ESP RainMaker Programming Guide

Espressif

Oct 23, 2020

CONTENTS

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

CHAPTER

ONE

C API REFERENCE

1.1 RainMaker Core

1.1.1 Core

Header File

- esp_rainmaker/include/esp_rmaker_core.h

Functions

ESP_EVENT_DECLARE_BASE(RMAKER_EVENT)
ESP RainMaker Event Base

esp_rmaker_param_val_t **esp_rmaker_bool**(bool *bval*)
Initialise a Boolean value

Return Value structure.

Parameters

- `[in] bval`: Initialising value.

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

- `[in] fval`: Initialising value.

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.

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.

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.

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().

ESP_OK on success.

Note This should be called after rainmaker has stopped.

Return error in case of failure.

Parameters

- `[in] node`: Node Handle returned by esp_rmaker_node_init().

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.

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.

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.

Parameters

- `node`: Node handle.

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.

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.
- `[in] fw_version`: New firmware version.

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

Add model for a node (Not recommended)

Model is set internally to the project name. This API can be used to override that name.

Return ESP_OK on success.

Return error in case of failure.

Parameters

- `node`: Node handle.
- `[in] model`: New model string.

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.

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.

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] write_cb`: Write callback.
- `[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_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.
- `[in] val`: Value of the attribute.

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

Get device name from handle

Get device type from handle

Return NULL terminated device name string on success.

Return NULL in case of failure.

Parameters

- `[in] device`: Device 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* **const** char **const** *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* **const** char **const** *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* **const** *esp_rmaker_param_t* **const** *device*, **const** *esp_rmaker_param_t* **const** *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.
- `[in] param`: Parameter handle.

esp_rmaker_param_t ***esp_rmaker_param_create**(**const** char **const** *param_name*, **const** char **const** *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. Eg, 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 documetation for supported UI Types.

Return ESP_OK on success.

Return error in case of failure.

Parameters

- `[in] param`: Parameter handle.
- `[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] param`: Parameter handle.
- `[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] param`: Parameter handle.
- `[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] param`: Parameter handle.
- `[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.
- `[in] val`: New value of the parameter.

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_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_queue_work**(*esp_rmaker_work_fn_t work_fn*, void **priv_data*)
Queue execution of a function in ESP RainMaker's context

This API queues a work function for execution in the ESP RainMaker 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.

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)

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

Macros

ESP_RMAKER_CONFIG_VERSION

MAX_VERSION_STRING_LEN

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 paassed while creating the device.
- `[in] ctx`: Context associated with the request.

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)
Callback for parameter value changes

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

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.

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_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()

enum 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

1.1.2 User Mapping

Header File

- esp_rainmaker/include/esp_rmaker_user_mapping.h

Functions

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

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

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)

1.1.3 Utilities

Header File

- esp_rainmaker/include/esp_rmaker_utils.h

Functions

esp_err_t **esp_rmaker_reboot**(uint8_t *seconds*)
Reboot the chip after a delay

This API just starts a reboot timer and returns immediately. The actual reboot is trigerred 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 chip should reboot.

esp_err_t **esp_rmaker_wifi_reset**(uint8_t *seconds*)
Reset Wi-Fi credentials and reboot

This will reset just the Wi-Fi credentials and 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 after the reset.

Note This function internally calls esp_rmaker_reboot() and returns immediately. The reboot happens asynchronously.

Return ESP_OK on success.

Return error on failure.

Parameters

- `[in] seconds`: Time in seconds after which the chip should reboot.

esp_err_t **esp_rmaker_factory_reset**(uint8_t *seconds*)
Reset to factory defaults and reboot

This will clear entire NVS partition and trigger a reboot. The RMAKER_EVENT_FACTORY_RESET event is triggered after the reset.

Note This function internally calls esp_rmaker_reboot() and returns. The reboot happens asynchronously.

Return ESP_OK on success.

Return error on failure.

Parameters

- `[in] seconds`: Time in seconds after which the chip should reboot.

esp_err_t **esp_rmaker_time_sync_init**(*esp_rmaker_time_config_t* **config*)
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.

bool **esp_rmaker_time_check**(void)
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

esp_err_t **esp_rmaker_time_wait_for_sync**(uint32_t *ticks_to_wait*)
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.

esp_err_t **esp_rmaker_time_set_timezone_posix**(**const** char **tz_posix*)
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

esp_err_t **esp_rmaker_time_set_timezone**(**const** char **tz*)
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

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_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.

sntp_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.

Type Definitions

typedef struct *esp_rmaker_time_config* **esp_rmaker_time_config_t**

1.1.4 Scheduling

Header File

- esp_rainmaker/include/esp_rmaker_schedule.h

Functions

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

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_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_DEVICE_SWITCH

ESP_RMAKER_DEVICE_LIGHTBULB

ESP_RMAKER_DEVICE_FAN

ESP_RMAKER_DEVICE_TEMP_SENSOR

ESP_RMAKER_SERVICE_OTA

ESP_RMAKER_SERVICE_TIME

ESP_RMAKER_SERVICE_SCHEDULE

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

esp_rmaker_param_t ***esp_rmaker_direction_param_create**(**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

esp_rmaker_param_t ***esp_rmaker_speed_param_create**(**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

esp_rmaker_param_t ***esp_rmaker_temperature_param_create**(**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

esp_rmaker_param_t ***esp_rmaker_ota_status_param_create**(**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

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

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

ESP_RMAKER_DEF_SCHEDULE_NAME

1.2.3 Standard Devices

Header File

- esp_rainmaker/include/esp_rmaker_standard_devices.h

Functions

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"

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] write_cb`: Write callback.
- `[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.

1.3 RainMaker MQTT

1.3.1 Header File

- esp_rainmaker/include/esp_rmaker_mqtt.h

1.3.2 Functions

esp_err_t **esp_rmaker_mqtt_init**(*esp_rmaker_mqtt_config_t* **config*)

Initialize ESP RainMaker MQTT

Return ESP_OK on success.

Return error in case of any error.

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.

Return ESP_OK on success.

Return error in case of any error.

esp_err_t **esp_rmaker_mqtt_disconnect**(void)
MQTT Disconnect

Disconnects from the MQTT broker.

Return ESP_OK on success.

Return error in case of any error.

esp_err_t **esp_rmaker_mqtt_publish**(**const** char *_topic_, void *_data_, size_t _data_len_)
Publish MQTT Message

Return ESP_OK on success.

Return 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

esp_err_t **esp_rmaker_mqtt_subscribe**(**const** char *_topic_, *esp_rmaker_mqtt_subscribe_cb_t* _cb_, 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

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.

1.3.3 Structures

struct esp_rmaker_mqtt_config_t
ESP RainMaker MQTT Configuration

Public Members

char ***mqtt_host**
MQTT Host

char ***client_id**
Client ID

char ***client_cert**
Client Certificate in NULL terminate PEM format

char ***client_key**
Client Key in NULL terminate PEM format

char ***server_cert**
Server Certificate in NULL terminate PEM format

1.3.4 Type Definitions

typedef void (***esp_rmaker_mqtt_subscribe_cb_t**)(**const** char *topic, void *payload, size_t payload_len, void *priv_data)
ESP RainMaker MQTT Subscribe callback prototype

Parameters

- `[in] topic`: Topic on which the message was received
- `[in] payload`: Data received in the message
- `[in] payload_len`: Length of the data
- `[in] priv_data`: The private data passed during subscription

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

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

const char ***server_cert**

The server certificate passed in esp_rmaker_enable_ota()

char ***priv**

The private data passed in esp_rmaker_enable_ota()

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_DEFAULT_OTA_SERVER_CERT`.

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

1.4.4 Type Definitions

typedef void ***esp_rmaker_ota_handle_t**
The OTA Handle to be used by the OTA callback

typedef esp_err_t (***esp_rmaker_ota_cb_t**)(*esp_rmaker_ota_handle_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

typedef bool (***esp_rmaker_post_ota_diag_t**)(void)
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.

Return true if diagnostics are successful, meaning that the new firmware is fine.

Return false if diagnostics fail and a roolback to previous firmware is required.

1.4.5 Enumerations

enum 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

enum 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.

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)
Initialize console

Initializes serial console and adds basic commands.

Return ESP_OK on success.

Return error in case of failures.

CHAPTER

TWO

PYTHON API REFERENCE

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

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.

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

2.1.3 Node

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*

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 **requestId** (*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

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

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.**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

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

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

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

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.**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.**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*) – *email* as key - Email address 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.**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

Return type None

rmaker_cmd.user.**signup**(*vars=None*)
User signup to the ESP Rainmaker.

Parameters **vars** (*dict*) – *email* as key - Email address 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

PYTHON MODULE INDEX

r

INDEX

F

G

L

M

N

O

P

R

S

U