Serial specification:
WRF01

DeviceDrive AS
08.11.2017

1 History

Date	Change	FW ver.	Author
10.08.2016	Added check_upgrade Added get_upgrade	3.0	ØK
28.09.2016	Added deep_sleep Reviewed and renewed document	3.0	PSB
18.11.2016	Changed Logo in document Formatted code Changed styling	3.0	ASB
21.11.2016	Rewrote error codes section	3.0	PSB
02.02.2017	Added send_file command	3.1	PSB
06.02.2017	Added correct Remote Error string Added correct connection status string	3.1	ASB
24.02.2017	Updated error code list	3.1	ØK
01.06.2017	Added handshake parameter to RAW OTA protocol Fixed error in CLIENT OTA specification ("size" should be "length")	4.0	ØK
25.09.2017	Fixed status result format	4.0	ASB
26.09.2017	Introduced MQTT messaging. Explained Master / IOT-hub communication life cycle. Added debug setup filtered by severity and flags. `ssl_enabled` is now deprecated (ssl is always enabled).	4.0	ØK
10.10.2017	Corrected response format the client receives after get_upgrade command	3.1	ASB
12.10.2017	Added Get Time	4.0	ASB
08.11.2017	Fixed mistake in Connection status. Explained silent connect flag correctly. Documented time_zone and dst_zone in setup command	4.0	ØK ASB

2 Contents

3 Introduction

The WRF01 chip is a complete hardware driver for connecting a client controller to a network endpoint (DeviceDrive cloud solution by default). All cloud communication goes through a serial connection, using JSON formatted strings.

WRF01 connects automatically to the local WiFi network and can easily be configured using HTTP requests to the local HTTP server implemented in the chip.

The **SmartLinkup** procedure allows for connecting the WRF01 to a given SSID/password without having to connect to the Soft Access point of the chip.

The client can also instruct the WRF01 to become visible as a local Access Point(AP), so that an application can send connection details directly to the device. This procedure is known as the **LinkUp** procedure.

Clients can send and receive messages in a single operation or in separate operations. In firmware version 4.0 the module connects using MQTT to allow for asynchronous messaging without having to poll the server.

To get started with the WRF01, check out the Getting Started documents or our YouTube channel https://www.youtube.com/channel/UCPXLPuDVMSlcc-MEGhW1kGw

4 Startup sequence

This section defines the startup sequence of WRF01, which takes place on GPIO pin 0, the serial port and the WiFi-module.

On startup, the following sequence occurs:

1. An undefined sequence of characters appears on the UART with undefined baud rate. This is the bootloader, and this data should be ignored.
2. The UART is set to 115200 baud, none parity, 1 stop bit, 8 data bits
3. 500ms after startup:
   a. Startup token 0x02 + 0x03 ([STX][ETX]) is sent over UART
   b. GPIO pin 0 is set to LOW, indicating that WRF01 is operable and ready to receive commands.
4. Client starts SmartLinkup procedure (alternatively you can use the old Linkup procedure exposing the Soft AP)
5. Mobile phone sends SSID, password and secret token to the module
6. WRF01 is connected to the local WiFi and the client gets connection message with mac address and RSSI value

5 Specifications

Character encoding	UTF-8, JSON formatted string
Max in/out-bound transmission length	1024 characters
Serial baud rate	115200 baud (8 data bits, 1 stop bit, no parity)
End of transmission character trigger	0x04 [EOT]

6 Serial JSON interface

The serial interface of WRF01 is used to communicate directly with a cloud solution through the DeviceDrive cloud hub.
Each transmission to or from the serial port are terminated with the EOT character (0x04)[EOT]
Hence no command or transmission will be executed before the EOT character is received.

6.1 Connection status

At start-up, WRF01 will attempt to connect to the local WiFi network. Once a connection is successfully established a connection status message is ready to be sent. The message is sent on the UART when the silent_connect flag is set to 0 and WiFi connection is established (see notes below),

Reply	`{ "configuration":{"mac":"18fe34a0d63f"}, "device_state":{"rssi":"-70"} }`
Mac	The MAC address of the WRF01 station interface.
Rssi	Station signal strength.

NOTE:
1. The status is not automatically sent on start up. Setup parameter silent_connect=0 or HTTP command “init” must be executed first.
2. If the station cannot connect within 25 seconds and the Soft AP is visible, the module will stop scanning to keep the module stabile on the Soft AP channel.
3. This output will also be sent upon successful connection after an Init HTTP operation. See WRF01 HTTP specification for details.
4. This output will be disabled if the "silent" flag is set in an Init HTTP operation.

6.2 Cloud transfer

Once the WRF01 is connected to the local WiFi, a client application can send any JSON string to the serial port, and that JSON string will be forwarded to the DeviceDrive cloud. Based on your type of product, the message will either be interpreted by our servers or forwarded untouched to your endpoint.

The received character buffer is sent once the [EOT] character is sent (0x04). The next inbound message from the cloud is reported back on the serial port, also terminated with [EOT].

Control sequence	Description	Response on UART
<Message> [EOT]	<Message> is sent and the corresponding reply message is polled from the in-queue. Other in-queue messages will be ignored. Only applicable for “forwarding” products (see DeviceDrive cloud offers). From v4.0 WRF01 will enable MQTT for “Internal” products, and messages from the cloud will be sent to the client asynchronously.	The reply message if any is returned. Otherwise DeviceDrive response with “SENT”.
[EOT]	Message is polled from the in-queue. No message is sent. For “Internal” products, this command will connect to MQTT to be ready to receive messages from the cloud.	The response message if any. Otherwise “EMPTY”
<Message> [ETX][EOT]	Message is sent, but no message is polled from the in-queue.	DeviceDrive response with “SENT”

Response message when no pending inbound message:

Response message	Description
`{"devicedrive":{"result":"`**`EMPTY`**`"}}<EOT>`	Nothing to send or receive.
`{"devicedrive":{"result":"`**`SENT`**`"}}<EOT>`	Message sent but inbound queue was empty.

If error messages are enabled, all errors will be returned in an ERROR format. See Error codes. This feature is disabled by default.

If the receive buffer overflows, an error message is issued. See the RX_OVERFLOW description for details.

6.3 JSON commands

NOTE:
WRF01 Commands are also JSON structured messages, but must be formatted the following way:
{"devicedrive":{ ... }}

If the sequence is formatted differently, the WRF01 will interpret it as a message to be sent to the cloud. If there are clutter or other characters on the line, the message will be sent as a cloud transfer message.

6.4 Reply format

When a JSON command is executed successfully, the following string will be issued:

Reply (JSON):	`{`
	` "devicedrive":{`
	`  "result":"OK"`
	` }`
	`}`

If an error occurs, the error message format is used as explained in section 0.

6.5 Setup

The client connected to the WRF01 can send setup commands to the WRF01. These commands can be sent as one message, or divided into smaller packets. These options are how debug_mode or error_mode will be handled, what prefix you want on the local AP's SSID, etc. For the complete list of configurations, see the table below.

The setup command is essential to setting up the WRF01, and should be done as soon as the WRF01 is powered on and ready.

Command:	`{"devicedrive":{`
	`  "command":"setup",`
	`  "<param>":"<value>",`
	`  …`
	` }`
	`}`

Parameter	Value
`debug_mode (*)`	If enabled, debug info is logged to UART2 TX on GPIO2. This can be used to troubleshoot client operations and may be attached to support cases. **all** – Send all debug info. **error** – Only log error messages. **info** – Log info and error messages. **dev** – Log all error messages. **none** – No debug info is sent (default). E.g. `"debug_mode": "info"` Debug is also filtered based on `debug_flags` parameter. **IMPORTANT:** Never enable debug in a production system. This is only for debugging purposes and involves significant overhead in the system.
`debug_flags (*)`	Flags selecting the internal logging modules to enable: 1 – Other 2 – Serial commands 4 – Cloud operations 8 – HTTP (client and server) 16 – Upgrades 32 – File transmission 64 – Wi-Fi (station and AP) 128 – MQTT Default: 255 = all modules enabled. E.g. 4+8=12 -> Enables Cloud and HTTP outputs `"debug_flags":12` **Note**: You can use DeviceDrive client simulator tool to set these values.
`error_mode (*)`	Setting error behaviour. **`none`** – No errors are sent to serial port. **`all`** – All errors are sent to serial port. **`local`** – All local errors are sent. Remote errors are given as a local error message (`REMOTE_ERROR`). **`remote`** – Only remote errors are sent on the serial port.
`ssid_prefix (*)`	The SSID prefix to apply next time when the WRF01 is set visible. E.g. “MyPrefix” will give “MyPrexix_aa:bb:cc:dd” (abcd is the last part of the mac address). Default = “DeviceDrive” (also applied if the value is empty).
`visibility`	Enabling or disabling Soft-AP visibility. Default = `0`.

	0 = Turn off AP -1 = Turn on AP indefinitely <xx> = Let AP stay on the next xx seconds. Using currently set ssid_prefix. **NOTE:** To avoid channel hopping, this command initiates a scan to determine the most likely station channel before activating the AP (if station is not connected already). This will delay the activation with some seconds (2-10seconds).
`ssl_enabled`	Deprecated. SSL is now always enabled (except when downloading firmware from cloud).
`network_ssid`	Station network ssid. Used by the WRF01 to connect to the local network.
`network_pwd`	Station network password. Used by the WRF01 to connect to the local network.
`silent_connect`	1 – (default)If network connect should not be issued. 0 –If network connect should be issued. The connection string will be issued even if the network is already connected, and will be delayed with at least 400ms after the OK response.
`token (*)`	Custom string to follow in the header of each cloud transfer. Can be used in linkup process and for authentication of the device in the target cloud. Max length: 32 characters.
`product_key (*)`	Id string identifying the DeviceDrive IoT product to which the device should be associated. Max length 32 characters.
`version`	Version of the client software. Max 16 characters.
`master_url(*)`	The master URL for sending data. This will override the default devicedrive server address. Beware that OTA will not be run from DeviceDrive if this function is active. Empty string will reset back to the DeviceDrive cloud servers.
`time_zone(*)`	The time zone the device should follow. This is supported from version 4.0 and the values range from -11 to 13. If this is not set, the default value is set to 0 which is Greenwich Mean Time(GMT).
`dst_zone(*)`	The Daylight Saving Time(DST) Zone the WRF01 should follow. This is supported from version 4.0. 0 - (default) should not use dst 1 – Use DST from last Sunday in March to last Sunday in October (Applicable for most of Europe)

(*) Not reset upon restart. Stored in flash memory.

6.6 Introspect

The Introspect is a JSON object that describes your device and its capabilities. This enables the mobile app to render the device controls and info tiles.

Command:	`{"devicedrive":{`
	 `  "command":"introspect",`
	 `      "interfaces": [`
	 `        [...]`
	 `    ]`
	 `  }`
	 `}`

interfaces

Example:
To define an oven with, 4 parameters called ambient temperature, target temperature, power and switch, you can send the following introspect document. The document defines the following properties:

- @ambient_temperature is a property, which is a read-only decimal.
- @target_temperature is a property, which is a read-write decimal.
- @power is a property, which is a read-write long.
- @switch is a property, which is a read-write boolean.

The decimals, long and strings will be shown in the app as tiles with information. The name will be formatted for readability. Boolean will create a button the user can switch on and off.

```
{"devicedrive":{
   "command":"introspect",
   "interfaces": [
      ["com.somecompany.heater",
              "@ambient_temperature>d",
              "@target_temperature=d",
              "@power=u",
              "@switch=b"]
   ]
}}
```

When your devices send data, you must match this format for the system to acknowledge the new data. If everything is correct, the new data should be visible in the mobile app.

```
{"com.somecompany.heater" : {
      "ambient_temperature":"23.5",
      "target_temperature":"25.0",
      "power":"700",
      "switch":"0"
   }
}
```

When you receive data from the mobile app through the read-write or write properties, you will only receive the new and updated value, like this:

```
{
   "com.somecompany.heater": {
     "switch": "0"
   }
}
```

6.7 Reboot

The client can send the reboot command to restart the WRF01 firmware.

Command:	`{"devicedrive":{"command":"reboot"}}`

6.8 SmartLinkup

Connect the device in seconds, without having to switch to the Soft AP network on the phone.

Uses the Espressif ESP-TOUCH linkup process where the SSID and password are transmitted from the mobile app without having to use the AP mode.

1) The app encodes the SSID and password in several raw Wifi packets
2) The WRF01 decodes the information and connects to the AP
3) When the device is connected, the IP address is sent back to the mobile app
4) The app uses the IP address to transfer the secure token on HTTPS to the WRF01

Command:	`{"devicedrive":{"command":"smart_linkup", "timeout":<timeout seconds>}}`

Parameter	**Value**
Timeout seconds	Timeout in seconds for the attempt to complete the smart linkup. Range 15..255 seconds.

Disclaimer: The ESP-TOUCH process does not encrypt the Wifi password, so that it could potentially be picked up if someone is sniffing for this activity in the vicinity. SmartLinkup encrypts the token as part of the HTTPS communication.

App support: SmartLinkup is supported by DeviceDrive.LinkUp.SDK.Xamarin.

On success: The connection string is issued

Error code on timeout or failure: `SMART_LINKUP_FAILED`

`NOTES:`

- Cloud operations will return NOT_ONLINE during the process.
- Setting ssid, password or token will cancel the SmartLiknup process (setup command or HTTP init).
- Setting visibility will cancel the SmartLinkup process.
- Status SMART_LINKUP will be issued on the status command.

6.9 Status

The client can send a command to get the current state of the device, with the connection status, IP address, local visibility mode and error codes.

Command:	`{"devicedrive":{"command":"status"}}`

Reply (JSON):

```
{"devicedrive":
    {"status":
        {
        "connection_status":"<connection status>",
        "connection_ip":"<ip address>",
        "local_visibility": "<visible status>",
        "last_error_code":"<error code>",
        "last_error_msg":"<error msg>",
        "successful_transfer_cnt":"<count>",
        "subscribed":0,
        "debug_enabled":1
        }
    }
}
```

NOTE:
The status reply is the same as in the http status request, only wrapped with the devicedrive json node. Please refer to the http specification for details on the response data.

Parameter	**Description**
`connection_status`	The current status of the network connection. See next table for details.
`connection_ip`	The current IP the device has on the local network
`local_visibility`	**ON** - The AP is on and visible. The device is ready for Linkup. **OFF** - The AP is off.
`last_error_code`	If the WRF has encountered a problem with a message or command, the error code will show here
`last_error_msg`	In addition to an error code, you can get the whole error message of what went wrong.
`successful_transfer_cnt`	How many messages has the WRF01 sent successfully to the cloud
`subscribed`	1 – If MQTT is connected and subscribed to the cloud. Polling not required. 0 – If MQTT is disconnected. Requires polling to receive messages.
`debug_enabled`	1 – If debug logging is enabled. Omitted if debug logging is disabled.

Connection status	Description
`CONNECTING`	WRF01 is trying to connect to the configured network.
`IDLE`	WRF01 is idle. Not trying to connect.
`GOT_IP`	WRF01 is online.
`CONNECTION_FAILED`	Problem connecting to the currently registered WiFi network. Still trying.
`NO_AP_FOUND`	The currently registered SSID is not found. Still trying.
`WRONG_PASSWORD`	Could not connect to the local Wifi because the currently registered password is wrong. Still trying.
`SMART_LINKUP`	WRF01 is in smart linkup mode.

6.10 Deep sleep

The client code can set the WRF01 in deep sleep, ensuring low battery drain and fast wakeup time. When the WRF01 is in deep sleep, only the internal RTC is alive.

To ensure that the WRF01 can wake up from deep sleep, two pins from the WRF01 must be connected.

- XPD_DCDC (GPIO16)
- EXT_RSTB (Reset)

The WRF01 uses these pins to send reset signal to itself, when a wakeup occurs either through a timer or an external pin.
The external pin is connected to the combination of XPD_DCDC and EXT_RSTB pins, and a low pulse will generate a wakeup. Wakeup will cause a normal start up routine.

Command:	`{"devicedrive":{`
	`"command":"deep_sleep",`
	`"seconds":"<duration>"`
	`}`
	`}`

Duration (integer)	The duration to sleep in secods. 0 = Sleep until hardware wake-up

6.11 Clear

The client can delete the current WiFi settings and Token. After this command is issued, a new LinkUp procedure is required, as the wifi settings and token is necessary to send data to the cloud servers.

NOTE:
The actual clear operation is done after a reply is sent back to the client. Wait 300ms after the reply before continuing normal operations.

Command:	`{"devicedrive":{"command":"clear"}}`

6.12 Factory reset

The client can completely clear the WRF01. This command will erase wifi information, flash configuration including the product key and token.

This means that a new setup command must be issued.

NOTE:
The actual clear operation is done after a reply is sent back to the client. Wait 300ms after the reply before continuing normal operations.

Command:	`{"devicedrive":{"command":"factory_reset"}}`

6.13 Get Time (Version >= 4.0)

The client can request the time from the WRF01. The WRF01 must have access to the Internet to access its SNTP servers. If the WRF01 cannot access the SNTP servers, it returns with error NO_TIME.

NOTE:
The clock follows time zone and DST zone. To set these, see Setup command or Init call in DeviceDrive_WRF01_http_spesification.

The SNTP servers utilized by the WRF01: europe.pool.ntp.org, ntp.sjtu.edu.cn, us.pool.ntp.org.

Command:	`{"devicedrive":{"command":"get_time"}}`
Response:	`{"devicedrive": {"time": [<timestamp>,<week_day> <day>,<month>,<year>, <hour>,<min>,<sec> <timezone>, <dst> ] } }`
Description:	`timestamp:(UINT64) Seconds since 1970.01.01 00:00:00, Time Zone and DST are incorporated.`
	`week_day: (INT) range from 0-6 (Mon-Sun)`
	`day: (INT) day of the month`
	`month: (INT) range from 1-12 (Jan-Dec)`
	`year: (INT)`
	`hour: (INT) range from 0-23`
	`min: (INT) range from 0-59`
	`sec: (INT) range from 0-59`
	`timezone: (INT) range from -11 to 13 where GMT is 0`
	`dst: (INT) 1 if Daylight Saving Time (DST) is currently active, 0 if not`

Example:

Actual current time	Tue Oct 10 13:47:30 2017 GMT +2 DST
Command	`{"devicedrive":{"command":"get_time"}}`
Response	`{"devicedrive": {"time": [1507643250,2, 10,10,2017, 13,47,30 2,1 ] } }`

6.14 Send file (Version >= 3.1)

When using an internal product and SSL, the WRF01 can upload a file to the cloud service. This could be a picture or a time series of data, collected over time.

The upload is initiated by sending a prepare message with the desired filename and the complete file size. This will put the WRF01 in a special mode for transmitting files. When the WRF01 is ready to receive your file, you will get a confirmation result message, in the standard JSON format. The WRF01 has some restrictions on file size, so you will receive a “max_packet_size” value. Make sure the size of your individual packets is this length or less.

Command:	`{"devicedrive":{`
	`   "command":"send_file",`
	`   "length":"<file_size>",`
	`   "file_name":"<file_name>"`
	`}}`

	`length : (INT) total size of the file`
	`file_name : (STR) name of file`

Response	`{"devicedrive":{`
	`    "max_packet_size" : "<size>",`
	`    "result" : "START_TRANSMISSION"`
	`}}`

	`max_packet_size : (INT)`

When START_TRANSMISSION is received, the WRF01 suspends all other cloud and local operations, and is ready to receive the data file for upload. To initiate the first packet, indicate the start with [STX] followed by data, limited by the max_packet_size, then CRC and last [EOT] The package should looke like this:

[STX]<data><CRC>[EOT]

Please see CRC32 Algorithm for details on how to calculate the CRC. If the CRC matches and the packet is transferred successfully, the WRF will respond with [ACK] (0x06) and you can continue with the next packet.

Otherwise the WRF01 will respond with [NAK] (0x15).

If a [NAK] is received:

You can retransmit the packet and wait for ACK/NAK again

Or you can cancel the operation.

Use [CAN] (0x18) to escape file upload mode, and start the procedure from the send_file command.

If [CAN] is received from WRF01, the operation has failed, and an error message will follow as described below. After the last packet is send, wait for [ACK], then send another [EOT] to confirm that the transmission is done. You will now receive a confirmation that the file has been uploaded to the cloud:

{"devicedrive":{"result":"FILE_SENT"}}

Example:

```
Request --->
{"devicedrive":{"command":"send_file", "length": "600","file_name": "myImage.jpg"}}[EOT]
<--- Response
{"devicedrive":{"max_packet_size" : "361","result" : "START_TRANSMISSION"}}[EOT]
Request --->
[STX] <361 byte of data> <CRC>[EOT]
<--- Response
[ACK]
---> Request
[STX] <239 byte of data> <CRC>[EOT]
<--- Response
[ACK]
---> Request
[EOT]
<--- Response {"devicedrive":{"result":"FILE_SENT"}}[EOT]
```

NOTE:

Although this feature is implemented in the WRF01, there is currently no endpoint API to retrieve the files from the cloud. This will be implemented later

6.15 OTAU support

Over the Air Upgrade (OTAU) is supported for both the WRF01 and the client by the WRF01 in conjunction with the https://manage.devicedrive.com. Client firmware files are uploaded by the manufacturer for each registered product type and version, so that they become available for the device.

> **Security warning:** The WRF01 does not support SSL encryption for firmware downloads, due to the sheer size of files transferred. You should not use this feature if your firmware files contain sensitive information.

File integrity is handled with CRC control of the received file in the WRF01 flash before sending the file to the client.

6.15.1 Upgrade (Deprecated) (Version < 2.2.1)

Deprecated: Use the check_upgrade **and** get_upgrade **instead.**

6.15.2 Check upgrade (Version >= 2.1.1)

The client can request a list of available upgrades if any exists from the cloud. Available upgrades are returned in any message from the cloud to the WRF01, so no upgrades will be available if no messages (or replies) are received from the cloud since reboot.

The supported firmware modules for OTAU are:

- "WRF01" - Firmware for WRF01.

- "CLIENT" - Firmware for the client controller.

Command:	`{"devicedrive":{"command":"check_upgrade"}}`

Reply format (JSON):	`{"devicedrive":{"upgrade":[ ... ]}}`
upgrade	Array of firmware modules to be upgraded for this device. The list is based on the planned upgrades in the cloud OTA system.

Example of reply

Upgrade	Description
{"devicedrive":{"upgrade":[]}}	No upgrades available
{"devicedrive":{"upgrade":["WRF01"]}}	Upgrade available for the WRF01
{"devicedrive":{"upgrade":["CLIENT"]}}	Upgrade available for the CLIENT firmware
{"devicedrive":{"upgrade":["WRF01","CLIENT"]}}	Upgrade available for both the WRF01 and the CLIENT

Use get_upgrade to get the firmware

6.16 Get upgrade (Version >= 2.1.1)

The client can issue this command to fetch the upgrades available based on the list in check_upgrade

After downloading the file to the WRF01 flash, it will either upgrade the WRF01 or transfer the file to the client based on the protocol defined in the request.

Command:	`{"devicedrive":{`
	`    "command":"get_upgrade",`
	`    "module":"<module>",`
	`    "file_no":<number>,`
	`    "delay":<milliseconds>,`
	`    "pin_toggle":"<toggle pattern>",`
	`    "protocol":"<protocol>"`
	`    }`
	`}`

Parameter	Description
module	Can be "WRF01" or "CLIENT", but not both. If WRF01 is selected, the rest of this command may be ignored.
file_no	The file number of the desired firmware. See description of how to package multiple files in an upgrade below. Default is 0.
delay	Number of milliseconds from the file is downloaded, after the reply, before the upgrade starts. Could be used to prepare the client bootloader for the upgrade. Default value is 0
pin_toggle	If an external reset pin for the client hardware is connected to the WRF01, the WRF01 can send a pulse train to the client to restart the hardware. You can in total control two pins. Connect one to GPIO4 and the other to GPIO5. GPIO4 is called ResetPin (R) and GPIO5 is called ProgramPin (P). 0 - Both pins are low 1 - R high, P low 2- R low, P high 3 - R high, P low. Default value is 0 There can be max 10 characters. Each character will hold the pin in a state for 5 ms. **Example:** "031" - Both low for 5 ms, both high for 5 ms, R high and P low for 5 ms.
protocol	The WRF01 supports the following bootloader protocols: RAW - Will write the firmware file directly as is over UART HANDSHAKE – Same as RAW but the bootloader must now request the number of bytes to receive. Repeated until entire file is downloaded. ARDUINO_ZERO - Follows SAM-BA protocol with XModem for bootloading the Arduino Zero.

Notes:

1. The pin toggle pattern is configured so that GPIO4 is connected to the Arduino's reset pin.
2. Do not power down the device until 50ms after receiving the reply to be sure that the WRF01 get time to perform a valid reboot. Otherwise you risk that it boots up in the old firmware version, and you must upgrade again.
3. For WRF01 upgrades the command can be shortened by omitting file_no, delay, pin_toggle and protocol. See example below.
4. A client firmware upgrade can consist of **multiple files** and are handled by the management portal. A client can request files based on a file number in the request for get_upgrade. This is done automatically for WRF01 upgrades. To upload multiple files, they must end with the number (e.g. firmwarefile1.bin, and firmwarefile2.bin), and

packaged using tar.gz. Max size per file is 500KB, as they are stored in the free area of the WRF01 flash.

Examples:

Description	Example
Upgrading WRF01 module	{"devicedrive": { "command":"get_upgrade", "module":"WRF01" } }
Upgrading Client module through RAW protocol, without any delay or pin-toggle	{"devicedrive": { "command":"get_upgrade", "module":"CLIENT", "protocol":"RAW" } }
Upgrading an Arduino Zero from the WRF01:	{"devicedrive": { "command":"get_upgrade", "module":"CLIENT", "delay":500, "pin_toggle":"010", "protocol":"ARDUINO_ZERO" } }

Replies for WRF01 upgrade:

After a WRF01 upgrade command has been issued, a standard `{"devicedrive":{"result":"OK"}}` is replied, followed by a reboot after the upgrade is finished.

Replies for Client upgrade:

Before the actual upgrade is sent, the client receives a reply with the details of the firmware file to be downloaded. You can use the delay to have time to crosscheck this reply.

Reply	{"devicedrive":{"upgrade": {"length":<length>,"crc":<crc>}}}

Parameter	Description
Length	Number of bytes in the firmware ready to transfer
Crc	The WRF01 does a CRC check on the file and sends the result CRC to the client. The WRF01 uses a CRC32 algorithm. See the code below.

CRC32 Algorithm:

```
int * init_crc_table(void)
{
    int* crc_table;
    unsigned int c;
    unsigned int i, j;
    crc_table = (unsigned int*)malloc(256 * 4);
    for (i = 0; i < 256; i++) {
        c = (unsigned int)i;
        for (j = 0; j < 8; j++) {
            if (c & 1)
                c = 0xedb88320L ^ (c >> 1);
            else
                c = c >> 1;
        }
        crc_table[i] = c;
    }
}

unsigned int crc32(unsigned int crc, int* crc_table, unsigned char *buffer,
                   unsigned int size)
{
    unsigned int i;
    for (i = 0; i < size; i++) {
        crc = crc_table[(crc ^ buffer[i]) & 0xff] ^ (crc >> 8);
    }
    return crc;
}

/* Replace this function with your implementation */
unsigned int calc_crc(char* buffer, int size)
{
    int *crc_table = init_crc_table();
    unsigned int crc = 0xffffffff;
    crc = crc32(crc, crc_table, buffer, size);
    free(crc_table);
    return ~crc;
}
```

6.16.1 Handshake protocol (Version >= 4.0)

The “HANDSHAKE” protocol lets you synchronize the firmware download with the pace at which it can be consumed by the bootloader. This solves the problem of not being able to save to flash at the same pace as the UART stream.

Initial handshake:

1) WRF01 sends the ACK character (0x06)
2) Client bootloader sends the ACK character (0x06)
   If a different character is sent by the client bootloader, WRF01 will interpret it as the start of a new regular WRF01 message. This prevents the problem of client code rebooting and not being in bootloader mode.

After the initial handshake, the protocol is a repetition of the following steps:

1) Client bootloader sends the number of bytes to receive as 2 bytes little endian: [LSB][MSB]
   Eg. 10 bytes is [0x0A][0x00]
2) WRF01 delivers the number of bytes requested until there are no more bytes.

The client bootloader must repeat the steps above until the entire firmware file is consumed and then starts with the new program.

To abort the process before it is finished, the client can send the special value [0x00][0x00]. The WRF01 will then exit to normal mode immediately.

Note: The client binary file can be encoded with a custom header including length and signature so that the bootloader can know the length from the content itself and check the file integrity against a signature to prevent malicious upgrades.

6.17 Error messages

Any error during command handling or cloud transfer will be issued as an error message on the serial port.

NOTE: If we have started to receive data from the host before the error occurred, the EOT character (0x04) will be issued before the error message to terminate the incomplete data.

JSON format:	`{"devicedrive":{"ErrorCode":"<error code>", "ErrorMsg":"<error msg>"}}`
Error code	The error code string. E.g. “`NOT_ONLINE`” Please see section 7 for a complete list of error codes.

Examples:

Local error	`{"devicedrive":{"error":"NOT_ONLINE"}}`
Remote error	`{"DeviceDrive":{"ErrorCode":"INVALID_JSON"}}`
NOTE: The difference between local and remote error, is the capital D’s in DeviceDrive for remote error and small d’s for local errors	

7 Error Codes

Errors are split into two levels. As a user, you can select All, None, Local or Remote errors only. Local errors are codes from the WRF01 itself, while Remote Errors originate in the Cloud services.

Error codes are reported in all error situations as a JSON string (see section above).

7.1 Local Error codes

This is a complete list of all supported local error codes.

Error code	Description
`COMMAND_FAILED`	The given command failed to execute. Please see the error message and log output for further details. **NOTE:** This error also occurs when sending the "introspect" command if the document is rejected by the server. E.g. because of **invalid token or product key**.
`DNS_FAILED`	Could not perform a required DNS lookup. Please try again or restart WRF01 if the problem persists.
`FILE_TRANSFER_FAILED`	The file transfer failed to execute. Please see log output for details.
`INVALID_JSON_COMMAND`	Failed to parse JSON for the given command.
`MASTER_REQUEST_FAILED`	Issued if an error occurred while connecting to the MASTER agent server.
`NONE`	Issued by "last_error_code" if there have not yet been any errors since startup or last Init (HTTP command).
`NOT_ONLINE`	System is not online. Sent on cloud transfer or update if the system is not yet online.
`RECEIVE_REQUEST_FAILED`	Issued if an error occurred while receiving JSON message from the cloud.
`REMOTE_ERROR`	An error is sent as JSON from the server. The error is overridden by this local error message if error mode is `local`.
`RX_OVERFLOW`	Issued if the receive buffer of the serial port is overflowed before it is handled. The receive buffer can take up to 2700 characters. The error can also be issued if WRF01 is busy with transmission of content to the cloud, since the buffer is not handled at that time. Client applications should always wait for cloud reply before issuing subsequent messages.
`SEND_REQUEST_FAILED`	Issued if an error occurred while sending JSON to the cloud system.
`SET_AP_MODE_FAILED`	Issued if the internal initialisation of the Wifi module fails.
`SYSTEM_BUSY`	Cloud operation is in progress. Client must wait until the operation is finished (wait for response from WRF01). Please make sure errors are enabled to allow response even if the operation fails.
`UNKNOWN_COMMAND`	Issued if the given command is not recognized.
`UPGRADE_ERROR`	Error running firmware upgrade from cloud. Please retry.
`SMART_LINKUP_FAILED`	Issued if the SmartLinkup process failed or timed out.
`NO_TIME`	Issued if WRF01 are unable to retrieve time from SNTP servers. Check that the WRF01 has internet if this occurs.
`MQTT_FAILED`	Issued if the MQTT connection is broken (after 3 retries) and the client must reconnect using a poll operation.

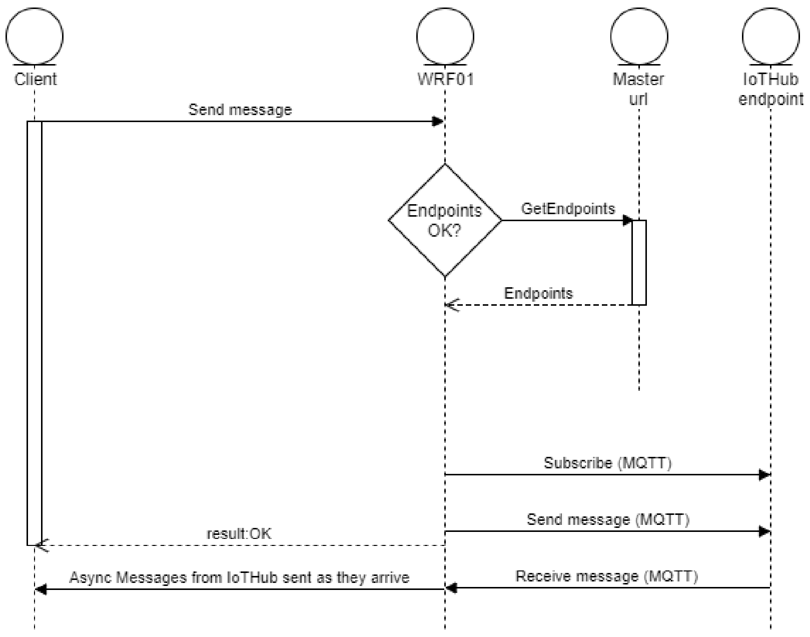
7.2 Cloud Error codes

This is a complete list of all supported remote error codes.

Error code	Description
UNDEFINED_ERROR	Errors that occurs in the cloud services. If you receive this, please contact DeviceDrive support
INVALID_JSON	The JSON document is invalid, or badly formatted
INVALID_INTROSPECT	The introspection document is not formatted correctly. Consult the documentation
INVALID_HEADER	The WRF01 header is malformed or missing required fields
FORWARDING_ERROR	Typically occurs if the forwarding URL is missing in the Management portal
MISSING_HEADER	No header received in the cloud portal.
INVALID_TOKEN	Occurs if the token stored in the WRF01 does not match the token assigned by the LinkUp system when device is configured as an Internal product
INVALID_APP	Occurs if the user tries to link up a product, that the app does not have permission to control.

7.3 Cloud messaging

When a send or receive operation is initiated, WRF01 checks to see if we already have valid communication endpoints. If not, they will be retrieved from the master server ("internal" DeviceDrive products only).



If the WRF01 is not already connected the WRF01 connects to the MQTT endpoint and subscribes to incoming messages.

A send operation will send the message directly on the open MQTT connection.

The MQTT connection stays active until:

1) The WRF01 is disconnected from the network (e.g. reboot or bad Wi-fi link)
2) or if a REST operation is required (e.g. file transfer or OTA download)
3) The soft AP is set active (visibility activated)

Any subsequent receive-operations will be ignored while the MQTT connection is active. Any incoming message to the device will be forwarded to the client directly (even if no receive operation was ever requested).

Any subsequent send-operation will send the message directly on the open MQTT.