ESP8266 DOWNLOAD PROTOCOL

Basic rules:

- All the command and data frames are sent under SLIP(Serial Line Internet Protocol).
- All the data length and checksum are counted or calculated with raw data without SLIP.
- Each packets begin with 0xC0. Each packets end with 0xC0
- If 0xDB in data frame, replace with 0xDB 0xDD. If 0xC0 in data frame, replace with 0xDB 0xDC
- Data frame formed in struct like this:

  response:{ header; payload; }

- Data fields are all little-endian

ESP8266

Request packet:

```
struct header {
    uint8 ucMsgType;
    uint8 ucMsgID;
    uint16 ucMsgLength;
    uint32 ucReserved;
}
struct request {
    header _header;
    uint8_t payload[];
}
```

Response packet:

```
struct header {
    uint8_t ucMsgType;
    uint8_t ucMsgID;
    uint16_t ucMsgLength;
    uint32_t ucMagicCode;
};
struct body {
    uint8_t RspFlag;
    uint8_t RspFailType;
};
struct response {
    header _header;
    body _payload;
};
```

ESP8266:

	MsgType	**MsgID**	**LEN**	**RSVD**	**FLAG**	**FAILTYPE**
Length(Bytes)	1	1	2	4	1	1

ESP32

	MsgType	**MsgID**	**LEN**	**RSVD**	**FLAG**	**FAILTYPE**
Length(Bytes)	1	1	2	4	2	2

MsgType	**Type**
0	Data Request
1	Data Confirm

MsgID	Type
0x0	Reserved
0x1	Reserved
0x2	Flash download begin
0x3	Flash download data
0x4	Flash download end
0x5	Memory download begin
0x6	Memory download end
0x7	Memory download data
0x8	Data synchronize frame
0x9	Write register
0xa	Read register
0xb	SPI parameter configure

RESPONSE ERROR CODE

Response FailType	ERROR
0	UNKNOW
1	INVALID INPUT
2	MALLOC FAILED
3	SEND MSG FAILED
4	RECV MSG FAILED
5	EXECUTE FAILED

Response Flag	Description
0	OK
1	ERROR

1. Baud Sync

The hardware of ESP8266\ESP32 has a baudrate auto detect function. Before we send data to the chip, we have to synchronize the data rate. The master side will sync packet to chip, and wait for reply, if the reply is correct, that means ESP chip got the correct data rate.

> *The whole download task will use the same baudrate as the sync procedure.*

> *The highest baudrate is limited, usually should be no higher than 921600, different from different UART hardware.*

1. 1. Sync request data frame

Original data before SLIP:

```
00 08 24 00 00 00 00 00
04 03 02 01 55 55 55 55
55 55 55 55 55 55 55 55
55 55 55 55 55 55 55 55
55 55 55 55 55 55 55 55
55 55 55 55
```

Frame Header

	MsgType	**MsgID**	**LEN**	**RSVD**
Value	0x0	0x8	0x0024	0x00000000
meaning	request	sync	body length	reserved

Sync Frame Body

MagicCode	**0x01020304**
SyncData	(0x55) * 32

> *fields in header are all little-endian.*
> *MagicCode can be any value you want to set.*

SLIP data:

> *C0 00 08 24 00 00 00 00 00 04 03 02 01 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 C0*

1.2 Sync response data frame

Original data before SLIP:

01 08 02 00 04 03 02 01 00 00

Frame Header

	MsgType	MsgID	LEN	MGCD
Value	0x1	0x8	0x0002	0x01020304
Meaning	response	sync	length	magic code

> *Magic code is the data you set in request frame.*

Sync Frame Body(Success)

Response Flag	Response Fail Flag
0x0	0x0
Success	NULL

SLIP data:

> *C0 01 08 02 00 04 03 02 01 00 00 C0*
> *If the response data frame is correct, we can move on to the next step.*

2. Flash download begin

```
After data rate synchronization, we must send flash download begin command.
ESP8266 chip will :
[a]. initialize SPI
[b]. erase flash according the start address and size.
```

2.1 Flash begin request data frame

Original data before SLIP:

```
00 02 10 00 00 00 00 00
64 6B 05 00 AE 00 00 00
00 08 00 00 00 90 00 00
```

Frame Header

	Length	Value	Meaning
MsgType	uint8_t	0x0	request
MsgID	uint8_t	0x02	Flash download begin
LEN	uint16_t	0x0010	Data length(16 bytes of payload)
RSVD	uint32_t	0x00000000	Reserved

Flash Begin Payload

	Length	Value	Meaning
FileSize	uint32_t	0x00056B64	Total length(355,172Bytes)
PacketNum	uint32_t	0x000000AE	Packet number(FileSize\/PacketSize)
PacketSize	uint32_t	0x00000800	Data size of each packets to send (2048 Bytes each packet)
Address	uint32_t	0x00009000	Download address

> *Note:*
> *ESP8266 will earse flash sectors accoring the "Address" and "FileSize" in flash begin frame.*
> *But in some circumstances, the chip will erase some extra flash sectors.*
> *We recommand that calculate the FileSize(erase_size) like this:*

```
sectors_per_block = 16
sector_size = 4096
offset = start_address
num_sectors = (size + sector_size - 1) / sector_size
start_sector = offset / sector_size
head_sectors = sectors_per_block - (start_sector %
sectors_per_block)
if num_sectors < head_sectors:
    head_sectors = num_sectors
if num_sectors < 2 * head_sectors:
    erase_size = (num_sectors + 1) / 2 * sector_size
else:
    erase_size = (num_sectors - head_sectors) * sector_size
```

So that when the start_sector is odd, ESP8266 will erase one extra sector. When the start_sector is even, ESP8266 will erase exact the given number of sectors.

SLIP DATA:

> *C0 00 02 10 00 00 00 00 00 64 6B 05 00 AE 00 00 00 00 08 00 00 00 90 00 00 C0*

2.2 flash begin response data frame

Original data before SLIP:

> *01 02 02 00 04 03 02 01 00 00*

Frame Header

	Length	Value	Meaning
MsgType	uint8_t	0x1	Response
MsgID	uint8_t	0x02	Flash download begin
LEN	uint16_t	0x0002	Data length(2 bytes of payload)
RSVD	uint32_t	0x01020304	MagicCode in sync packet

Sync Frame Body(Success)

	Length	Value	Meaning
Result	uint8_t	0x0	Success
FailType	uint8_t	0x00	NULL

SLIP DATA:

> *C0 01 02 02 00 04 03 02 01 C0*

3. Flash download data

```
After flash download begin command get a success response, we can send the
data packets in the length of PacketSize.
```

3.1 flash data request data frame:

Original data before SLIP:

```
00 03 10 08 E9 00 00 00
00 08 00 00 00 00 00 00
00 00 00 00 00 00 00 00
E9 03 00 00 94 02 10 40
00 00 10 40 74 05 00 00
......
```

Frame Header

	Length	Value	Meaning
MsgType	uint8_t	0x00	Request
MsgID	uint8_t	0x03	Flash download data
LEN	uint16_t	0x0810	Data length(2064 bytes of payload)
CheckSum	uint32_t	0x000000E9	Check sum of this data frame

> *How to calculate checksum of payload:*

```
def checksum(packet_data):
    checksum_magic = 0xEF
    checksum = checksum_magic;
    for each_byte in packet_data:
        checksum ^= ord(each_byte)
return checksum
```

> *The result will be a one-byte data.*

Flash Data Payload

	Type	Value	Meaning
PacketLen	uint32_t	0x00000800	Packet length(2048 bytes of each data frame, length of data[] field)
SequenceNum	uint32_t	0x00000000	Packet sequence(increase every time from zero)
Reserved	uint32_t	0x00000000	Reserved
Reserved	uint32_t	0x00000000	Reserved
data[ ]	PacketSize	N\/A	Download data, the data field is a byte stream in sequence. (with the size of PacketLen)

> *Sequence number should be increased after each packet.*
> *PacketLen must be accord with the "PacketSize" setting in flash begin command.*
> *For the last packet, if the length is shorter than PacketSize, fill the remaining position with oxFF.*

SLIP DATA:

```
C0 00 03 10 08 E9 00 00
00 00 08 00 00 00 00 00
00 00 00 00 00 00 00 00
00 E9 03 00 00 94 02 10
40 00 00 10 40 74 05 00
00...... C0
```

3.2 flash data response data frame

Original data before SLIP:

> *01 03 02 00 04 03 02 01 00 00*

Frame Header

	Length	Value	Meaning
MsgType	uint8_t	0x1	Response
MsgID	uint8_t	0x03	Flash download data
LEN	uint16_t	0x0002	Data length(2 bytes of payload)
RSVD	uint32_t	0x01020304	MagicCode in sync packet

Sync Frame Body(Success)

	Length	Value	Meaning
Result	uint8_t	0x0	Success
FailType	uint8_t	0x00	NULL

SLIP DATA:

> *C0 01 02 02 00 04 03 02 01 00 00 C0*

4. Flash download stop

>

After all the packets are sent, we should send a flash download stop command. (Actually it is not needed, after sending flash download stop message, ESP8266 will do nothing rather than just lock SPI.)