

OTAD

1, Overview:

OTAD is OTA-related daemon process. This process main responsibility is to do firmware upgrade involves common things. Such as FBF file integrity checking, Flash burning and so on. OTAD interact with other APP through UBUS. OTAD implemented method call and notification. APP can launch firmware upgrade through method call. APP can listen notification get download process information.

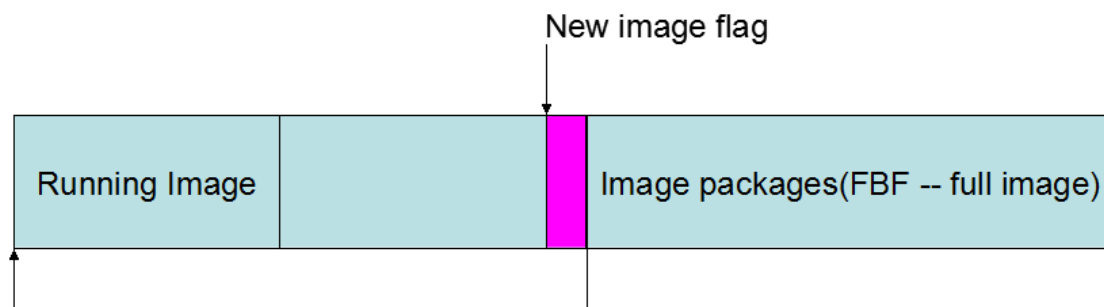
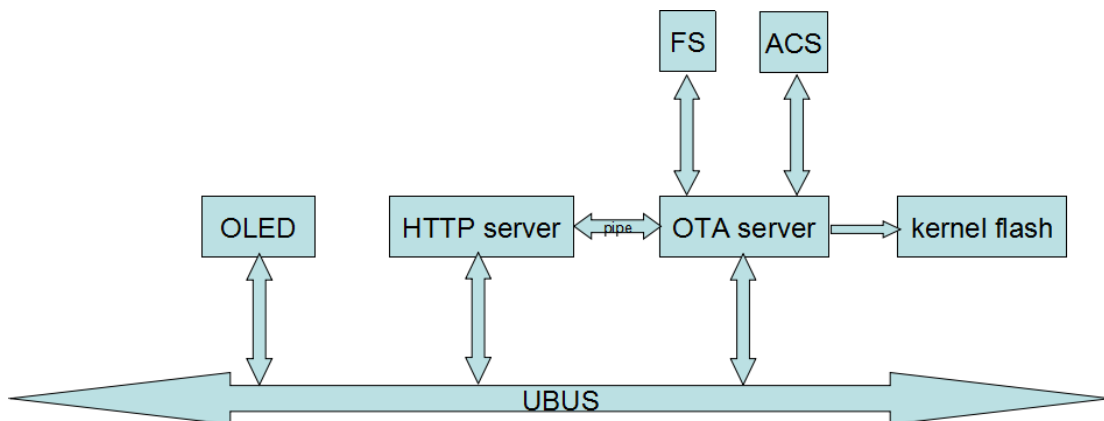
In addition to providing common service for other APP using, OTAD also achieve Marvell defined new firmware discovery protocol.

For end user has three ways to upgrade device:

- a) Device auto detect new firmware through internet. Such as TR069 ACS or other OTA protocol
- b) Upgrade device through WEB UI
- c) Firmware store on SD card

It means OTAD need achieve different ways get firmware data. OTAD get data from the path, verify and burn to flash.

2, OTAD diagram



OBM extract firmware and move to running address.
(If has new image flag, and FBF can be checking pass)

3, UBUS method call

UBUS method call:

start_download: this method call will trigger OTAD start real firmware downloading and verification.

Method call parameters:

- 1) int32: type. Indicate target URL type. Because, we support three types request: http/SD /WEB UI
- 2) string: url. Indicate firmware storage location. So, it possible be http link, or file path on SD card, or UNIX type socket
- 3) string: username. Sometimes, we need user name and password for http downloading
- 4) string: password

OTAD method call related data structure:

```
static const struct ubus_method start_download[] = {
    UBUS_METHOD("download", download_func, download_policy),
};
enum {
    DOWNLOAD_TYPE,
    DOWNLOAD_URL,
    DOWNLOAD_USERNAME,
    DOWNLOAD_PASSWORD,
    __DOWNLOAD_MAX
};
enum {
    DOWNLOAD_TYPE_HTTP,
    DOWNLOAD_TYPE_SD,
    DOWNLOAD_TYPE_UNIX SOCK,
    __DOWNLOAD_TYPE_MAX
};
static const struct blobmsg_policy download_policy[] = {
    [DOWNLOAD_ID] = { .name = "type", .type = BLOBMSG_TYPE_INT32 },
    [DOWNLOAD_URL] = { .name = "url", .type = BLOBMSG_TYPE_STRING },
    [DOWNLOAD_NAME] = { .name = "username", .type = BLOBMSG_TYPE_STRING },
    [DOWNLOAD_PSW] = { .name = "password", .type = BLOBMSG_TYPE_STRING },
};
static int download_func(struct ubus_context *ctx, struct ubus_object *obj,
    struct ubus_request_data *req, const char *method,
    struct blob_attr *msg)
{
    // Check if other downloading in process, if yes return error number
    // Check input parameters
    // If all parameters are valid return 0 else return error number
    switch (download_type) {
        case DOWNLOAD_TYPE_HTTP:
            // Start downloading from HTTP link
            // Here, maybe we need create pthread for the actual download function (because,
```

```

        maybe downloading through http need time is too long)
        break;
    case DOWNLOAD_TYPE_SD:
        // Read data from SD card
        Break;
    case DOWNLOAD_UNIX SOCK:
        // Read data from pipe or local sock
        Break;
    default:
        assert;
}
}

void * __download_http_func(void * data)
{
    // Send http request to store server
    // Process server response. In response header, we can get firmware size
    while (received all size) {
        // Receive data
        // Process data, if detect error (such as check sum error), abort process
        // Notify downloading process via ubus_notify
    }
    // Set flag for OBM upgrade firmware at next reboot, if firmware download successful
    // Notify download result via ubus_notify
}

```

4, Notification

APP can listen notification get download process information.

APP:

```

static struct ubus_subscriber notification_event;
static int register_otad_notification(void)
{
    ctx = ubus_connect("/var/run/ubus.sock");
    ubus_add_uloop(ctx);
    ret = ubus_register_subscriber(ctx, &notification_event);
    notification_event.cb = otad_notify;
    ubus_lookup_id(ctx, "ota", &id);
    ubus_subscribe(ctx, &notification_event, id);
}

static int
otad_notify(struct ubus_context *ctx, struct ubus_object *obj,
            struct ubus_request_data *req, const char *method,
            struct blob_attr *msg)
{

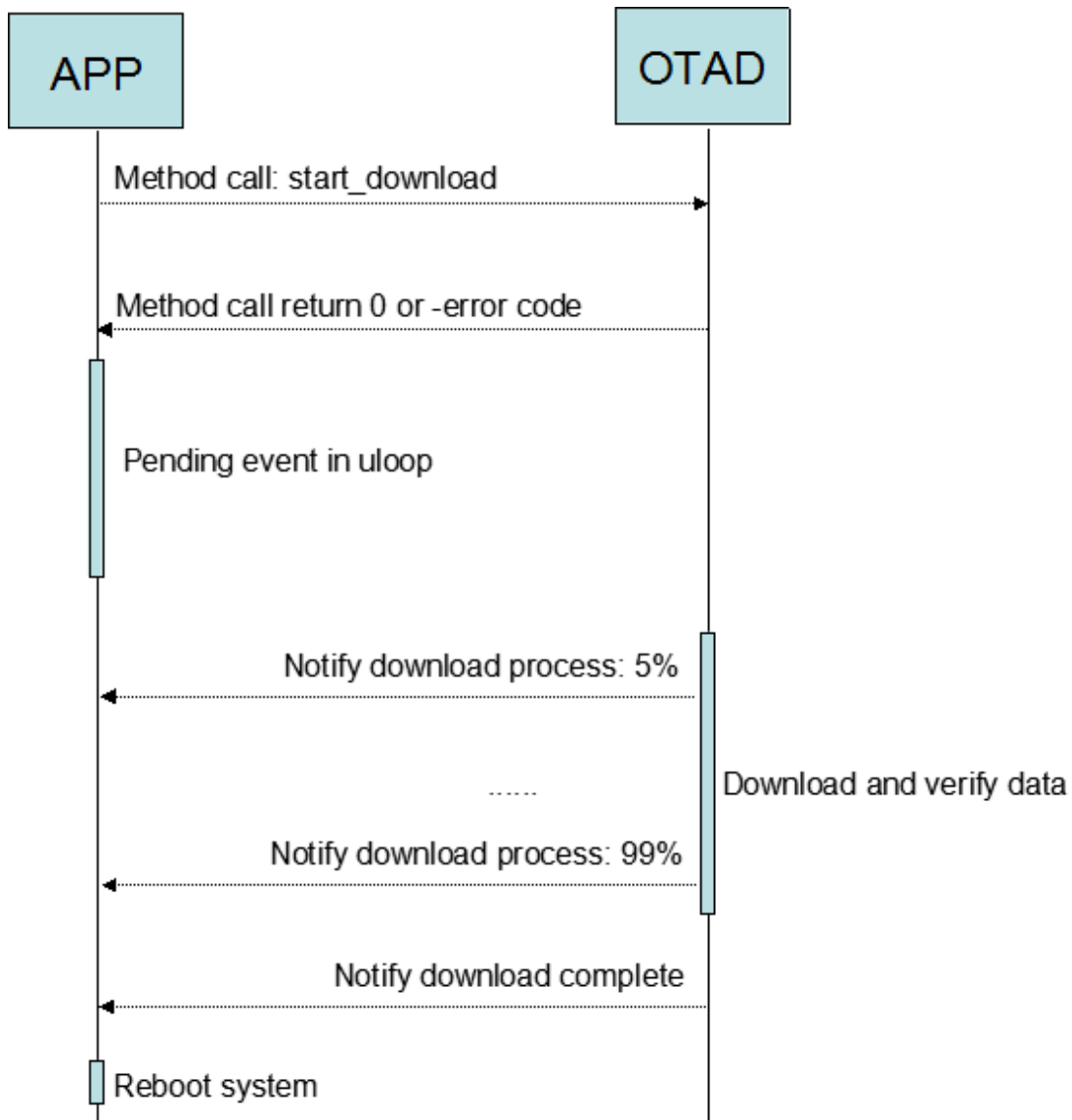
```

```

    // User can get downloading process information from here
}

```

5, APP and daemon work flow



6, Real download process

Real download, we will depend on http client. The client maybe is Curl.

```

static size_t curl_cb(void *buffer, size_t size, size_t nmemb, void *stream)
{
    // Add code here
    // FBF check and write to flash
    return size * nmemb;
}

```

```

static void __download(const char * url)

```

```

{
    CURL *curl = curl_easy_init();
    unsigned int response_code;

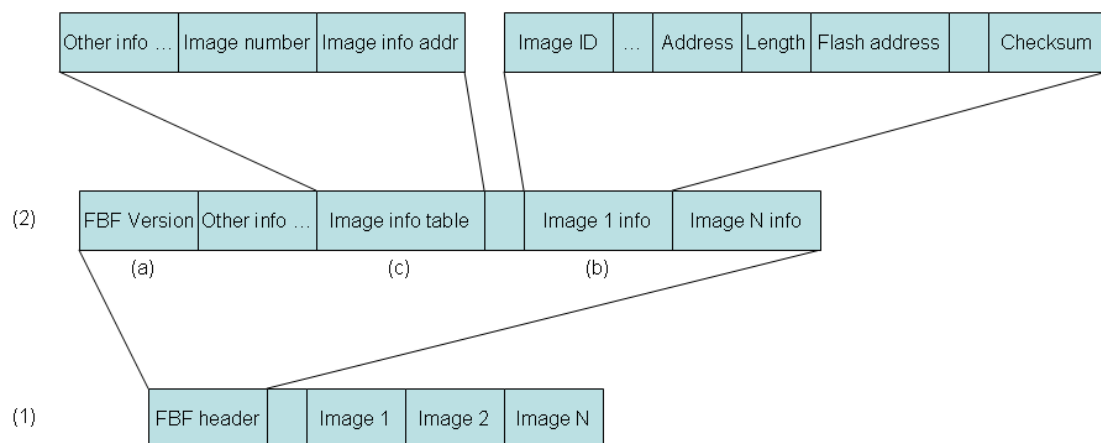
    curl_easy_setopt(curl, CURLOPT_URL, url);
    curl_easy_setopt(curl, CURLOPT_TIMEOUT, 20);
    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, curl_cb);

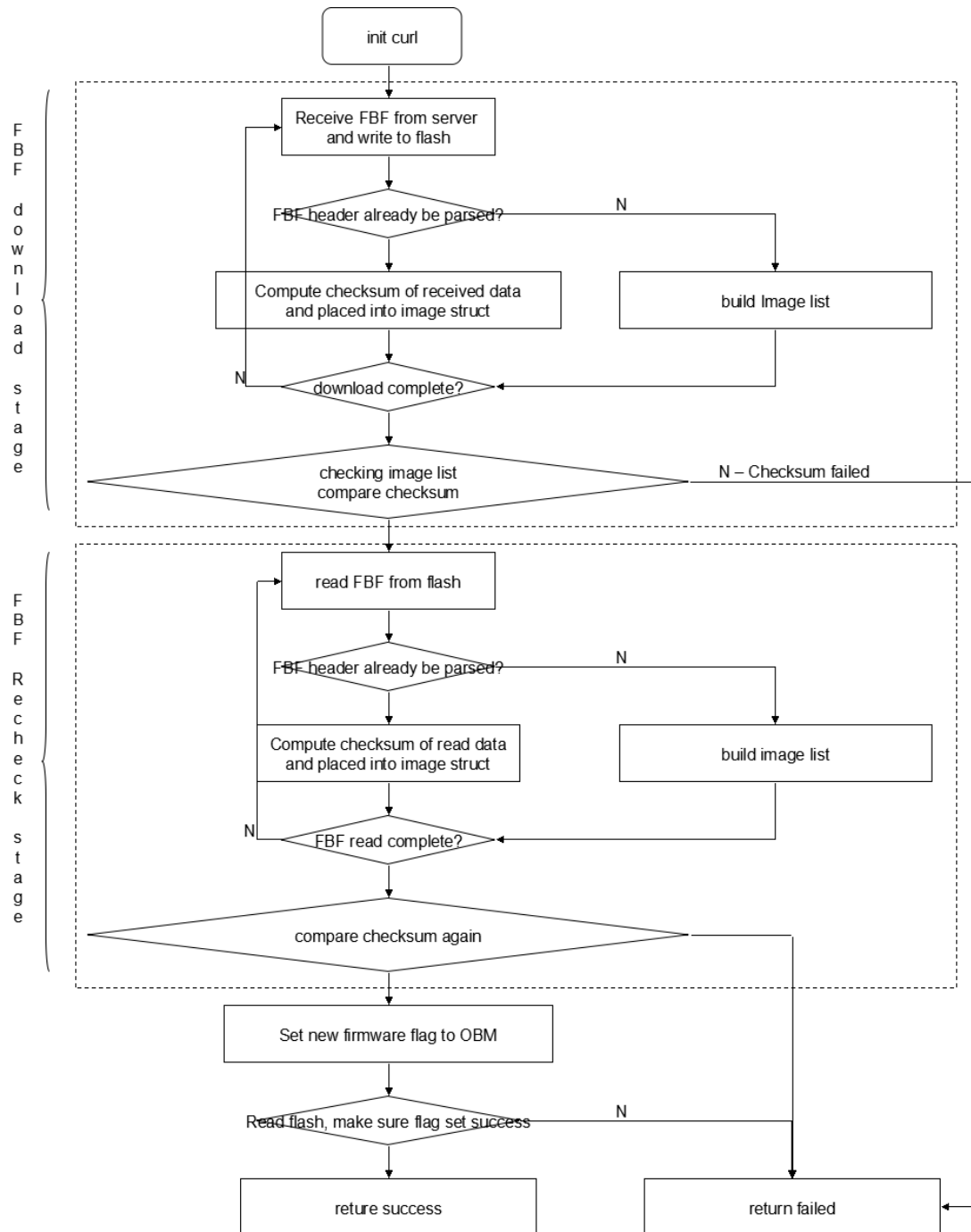
    curl_easy_perform(curl);
    curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE, &response_code);
    curl_easy_cleanup(curl);

    // Check curl_cb process result. If return success then read flash re-verify again else return
    failed
    // Read FBF file from flash and re-verify it
}

```

The FBF format like this:





7, Supported method call list:

a, function name: download

parameters list:

type: http/udp/sd

size: firmware size

url: firmware stored link

username: username for access firmware

password: password for access firmware

b, function name: query

parameters list:

type: 1 --> query download states: updating/success/failed/not start

0 --> query new version