Github OTA

Release 0.0.1

Justin Hammond

Nov 28, 2022

CONTENTS:

This documentation is work in progress. Please refer to the [examples](https://github.com/Fishwaldo/esp_ghota/tree/master/examples) directory for working examples

CHAPTER

ONE

GITHUB OTA FOR ESP32 DEVICES

Automate your OTA and CI/CD pipeline with Github Actions to update your ESP32 devices in the field direct from github releases

1.1 Features

- Uses the esp_htps_ota library under the hood to update firmware images
- Can also update spiffs/littlefs/fatfs partitions
- Uses SemVer to compare versions and only update if a newer version is available
- Plays nicely with App rollback and anti-rollback features of the esp-idf bootloader
- Download firmware and partitiion images from the github release page directly
- Supports multiple devices with different firmware images
- Includes a sample Github Actions that builds and releases images when a new tag is pushed
- Updates can be triggered manually, or via a interval timer
- Uses a streaming JSON parser for to reduce memory usage (Github API responses can be huge)
- Supports Private Repositories (Github API token required*)
- Supports Github Enterprise
- Supports Github Personal Access Tokens to overcome Github API Ratelimits
- Sends progress of Updates via the esp_event_loop

Note: You should be careful with your GitHub PAT and putting it in the source code. I would suggest that you store the PAT in NVS, and the user enters it when running, as otherwise the PAT would be easily extractable from your firmware images.

1.2 Usage

via the Espressif Component Registry:

```
idf.py add-dependency Fishwaldo/ghota^1.0.0
```

1.3 Example

After Initilizing Network Access, Start a timer to periodically check for new releases:

```
    ghota_config_t ghconfig = {
        .filenamematch = "GithubOTA-esp32.bin", // Glob Pattern to match against the␣
↪Firmware file
        .storagenamematch = "storage-esp32.bin", // Glob Pattern to match against the␣
↪storage firmware file
        .storagepartitionname = "storage", // Update the storage partition
        .updateInterval = 60, // Check for updates every 60 minuites
    };
    ghota_client_handle_t *ghota_client = ghota_init(&ghconfig);
    if (ghota_client == NULL) {
        ESP_LOGE(TAG, "ghota_client_init failed");
        return;
    }
    esp_event_handler_register(GHOTA_EVENTS, ESP_EVENT_ANY_ID, &ghota_event_callback,␣
↪ghota_client); // Register a handler to get updates on progress
    ESP_ERROR_CHECK(ghota_start_update_timer(ghota_client)); // Start the timer to check␣
↪for updates
```

Manually Checking for updates:

```
    ghota_config_t ghconfig = {
        .filenamematch = "GithubOTA-esp32.bin",
        .storagenamematch = "storage-esp32.bin",
        .storagepartitionname = "storage",
        .updateInterval = 60,
    };
    ghota_client_handle_t *ghota_client = ghota_init(&ghconfig);
    if (ghota_client == NULL) {
        ESP_LOGE(TAG, "ghota_client_init failed");
        return;
    }
    esp_event_handler_register(GHOTA_EVENTS, ESP_EVENT_ANY_ID, &ghota_event_callback,␣
↪ghota_client);
    ESP_ERROR_CHECK(ghota_check(ghota_client));

    semver_t *cur = ghota_get_current_version(ghota_client);
    if (cur) {
        ESP_LOGI(TAG, "Current version: %d.%d.%d", cur->major, cur->minor, cur->patch);
        semver_free(cur);
    }

    semver_t *new = ghota_get_latest_version(ghota_client);
    if (new) {
        ESP_LOGI(TAG, "New version: %d.%d.%d", new->major, new->minor, new->patch);
        semver_free(new);
    }
    ESP_ERROR_CHECK(ghota_update(ghota_client));
    ESP_ERROR_CHECK(ghota_free(ghota_client));
```

1.4 Configuration

The following configuration options are available:

```
* config.filenamematch <- Glob pattern to match against the firmware file from the␣
↪Github Releases page.
* config.storagenamematch <- Glob pattern to match against the storage file from the␣
↪Github Releases page.
* config.storagepartitionname <- Name of the storage partition to update (as defined in␣
↪partitions.csv)
* config.hostname <- Hostname of the Github API (default: api.github.com)
* config.orgname <- Name of the Github User or Organization
* config.reponame <- Name of the Github Repository
* config.updateInterval <- Interval in minutes to check for updates
```

1.5 Github Actions

The Github Actions included in this repository can be used to build and release firmware images to Github Releases. This is a good way to automate your CI/CD pipeline, and update your devices in the field. In this example, we build two variants of the Firmware - on for a ESP32 and one for a ESP32-S3 device Using the filenamematch and storage-namematch config options, we can match against the correct firmware image for the device.

```yaml
on:
  push:
  pull_request:
    branches: [master]

permissions:
  contents: write
name: Build
jobs:
  build:
    strategy:
      fail-fast: true
      matrix:
        targets: [esp32, esp32s3]
    runs-on: ubuntu-latest
    steps:
    - name: Checkout repo
      uses: actions/checkout@v3
      with:
        submodules: 'recursive'
    - name: esp-idf build
      uses: espressif/esp-idf-ci-action@v1
      with:
        esp_idf_version: v4.4
        target: ${{ matrix.targets }}
    - name: Rename artifact
      run: |
        cp build/GithubOTA.bin GithubOTA-${{ matrix.targets }}.bin
        cp build/storage.bin storage-${{ matrix.targets }}.bin
```

(continues on next page)

(continued from previous page)

```
    - name: Archive Firmware Files
      uses: actions/upload-artifact@v3
      with:
        name: ${{ matrix.targets }}-firmware
        path: "*-${{ matrix.targets }}.bin"

  release:
    needs: build
    runs-on: ubuntu-latest
    steps:
    - name: Download Firmware Files
      uses: actions/download-artifact@v2
      with:
        path: release
    - name: Release Firmware
      uses: ncipollo/release-action@v1
      if: startsWith(github.ref, 'refs/tags/')
      with:
        artifacts: release/*/*.bin
        generateReleaseNotes: true
        allowUpdates: true
        token: ${{ secrets.GITHUB_TOKEN }}
```

CHAPTER

TWO

MAIN API

esp_ghota.h contains the main API for the library.

2.1 API Details

Typedefs

typedef struct *ghota_config_t* **ghota_config_t**

Github OTA Configuration.

typedef struct *ghota_client_handle_t* **ghota_client_handle_t**

Enums

enum **ghota_event_e**

Github OTA events These events are posted to the event loop to track progress of the OTA process.

Values:

enumerator **GHOTA_EVENT_START_CHECK**

Github OTA check started

enumerator **GHOTA_EVENT_UPDATE_AVAILABLE**

Github OTA update available

enumerator **GHOTA_EVENT_NOUPDATE_AVAILABLE**

Github OTA no update available

enumerator **GHOTA_EVENT_START_UPDATE**

Github OTA update started

enumerator **GHOTA_EVENT_FINISH_UPDATE**

Github OTA update finished

enumerator **GHOTA_EVENT_UPDATE_FAILED**

Github OTA update failed

enumerator **GHOTA_EVENT_START_STORAGE_UPDATE**

Github OTA storage update started. If the storage is mounted, you should unmount it when getting this call

enumerator **GHOTA_EVENT_FINISH_STORAGE_UPDATE**

Github OTA storage update finished. You can mount the new storage after getting this call if needed

enumerator **GHOTA_EVENT_STORAGE_UPDATE_FAILED**

Github OTA storage update failed

enumerator **GHOTA_EVENT_FIRMWARE_UPDATE_PROGRESS**

Github OTA firmware update progress

enumerator **GHOTA_EVENT_STORAGE_UPDATE_PROGRESS**

Github OTA storage update progress

enumerator **GHOTA_EVENT_PENDING_REBOOT**

Github OTA pending reboot

Functions

ESP_EVENT_DECLARE_BASE(GHOTA_EVENTS)

ghota_client_handle_t ***ghota_init**(*ghota_config_t* *config)

Initialize the github ota client.

Parameters

config – [in] Configuration for the github ota client

Returns

ghota_client_handle_t* handle to pass to all subsequent calls. If it returns NULL, there is a error in your config

esp_err_t **ghota_set_auth**(*ghota_client_handle_t* *handle, const char *username, const char *password)

Set the Username and Password to access private repositories or get more API calls.

Anonymus API calls are limited to 60 per hour. If you want to get more calls, you need to set a username and password. Be aware that this will be stored in the flash and can be read by anyone with access to the flash. The password should be a Github Personal Access Token and for good security you should limit what it can do

Parameters

- **handle** – the handle returned by ghota_init
- **username** – the username to authenticate with
- **password** – this Github Personal Access Token

Returns

esp_err_t ESP_OK if all is good, ESP_FAIL if there is an error

esp_err_t **ghota_free**(*ghota_client_handle_t* *handle)

Free the ghota client handle and all resources.

Parameters
handle – the Handle

Returns
esp_err_t if there was a error

esp_err_t **ghota_check**(*ghota_client_handle_t* *handle)

Perform a check for updates.

This will just check if there is a available update on Github releases with download resources that match your configuration for firmware and storage files. If it returns ESP_OK, you can call ghota_get_latest_version to get the version of the latest release

Parameters
handle – the ghota_client_handle_t handle

Returns
esp_err_t ESP_OK if there is a update available, ESP_FAIL if there is no update available or an error

esp_err_t **ghota_update**(*ghota_client_handle_t* *handle)

Downloads and writes the latest firmware and storage partition (if available)

You should only call this after calling ghota_check and ensuring that there is a update available.

Parameters
handle – the ghota_client_handle_t handle

Returns
esp_err_t ESP_FAIL if there is a error. If the Update is successful, it will not return, but reboot the device

semver_t ***ghota_get_current_version**(*ghota_client_handle_t* *handle)

Get the currently running version of the firmware.

This will return the version of the firmware currently running on your device. consult semver.h for functions to compare versions

Parameters
handle – the ghota_client_handle_t handle

Returns
semver_t the version of the latest release

semver_t ***ghota_get_latest_version**(*ghota_client_handle_t* *handle)

Get the version of the latest release on Github. Only valid after calling ghota_check.

Parameters
handle – the ghota_client_handle_t handle

Returns
semver_t* the version of the latest release on Github

esp_err_t **ghota_start_update_task**(*ghota_client_handle_t* *handle)

Start a new Task that will check for updates and update if available.

This is equivalent to calling ghota_check and ghota_update if there is a new update available. If no update is available, it will not update the device.

Progress can be monitored by registering for the GHOTA_EVENTS events on the Global Event Loop

Parameters
handle – ghota_client_handle_t handle

Returns
esp_err_t ESP_OK if the task was started, ESP_FAIL if there was an error

esp_err_t **ghota_start_update_timer**(*ghota_client_handle_t* *handle)

Install a Timer to automatically check for new updates and update if available.

Install a timer that will check for new updates every updateInterval seconds and update if available.

Parameters
handle – ghota_client_handle_t handle

Returns
esp_err_t ESP_OK if no error, otherwise ESP_FAIL

char ***ghota_get_event_str**(*ghota_event_e* event)

convience function to return a string representation of events emited by this library

Parameters
event – the eventid passed to the event handler

Returns
char* a string representing the event

struct **ghota_config_t**

#include <*esp_ghota.h*> Github OTA Configuration.

Public Members

char **filenamematch**[CONFIG_MAX_FILENAME_LEN]

Filename to match against on Github indicating this is a firmware file

char **storagenamematch**[CONFIG_MAX_FILENAME_LEN]

Filename to match against on Github indicating this is a storage file

char **storagepartitionname**[17]

Name of the storage partition to update

char ***hostname**

Hostname of the Github server. Defaults to api.github.com

char ***orgname**

Name of the Github organization

char ***reponame**

Name of the Github repository

uint32_t **updateInterval**

Interval in Minutes to check for updates if using the ghota_start_update_timer function

CHAPTER

THREE

SEMVER.H

The Semver support comes from https://github.com/h2non/semver.c

You can use this to compare released and installed versions of the firmware.

consult the above github repo for more information on using the functions defined in this header.

Defines

SEMVER_VERSION

Typedefs

typedef struct *semver_version_s* **semver_t**

semver_t struct

Functions

int **semver_satisfies**(*semver_t* x, *semver_t* y, const char *op)

Set prototypes

int **semver_satisfies_caret**(*semver_t* x, *semver_t* y)

int **semver_satisfies_patch**(*semver_t* x, *semver_t* y)

int **semver_compare**(*semver_t* x, *semver_t* y)

int **semver_compare_version**(*semver_t* x, *semver_t* y)

int **semver_compare_prerelease**(*semver_t* x, *semver_t* y)

int **semver_gt**(*semver_t* x, *semver_t* y)

int **semver_gte**(*semver_t* x, *semver_t* y)

int **semver_lt**(*semver_t* x, *semver_t* y)

int **semver_lte**(*semver_t* x, *semver_t* y)

int **semver_eq**(*semver_t* x, *semver_t* y)

int **semver_neq**(*semver_t* x, *semver_t* y)

int **semver_parse**(const char *str, *semver_t* *ver)

int **semver_parse_version**(const char *str, *semver_t* *ver)

void **semver_render**(*semver_t* *x, char *dest)

int **semver_numeric**(*semver_t* *x)

void **semver_bump**(*semver_t* *x)

void **semver_bump_minor**(*semver_t* *x)

void **semver_bump_patch**(*semver_t* *x)

void **semver_free**(*semver_t* *x)

int **semver_is_valid**(const char *s)

int **semver_clean**(char *s)

struct **semver_version_s**

#include <semver.h> semver_t struct

Public Members

int **major**

int **minor**

int **patch**

char ***metadata**

char ***prerelease**

CHAPTER

FOUR

RECENT CHANGES

Prior to Version 1.0.0, the following changes were made:

- **Update Readme and add PlatformIO library.json file** by *Justin Hammond* at *2022-11-28 11:59:53*
- **Create FUNDING.yml** by *Justin Hammond* at *2022-11-27 18:14:09*
- **Update component Version** by *Justin Hammond* at *2022-11-27 17:35:51*
- **add esp-idf action** by *Justin Hammond* at *2022-11-27 17:32:05*
- **We need the build directory to build our docs** by *Justin Hammond* at *2022-11-27 17:21:01*
- **Documentation** by *Justin Hammond* at *2022-11-27 17:13:49*
- **Fix Artifact Copy** by *Justin Hammond* at *2022-11-17 18:34:05*
- **Update espidf version** by *Justin Hammond* at *2022-11-17 18:29:39*
- **Update CI build** by *Justin Hammond* at *2022-11-17 18:27:29*
- **Update main.yml** by *Justin Hammond* at *2022-11-17 18:25:20*
- **Initial Version** by *Justin Hammond* at *2022-11-17 17:39:48*
- **Initial commit** by *Justin Hammond* at *2022-11-17 16:14:25*

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX