_rmaker_core.h

Functions

**const char \*esp_rmaker_device_cb_src_to_str(esp_rmaker_req_src_t src)**

Convert device callback source to string

Device read/write callback can be via different sources. This is a helper API to give the source in string format for printing.

Example Usage:

```c
static esp_err_t write_cb(const esp_rmaker_device_t *device, const esp_rmaker_param_t *param, const esp_rmaker_param_val_t val, void *priv_data, esp_rmaker_write_ctx_t *ctx)
{
    if (ctx) {
        ESP_LOGI(TAG, "Received write request via : %s", esp_rmaker_device_cb_src_to_str(ctx->src));
    }
}
```

**Return** NULL terminated source string on success

**Return** NULL on failure

**Parameters**

- **[in] src**: The src field as received in the callback context.

**esp_rmaker_param_val_t esp_rmaker_bool(bool bval)**

Initialise a Boolean value

**Return** Value structure.

**Parameters**

- **[in] bval**: Initialising value.
ESP RainMaker Programming Guide

```
esp_rmaker_param_val_t esp_rmaker_int (int ival)
    Initialise an Integer value

    Return Value structure.
    Parameters
    • [in] ival: Initialising value.

esp_rmaker_param_val_t esp_rmaker_float (float fval)
    Initialise a Float value

    Return Value structure.
    Parameters

esp_rmaker_param_val_t esp_rmaker_str (const char *sval)
    Initialise a String value

    Return Value structure.
    Parameters
    • [in] sval: Initialising value.

esp_rmaker_param_val_t esp_rmaker_obj (const char *val)
    Initialise a json object value
    param[in] val initialising value

    Note the object will not be validated internally. it is the application’s responsibility to ensure that the object is
    a valid json object. eg. esp_rmaker_obj(“{“name”:”value”}”);
    return value structure

esp_rmaker_param_val_t esp_rmaker_array (const char *val)
    Initialise a json array value
    param[in] val initialising value

    Note the array will not be validated internally. it is the application’s responsibility to ensure that the array is a
    valid json array. eg. esp_rmaker_array(“[1,2,3]”);
    return value structure

esp_rmaker_node_t *esp_rmaker_node_init (const esp_rmaker_config_t *config, const char *name, const char *type)
    Initialize ESP RainMaker Node

    This initializes the ESP RainMaker agent and creates the node. The model and firmware version for the node
    are set internally as per the project name and version. These can be overridden (but not recommended) using
    the esp_rmaker_node_add_fw_version() and esp_rmaker_node_add_model() APIs.

    Note This should be the first call before using any other ESP RainMaker API.

    Return Node handle on success.
    Return NULL in case of failure.
    Parameters
    • [in] config: Configuration to be used by the ESP RainMaker.
```
• [in] name: Name of the node.
• [in] type: Type of the node.

```c
esp_err_t esp_rmaker_start(void)
Start ESP RainMaker Agent
This call starts the actual ESP RainMaker thread. This should preferably be called after a successful Wi-Fi connection in order to avoid unnecessary failures.

Return ESP_OK on success.
Return error in case of failure.
```

```c
esp_err_t esp_rmaker_stop(void)
Stop ESP RainMaker Agent
This call stops the ESP RainMaker Agent instance started earlier by esp_rmaker_start().

Return ESP_OK on success.
Return error in case of failure.
```

```c
esp_err_t esp_rmaker_node_deinit(const esp_rmaker_node_t *node)
Deinitialize ESP RainMaker Node
This API deinitializes the ESP RainMaker agent and the node created using esp_rmaker_node_init().

@returns ESP_OK on success.
Note This should be called after rainmaker has stopped.
Return error in case of failure.
```

```c
const esp_rmaker_node_t *esp_rmaker_get_node(void)
Get a handle to the Node
This API returns handle to a node created using esp_rmaker_node_init().

Return Node handle on success.
Return NULL in case of failure.
```

```c
char *esp_rmaker_get_node_id(void)
Get Node Id
Returns pointer to the NULL terminated Node ID string.

Return Pointer to a NULL terminated Node ID string.
```

```c
esp_rmaker_node_info_t *esp_rmaker_node_get_info(const esp_rmaker_node_t *node)
Get Node Info
Returns pointer to the node info as configured during initialisation.

Return Pointer to the node info on success.
Return NULL in case of failure.
```

1.1. RainMaker Core
Parameters

- node: Node handle.

```c
esp_err_t esp_rmaker_node_add_attribute(const esp_rmaker_node_t *node, const char *attr_name, const char *val)
```

Add Node attribute

Adds a new attribute as the metadata for the node. For the sake of simplicity, only string values are allowed.

Return ESP_OK on success.
Return error in case of failure.

Parameters

- node: Node handle.
- [in] attr_name: Name of the attribute.
- [in] val: Value for the attribute.

```c
esp_err_t esp_rmaker_node_add_fw_version(const esp_rmaker_node_t *node, const char *fw_version)
```

Add FW version for a node (Not recommended)

FW version is set internally to the project version. This API can be used to override that version.

Return ESP_OK on success.
Return error in case of failure.

Parameters

- node: Node handle.

```c
esp_err_t esp_rmaker_node_add_model(const esp_rmaker_node_t *node, const char *model)
```

Add model for a node

Model is set internally to the project name. This API can be used to override that name, now that a new field “project” has also been added internally to the node info.

Return ESP_OK on success.
Return error in case of failure.

Parameters

- node: Node handle.

```c
esp_err_t esp_rmaker_node_add_subtype(const esp_rmaker_node_t *node, const char *subtype)
```

Add subtype for a node

Return ESP_OK on success.
Return error in case of failure.

Parameters

- node: Node handle.
* [in] subtype: Subtype string.

```c
esp_rmaker_device_t *esp_rmaker_device_create(const char *dev_name, const char *type, void *priv_data)
```

Create a Device

This API will create a virtual “Device”. This could be something like a Switch, Lightbulb, etc.

**Note** The device created needs to be added to a node using esp_rmaker_node_add_device().

**Return** Device handle on success.

**Return** NULL in case of any error.

**Parameters**

- [in] dev_name: The unique device name.
- [in] type: Optional device type. Can be kept NULL.
- [in] priv_data: (Optional) Private data associated with the device. This will be passed to callbacks. It should stay allocated throughout the lifetime of the device.

```c
esp_rmaker_device_t *esp_rmaker_service_create(const char *serv_name, const char *type, void *priv_data)
```

Create a Service

This API will create a “Service”. It is exactly same like a device in terms of structure and so, all APIs for device are also valid for a service. A service could be something like OTA, diagnostics, etc.

**Note** Name of a service should not clash with name of a device.

**Note** The service created needs to be added to a node using esp_rmaker_node_add_device().

**Return** Device handle on success.

**Return** NULL in case of any error.

**Parameters**

- [in] serv_name: The unique service name.
- [in] type: Optional service type. Can be kept NULL.
- [in] priv_data: (Optional) Private data associated with the service. This will be passed to callbacks. It should stay allocated throughout the lifetime of the device.

```c
esp_err_t esp_rmaker_device_delete(const esp_rmaker_device_t *device)
```

Delete a Device/Service

This API will delete a device created using esp_rmaker_device_create().

**Note** The device should first be removed from the node using esp_rmaker_node_remove_device() before deleting.

**Return** ESP_OK on success.

**Return** error in case of failure.

**Parameters**

- [in] device: Device handle.
esp_err_t esp_rmaker_device_add_cb(const esp_rmaker_device_t *device, esp_rmaker_device_write_cb_t write_cb, esp_rmaker_device_read_cb_t read_cb)

Add callbacks for a device/service
Add read/write callbacks for a device that will be invoked as per requests received from the cloud (or other paths as may be added in future).

Return ESP_OK on success.
Return error in case of failure.

Parameters
• [in] device: Device handle.
• [in] read_cb: Read callback.

esp_err_t esp_rmaker_node_add_device(const esp_rmaker_node_t *node, const esp_rmaker_device_t *device)

Add a device to a node

Return ESP_OK on success.
Return error in case of failure.

Parameters
• [in] node: Node handle.
• [in] device: Device handle.

esp_err_t esp_rmaker_node_remove_device(const esp_rmaker_node_t *node, const esp_rmaker_device_t *device)

Remove a device from a node

Return ESP_OK on success.
Return error in case of failure.

Parameters
• [in] node: Node handle.
• [in] device: Device handle.

esp_rmaker_device_t *esp_rmaker_node_get_device_by_name(const esp_rmaker_node_t *node, const char *device_name)

Get device by name
Get handle for a device based on the name.

Return Device handle on success.
Return NULL in case of failure.

Parameters
• [in] node: Node handle.
• [in] device_name: Device name to search.
```c
esp_err_t esp_rmaker_device_add_attribute(const esp_rmaker_device_t *device, const char *attr_name, const char *val)
```

Add a Device attribute

**Note** Device attributes are reported only once after a boot-up as part of the node configuration. Eg. Serial Number

**Return** ESP_OK if the attribute was added successfully.

**Return** error in case of failure.

**Parameters**
- [in] device: Device handle.
- [in] attr_name: Name of the attribute.

```c
esp_err_t esp_rmaker_device_add_subtype(const esp_rmaker_device_t *device, const char *subtype)
```

Add a Device subtype

This can be something like esp.subtype.rgb-light for a device of type esp.device.lightbulb. This would primarily be used by the phone apps to render different icons for the same device type.

**Return** ESP_OK if the subtype was added successfully.

**Return** error in case of failure.

**Parameters**
- [in] device: Device handle.
- [in] subtype: String describing the sub type.

```c
esp_err_t esp_rmaker_device_add_model(const esp_rmaker_device_t *device, const char *model)
```

Add a Device model

This would primarily be used by the phone apps to render different icons for the same device type.

**Return** ESP_OK if the model was added successfully.

**Return** error in case of failure.

**Parameters**
- [in] device: Device handle.
- [in] model: String describing the model.

```c
char *esp_rmaker_device_get_name(const esp_rmaker_device_t *device)
```

Get device name from handle

**Return** NULL terminated device name string on success.

**Return** NULL in case of failure.

**Parameters**
- [in] device: Device handle.

---

### 1.1. RainMaker Core

1.1.1. RainMaker Core
char *esp_rmaker_device_get_type(const esp_rmaker_device_t *device)
Get device type from handle

Return NULL terminated device type string on success.
Return NULL in case of failure, or if the type wasn’t provided while creating the device.
Parameters
  • [in] device: Device handle.

esp_err_t esp_rmaker_device_add_param(const esp_rmaker_device_t *device, const esp_rmaker_param_t *param)
Add a parameter to a device/service

Return ESP_OK on success.
Return error in case of failure.
Parameters
  • [in] device: Device handle.
  • [in] param: Parameter handle.

esp_rmaker_param_t *esp_rmaker_device_get_param_by_type(const esp_rmaker_device_t *device, const char *param_type)
Get parameter by type
Get handle for a parameter based on the type.

Note If there are multiple parameters with the same type, this will return the first one. The API esp_rmaker_device_get_param_by_name() can be used to get a specific parameter, because the parameter names in a device are unique.

Return Parameter handle on success.
Return NULL in case of failure.
Parameters
  • [in] device: Device handle.
  • [in] param_type: Parameter type to search.

esp_rmaker_param_t *esp_rmaker_device_get_param_by_name(const esp_rmaker_device_t *device, const char *param_name)
Get parameter by name
Get handle for a parameter based on the name.

Return Parameter handle on success.
Return NULL in case of failure.
Parameters
  • [in] device: Device handle.
  • [in] param_name: Parameter name to search.
**esp_err_t esp_rmaker_device_assign_primary_param(const esp_rmaker_device_t *device, const esp_rmaker_param_t *param)**

Assign a primary parameter

Assign a parameter (already added using esp_rmaker_device_add_param()) as a primary parameter, which can be used by clients (phone apps specifically) to give prominence to it.

**Return** ESP_OK if the parameter was assigned as the primary successfully.

**Return** error in case of failure.

**Parameters**

- [in] `device`: Device handle.

**esp_rmaker_param_t* esp_rmaker_param_create(const char *param_name, const char *type, esp_rmaker_param_val_t val, uint8_t properties)**

Create a Parameter

Parameter can be something like Temperature, Outlet state, Lightbulb brightness, etc.

Any changes should be reported using the esp_rmaker_param_update_and_report() API. Any remote changes will be reported to the application via the device callback, if registered.

**Note** The parameter created needs to be added to a device using esp_rmaker_device_add_param(). Parameter name should be unique in a given device.

**Return** Parameter handle on success.

**Return** NULL in case of failure.

**Parameters**

- [in] `param_name`: Name of the parameter. a*
- [in] `type`: Optional parameter type. Can be kept NULL.
- [in] `val`: Value of the parameter. This also specifies the type that will be assigned to this parameter. You can use esp_rmaker_bool(), esp_rmaker_int(), esp_rmaker_float() or esp_rmaker_str() functions as the argument here. E.g, esp_rmaker_bool(true).
- [in] `properties`: Properties of the parameter, which will be a logical OR of flags in esp_param_property_flags_t.

**esp_err_t esp_rmaker_param_add_ui_type(const esp_rmaker_param_t *param, const char *ui_type)**

Add a UI Type to a parameter

This will be used by the Phone apps (or other clients) to render appropriate UI for the given parameter. Please refer the RainMaker documentation for supported UI Types.

**Return** ESP_OK on success.

**Return** error in case of failure.

**Parameters**

- [in] `ui_type`: String describing the UI Type.
esp_err_t esp_rmaker_param_add_bounds(const esp_rmaker_param_t *param, esp_rmaker_param_val_t min, esp_rmaker_param_val_t max, esp_rmaker_param_val_t step)

Add bounds for an integer/float parameter

This can be used to add bounds (min/max values) for a given integer parameter. Eg. brightness will have bounds as 0 and 100 if it is a percentage. Eg. esp_rmaker_param_add_bounds(brightness_param, esp_rmaker_int(0), esp_rmaker_int(100), esp_rmaker_int(5));

Note The RainMaker core does not check the bounds. It is upto the application to handle it.

Return ESP_OK on success. return error in case of failure.

Parameters
- [in] min: Minimum allowed value.
- [in] max: Maximum allowed value.
- [in] step: Minimum stepping (set to 0 if no specific value is desired).

esp_err_t esp_rmaker_param_add_valid_str_list(const esp_rmaker_param_t *param, const char *strs[], uint8_t count)

Add a list of valid strings for a string parameter

This can be used to add a list of valid strings for a given string parameter.

Eg. static const char *valid_strs[] = {"None","Yes","No","Can’t Say");
esp_rmaker_param_add_valid_str_list(param, valid_strs, 4);

Note The RainMaker core does not check the values. It is upto the application to handle it.

Return ESP_OK on success. return error in case of failure.

Parameters
- [in] strs: Pointer to an array of strings. Note that this memory should stay allocated throughout the lifetime of this parameter.
- [in] count: Number of strings in the above array.

esp_err_t esp_rmaker_param_add_array_max_count(const esp_rmaker_param_t *param, int count)

Add max count for an array parameter

This can be used to put a limit on the maximum number of elements in an array.

Note The RainMaker core does not check the values. It is upto the application to handle it.

Return ESP_OK on success. return error in case of failure.

Parameters
- [in] count: Max number of elements allowed in the array.
esp_err_t esp_rmaker_param_update_and_report (const esp_rmaker_param_t *param, esp_rmaker_param_val_t val)

Update and report a parameter

Calling this API will update the parameter and report it to ESP RainMaker cloud. This should be used whenever there is any local change.

Return ESP_OK if the parameter was updated successfully.
Return error in case of failure.

Parameters
  • [in] param: Parameter handle.

esp_err_t esp_rmaker_param_update_and_notify (const esp_rmaker_param_t *param, esp_rmaker_param_val_t val)

Update and notify a parameter

Calling this API will update the parameter and report it to ESP RainMaker cloud similar to esp_rmaker_param_update_and_report(). However, additionally, it will also trigger a notification on the phone apps (if enabled).

Alternatively, the esp_rmaker_raise_alert() API can also be used to trigger notification on the phone apps with pre-formatted text.

Note This should be used only when some local change requires explicit notification even when the phone app is in background, not otherwise. Eg. Alarm got triggered, temperature exceeded some threshold, etc.

Return ESP_OK if the parameter was updated successfully.
Return error in case of failure.

Parameters
  • [in] param: Parameter handle.

esp_err_t esp_rmaker_raise_alert (const char *alert_str)

Trigger an alert on the phone app

This API will trigger a notification alert on the phone apps (if enabled) using the formatted text provided. Note that this does not send a notification directly to the phone, but reports the alert to the ESP RainMaker cloud which then uses the Notification framework to send notifications to the phone apps. The value does not get stored anywhere, nor is it linked to any node parameters.

Note This should be used only if some event requires explicitly alerting the user even when the phone app is in background, not otherwise. Eg. “Motion Detected”, “Fire alarm triggered”

Return ESP_OK on success.
Return error in case of failure.

Parameters
  • [in] alert_str: NULL terminated pre-formatted alert string. Maximum length can be ESP_RMAKER_MAX_ALERT_LEN, excluding NULL character.

char * esp_rmaker_param_get_name (const esp_rmaker_param_t *param)

Get parameter name from handle
Return  NULL terminated parameter name string on success.
Return  NULL in case of failure.
Parameters
  • [in]  param: Parameter handle.

char *esp_rmaker_param_get_type(const esp_rmaker_param_t *param)
Get parameter type from handle
Return  NULL terminated parameter type string on success.
Return  NULL in case of failure, or if the type wasn’t provided while creating the parameter.
Parameters
  • [in]  param: Parameter handle.

esp_rmaker_param_val_t *esp_rmaker_param_get_val(esp_rmaker_param_t *param)
Get parameter value
This gives the parameter value that is stored in the RainMaker core.

Note  This does not call any explicit functions to read value from hardware/driver.
Return  Pointer to parameter value on success.
Return  NULL in case of failure.
Parameters
  • [in]  param: Parameter handle

esp_err_t esp_rmaker_report_node_details(void)
Report the node details to the cloud
This API reports node details i.e. the node configuration and values of all the parameters to the ESP RainMaker cloud. Eg. If a new device is created (with some parameters and attributes), then this API should be called after that to send the node details to the cloud again and the changes to be reflected in the clients (like phone apps).

Note  Please use this API only if you need to create or delete devices after esp_rmaker_start() has already been called, for use cases like bridges or hubs.
Return  ESP_OK if the node details are successfully queued to be published.
Return  error in case of failure.

esp_err_t esp_rmaker_timezone_service_enable(void)
Enable Timezone Service
This enables the ESP RainMaker standard timezone service which can be used to set timezone, either in POSIX or location string format. Please refer the specifications for additional details.
Return  ESP_OK on success
Return  error on failure

esp_err_t esp_rmaker_system_service_enable(esp_rmaker_system_serv_config_t *config)
Enable System Service
This enables the ESP RainMaker standard system service which can be used for operations like reboot, factory reset and Wi-Fi reset.

Please refer to the specifications for additional details.

**Return** ESP_OK on success

**Return** error on failure

**Parameters**

- **[in] config**: Configuration for the system service.

bool `esp_rmaker_local_ctrl_service_started`(void)

Check if local_ctrl service has started

**Return** true if service has started

**Return** false if the service has not started

**Return**

**Return**

**Return**

**Parameters**

- **[in] config**: Configuration for the system service.

`esp_err_t esp_rmaker_ota_enable_default`(void)

Enable Default RainMaker OTA Firmware Upgrade

This enables the default recommended RainMaker OTA Firmware Upgrade, which is “Using the Topics”, which allows performing OTA from Dashboard. This OTA can be triggered by Admin Users only. On Public RainMaker deployment, for nodes using “Self Claiming”, since there is no associated admin user, the Primary user will automatically become the admin and can perform OTA from dashboard.

**Return** ESP_OK on success

**Return** error on failure

**Parameters**

- **[in] config**: Configuration for the system service.

`esp_err_t esp_rmaker_local_ctrl_enable`(void)

`esp_err_t esp_rmaker_local_ctrl_disable`(void)

This API signs the challenge with RainMaker private key.

**Return** ESP_OK on success. response is dynamically allocated, free the response on success.

**Return** Apt error on failure.

**Parameters**

- **[in] challenge**: Pointer to the data to be signed
- **[in] inlen**: Length of the challenge
- **[out] response**: Pointer to the signature.
- **[out] outlen**: Length of the signature

`esp_err_t esp_rmaker_local_ctrl_enable`(void)

`esp_err_t esp_rmaker_local_ctrl_disable`(void)
Unions

union esp_rmaker_val_t
    #include <esp_rmaker_core.h> ESP RainMaker Value

    Public Members

    bool b
        Boolean
    int i
        Integer
    float f
        Float
    char *s
        NULL terminated string

Structures

struct esp_rmaker_node_info_t
    ESP RainMaker Node information

    Public Members

    char *name
        Name of the Node
    char *type
        Type of the Node
    char *fw_version
        Firmware Version (Optional). If not set, PROJECT_VER is used as default (recommended)
    char *model
        Model (Optional). If not set, PROJECT_NAME is used as default (recommended)
    char *subtype
        Subtype (Optional).
    char **secure_boot_digest
        An array of digests read from efuse. Should be freed after use

struct esp_rmaker_config_t
    ESP RainMaker Configuration
Public Members

`bool enable_time_sync`
Enable Time Sync Setting this true will enable SNTP and fetch the current time before attempting to connect to the ESP RainMaker service

**struct esp_rmaker_param_val_t**
ESP RainMaker Parameter Value

Public Members

`esp_rmaker_val_type_t type`
Type of Value

`esp_rmaker_val_t val`
Actual value. Depends on the type

**struct esp_rmaker_write_ctx_t**
Write request Context

Public Members

`esp_rmaker_req_src_t src`
Source of request

**struct esp_rmaker_read_ctx_t**
Read request context

Public Members

`esp_rmaker_req_src_t src`
Source of request

**struct esp_rmaker_system_serv_config_t**
System service configuration

Public Members

`uint16_t flags`
Logical OR of system service flags (SYSTEM_SERV_FLAG_REBOOT, SYSTEM_SERV_FLAG_FACTORY_RESET, SYSTEM_SERV_FLAG_WIFI_RESET) as required or SYSTEM_SERV_FLAGS_ALL.

`int8_t reboot_seconds`
Time in seconds after which the device should reboot. Value of zero would trigger an immediate reboot if a write is received for the Reboot parameter. Recommended value: 2

`int8_t reset_seconds`
Time in seconds after which the device should reset (Wi-Fi or factory). Value of zero would trigger an immediate action if a write is received for the Wi-Fi reset or Factory reset parameter. Recommended value: 2

`int8_t reset_reboot_seconds`
Time in seconds after which the device should reboot after it has been reset. Value of zero would mean that there won’t be any reboot after the reset. Recommended value: 2

1.1. RainMaker Core
Macros

ESP_RMAKER_CONFIG_VERSION
ESP_RMAKER_MAX_ALERT_LEN
SYSTEM_SERV_FLAG_REBOOT
  System Service Reboot Flag
SYSTEM_SERV_FLAG_FACTORY_RESET
  System Service Factory Reset Flag
SYSTEM_SERV_FLAG_WIFI_RESET
  System Service Wi-Fi Reset Flag
SYSTEM_SERV_FLAGS_ALL
  System Service All Flags

Type Definitions

typedef size_t esp_rmaker_handle_t
  Generic ESP RainMaker handle

typedef esp_rmaker_handle_t esp_rmaker_node_t
  ESP RainMaker Node Handle

typedef esp_rmaker_handle_t esp_rmaker_device_t
  ESP RainMaker Device Handle

typedef esp_rmaker_handle_t esp_rmaker_param_t
  ESP RainMaker Parameter Handle

typedef esp_err_t (*esp_rmaker_device_write_cb_t)(const esp_rmaker_device_t *device,
  const esp_rmaker_param_t *param,
  const esp_rmaker_param_val_t *val,
  void *priv_data, esp_rmaker_write_ctx_t *ctx)

Callback for parameter value write requests.
The callback should call the esp_rmaker_param_update_and_report() API if the new value is to be set and
reported back.

Return  ESP_OK on success.
Return  error in case of failure.

Parameters

  • [in] device: Device handle.
  • [in] param: Parameter handle.
  • [in] param: Pointer to esp_rmaker_param_val_t. Use appropriate elements as per the value type.
  • [in] priv_data: Pointer to the private data passed while creating the device.
  • [in] ctx: Context associated with the request.

Callback for parameter value changes

typedef esp_err_t (*esp_rmaker_device_read_cb_t)(const esp_rmaker_device_t *device,
  const esp_rmaker_param_t *param,
  void *priv_data, esp_rmaker_read_ctx_t *ctx)
The callback should call the esp_rmaker_param_update_and_report() API if the new value is to be set and reported back.

Note Currently, the read callback never gets invoked as the communication between clients (mobile phones, CLI, etc.) and node is asynchronous. So, the read request does not reach the node. This callback will however be used in future.

Return ESP_OK on success.

Return error in case of failure.

Parameters

- [in] device: Device handle.
- [in] priv_data: Pointer to the private data passed while creating the device.
- [in] ctx: Context associated with the request.

Enumerations

enum esp_rmaker_event_t
ESP RainMaker Events

Values:

RMAKER_EVENT_INIT_DONE = 1
RainMaker Core Initialisation Done

RMAKER_EVENT_CLAIM_STARTED
Self Claiming Started

RMAKER_EVENT_CLAIM_SUCCESSFUL
Self Claiming was Successful

RMAKER_EVENT_CLAIM_FAILED
Self Claiming Failed

RMAKER_EVENT_USER_NODE_MAPPING_DONE
Node side communication for User-Node mapping done. Actual mapping state will be managed by the ESP RainMaker cloud based on the user side communication. Associated data is the NULL terminated user id.

RMAKER_EVENT_LOCAL_CTRL_STARTED
Local control started. Associated data is the NULL terminated Service Name

RMAKER_EVENT_USER_NODE_MAPPING_RESET

RMAKER_EVENT_LOCAL_CTRL_STOPPED
Local control stopped.

denum esp_rmaker_val_type_t
ESP RainMaker Parameter Value type

Values:

RMAKER_VAL_TYPE_INVALID = 0
Invalid

RMAKER_VAL_TYPE_BOOLEAN
Boolean
**RMAKER_VAL_TYPE_INTEGER**

Integer. Mapped to a 32 bit signed integer

**RMAKER_VAL_TYPE_FLOAT**

Floating point number

**RMAKER_VAL_TYPE_STRING**

NULL terminated string

**RMAKER_VAL_TYPE_OBJECT**

NULL terminated JSON Object string Eg. {"name":"value"}

**RMAKER_VAL_TYPE_ARRAY**

NULL terminated JSON Array string Eg. [1,2,3]

**enum esp_param_property_flags_t**

Param property flags

Values:

- **PROP_FLAG_WRITE** = (1 << 0)
- **PROP_FLAG_READ** = (1 << 1)
- **PROP_FLAG_TIME_SERIES** = (1 << 2)
- **PROP_FLAG_PERSIST** = (1 << 3)

**enum esp_rmaker_req_src_t**

Parameter read/write request source

Values:

- **ESP_RMAKER_REQ_SRC_INIT**
  Request triggered in the init sequence i.e. when a value is found in persistent memory for parameters with **PROP_FLAG_PERSIST**.
- **ESP_RMAKER_REQ_SRC_CLOUD**
  Request received from cloud
- **ESP_RMAKER_REQ_SRC_SCHEDULE**
  Request received when a schedule has triggered
- **ESP_RMAKER_REQ_SRC_SCENE_ACTIVATE**
  Request received when a scene has been activated
- **ESP_RMAKER_REQ_SRC_SCENE_DEACTIVATE**
  Request received when a scene has been deactivated
- **ESP_RMAKER_REQ_SRC_LOCAL**
  Request received from a local controller
- **ESP_RMAKER_REQ_SRC_MAX**
  This will always be the last value. Any value equal to or greater than this should be considered invalid.
1.1.2 User Mapping

Header File

- esp_rainmaker/include/esp_rmaker_user_mapping.h

Functions

```c
esp_rmaker_user_mapping_state_t esp_rmaker_user_node_mapping_get_state(void)
Get User-Node mapping state
This returns the current user-node mapping state.

Return user mapping state
```

```c
esp_err_t esp_rmaker_user_mapping_endpoint_create(void)
Create User Mapping Endpoint
This will create a custom provisioning endpoint for user-node mapping. This should be called after
wifi_prov_mgr_init() but before wifi_prov_mgr_start_provisioning()

Return ESP_OK on success
Return error on failure
```

```c
esp_err_t esp_rmaker_user_mapping_endpoint_register(void)
Register User Mapping Endpoint
This will register the callback for the custom provisioning endpoint for user-node mapping which
was created with esp_rmaker_user_mapping_endpoint_create(). This should be called immediately after
wifi_prov_mgr_start_provisioning().

Return ESP_OK on success
Return error on failure
```

```c
esp_err_t esp_rmaker_start_user_node_mapping(char *user_id, char *secret_key)
Add User-Node mapping
This call will start the user-node mapping workflow on the node. This is automatically called if you have used
esp_rmaker_user_mapping_endpoint_register(). Use this API only if you want to trigger the user-node mapping
after the Wi-Fi provisioning has already been done.

Return ESP_OK if the workflow was successfully triggered. This does not guarantee success of the actual
mapping. The mapping status needs to be checked separately by the clients.
Return error on failure.
```

Parameters

- [in] user_id: The User identifier received from the client (Phone app/CLI)
- [in] secret_key: The Secret key received from the client (Phone app/CLI)
Enumerations

```c
enum esp_rmaker_user_mapping_state_t
    User-Node Mapping states
    
    Values:

    ESP_RMAKER_USER_MAPPING_RESET = 0
        Mapping does not exist or is not initialized

    ESP_RMAKER_USER_MAPPING_STARTED
        Mapping has started

    ESP_RMAKER_USER_MAPPING_REQ_SENT
        Mapping request sent to cloud

    ESP_RMAKER_USER_MAPPING_DONE
        Mapping is done
```

1.1.3 Scheduling

Header File

- esp_rainmaker/include/esp_rmaker_schedule.h

Functions

```c
esp_err_t esp_rmaker_schedule_enable (void)
    Enable Schedules

    This API enables the scheduling service for the node. For more information, check here
    It is recommended to set the timezone while using schedules. Check here for more information on timezones

    Note This API should be called after esp_rmaker_node_init() but before esp_rmaker_start().

    Return ESP_OK on success.
    Return error in case of failure.
```

1.1.4 Scenes

Header File

- esp_rainmaker/include/esp_rmaker_scenes.h
**Functions**

```c
esp_err_t esp_rmaker_scenes_enable (void)
```

Enable Scenes

This API enables the scenes service for the node. For more information, check [here](#).

**Note** This API should be called after esp_rmaker_node_init() but before esp_rmaker_start().

**Return** ESP_OK on success.

**Return** error in case of failure.

### 1.2 RainMaker Standard Types

#### 1.2.1 Standard Types

**Header File**

- `esp_rainmaker/include/esp_rmaker_standard_types.h`

**Macros**

- `ESP_RMAKER_UI_TOGGLE`
- `ESP_RMAKER_UI_SLIDER`
- `ESP_RMAKER_UI_DROPDOWN`
- `ESP_RMAKER_UI_TEXT`
- `ESP_RMAKER_UI_HUE_SLIDER`
- `ESP_RMAKER_UI_HUE_CIRCLE`
- `ESP_RMAKER_UI_PUSHBUTTON`
- `ESP_RMAKER_UI_TRIGGER`
- `ESP_RMAKER_UI_HIDDEN`
- `ESP_RMAKER_PARAM_NAME`
- `ESP_RMAKER_PARAM_POWER`
- `ESP_RMAKER_PARAM_BRIGHTNESS`
- `ESP_RMAKER_PARAM_HUE`
- `ESP_RMAKER_PARAM_SATURATION`
- `ESP_RMAKER_PARAM_INTENSITY`
- `ESP_RMAKER_PARAM_CCT`
- `ESP_RMAKER_PARAM_SPEED`
- `ESP_RMAKER_PARAM_DIRECTION`
- `ESP_RMAKER_PARAM_TEMPERATURE`
- `ESP_RMAKER_PARAM_OTA_STATUS`
ESP_RMAKER_PARAM_OTA_INFO
ESP_RMAKER_PARAM_OTA_URL
ESP_RMAKER_PARAM_TIMEZONE
ESP_RMAKER_PARAM_TIMEZONE_POSIX
ESP_RMAKER_PARAM_SCHEDULES
ESP_RMAKER_PARAM_SCENES
ESP_RMAKER_PARAM_REBOOT
ESP_RMAKER_PARAM_FACTORY_RESET
ESP_RMAKER_PARAM_WIFI_RESET
ESP_RMAKER_PARAM_LOCAL_CONTROL_POP
ESP_RMAKER_PARAM_LOCAL_CONTROL_TYPE
ESP_RMAKER_PARAM_TOGGLE
ESP_RMAKER_PARAM_RANGE
ESP_RMAKER_PARAM_MODE
ESP_RMAKER_PARAM_BLINDS_POSITION
ESP_RMAKER_PARAM_GARAGE_POSITION
ESP_RMAKER_PARAM_LIGHT_MODE
ESP_RMAKER_PARAM_AC_MODE
ESP_RMAKER_DEVICE_SWITCH
ESP_RMAKER_DEVICE_LIGHTBULB
ESP_RMAKER_DEVICE_FAN
ESP_RMAKERDEVICE_TEMP_SENSOR
ESP_RMAKERDEVICE_LIGHT
ESP_RMAKERDEVICE_OUTLET
ESP_RMAKERDEVICE_PLUG
ESP_RMAKERDEVICE_SOCKET
ESP_RMAKERDEVICE_LOCK
ESP_RMAKERDEVICE_BLINDS_INTERNAL
ESP_RMAKERDEVICE_BLINDS_EXTERNAL
ESP_RMAKERDEVICE_GARAGE_DOOR
ESP_RMAKERDEVICE_GARAGE_LOCK
ESP_RMAKERDEVICE_SPEAKER
ESP_RMAKERDEVICE_AIR_CONDITIONER
ESP_RMAKERDEVICE_THERMOSTAT
ESP_RMAKERDEVICE_TV
ESP_RMAKERDEVICE_WASHER
1.2.2 Standard Parameters

Header File

- esp_rainmaker/include/esp_rmaker_standard_params.h

Functions

esp_rmaker_param_t *esp_rmaker_name_param_create(const char *param_name, const char *val)

Create standard name param

This will create the standard name parameter. This should be added to all devices for which you want a user customisable name. The value should be same as the device name.

All standard device creation APIs will add this internally. No application registered callback will be called for this parameter, and changes will be managed internally.

Return  Parameter handle on success.
Return  NULL in case of failures.

Parameters

- [in] param_name: Name of the parameter

esp_rmaker_param_t *esp_rmaker_power_param_create(const char *param_name, bool val)

Create standard Power param

This will create the standard power parameter.

Return  Parameter handle on success.
Return  NULL in case of failures.

Parameters

- [in] param_name: Name of the parameter
- [in] val: Default Value of the parameter

esp_rmaker_param_t *esp_rmaker_brightness_param_create(const char *param_name, int val)

Create standard Brightness param

This will create the standard brightness parameter.
Return Parameter handle on success.
Return NULL in case of failures.

Parameters
- [in] param_name: Name of the parameter
- [in] val: Default Value of the parameter

```
esp_rmaker_param_t *esp_rmaker_hue_param_create(const char *param_name, int val)
```
Create standard Hue param
This will create the standard hue parameter.

Return Parameter handle on success.
Return NULL in case of failures.

Parameters
- [in] param_name: Name of the parameter
- [in] val: Default Value of the parameter

```
esp_rmaker_param_t *esp_rmaker_saturation_param_create(const char *param_name, int val)
```
Create standard Saturation param
This will create the standard saturation parameter.

Return Parameter handle on success.
Return NULL in case of failures.

Parameters
- [in] param_name: Name of the parameter
- [in] val: Default Value of the parameter

```
esp_rmaker_param_t *esp_rmaker_intensity_param_create(const char *param_name, int val)
```
Create standard Intensity param
This will create the standard intensity parameter.

Return Parameter handle on success.
Return NULL in case of failures.

Parameters
- [in] param_name: Name of the parameter
- [in] val: Default Value of the parameter

```
esp_rmaker_param_t *esp_rmaker_cct_param_create(const char *param_name, int val)
```
Create standard CCT param
This will create the standard cct parameter.

Return Parameter handle on success.
Return NULL in case of failures.
Parameters

- [in] param_name: Name of the parameter
- [in] val: Default Value of the parameter

\texttt{esp_rmaker_param_t} *\texttt{esp_rmaker_direction_param_create}(\texttt{const} char *param_name, int val)

Create standard Direction param
This will create the standard direction parameter.

Return Parameter handle on success.
Return NULL in case of failures.

Parameters

- [in] param_name: Name of the parameter
- [in] val: Default Value of the parameter

\texttt{esp_rmaker_param_t} *\texttt{esp_rmaker_speed_param_create}(\texttt{const} char *param_name, int val)

Create standard Speed param
This will create the standard speed parameter.

Return Parameter handle on success.
Return NULL in case of failures.

Parameters

- [in] param_name: Name of the parameter
- [in] val: Default Value of the parameter

\texttt{esp_rmaker_param_t} *\texttt{esp_rmaker_temperature_param_create}(\texttt{const} char *param_name, float val)

Create standard Temperature param
This will create the standard temperature parameter.

Return Parameter handle on success.
Return NULL in case of failures.

Parameters

- [in] param_name: Name of the parameter
- [in] val: Default Value of the parameter

\texttt{esp_rmaker_param_t} *\texttt{esp_rmaker_ota_status_param_create}(\texttt{const} char *param_name)

Create standard OTA Status param
This will create the standard ota status parameter. Default value is set internally.

Return Parameter handle on success.
Return NULL in case of failures.

Parameters

- [in] param_name: Name of the parameter

1.2. RainMaker Standard Types
esp_rmaker_param_t *esp_rmaker_ota_info_param_create(const char *param_name)

Create standard OTA Info param

This will create the standard ota info parameter. Default value is set internally.

Return Parameter handle on success.
Return NULL in case of failures.

Parameters

• [in] param_name: Name of the parameter

esp_rmaker_param_t *esp_rmaker_ota_url_param_create(const char *param_name)

Create standard OTA URL param

This will create the standard ota url parameter. Default value is set internally.

Return Parameter handle on success.
Return NULL in case of failures.

Parameters

• [in] param_name: Name of the parameter

esp_rmaker_param_t *esp_rmaker_timezone_param_create(const char *param_name, const char *val)

Create standard Timezone param

This will create the standard timezone parameter.

Return Parameter handle on success.
Return NULL in case of failures.

Parameters

• [in] param_name: Name of the parameter
• [in] val: Default Value of the parameter (Eg. “Asia/Shanghai”). Can be kept NULL.

esp_rmaker_param_t *esp_rmaker_timezone_posix_param_create(const char *param_name, const char *val)

Create standard POSIX Timezone param

This will create the standard posix timezone parameter.

Return Parameter handle on success.
Return NULL in case of failures.

Parameters

• [in] param_name: Name of the parameter
• [in] val: Default Value of the parameter (Eg. “CST-8”). Can be kept NULL.

esp_rmaker_param_t *esp_rmaker_schedules_param_create(const char *param_name, int max_schedules)

Create standard Schedules param

This will create the standard schedules parameter. Default value is set internally.
Return Parameter handle on success.

Return NULL in case of failures.

Parameters

• [in] param_name: Name of the parameter
• [in] max_schedules: Maximum number of schedules allowed

```
esp_rmaker_param_t *esp_rmaker_scenes_param_create(const char *param_name, int max_scenes)
```

Create standard Scenes param

This will create the standard scenes parameter. Default value is set internally.

Return Parameter handle on success.

Return NULL in case of failures.

Parameters

• [in] param_name: Name of the parameter
• [in] max_scenes: Maximum number of scenes allowed

```
esp_rmaker_param_t *esp_rmaker_reboot_param_create(const char *param_name)
```

Create standard Reboot param

This will create the standard reboot parameter. Set value to true (via write param) for the action to trigger.

Return Parameter handle on success.

Return NULL in case of failures.

Parameters

• [in] param_name: Name of the parameter

```
esp_rmaker_param_t *esp_rmaker_factory_reset_param_create(const char *param_name)
```

Create standard Factory Reset param

This will create the standard factory reset parameter. Set value to true (via write param) for the action to trigger.

Return Parameter handle on success.

Return NULL in case of failures.

Parameters

• [in] param_name: Name of the parameter

```
esp_rmaker_param_t *esp_rmaker_wifi_reset_param_create(const char *param_name)
```

Create standard Wi-Fi Reset param

This will create the standard Wi-Fi Reset parameter. Set value to true (via write param) for the action to trigger.

Return Parameter handle on success.

Return NULL in case of failures.

Parameters

• [in] param_name: Name of the parameter
```c
esp_rmaker_param_t *esp_rmaker_local_control_pop_param_create(const char *param_name, const char *val)
```

Create standard Local Control POP param

This will create the standard Local Control POP parameter.

**Return**  Parameter handle on success.

**Return**  NULL in case of failures.

**Parameters**

- `[in]` param_name: Name of the parameter
- `[in]` val: Default Value of the parameter (Eg. “abcd1234”). Can be kept NULL.

```c
esp_rmaker_param_t *esp_rmaker_local_control_type_param_create(const char *param_name, int val)
```

Create standard Local Control Type param

This will create the standard Local Control security type parameter.

**Return**  Parameter handle on success.

**Return**  NULL in case of failures.

**Parameters**

- `[in]` param_name: Name of the parameter
- `[in]` val: Default Value of the parameter

**Macros**

- ESP_RMAKER_DEF_NAME_PARAM
- ESP_RMAKER_DEF_POWER_NAME
- ESP_RMAKER_DEF_BRIGHTNESS_NAME
- ESP_RMAKER_DEF_HUE_NAME
- ESP_RMAKER_DEF_SATURATION_NAME
- ESP_RMAKER_DEF_INTENSITY_NAME
- ESP_RMAKER_DEF_CCT_NAME
- ESP_RMAKER_DEF_DIRECTION_NAME
- ESP_RMAKER_DEF_SPEED_NAME
- ESP_RMAKER_DEF_TEMPERATURE_NAME
- ESP_RMAKER_DEF_OTA_STATUS_NAME
- ESP_RMAKER_DEF_OTA_INFO_NAME
- ESP_RMAKER_DEF_OTA_URL_NAME
- ESP_RMAKER_DEF_TIMEZONE_NAME
- ESP_RMAKER_DEF_TIMEZONE_POSIX_NAME
1.2.3 Standard Devices

Header File

- esp_rainmaker/include/esp_rmaker_standard_devices.h

Functions

```c
esp_rmaker_device_t *esp_rmaker_switch_device_create(const char *dev_name, void *priv_data, bool power)
```

Create a standard Switch device

This creates a Switch device with the mandatory parameters and also assigns the primary parameter. The default parameter names will be used. Refer esp_rmaker_standard_params.h for default names.

Return Device handle on success.

Return NULL in case of failures.

Parameters

- [in] dev_name: The unique device name
- [in] priv_data: (Optional) Private data associated with the device. This should stay allocated throughout the lifetime of the device #
- [in] power: Default value of the mandatory parameter “power”

```c
esp_rmaker_device_t *esp_rmaker_lightbulb_device_create(const char *dev_name, void *priv_data, bool power)
```

Create a standard Lightbulb device

This creates a Lightbulb device with the mandatory parameters and also assigns the primary parameter. The default parameter names will be used. Refer esp_rmaker_standard_params.h for default names.

Return Device handle on success.

Return NULL in case of failures.

Parameters

- [in] dev_name: The unique device name
- [in] priv_data: (Optional) Private data associated with the device. This should stay allocated throughout the lifetime of the device
- [in] power: Default value of the mandatory parameter “power”
**esp_rmaker_device_t** *esp_rmaker_fan_device_create*(const char *dev_name, void *priv_data, bool power)

Create a standard Fan device

This creates a Fan device with the mandatory parameters and also assigns the primary parameter. The default parameter names will be used. Refer esp_rmaker_standard_params.h for default names.

Return Device handle on success.

Return NULL in case of failures.

Parameters

• [in] dev_name: The unique device name

• [in] priv_data: (Optional) Private data associated with the device. This should stay allocated throughout the lifetime of the device

• [in] power: Default value of the mandatory parameter “power”

**esp_rmaker_device_t** *esp_rmaker_temp_sensor_device_create*(const char *dev_name, void *priv_data, float temperature)

Create a standard Temperature Sensor device

This creates a Temperature Sensor device with the mandatory parameters and also assigns the primary parameter. The default parameter names will be used. Refer esp_rmaker_standard_params.h for default names.

Return Device handle on success.

Return NULL in case of failures.

Parameters

• [in] dev_name: The unique device name

• [in] priv_data: (Optional) Private data associated with the device. This should stay allocated throughout the lifetime of the device

• [in] temperature: Default value of the mandatory parameter “temperature”

### 1.2.4 Standard Services

**Header File**

- esp_rainmaker/include/esp_rmaker_standard_services.h

**Functions**

**esp_rmaker_device_t** *esp_rmaker_ota_service_create*(const char *serv_name, void *priv_data)

Create a standard OTA service

This creates an OTA service with the mandatory parameters. The default parameter names will be used. Refer esp_rmaker_standard_params.h for default names.

Return service_handle on success.

Return NULL in case of any error.

Parameters
• [in] serv_name: The unique service name
• [in] priv_data: (Optional) Private data associated with the service. This should stay allocated throughout the lifetime of the service.

```
esp_rmaker_device_t *esp_rmaker_time_service_create(const char *serv_name, const char *timezone, const char *timezone_posix, void *priv_data)
```

Create a standard OTA service
This creates an OTA service with the mandatory parameters. The default parameter names will be used. Refer esp_rmaker_standard_params.h for default names.

**Return** service_handle on success.
**Return** NULL in case of any error.

**Parameters**
• [in] serv_name: The unique service name
• [in] timezone: Default value of timezone string (Eg. “Asia/Shanghai”). Can be kept NULL.
• [in] timezone_posix: Default value of posix timezone string (Eg. “CST-8”). Can be kept NULL.
• [in] priv_data: (Optional) Private data associated with the service. This should stay allocated throughout the lifetime of the service.

```
esp_rmaker_device_t *esp_rmaker_create_schedule_service(const char *serv_name, esp_rmaker_device_write_cb_t write_cb, esp_rmaker_device_read_cb_t read_cb, int max_schedules, void *priv_data)
```

Create a standard Schedule service
This creates a Schedule service with the mandatory parameters. The default parameter names will be used. Refer esp_rmaker_standard_params.h for default names.

**Return** service_handle on success.
**Return** NULL in case of any error.

**Parameters**
• [in] serv_name: The unique service name
• [in] read_cb: Read callback.
• [in] max_schedules: Maximum number of schedules supported.
• [in] priv_data: (Optional) Private data associated with the service. This should stay allocated throughout the lifetime of the service.
Create a standard Scenes service

This creates a Scenes service with the mandatory parameters. The default parameter names will be used. Refer esp_rmaker_standard_params.h for default names.

**Return** service_handle on success.

**Return** NULL in case of any error.

**Parameters**

- **[in]** serv_name: The unique service name
- **[in]** write_cb: Write callback.
- **[in]** read_cb: Read callback.
- **[in]** max_scenes: Maximum number of scenes supported.
- **[in]** deactivation_support: Deactivation callback support.
- **[in]** priv_data: (Optional) Private data associated with the service. This should stay allocated throughout the lifetime of the service.

Create a standard System service

This creates an empty System service. Appropriate parameters should be added by the caller.

**Return** service_handle on success.

**Return** NULL in case of any error.

**Parameters**

- **[in]** serv_name: The unique service name
- **[in]** priv_data: (Optional) Private data associated with the service. This should stay allocated throughout the lifetime of the service.

Create a standard Local Control service

This creates a Local Control service with the mandatory parameters. The default parameter names will be used. Refer esp_rmaker_standard_params.h for default names.

**Return** service_handle on success.

**Return** NULL in case of any error.

**Parameters**

- **[in]** serv_name: The unique service name
- **[in]** pop: Proof of possession
1.3 RainMaker MQTT

1.3.1 Header File

- esp_rainmaker/include/esp_rmaker_mqtt.h

1.3.2 Functions

esp_rmaker_mqtt_conn_params_t *esp_rmaker_mqtt_get_conn_params (void)

- Returns ESP_OK on success.
- Returns error in case of any error.

Parameters
- [in] config: The MQTT configuration data

void esp_rmaker_mqtt_deinit (void)

esp_err_t esp_rmaker_mqtt_init (esp_rmaker_mqtt_conn_params_t *conn_params)

- Initialize ESP RainMaker MQTT
- Parameters
- [in] config: The MQTT configuration data

esp_err_t esp_rmaker_mqtt_connect (void)

- MQTT Connect
- Starts the connection attempts to the MQTT broker as per the configuration provided during initializing. This should ideally be called after successful network connection.
- Returns ESP_OK on success.
- Returns error in case of any error.

esp_err_t esp_rmaker_mqtt_disconnect (void)

- MQTT Disconnect
- Disconnects from the MQTT broker.
- Returns ESP_OK on success.
- Returns error in case of any error.

esp_err_t esp_rmaker_mqtt_publish (const char *topic, void *data, size_t data_len, uint8_t qos, int *msg_id)

- Publish MQTT Message
- Returns ESP_OK on success.
- Returns error in case of any error.

Parameters
- [in] topic: The MQTT topic on which the message should be published.
• [in] data: Data to be published
• [in] data_len: Length of the data
• [in] qos: Quality of Service for the Publish. Can be 0, 1 or 2. Also depends on what the MQTT broker supports.

esp_err_t esp_rmaker_mqtt_subscribe(const char *topic, esp_rmaker_mqtt_subscribe_cb_t cb, uint8_t qos, void *priv_data)

Subscribe to MQTT topic

Return ESP_OK on success.
Return error in case of any error.

Parameters
• [in] topic: The topic to be subscribed to.
• [in] cb: The callback to be invoked when a message is received on the given topic.
• [in] priv_data: Optional private data to be passed to the callback
• [in] qos: Quality of Service for the Subscription. Can be 0, 1 or 2. Also depends on what the MQTT broker supports.

esp_err_t esp_rmaker_mqtt_unsubscribe(const char *topic)

Unsubscribe from MQTT topic

Return ESP_OK on success.
Return error in case of any error.

Parameters
• [in] topic: Topic from which to unsubscribe.

esp_err_t esp_rmaker_mqtt_setup(esp_rmaker_mqtt_config_t mqtt_config)

void esp_rmaker_create_mqtt_topic(char *buf, size_t buf_size, const char *topic_suffix, const char *rule)

Creates appropriate MQTT Topic String based on CONFIG_ESP_RMAKER_MQTT_USE_BASIC_INGEST_TOPICS

Parameters
• [out] buf: Buffer to hold topic string
• [in] buf_size: Size of buffer
• [in] topic_suffix: MQTT Topic suffix
• [in] rule: Basic Ingests Rule Name

bool esp_rmaker_mqtt_is_budget_available(void)

Check if budget is available to publish an mqtt message.

Return true if budget is available
Return false if budget is exhausted

Note esp_rmaker_mqtt_publish API already does this check. In addition to that, some use-cases might still need to check for this.
bool esp_rmaker_is_mqtt_connected()
    Check if device is connected to MQTT Server.

    Return true if device is connected
    Return false if device is not connected

1.4 RainMaker OTA

1.4.1 Header File

- esp_rainmaker/include/esp_rmaker_ota.h

1.4.2 Functions

esp_err_t esp_rmaker_ota_enable(esp_rmaker_ota_config_t *ota_config, esp_rmaker_ota_type_t type)
    Enable OTA

    Calling this API enables OTA as per the ESP RainMaker specification. Please check the various ESP RainMaker configuration options to use the different variants of OTA. Refer the documentation for additional details.

    Return ESP_OK on success
    Return error on failure

Parameters

- [in] ota_config: Pointer to an OTA configuration structure
- [in] type: The OTA workflow type

esp_err_t esp_rmaker_ota_report_status(esp_rmaker_ota_handle_t ota_handle, ota_status_t status, char *additional_info)
    Report OTA Status

    This API must be called from the OTA Callback to indicate the status of the OTA. The OTA_STATUS_IN_PROGRESS can be reported multiple times with appropriate additional information. The final success/failure should be reported only once, at the end.

    This can be ignored if you are using the default internal OTA callback.

    Return ESP_OK on success
    Return error on failure

Parameters

- [in] ota_handle: The OTA handle received by the callback
- [in] status: Status to be reported
- [in] additional_info: NULL terminated string indicating additional information for the status
esp_err_t esp_rmaker_ota_default_cb(esp_rmaker_ota_handle_t handle, esp_rmaker_ota_data_t *ota_data)

Default OTA callback

This is the default OTA callback which will get used if you do not pass your own callback. You can call this even from your callback, in case you want better control on when the OTA can proceed and yet let the actual OTA process be managed by the RainMaker Core.

Return ESP_OK if the OTA was successful
Return ESP_FAIL if the OTA failed.

Parameters

- [in] handle: An OTA handle assigned by the ESP RainMaker Core
- [in] ota_data: The data to be used for the OTA

esp_err_t esp_rmaker_ota_fetch(void)

Fetch OTA Info

For OTA using Topics, this API can be used to explicitly ask the backend if an OTA is available. If it is, then the OTA callback would get invoked.

Return ESP_OK if the OTA fetch publish message was successful.
Return error on failure

esp_err_t esp_rmaker_ota_fetch_with_delay(int time)

Fetch OTA Info with a delay

For OTA using Topics, this API can be used to explicitly ask the backend if an OTA is available after a delay (in seconds) passed as an argument.

Return ESP_OK if the OTA fetch timer was created.
Return error on failure

Parameters

- [in] time: Delay (in seconds)

esp_err_t esp_rmaker_ota_mark_valid(void)

Mark OTA as valid

This should be called if the OTA validation has been kept pending by returning OTA_DIAG_STATUS_PENDING in the ota_diag callback and then, the validation was eventually successful. This can also be used to mark the OTA valid even before RainMaker core does its own validations (primarily MQTT connection).

Return ESP_OK on success
Return error on failure

esp_err_t esp_rmaker_ota_mark_invalid(void)

Mark OTA as invalid

This should be called if the OTA validation has been kept pending by returning OTA_DIAG_STATUS_PENDING in the ota_diag callback and then, the validation eventually failed. This can even be used to rollback at any point of time before RainMaker core’s internal logic and the application’s logic mark the OTA as valid.
Return ESP_OK on success
Return error on failure

1.4.3 Structures

struct esp_rmaker_ota_data_t
OTA Data

Public Members

char *url
The OTA URL received from ESP RainMaker Cloud

int filesize
Size of the OTA File. Can be 0 if the file size isn’t received from the ESP RainMaker Cloud

char *fw_version
The firmware version of the OTA image

char *ota_job_id
The OTA Job ID received from cloud

const char *server_cert
The server certificate passed in esp_rmaker_enable_ota()

char *priv
The private data passed in esp_rmaker_enable_ota()

char *metadata
OTA Metadata. Applicable only for OTA using Topics. Will be received (if applicable) from the backend, along with the OTA URL

struct esp_rmaker_ota_diag_priv_t

Public Members

esp_rmaker_ota_diag_state_t state
OTA diagnostic state

bool rmaker_ota
Flag to indicate whether the OTA which has triggered the Diagnostics checks for rollback was triggered via RainMaker or not. This would be useful only when your application has some other mechanism for OTA too.

struct esp_rmaker_ota_config_t
ESP RainMaker OTA Configuration
Public Members

`esp_rmaker_ota_cb_t ota_cb`
OTA Callback. The callback to be invoked when an OTA Job is available. If kept NULL, the internal default callback will be used (Recommended).

`esp_rmaker_post_ota_diag_t ota_diag`
OTA Diagnostics Callback. A post OTA diagnostic handler to be invoked if app rollback feature is enabled. If kept NULL, the new firmware will be assumed to be fine, and no rollback will be performed.

`const char *server_cert`
Server Certificate. The certificate to be passed to the OTA callback for server authentication. This is mandatory, unless you have disabled it in ESP HTTPS OTA config option. If you are using the ESP RainMaker OTA Service, you can just set this to `ESP_RMAKER_OTA_DEFAULT_SERVER_CERT`.

`void *priv`
Private Data. Optional private data to be passed to the OTA callback.

1.4.4 Type Definitions

```c
typedef void *esp_rmaker_ota_callback_t
The OTA Handle to be used by the OTA callback
```

```c
typedef esp_err_t (*esp_rmaker_ota_callback_t)(esp_rmaker_ota_callback_t handle,
                           esp_rmaker_ota_data_t *ota_data)
```
Function prototype for OTA Callback

This function will be invoked by the ESP RainMaker core whenever an OTA is available. The `esp_rmaker_report_ota_status()` API should be used to indicate the progress and success/fail status.

`Return` ESP_OK if the OTA was successful

`Return` ESP_FAIL if the OTA failed.

Parameters

- `[in] handle: An OTA handle assigned by the ESP RainMaker Core`
- `[in] ota_data: The data to be used for the OTA`

```c
typedef esp_rmaker_ota_diag_status_t (*esp_rmaker_post_ota_diag_t)(esp_rmaker_ota_diag_priv_t
                           *ota_diag_priv, void
                           *priv)
```
Function Prototype for Post OTA Diagnostics

If the Application rollback feature is enabled, this callback will be invoked as soon as you call `esp_rmaker_ota_enable()`, if it is the first boot after an OTA. You may perform some application specific diagnostics and report the status which will decide whether to roll back or not.

This will be invoked once again after MQTT has connected, in case some additional validations are to be done later.

If OTA state == OTA_DIAG_STATE_INIT, then return OTA_DIAG_STATUS_FAIL to indicate failure and rollback. return OTA_DIAG_STATUS_SUCCESS or OTA_DIAG_STATUS_PENDING to tell internal OTA logic to continue further.

If OTA state == OTA_DIAG_STATE_POST_MQTT, then return OTA_DIAG_STATUS_FAIL to indicate failure and rollback. return OTA_DIAG_STATUS_SUCCESS to indicate validation was successful and mark OTA as valid return OTA_DIAG_STATUS_PENDING to indicate that some additional validations will
be done later and the OTA will eventually be marked valid/invalid using esp_rmaker_ota_mark_valid() or esp_rmaker_ota_mark_invalid() respectively.

Return esp_rmaker_ota_diag_status_t as applicable

1.4.5 Enumerations

define esp_rmaker_ota_event_t
  ESP RainMaker Events
  Values:
    RMAKER_OTA_EVENT_INVALID = 0
    RMAKER_OTA_EVENT_STARTING
    RainMaker OTA is Starting
    RMAKER_OTA_EVENT_IN_PROGRESS
    RainMaker OTA has Started
    RMAKER_OTA_EVENT_SUCCESSFUL
    RainMaker OTA Successful
    RMAKER_OTA_EVENT_FAILED
    RainMaker OTA Failed
    RMAKER_OTA_EVENT_REJECTED
    RainMaker OTA Rejected
    RMAKER_OTA_EVENT_DELAYED
    RainMaker OTA Delayed
    RMAKER_OTA_EVENT_REQ_FOR_REBOOT
    OTA Image has been flashed and active partition changed. Reboot is requested. Applicable only if Auto reboot is disabled

define ota_status_t
  OTA Status to be reported to ESP RainMaker Cloud
  Values:
    OTA_STATUS_IN_PROGRESS = 1
    OTA is in Progress. This can be reported multiple times as the OTA progresses.
    OTA_STATUS_SUCCESS
    OTA Succeeded. This should be reported only once, at the end of OTA.
    OTA_STATUS_FAILED
    OTA Failed. This should be reported only once, at the end of OTA.
    OTA_STATUS_DELAYED
    OTA was delayed by the application
    OTA_STATUS_REJECTED
    OTA rejected due to some reason (wrong project, version, etc.)

define esp_rmaker_ota_type_t
  OTA Workflow type
  Values:
    OTA_USING_PARAMS = 1
    OTA will be performed using services and parameters.
OTA USING TOPICS
OTA will be performed using pre-defined MQTT topics.

enum esp_rmaker_ota_diag_status_t
Values:

OTA_DIAG_STATUS_FAIL
OTA Diagnostics Failed. Rollback the firmware.

OTA_DIAG_STATUS_PENDING
OTA Diagnostics Pending. Additional validations will be done later.

OTA_DIAG_STATUS_SUCCESS
OTA Diagnostics Succeeded. Firmware can be considered valid.

enum esp_rmaker_ota_diag_state_t
Values:

OTA_DIAG_STATE_INIT
OTA State: Initialised.

OTA_DIAG_STATE_POST_MQTT
OTA state: MQTT has connected.

1.5 RainMaker Console

1.5.1 Header File

- esp_rainmaker/include/esp_rmaker_console.h

1.5.2 Functions

esp_err_t esp_rmaker_console_init (void)
Initializes console

  Initializes serial console and adds basic commands.

  Return  ESP_OK on success.

  Return  error in case of failures.

1.6 RainMaker Common

1.6.1 Utilities

Header File

- rmaker_common/include/esp_rmaker_utils.h
Functions

esp_err_t esp_rmaker_reboot (int8_t seconds)
Reboot the device after a delay

This API just starts a reboot timer and returns immediately. The actual reboot is triggered asynchronously in the
timer callback. This is useful if you want to reboot after a delay, to allow other tasks to finish their operations
(Eg. MQTT publish to indicate OTA success). The RMAKER_EVENT_REBOOT event is triggered when the
reboot timer is started.

Return ESP_OK on success.
Return error on failure.

Parameters
• [in] seconds: Time in seconds after which the device should reboot.

esp_err_t esp_rmaker_wifi_reset (int8_t reset_seconds, int8_t reboot_seconds)
Reset Wi-Fi credentials and (optionally) reboot

This will reset just the Wi-Fi credentials and (optionally) trigger a reboot. This is useful when you
want to keep all the entries in NVS memory intact, but just change the Wi-Fi credentials. The
RMAKER_EVENT_WIFI_RESET event is triggered when this API is called. The actual reset will happen
after a delay if reset_seconds is not zero.

Note This reset and reboot operations will happen asynchronously depending on the values passed to the API.

Return ESP_OK on success.
Return error on failure.

Parameters
• [in] reset_seconds: Time in seconds after which the reset should get triggered. This will help
other modules take some actions before the device actually resets. If set to zero, the operation would
be performed immediately.
• [in] reboot_seconds: Time in seconds after which the device should reboot. If set to negative
value, the device will not reboot at all.

esp_err_t esp_rmaker_factory_reset (int8_t reset_seconds, int8_t reboot_seconds)
Reset to factory defaults and reboot

This will clear entire NVS partition and (optionally) trigger a reboot. The
RMAKER_EVENT_FACTORY_RESET event is triggered when this API is called. The actual reset
will happen after a delay if reset_seconds is not zero.

Note This reset and reboot operations will happen asynchronously depending on the values passed to the API.

Return ESP_OK on success.
Return error on failure.

Parameters
• [in] reset_seconds: Time in seconds after which the reset should get triggered. This will help
other modules take some actions before the device actually resets. If set to zero, the operation would
be performed immediately.
• [in] reboot_seconds: Time in seconds after which the device should reboot. If set to negative
value, the device will not reboot at all.
Initialize time synchronization

This API initializes SNTP for time synchronization.

Return  ESP_OK on success
Return  error on failure

Parameters
  • [in] config: Configuration to be used for SNTP time synchronization. The default configuration is used if NULL is passed.

Check if current time is updated

This API checks if the current system time is updated against the reference time of 1-Jan-2019.

Return  true if time is updated
Return  false if time is not updated

Wait for time synchronization

This API waits for the system time to be updated against the reference time of 1-Jan-2019. This is a blocking call.

Return  ESP_OK on success
Return  error on failure

Parameters
  • [in] ticks_to_wait: Number of ticks to wait for time synchronization. Accepted values: 0 to portMAX_DELAY.

Set POSIX timezone

Set the timezone (TZ environment variable) as per the POSIX format specified in the GNU libc documentation. Eg. For China: “CST-8” For US Pacific Time (including daylight saving information): “PST8PDT,M3.2.0,M11.1.0”

Return  ESP_OK on success
Return  error on failure

Parameters
  • [in] tz_posix: NULL terminated TZ POSIX string

Set timezone location string

Set the timezone as a user friendly location string. Check here for a list of valid values.
Eg. For China: “Asia/Shanghai” For US Pacific Time: “America/Los_Angeles”

Note Setting timezone using this API internally also sets the POSIX timezone string.
Return ESP_OK on success
Return error on failure

Parameters

• [in] tz: NULL terminated Timezone location string

char *esp_rmaker_time_get_timezone_posix (void)
Get the current POSIX timezone
This fetches the current timezone in POSIX format, read from NVS.

Return Pointer to a NULL terminated POSIX timezone string on success. Freeing this is the responsibility of the caller.
Return NULL on failure.

char *esp_rmaker_time_get_timezone (void)
Get the current timezone
This fetches the current timezone in POSIX format, read from NVS.

Return Pointer to a NULL terminated timezone string on success. Freeing this is the responsibility of the caller.
Return NULL on failure.

esp_err_t esp_rmaker_get_local_time_str (char *buf, size_t buf_len)
Get printable local time string
Get a printable local time string, with information of timezone and Daylight Saving. Eg. “Tue Sep 1 09:04:38 2020 -0400[EDT], DST: Yes” “Tue Sep 1 21:04:04 2020 +0800[CST], DST: No”

Return ESP_OK on success
Return error on failure

Parameters

• [out] buf: Pointer to a pre-allocated buffer into which the time string will be populated.
• [in] buf_len: Length of the above buffer.

Structures

struct esp_rmaker_time_config

Public Members

char *sntp_server_name
If not specified, then ‘CONFIG_ESP_RMAKER_SNTP_SERVER_NAME’ is used as the SNTP server.

snntp_sync_time_cb_t sync_time_cb
Optional callback to invoke, whenever time is synchronised. This will be called periodically as per the SNTP polling interval (which is 60min by default). If kept NULL, the default callback will be invoked, which will just print the current local time.
Macros

MEM_ALLOC_EXTRAM(size)
MEM_CALLOC_EXTRAM(num, size)
MEM_REALLOC_EXTRAM(ptr, size)

Type Definitions

typedef struct esp_rmaker_time_config esp_rmaker_time_config_t

1.6.2 Factory Storage

Header File

- rmaker_common/include/esp_rmaker_factory.h

Functions

esp_err_t esp_rmaker_factory_init(void)
Initialize Factory NVS
This initializes the Factory NVS partition which will store data that should not be cleared even after a reset to factory.

Return ESP_OK on success.
Return error on failure.

void *esp_rmaker_factory_get(const char *key)
Get value from factory NVS
This will search for the specified key in the Factory NVS partition, allocate the required memory to hold it, copy the value and return the pointer to it. It is responsibility of the caller to free the memory when the value is no more required.

Return pointer to the value on success.
Return NULL on failure.

Parameters

- [in] key: The key of the value to be read from factory NVS.

size_t esp_rmaker_factory_get_size(const char *key)
Get size of value from factory NVS
This will search for the specified key in the Factory NVS partition, and return the size of the value associated with the key.

Return size of the value on success.
Return 0 on failure.
Parameters
• [in] **key**: The key of the value to be read from factory NVS.

```c
esp_err_t esp_rmaker_factory_set (const char *key, void *value, size_t len)
```

Set a value in factory NVS

This will write the value for the specified key into factory NVS.

**Return** ESP_OK on success.

**Return** error on failure.

**Parameters**
- [in] **key**: The key for the value to be set in factory NVS.
- [in] **data**: Pointer to the value.
- [in] **len**: Length of the value.

### 1.6.3 Work Queue

**Header File**

- rmaker_common/include/esp_rmaker_work_queue.h

**Functions**

```c
esp_err_t esp_rmaker_work_queue_init (void)
```

Initializes the Work Queue

This initializes the work queue, which is basically a mechanism to run tasks in the context of a dedicated thread. You can start queueing tasks after this, but they will get executed only after calling esp_rmaker_work_queue_start().

**Return** ESP_OK on success.

**Return** error in case of failure.

```c
esp_err_t esp_rmaker_work_queue_deinit (void)
```

De-initialize the Work Queue

This de-initializes the work queue. Note that the work queue needs to be stopped using esp_rmaker_work_queue_stop() before calling this.

**Return** ESP_OK on success.

**Return** error in case of failure.

```c
esp_err_t esp_rmaker_work_queue_start (void)
```

Start the Work Queue

This starts the Work Queue thread which then starts executing the tasks queued.

**Return** ESP_OK on success.

**Return** error in case of failure.
esp_err_t esp_rmaker_work_queue_stop (void)
    Stop the Work Queue
    This stops a running Work Queue.

    Return  ESP_OK on success.
    Return  error in case of failure.

esp_err_t esp_rmaker_work_queue_add_task (esp_rmaker_work_fn_t work_fn, void *priv_data)
    Queue execution of a function in the Work Queue’s context
    This API queues a work function for execution in the Work Queue Task’s context.

    Return  ESP_OK on success.
    Return  error in case of failure.

Parameters
    •  [in]  work_fn: The Work function to be queued.
    •  [in]  priv_data: Private data to be passed to the work function.

Type Definitions

typedef void (*esp_rmaker_work_fn_t) (void *priv_data)
    Prototype for ESP RainMaker Work Queue Function

Parameters
    •  [in]  priv_data: The private data associated with the work function.

1.6.4 Common Events

Header File

    •  rmaker_common/include/esp_rmaker_common_events.h

Functions

ESP_EVENT_DECLARE_BASE (RMAKER_COMMON_EVENT)
    ESP RainMaker Common Event Base

Enumerations

enum esp_rmaker_common_event_t
    Values:

    RMAKER_EVENT_REBOOT
        Node reboot has been triggered. The associated event data is the time in seconds (type: uint8_t) after which
        the node will reboot. Note that this time may not be accurate as the events are received asynchronously.

    RMAKER_EVENT_WIFI_RESET
        Wi-Fi credentials reset. Triggered after calling esp_rmaker_wifi_reset()
**RMAKER_EVENT_FACTORY_RESET**
Node reset to factory defaults. Triggered after calling esp_rmaker_factory_reset()

**RMAKER_MQTT_EVENT_CONNECTED**
Connected to MQTT Broker

**RMAKER_MQTT_EVENT_DISCONNECTED**
Disconnected from MQTT Broker

**RMAKER_MQTT_EVENT_PUBLISHED**
MQTT message published successfully. Event data will contain the message ID (integer) of published message.

**RMAKER_EVENT_TZ_POSIX_CHANGED**
POSIX Timezone Changed. Associated data would be NULL terminated POSIX Timezone Eg. “PST8PDT,M3.2.0,M11.1.0”

**RMAKER_EVENT_TZ_CHANGED**
Timezone Changed. Associated data would be NULL terminated Timezone. Eg. “America/Los_Angeles”
Note that whenever this event is received, the RMAKER_EVENT_TZ_POSIX_CHANGED event will also be received, but not necessarily vice versa.

**RMAKER_MQTT_EVENT_MSG_DELETED**
MQTT message deleted from the outbox if the message couldn’t have been sent and acknowledged. Event data will contain the message ID (integer) of deleted message. Valid only if CONFIG_MQTT_REPORT_DELETED_MESSAGES is enabled.
2.1 Library

2.1.1 User

class rmaker_lib.user.User(username)
   User class used to instantiate instances of user to perform various user signup/login operations.

   Parameters
   username (str) – Name of User

   forgot_password(password=None, verification_code=None)
   Forgot password request to reset the password.

   Parameters
   • password (str) – Password of user, defaults to None
   • verification_code (int) – Verification code received during forgot password request, defaults to None

   Raises
   • NetworkError – If there is a network connection issue during password reset
   • Exception – If there is an HTTP issue during forgot password

   Returns True on Success

   Return type bool

handle_otp_based_login(login_session=None)
   OTP based login for ESP RainMaker.

   Parameters
   login_session (str) – Session param received in first login request

   Raises
   • NetworkError – If there is a network connection issue during login
   • Exception – If there is an HTTP issue during login or JSON format issue in HTTP response
   • AuthenticationError – If login failed with the given parameters

   Returns rmaker_lib.session.Session on Success

   Return type Dict

login(password=None)
   User login to the ESP Rainmaker.
Parameters **password** *(str)* – Password of user, defaults to *None*

Raises

- **NetworkError** – If there is a network connection issue during login
- **Exception** – If there is an HTTP issue during login or JSON format issue in HTTP response
- **AuthenticationError** – If login failed with the given parameters

Returns *rmaker_lib.session.Session* on Success

Return type *object*

**signup** *(code)*

Sign up of new User for ESP Rainmaker.

Parameters **code** *(int)* – Verification code received in signup request for user

Raises

- **NetworkError** – If there is a network connection issue during signup
- **Exception** – If there is an HTTP issue during signup

Returns True on Success

Return type *bool*

**signup_request** *(password)*

Sign up request of new User for ESP Rainmaker.

Parameters **password** *(str)* – Password to set for new user

Raises

- **NetworkError** – If there is a network connection issue during signup request
- **Exception** – If there is an HTTP issue during signup request

Returns True on Success

Return type *bool*

### 2.1.2 Session

**class rmaker_lib.session.Session**

Session class for logged in user.

**get_mqtt_host ()**

Get the MQTT Host endpoint.

Raises

- **NetworkError** – If there is a network connection issue while getting MQTT Host endpoint
- **Exception** – If there is an HTTP issue while getting MQTT host or JSON format issue in HTTP response

Returns MQTT Host on Success, None on Failure

Return type *str | None*

**get_nodes ()**

Get list of all nodes associated with the user.
2.1.3 Node

```python
class rmaker_lib.node.Node(nodeid, session)
```
Node class used to instantiate instances of node to perform various node operations.

Parameters

- **nodeid** (*str*) – Node Id of node
- **session** (*object*) – `rmaker_lib.session.Session`

## Raises

- **NetworkError** – If there is a network connection issue while getting nodes associated with user
- **Exception** – If there is an HTTP issue while getting nodes

## Returns
Nodes associated with user on Success

## Return type
`dict`
add_user_for_sharing\((data)\)

Perform sharing operations -

1. Request to add user for sharing nodes

Parameters data (dict) – 1. To add nodes - Data containing user_name and nodes as keys

Raises

• SSLError – If there is an SSL issue
• HTTPError – If the HTTP response is an HTTPError
• NetworkError – If there is a network connection issue
• Timeout – If there is a timeout issue
• RequestException – If there is an issue during the HTTP request
• Exception – If there is an HTTP issue while performing sharing operation or JSON format issue in HTTP response

Returns HTTP response on Success

Return type dict

add_user_node_mapping\((secret_key)\)

Add user node mapping.

Parameters secret_key (str) – The randomly generated secret key that will be used for User-Node mapping

Raises

• NetworkError – If there is a network connection issue while adding user node mapping
• Exception – If there is an HTTP issue while adding user node mapping or JSON format issue in HTTP response

Returns Request Id on Success, None on Failure

Return type str | None

get_mapping_status\((request_id)\)

Check status of user node mapping request.

Parameters request_id (str) – Request Id

Raises

• NetworkError – If there is a network connection issue while getting user node mapping status
• Exception – If there is an HTTP issue while getting user node mapping status or JSON format issue in HTTP response

Returns Request Status on Success, None on Failure

Type str | None

get_node_config\()

Get node configuration.

Raises

• NetworkError – If there is a network connection issue while getting node configuration
• Exception – If there is an HTTP issue while getting node config

## Returns
Configuration of node on Success

## Return type
dict

### get_node_params ()
Get parameters of the node.

## Raises

- **NetworkError** – If there is a network connection issue while getting node params
- **Exception** – If there is an HTTP issue while getting node params or JSON format issue in HTTP response

## Returns
Node Parameters on Success, None on Failure

## Return type
dict | None

### get_node_status ()
Get online/offline status of the node.

## Raises

- **NetworkError** – If there is a network connection issue while getting node status
- **Exception** – If there is an HTTP issue while getting node status

## Returns
Status of node on Success

## Return type
dict

### get_nodeid ()
Get nodeid of device

## Returns
Node Id of node on Success

## Return type
str

### get_shared_nodes_request (params)
Get request sent to share nodes with user

## Parameters

- **params** *(dict)* – Query parameters containing request_id and primary_user as keys

## Raises

- **SSLError** – If there is an SSL issue
- **HTTPError** – If the HTTP response is an HTTPError
- **NetworkError** – If there is a network connection issue
- **Timeout** – If there is a timeout issue
- **RequestException** – If there is an issue during the HTTP request
- **Exception** – If there is an HTTP issue while getting sharing request or JSON format issue in HTTP response

## Returns
HTTP response on Success

## Return type
dict

### get_sharing_details_of_nodes ()
Get sharing details of nodes associated with user

## Raises

2.1. Library 55
• **SSLError** – If there is an SSL issue
• **HTTPError** – If the HTTP response is an HTTPError
• **NetworkError** – If there is a network connection issue
• **Timeout** – If there is a timeout issue
• **RequestException** – If there is an issue during the HTTP request
• **Exception** – If there is an HTTP issue while getting shared nodes or JSON format issue in HTTP response

Returns HTTP response on Success

Return type dict

**remove_shared_nodes_request**( req_id )
Remove/Cancel request sent to share nodes with user

Parameters **req_id** – Id of sharing request

Raises
• **SSLError** – If there is an SSL issue
• **HTTPError** – If the HTTP response is an HTTPError
• **NetworkError** – If there is a network connection issue
• **Timeout** – If there is a timeout issue
• **RequestException** – If there is an issue during the HTTP request
• **Exception** – If there is an HTTP issue while removing sharing request or JSON format issue in HTTP response

Returns HTTP response on Success

Return type dict

**remove_user_from_shared_nodes**( data )
Remove user from shared nodes

Parameters **data**(dict) – Data containing user_name and nodes as keys

Raises
• **SSLError** – If there is an SSL issue
• **HTTPError** – If the HTTP response is an HTTPError
• **NetworkError** – If there is a network connection issue
• **Timeout** – If there is a timeout issue
• **RequestException** – If there is an issue during the HTTP request
• **Exception** – If there is an HTTP issue while removing shared nodes or JSON format issue in HTTP response

Returns HTTP response on Success

Return type dict

**remove_user_node_mapping**( )
Remove user node mapping request.

Raises
• **NetworkError** – If there is a network connection issue while removing user node mapping

• **Exception** – If there is an HTTP issue while removing user node mapping or JSON format issue in HTTP response

**Returns**  
Request Id on Success, None on Failure

**Return type**  
str | None

**request_op** *(data)*  
Perform sharing operations -

1. Accept or decline sharing request

**Parameters**  
`data (dict)` – 1. Data containing `request_id` and `accept` as keys

**Raises**

• **SSLError** – If there is an SSL issue

• **HTTPError** – If the HTTP response is an HTTPError

• **NetworkError** – If there is a network connection issue

• **Timeout** – If there is a timeout issue

• **RequestException** – If there is an issue during the HTTP request

• **Exception** – If there is an HTTP issue while performing request operation or JSON format issue in HTTP response

**Returns**  
HTTP response on Success

**Return type**  
dict

**set_node_params** *(data)*  
Set parameters of the node.

**Parameters**  
`data (dict)` – Parameters to be set for the node

**Raises**

• **NetworkError** – If there is a network connection issue while setting node params

• **Exception** – If there is an HTTP issue while setting node params or JSON format issue in HTTP response

**Returns**  
True on Success

**Return type**  
bool

### 2.2 Commands

#### 2.2.1 Node

**rmaker_cmd.node.add_user_to_share_nodes** *(nodes=None, user=None)*  
Add user to share nodes

**Parameters**

• **vars (str)** – `nodes` as key - Node Id of the node(s)
• **vars** – *user* as key - User name

**Raises** Exception – If there is an issue while adding user to share nodes

**Returns** API response

**Return type** dict

```python
rmaker_cmd.node.claim_node(vars=None)
```

Claim the node connected to the given serial port (Get cloud credentials)

**Parameters**
- **vars** *(str | None)* – port as key - Serial Port, defaults to None

**Raises** Exception – If there is an HTTP issue while claiming

**Returns** None on Success

**Return type** None

```python
rmaker_cmd.node.get_mqtt_host(vars=None)
```

Returns MQTT Host endpoint

**Parameters**
- **vars** *(dict | None)* – No Parameters passed, defaults to None

**Raises**
- **NetworkError** – If there is a network connection issue while getting MQTT Host endpoint

- **Exception** – If there is an HTTP issue while getting MQTT Host endpoint or JSON format issue in HTTP response

**Returns** MQTT Host endpoint

**Return type** str

```python
rmaker_cmd.node.get_node_config(vars=None)
```

Shows the configuration of the node.

**Parameters**
- **vars** *(dict | None)* – nodeid as key - Node ID for the node, defaults to None

**Raises** Exception – If there is an HTTP issue while getting node config

**Returns** None on Success

**Return type** None

```python
rmaker_cmd.node.get_node_status(vars=None)
```

Shows the online/offline status of the node.

**Parameters**
- **vars** *(dict | None)* – nodeid as key - Node ID for the node, defaults to None

**Raises** Exception – If there is an HTTP issue while getting node status

**Returns** None on Success

**Return type** None

```python
rmaker_cmd.node.get_nodes(vars=None)
```

List all nodes associated with the user.

**Parameters**
- **vars** *(dict | None)* – No Parameters passed, defaults to None

**Raises** Exception – If there is an HTTP issue while getting nodes

**Returns** None on Success

**Return type** None
**rmaker_cmd.node.get_params**(vars=None)

Get parameters of the node.

**Parameters**

- **vars**(dict | None) – nodeid as key - Node ID for the node, defaults to None

**Raises** Exception – If there is an HTTP issue while getting params or JSON format issue in HTTP response

**Returns** None on Success

**Return type** None

**rmaker_cmd.node.list_sharing_details**(node_id=None, primary_user=False, request_id=None, list_requests=False)

List sharing details of all nodes associated with user or List pending requests

**Parameters**

- **vars**(bool) – node_id as key - Node Id of the node(s) (if not provided, is set to all nodes associated with user)

- **vars** – primary_user as key - User is primary or secondary (if provided, user is primary user)

- **vars** – request_id as key - Id of sharing request

- **vars** – list_requests as key - If True, list pending requests If False, list sharing details of nodes

**Raises** Exception – If there is an issue while listing details

**Returns** API response

**Return type** dict

**rmaker_cmd.node.ota_upgrade**(vars=None)

Upload OTA Firmware Image and Set image url returned in response as node params

**rmaker_cmd.node.remove_node**(vars=None)

Removes the user node mapping.

**Parameters**

- **vars**(dict | None) – nodeid as key - Node ID for the node, defaults to None

**Raises**

- **NetworkError** – If there is a network connection issue during HTTP request for removing node

- **Exception** – If there is an HTTP issue while removing node or JSON format issue in HTTP response

**Returns** None on Success

**Return type** None

**rmaker_cmd.node.remove_sharing**(nodes=None, user=None, request_id=None)

Remove user from shared nodes or Remove sharing request

**Parameters**

- **vars**(str) – nodes as key - Node Id for the node

- **vars** – user as key - User name

- **vars** – request_id as key - Id of sharing request

**Raises** Exception – If there is an issue while remove operation
Returns API response
Return type dict

rmaker_cmd.node.set_params(vars=None)
Set parameters of the node.

Parameters
vars (dict | None) – nodeid as key - Node ID for the node,
data as key - JSON data containing parameters to be set or
filepath as key - Path of the JSON file containing parameters to be set,
defaults to None

Raises Exception – If there is an HTTP issue while setting params or JSON format issue in HTTP response

Returns None on Success
Return type None

2.2.2 User

rmaker_cmd.user.forgot_password(vars=None)
Forgot password request to reset the password.

Parameters
vars (dict) – user_name as key - Email address/ phone number of the user, defaults to None

Raises Exception – If there is an HTTP issue while changing password for user

Returns None on Success and Failure
Return type None

rmaker_cmd.user.get_password()
Get Password as input and perform basic password validation checks.

Raises SystemExit – If there is an issue in getting password

Returns Password for User on Success
Return type str

rmaker_cmd.user.get_user_details(vars=None)
Get details of current logged-in user

rmaker_cmd.user.login(vars=None)
First time login of the user.
Parameters **vars** (*dict*) – *email* as key - Email address of the user, defaults to *None*

Raises **Exception** – If there is any issue in login for user

Returns None on Success and Failure

Return type None

rmaker_cmd.user.logout (*vars=None*)

Logout of the current session.

Raises **Exception** – If there is any issue in logout for user

Returns None on Success and Failure

Return type None

rmaker_cmd.user.signup (*vars=None*)

User signup to the ESP Rainmaker.

Parameters **vars** (*dict*) – *user_name* as key - Email address/Phone number of the user, defaults to *None*

Raises **Exception** – If there is any issue in signup for user

Returns None on Success

Return type None

### 2.2.3 Provision

rmaker_cmd.provision.provision (*vars=None*)

Provisioning of the node.

Raises

- **NetworkError** – If there is a network connection issue during provisioning
- **Exception** – If there is an HTTP issue during provisioning

Parameters **vars** (*dict*) – *pop* - Proof of Possession of the node, defaults to *None*

Returns None on Success and Failure

Return type None

### 2.2.4 Browser Login

rmaker_cmd.browserlogin.browser_login()

Opens browser with login url using Httpd Server.

Raises **Exception** – If there is an HTTP issue while logging in through browser

Returns None on Success and Failure

Return type None
r
rmaker_cmd.browserlogin, 61
rmaker_cmd.node, 57
rmaker_cmd.provision, 61
rmaker_cmd.user, 60
rmaker_lib.node, 53
rmaker_lib.session, 52
rmaker_lib.user, 51
add_user_for_sharing() (rmaker_lib.node.Node method), 53
add_user_node_mapping() (rmaker_lib.node.Node method), 54
add_user_to_share_nodes() (in module rmaker_cmd.node), 57

browser_login() (in module rmaker_cmd.browserlogin), 61

claim_node() (in module rmaker_cmd.node), 58

ESP_EVENT_DECLARE_BASE (C++ function), 48
esp_param_property_flags_t (C++ enum), 20
esp_rmaker_array (C++ function), 4
esp_rmaker_brightness_param_create (C++ function), 25
esp_rmaker_cct_param_create (C++ function), 26
esp_rmaker_common_event_t (C++ enum), 48
esp_rmaker_config_t (C++ class), 16
esp_rmaker_config_t::enable_time_sync (C++ member), 17
ESP_RMAKER_CONFIG_VERSION (C macro), 18
esp_rmaker_console_init (C++ function), 42
esp_rmaker_create_local_control_service (C++ function), 34
esp_rmaker_create_mqtt_topic (C++ function), 36
esp_rmaker_create_scenes_service (C++ function), 33
esp_rmaker_create_schedule_service (C++ function), 33
esp_rmaker_create_system_service (C++ function), 34
ESP_RMAKER_DEF_BRIGHTNESS_NAME (C macro), 30
ESP_RMAKER_DEF_CCT_NAME (C macro), 30
ESP_RMAKER_DEF_DIRECTION_NAME (C macro), 30
ESP_RMAKER_DEF_FACTORY_RESET_NAME (C macro), 31
ESP_RMAKER_DEF_HUE_NAME (C macro), 30
ESP_RMAKER_DEF_INTENSITY_NAME (C macro), 30
ESP_RMAKER_DEF_LOCAL_CONTROL_POP (C macro), 31
ESP_RMAKER_DEF_LOCAL_CONTROL_TYPE (C macro), 31
ESP_RMAKER_DEF_NAME_PARAM (C macro), 30
ESP_RMAKER_DEF_OTA_INFO_NAME (C macro), 30
ESP_RMAKER_DEF_OTA_STATUS_NAME (C macro), 30
ESP_RMAKER_DEF_OTA_URL_NAME (C macro), 30
ESP_RMAKER_DEF_POWER_NAME (C macro), 30
ESP_RMAKER_DEF_REBOOT_NAME (C macro), 31
ESP_RMAKER_DEF_SCENES_NAME (C macro), 31
ESP_RMAKER_DEF_SCHEDULE_NAME (C macro), 30
ESP_RMAKER_DEF_SATURATION_NAME (C macro), 30
ESP_RMAKER_DEF_TEMPERATURE_NAME (C macro), 30
ESP_RMAKER_DEF_TIMEZONE_NAME (C macro), 30
ESP_RMAKER_DEF_TIMEZONE_POSIX_NAME (C macro), 30
ESP_RMAKER_DEF_WIFI_RESET_NAME (C macro), 31
esp_rmaker_device_add_attribute (C++ function), 8
esp_rmaker_device_add_cb (C++ function), 9
esp_rmaker_device_add_model (C++ function), 9
esp_rmaker_device_add_param (C++ function), 10
esp_rmaker_device_add_subtype (C++ function), 9
ESP_RMAKER_DEVICE_AIR_CONDITIONER (C macro), 24
ESP_RMAKER_DEF_BRIGHTNESS_NAME (C macro), 30
esp_rmaker_node_info_t::fw_version (C++ member), 16
esp_rmaker_node_info_t::model (C++ member), 16
esp_rmaker_node_info_t::name (C++ member), 16
esp_rmaker_node_info_t::secure_boot_digest (C++ member), 16
esp_rmaker_node_info_t::type (C++ member), 16
esp_rmaker_node_info_t::subtype (C++ member), 16
esp_rmaker_node_init (C++ function), 4
esp_rmaker_node_remove_device (C++ function), 8
esp_rmaker_node_t (C++ type), 18
esp_rmaker_obj (C++ function), 4
esp_rmaker_ota_cb_t (C++ type), 40
esp_rmaker_ota_config_t (C++ class), 39
esp_rmaker_ota_config_t::ota_cb (C++ member), 40
esp_rmaker_ota_config_t::ota_diag (C++ member), 40
esp_rmaker_ota_config_t::priv (C++ member), 40
esp_rmaker_ota_config_t::server_cert (C++ member), 40
esp_rmaker_ota_data_t (C++ class), 39
esp_rmaker_ota_data_t::filesize (C++ member), 39
esp_rmaker_ota_data_t::fw_version (C++ member), 39
esp_rmaker_ota_data_t::metadata (C++ member), 39
esp_rmaker_ota_data_t::ota_job_id (C++ member), 39
esp_rmaker_ota_data_t::priv (C++ member), 39
esp_rmaker_ota_data_t::server_cert (C++ member), 39
esp_rmaker_ota_data_t::url (C++ member), 39
esp_rmaker_ota_default_cb (C++ function), 37
esp_rmaker_ota_diag_priv_t::ota (C++ member), 39
esp_rmaker_ota_diag_priv_t::state (C++ member), 39
esp_rmaker_ota_diag_state_t (C++ enum), 42
esp_rmaker_ota_diag_status_t (C++ enum), 42
esp_rmaker_ota_enable (C++ function), 37
esp_rmaker_ota_enable_default (C++ function), 15
esp_rmaker_ota_event_t (C++ enum), 41
esp_rmaker_ota_fetch (C++ function), 38
esp_rmaker_ota_fetch_with_delay (C++ function), 38
esp_rmaker_ota_handle_t (C++ type), 40
esp_rmaker_ota_info_param_create (C++ function), 40
esp_rmaker_ota_mark_invalid (C++ function), 38
esp_rmaker_ota_mark_valid (C++ function), 38
esp_rmaker_ota_report_status (C++ function), 37
esp_rmaker_ota_service_create (C++ function), 32
esp_rmaker_ota_status_param_create (C++ function), 27
esp_rmaker_ota_type_t (C++ enum), 41
esp_rmaker_ota_url_param_create (C++ function), 28
ESP_RMAKER_PARAM_AC_MODE (C macro), 24
esp_rmaker_param_add_array_max_count (C++ function), 12
esp_rmaker_param_add_bounds (C++ function), 11
esp_rmaker_param_add_ui_type (C++ function), 11
esp_rmaker_param_add_valid_str_list (C++ function), 12
ESP_RMAKER_PARAM_BLINDS_POSITION (C macro), 24
ESP_RMAKER_PARAM_BRIGHTNESS (C macro), 23
ESP_RMAKER_PARAM_CCT (C macro), 23
esp_rmaker_param_create (C++ function), 11
ESP_RMAKER_PARAM_DIRECTION (C macro), 23
ESP_RMAKER_PARAM_FACTORY_RESET (C macro), 24
ESP_RMAKER_PARAM_GARAGE_POSITION (C macro), 24
esp_rmaker_param_get_name (C++ function), 13
esp_rmaker_param_get_type (C++ function), 14
esp_rmaker_param_get_val (C++ function), 14
ESP_RMAKER_PARAM_HUE (C macro), 23
ESP_RMAKER_PARAM_INTENSITY (C macro), 23
ESP_RMAKER_PARAM_LIGHT_MODE (C macro), 24
ESP_RMAKER_PARAM_LOCAL_CONTROL_POP (C macro), 24
ESP_RMAKER_PARAM_LOCAL_CONTROL_TYPE (C macro), 24
ESP_RMAKER_PARAM_MODE (C macro), 24
ESP_RMAKER_PARAM_NAME (C macro), 23
ESP_RMAKER_PARAM_OTA_INFO (C macro), 23
ESP_RMAKER_PARAM_OTA_STATUS (C macro), 23
ESP_RMAKER_PARAM_OTA_URL (C macro), 24
ESP_RMAKER_PARAM_POWER (C macro), 23
ESP_RMAKER_PARAM_RANGE (C macro), 24
ESP_RMAKER_PARAM_REBOOT (C macro), 24
ESP_RMAKER_PARAM_SATURATION (C macro), 23
ESP_RMAKER_PARAM_SCENES (C macro), 24
ESP_RMAKER_PARAM_SCHEDULES (C macro), 24
ESP_RMAKER_PARAM_SPEED (C macro), 23
esp_rmaker_param_t (C++ type), 18
ESP_RMAKER_PARAM_TEMPERATURE (C macro), 23
ESP_RMAKER_PARAM_TIMEZONE (C macro), 24
ESP_RMAKER_PARAM_TIMEZONE_POSIX (C macro), 24
ESP_RMAKER_PARAM_TOGGLE (C macro), 24
esp_rmaker_param_update (C++ function), 12
esp_rmaker_param_update_and_notify (C++ function), 13
esp_rmaker_param_update_and_report (C++ function), 12
esp_rmaker_param_val_t (C++ class), 17
esp_rmaker_param_val_t::type (C++ member), 17
esp_rmaker_param_val_t::val (C++ member), 17
ESP_RMAKER_PARAM_WIFI_RESET (C macro), 24
esp_rmaker_post_ota_diag_t (C++ type), 40
esp_rmaker_power_param_create (C++ function), 25
esp_rmaker_raise_alert (C++ function), 13
esp_rmaker_read_ctx_t (C++ class), 17
esp_rmaker_read_ctx_t::src (C++ member), 17
esp_rmaker_reboot (C++ function), 43
esp_rmaker_reboot_param_create (C++ function), 29
esp_rmaker_report_node_details (C++ function), 14
ESP_RMAKER_REQ_SRC_CLOUD (C++ enumerator), 20
ESP_RMAKER_REQ_SRC_INIT (C++ enumerator), 20
ESP_RMAKER_REQ_SRC_LOCAL (C++ enumerator), 20
ESP_RMAKER_REQ_SRC_MAX (C++ enumerator), 20
ESP_RMAKER_REQ_SRC_SCENE_ACTIVATE (C++ enumerator), 20
ESP_RMAKER_REQ_SRC_SCENE_DEACTIVATE (C++ enumerator), 20
ESP_RMAKER_REQ_SRC_SCHEDULE (C++ enumerator), 20
esp_rmaker_req_src_t (C++ enum), 20
esp_rmaker_saturation_param_create (C++ function), 26
esp_rmaker_scenes_enable (C++ function), 23
esp_rmaker_scenes_param_create (C++ function), 29
esp_rmaker_schedule_enable (C++ function), 22
esp_rmaker_schedules_param_create (C++ function), 28
esp_rmaker_service_create (C++ function), 7
ESP_RMAKER_SERVICE_LOCAL_CONTROL (C macro), 25
ESP_RMAKER_SERVICE_OTA (C macro), 25
ESP_RMAKER_SERVICE_SCENES (C macro), 25
ESP_RMAKER_SERVICE_SCHEDULE (C macro), 25
ESP_RMAKER_SERVICE_SYSTEM (C macro), 25
ESP_RMAKER_SERVICE_TIME (C macro), 25
esp_rmaker_speed_param_create (C++ function), 27
esp_rmaker_start (C++ function), 5
esp_rmaker_start_user_node_mapping (C++ function), 21
esp_rmaker_stop (C++ function), 5
esp_rmaker_str (C++ function), 4
esp_rmaker_switch_device_create (C++ function), 31
esp_rmaker_system_serv_config_t (C++ class), 17
esp_rmaker_system_serv_config_t::flags (C++ member), 17
esp_rmaker_system_serv_config_t::reboot_seconds (C++ member), 17
esp_rmaker_system_serv_config_t::reset_reboot_seconds (C++ member), 17
esp_rmaker_system_serv_config_t::reset_seconds (C++ member), 17
esp_rmaker_system_service_enable (C++ function), 14
esp_rmaker_temp_sensor_device_create (C++ function), 32
esp_rmaker_temperature_param_create (C++ function), 27
esp_rmaker_test_cmd_resp (C++ function), 15
esp_rmaker_time_check (C++ function), 44
esp_rmaker_time_config (C++ class), 45
esp_rmaker_time_config::sntp_server_name (C++ member), 45
esp_rmaker_time_config::sync_time_cb (C++ member), 45
esp_rmaker_time_config::time_check (C++ member), 45
esp_rmaker_time_config::time_config (C++ member), 46
esp_rmaker_time_get_timezone (C++ function), 45
esp_rmaker_time_get_timezone_posix (C++ function), 45
esp_rmaker_time_service_create (C++ function), 33
esp_rmaker_time_set_timezone (C++ function), 44
esp_rmaker_time_set_timezone_posix (C++ function), 44
MEM_ALLOC_EXTRAM (C macro), 46
MEM_CALLOC_EXTRAM (C macro), 46
MEM_REALLOC_EXTRAM (C macro), 46
Node (class in rmaker_lib.node), 53
OTA_DIAG_STATE_INIT (C++ enumerator), 42
OTA_DIAG_STATE_POST_MQTT (C++ enumerator), 42
OTA_DIAG_STATUS_FAIL (C++ enumerator), 42
OTA_DIAG_STATUS_PENDING (C++ enumerator), 42
OTA_DIAG_STATUS_SUCCESS (C++ enumerator), 42
OTA_STATUS_DELAYED (C++ enumerator), 41
OTA_STATUS_FAILED (C++ enumerator), 41
OTA_STATUS_IN_PROGRESS (C++ enumerator), 41
OTA_STATUS_REJECTED (C++ enumerator), 41
OTA_STATUS_SUCCESS (C++ enumerator), 41
ota_status_t (C++ enum), 41
ota_upgrade() (in module rmaker_cmd.node), 59
OTA_USING_PARAMS (C++ enumerator), 41
OTA_USING_TOPICS (C++ enumerator), 42
PROP_FLAG_PERSIST (C++ enumerator), 20
PROP_FLAG_READ (C++ enumerator), 20
PROP_FLAG_TIME_SERIES (C++ enumerator), 20
PROP_FLAG_WRITE (C++ enumerator), 20
provision() (in module rmaker_cmd.provision), 61
remove_node() (in module rmaker_cmd.node), 59
remove_shared_nodes_request() (rmaker_lib.node.Node method), 56
remove_sharing() (in module rmaker_cmd.node), 59
remove_user_from_shared_nodes() (rmaker_lib.node.Node method), 56
remove_user_node_mapping() (rmaker_lib.node.Node method), 56
request_op() (rmaker_lib.node.Node method), 57
rmaker_cmd.browserlogin (module), 61
rmaker_cmd.node (module), 57
rmaker_cmd.provision (module), 61
rmaker_cmd.user (module), 60
RMAKER_EVENT_CLAIM_FAILED (C++ enumerator), 19
RMAKER_EVENT_CLAIM_SUCCESSFUL (C++ enumerator), 19
RMAKER_EVENT_FACTORY_RESET (C++ enumerator), 48
RMAKER_EVENT_INIT_DONE (C++ enumerator), 19
RMAKER_EVENT_LOCAL_CTRL_STARTED (C++ enumerator), 19
RMAKER_EVENT_LOCAL_CTRL_STOPPED (C++ enumerator), 19
RMAKER_EVENT_REBOOT (C++ enumerator), 48
RMAKER_EVENT_TZ_CHANGED (C++ enumerator), 49
RMAKER_EVENT_TZ_POSIX_CHANGED (C++ enumerator), 49
RMAKER_EVENT_USER_NODE_MAPPING_DONE (C++ enumerator), 19
RMAKER_EVENT_USER_NODE_MAPPING_RESET (C++ enumerator), 19
RMAKER_EVENT_WIFI_RESET (C++ enumerator), 48
rmaker_lib.node (module), 53
rmaker_lib.session (module), 52
rmaker_lib.user (module), 51
RMAKER_MQTT_EVENT_CONNECTED (C++ enumerator), 49
RMAKER_MQTT_EVENT_DISCONNECTED (C++ enumerator), 49
RMAKER_MQTT_EVENT_MSG_DELETED (C++ enumerator), 49
RMAKER_MQTT_EVENT_PUBLISHED (C++ enumerator), 49
RMAKER_MQTT_EVENT_PUBLISHED (C++ enumerator), 49
RMAKER_MQTT_EVENT_PUBLISHED (C++ enumerator), 49
RMAKER_MQTT_EVENT_PUBLISHED (C++ enumerator), 49
RMAKER_OTAA_EVENT_DELAYED (C++ enumerator), 41
RMAKER_OTAA_EVENT_FAILED (C++ enumerator), 41
RMAKER_OTAA_EVENT_IN_PROGRESS (C++ enumerator), 41
RMAKER_OTAA_EVENT_INVALID (C++ enumerator), 41
RMAKER_OTAA_EVENT_REJECTED (C++ enumerator), 41
RMAKER_OTAA_EVENT_REQ_FOR_REBOOT (C++ enumerator), 41
RMAKER_OTAA_EVENT_STARTING (C++ enumerator), 41
RMAKER_OTAA_EVENT_SUCCESSFUL (C++ enumerator), 41
RMAKER_VAL_TYPE_ARRAY (C++ enumerator), 20
RMAKER_VAL_TYPE_BOOLEAN (C++ enumerator), 19
RMAKER_VAL_TYPE_FLOAT (C++ enumerator), 20
RMAKER_VAL_TYPE_INTEGER (C++ enumerator), 19
RMAKER_VAL_TYPE_INVALID (C++ enumerator), 19
19
RMAKER_VAL_TYPE_OBJECT (C++ enumerator), 20
RMAKER_VAL_TYPE_STRING (C++ enumerator), 20

S
Session (class in rmaker_lib.session), 52
set_node_params () (rmaker_lib.node.Node method), 57
set_params () (in module rmaker_cmd.node), 60
sharing_request_op () (in module rmaker_cmd.node), 60
signup () (in module rmaker_cmd.user), 61
signup () (rmaker_lib.user.User method), 52
signup_request () (rmaker_lib.user.User method), 52
SYSTEM_SERV_FLAG_FACTORY_RESET (C macro), 18
SYSTEM_SERV_FLAG_REBOOT (C macro), 18
SYSTEM_SERV_FLAG_WIFI_RESET (C macro), 18
SYSTEM_SERV_FLAGS_ALL (C macro), 18

U
User (class in rmaker_lib.user), 51