

EPORT SDK API User Manual

Release v1.3
April 30, 2019

History:

Time	Author	Content	Version
2017.02.18	Ke Zhang	Initialize	V1.0
2017.10.24	Ke Zhang	Add web customization guide	V1.1
2018.02.08	Ke Zhang	Add user flash, GPIO APIs	V1.2
2019.04.30	Ke Zhang	Update EP10 GPIO APIs	V1.3

Contents

0. Introduction	4
1. System Setting	5
1.1 hfep_get_sys_version	5
1.2 hfep_set_sys_user	6
1.3 hfep_set_sys_pwd	6
1.4 hfep_get_sys_netcfg	7
1.5 hfep_set_sys_dhcp	7
1.6 hfep_set_sys_dns	8
1.7 hfep_set_sys_hostname	8
1.8 hfep_get_sys_hostname	9
1.9 hfep_set_sys_telnet	9
1.10 hfep_set_sys_web	10
1.11 hfep_set_sys_mac	10
1.12 hfep_set_reload	11
1.13 hfep_set_restart	11
2. UART Setting	12
2.1 hfep_set_uart_active	12
2.2 hfep_set_uart_baudrate	12
2.3 hfep_set_uart_databits	13
2.4 hfep_set_uart_stopbits	13
2.5 hfep_set_uart_parity	14
2.6 hfep_set_uart_buffer	14
2.7 hfep_set_uart_fc	15
2.8 hfep_set_uart_swfc	15
2.9 hfep_set_uart_cli_getin	16
2.10 hfep_set_uart_cli_wt	16
2.11 hfep_set_uart_proto	17
2.12 hfep_set_uart_frame_len	17
2.13 hfep_set_uart_frame_time	18
2.14 hfep_set_uart_edit	18
2.15 hfep_set_uart_clean	19
2.16 hfep_get_uart_config	19
3. Network Setting	20
3.1 hfep_set_network_rout	20
3.2 hfep_set_network_keepalive	20
3.3 hfep_set_network_timeout	21
3.4 hfep_set_network_security	21
3.5 hfep_set_network_bufsize	22
3.6 hfep_set_network_connectmode	22
3.7 hfep_set_network_active	23
3.8 hfep_get_network_config	23
3.9 hfep_new_tcpserver	24
3.10 hfep_new_tcpclient	25

3.11 hfep_new_udpserver	25
3.12 hfep_new_udpclient	26
3.13 hfep_del_network	26
4. Customer Network&UART Setting	27
4.1 Customer Structure	27
4.2 hfep_custom_net	27
4.3 hfep_custom_tcpserver	28
4.4 hfep_custom_tcpclient	28
4.5 hfep_custom_udpserver	29
4.6 hfep_custom_udpclient	29
4.7 hfep_custom_uart	30
4.8 hfep_custom_net_send	30
4.9 hfep_custom_uart_send	31
5. Custom Web	31
5.1 hfep_custom_web_register	31
5.2 hfep_custom_web_unregister	34
5.3 hfcj_get_object_item	35
5.4 hfcj_get_string	35
5.5 hfcj_get_number	36
5.6 hfcj_get_bool	36
5.7 hfcj_add_object_item	37
5.8 hfcj_add_number	37
5.9 hfcj_add_string	38
5.10 hfcj_add_bool	38
6. Flash & GPIO	39
6.1 hfuflash_read	39
6.2 hfuflash_write	39
6.3 hfsys_set_gpio_value	40
6.4 hfsys_get_gpio_value	40
6.5 ep10_gpio_config	41
6.6 ep10_gpio_set	41
6.7 ep10_gpio_get	42

0. Introduction

Released EPORT SDK already includes EPORT application. By default, it starts up automatically as one of the initial service. It performs the normal UART to Ethernet , Ethernet to UART functionality, CLI (command line), Web service, Telnet, NTP... But these functions cannot meet all customers' requirements.

EPORT SDK APIs provide a method to communicate with the existing EPORT process. Users can quickly develop their own applications based on the existing functions that EPORT has already implemented, such as system setting, UART & Network configuration, UART & Network data receive and transfer, web service, etc.

There are 2 kinds of APIs provided:

1. Platform defined APIs, including thread routines, memory utilities, timer application, mutex operation. In principle, user can directly use Linux C library to do the same, such as pthread_create, malloc, timer_create, pthread_mutex_init... So, they are NOT mandatory for user application. These APIs are there mainly for platform compatible, also they don't need EPORT process to run at the same time. To use them, include the dynamic library 'libhiflying.so' during compilation. For detail, check the header files 'hfthread.h', 'hfplatform.h', 'hftimer.h', 'hfsys.h'. They are not listed in this manual. There are some exceptions, user flash operation, gpio utilities, which will be listed in following section.

2. APIs need to communicate with EPORT process. To configure EPORT, to receive from or send data to EPORT predefined interface... Note that, configurations are automatically saved in Flash, means they will not be lost during power off/on.

1. System Setting

1.1 hfep_get_sys_version

Function prototype:

```
int HSF_API_EX hfep_get_sys_version(char *ver);
```

Description:

Get the version of EPORT service.

Parameters:

ver: Pointer to the memory store the version information, should be maintain(allocate or free) by caller.

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

1.2 hfep_set_sys_user

Function prototype:

```
int HSF_API_EX hfep_set_sys_user(char *user);
```

Description:

Set user name of EPORT service.

Parameters:

user: Pointer to the memory store the user name, should be maintain(allocate or free) by caller.

Return Value:

0: Success

Others: Fail

Notes:

User name is used to login EPORT service, ie. CLI, Web, Telnet...

Header file:

Hfplatform_ep_api.h

1.3 hfep_set_sys_pwd

Function prototype:

```
int HSF_API_EX hfep_set_sys_pwd(char *pwd);
```

Description:

Set user password of EPORT service.

Parameters:

pwd: Pointer to the memory store the user password, should be maintain(allocate or free) by caller.

Return Value:

0: Success

Others: Fail

Notes:

User password is used to login EPORT service, ie. CLI, Web, Telnet... together with user name.

Header file:

Hfplatform_ep_api.h

1.4 hfep_get_sys_netcfg

Function prototype:

```
int HSF_API_EX hfep_get_sys_netcfg(char *mac, char *ipaddr, char *gateway);
```

Description:

Get system mac address, ip address and gateway address.

Parameters:

mac: Pointer to the memory store the mac address. The return format should be 'xxxxxxxxxxxx'

ipaddr: Pointer to the memory store the ip address. The return format should be 'x.x.x.x'

gateway: Pointer to the memory store the gateway ip address. The return format should be 'x.x.x.x'

Return Value:

0: Success

Others: Fail

Notes:

None

Header file:

Hfplatform_ep_api.h

1.5 hfep_set_sys_dhcp

Function prototype:

```
int HSF_API_EX hfep_set_sys_dhcp(bool enable, char *ipaddr, char *gateway);
```

Description:

Set system DHCP configuration.

Parameters:

enable: true means enable dhcp Pointer to the memory store the mac address. The return format should be 'xxxxxxxxxxxx'

ipaddr: Pointer to the memory store the static ip address. The input format should be 'x.x.x.x'

gateway: Pointer to the memory store the static gateway ip address. The input format should be 'x.x.x.x'

Return Value:

0: Success
Others: Fail

Notes:

When enable is true, ipaddr and gateway can be NULL.

Header file:

Hfplatform_ep_api.h

1.6 hfep_set_sys_dns

Function prototype:

```
int HSF_API_EX hfep_set_sys_dns(char *dns);
```

Description:

Set system DNS.

Parameters:

dns: Pointer to the memory store the DNS ip address. The input format should be 'x.x.x.x'

Return Value:

0: Success
Others: Fail

Notes:

Only one ip address is supported.

Header file:

Hfplatform_ep_api.h

1.7 hfep_set_sys_hostname

Function prototype:

```
int HSF_API_EX hfep_set_sys_hostname(char *hostname);
```

Description:

Set host name.

Parameters:

hostname: Pointer to the memory store the host name.

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

1.8 hfep_get_sys_hostname

Function prototype:

```
int HSF_API_EX hfep_get_sys_hostname(char *hostname);
```

Description:

Get host name.

Parameters:

hostname: Pointer to the memory store the host name, should be maintain(allocate or free) by caller.

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

1.9 hfep_set_sys_telnet

Function prototype:

```
int HSF_API_EX hfep_set_sys_telnet(bool enable, bool echo, unsigned short port);
```

Description:

Set telnet parameters.

Parameters:

enable: true means enable telnet, false means disable it.

echo: true means using echo, otherwise it's false

port: telnet port(default is 23)

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

1.10 hfep_set_sys_web

Function prototype:

```
int HSF_API_EX hfep_set_sys_web(bool enable,unsigned short port);
```

Description:

Set web parameters (httpd).

Parameters:

enable: true means enable web service(httpd), false means disable it.
port: httpd port(default is 80)

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

1.11 hfep_set_sys_mac

Function prototype:

```
int HSF_API_EX hfep_set_sys_mac(char mac[6]);
```

Description:

Set device mac address.

Parameters:

mac:device mac address, 6 bytes address.

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

1.12 hfep_set_reload

Function prototype:

```
int HSF_API_EX hfep_set_reload(char type);
```

Description:

Reload configuration.

Parameters:

```
type :reload configuration type, should be one of flowing.
enum EP_CONFIG_TYPE
{
    CONFIG_TYPE_SYS =0,    //system configuration
    CONFIG_TYPE_SOCKET,    //network configuration
    CONFIG_TYPE_UART,      //UART configuration
    CONFIG_TYPE_SYSONLY,   //system configuration only
    CONFIG_TYPE_ALL,       //all configurations
    CONFIG_TYPE_ACT,       //configuration in active store area
    CONFIG_TYPE_BACK,      //configuration in backup area
    //CONFIG_TYPE_FSETTING,
};
```

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

1.13 hfep_set_restart

Function prototype:

```
int HSF_API_EX hfep_set_restart(int delay_ms);
```

Description:

Restart system.

Parameters:

delay_ms :delay time(ms) before system restart.

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

2. UART Setting

2.1 hfep_set_uart_active

Function prototype:

```
int HSF_API_EX hfep_set_uart_active(void);
```

Description:

Activate UART configuration.

Parameters:

None.

Return Value:

0: Success

Others: Fail

Notes:

Normally it is called after setting changed.

Header file:

Hfplatform_ep_api.h

2.2 hfep_set_uart_baudrate

Function prototype:

```
int HSF_API_EX hfep_set_uart_baudrate(char *baud);
```

Description:

Set UART baud rate.

Parameters:

baud: baud rate string, should be one of followings:

"2400", "4800", "9600", "19200", "38400", "57600", "115200", "230400", "460800"

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

2.3 hfep_set_uart_databits

Function prototype:

```
int HSF_API_EX hfep_set_uart_databits(char *bits);
```

Description:

Set UART bits.

Parameters:

bits: UART bits string, should be one of followings:
"5", "6", "7", "8"

Return Value:

0: Success
Others: Fail

Notes:

For EP10/EP20/HF511A, 8 bits is supported

Header file:

Hfplatform_ep_api.h

2.4 hfep_set_uart_stopbits

Function prototype:

```
int HSF_API_EX hfep_set_uart_stopbits(char *bits);
```

Description:

Set UART stop bits.

Parameters:

bits: UART stop bits string, should be one of followings:
"1", "1.5", "2"

Return Value:

0: Success
Others: Fail

Notes:

For EP10/EP20/HF511A, 1 & 2 stop bits are supported.

Header file:

Hfplatform_ep_api.h

2.5 hfep_set_uart_parity

Function prototype:

```
int HSF_API_EX hfep_set_uart_parity(char *par);
```

Description:

Set UART parity.

Parameters:

par: UART parity string, should be one of followings:
"NONE", "EVEN", "ODD", "SPACE", "MARK"

Return Value:

0: Success
Others: Fail

Notes:

For EP10/EP20/HF511A, "NONE", "EVEN", "ODD" are supported.

Header file:

Hfplatform_ep_api.h

2.6 hfep_set_uart_buffer

Function prototype:

```
int HSF_API_EX hfep_set_uart_buffer(char *bsz);
```

Description:

Set UART buffer size.

Parameters:

bsz: UART buffer size string, should be less than 12KB and large than 32B:

Return Value:

0: Success
Others: Fail

Notes:

For EP10/EP20/HF511A, default value is 8K.

Header file:

Hfplatform_ep_api.h

2.7 hfep_set_uart_fc

Function prototype:

```
int HSF_API_EX hfep_set_uart_fc(bool enable);
```

Description:

Enable/Disable UART flow control(RTS/CTS).

Parameters:

enable: true means enable UART flow control, otherwise disable.

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

2.8 hfep_set_uart_swfc

Function prototype:

```
int HSF_API_EX hfep_set_uart_swfc(bool enable, unsigned char xon, unsigned char xoff);
```

Description:

Enable/Disable UART software flow control.

Parameters:

enable: true means enable UART software flow control, otherwise disable.

xon: character to indicate 'ON'

xoff: character to indicate 'OFF'

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

2.9 hfep_set_uart_cli_getin

Function prototype:

```
int HSF_API_EX hfep_set_uart_cli_getin(char *getin, char type, char *sstr, char sstr_len);
```

Description:

Set UART CLI parameters.

Parameters:

getin: set method for CLI mode, should be one of following:

"Disable": Disable CLI mode

"Serial-String": Set trigger to enter CLI mode

"Always": Always in CLI mode

type: input type for "Serial-String" mode, 0: ascii 1: hex

sstr: input string for "Serial-String" mode

sstr_len: string length

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

2.10 hfep_set_uart_cli_wt

Function prototype:

```
int HSF_API_EX hfep_set_uart_cli_wt(int wt);
```

Description:

set UART CLI waiting time (seconds).

Parameters:

wt: should be large than 0 and less than 300.

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

2.11 hfep_set_uart_proto

Function prototype:

```
int HSF_API_EX hfep_set_uart_proto(char *protocol);
```

Description:

set UART protocol.

Parameters:

protocol: should be one of following.
"NONE", "Modbus", "Frame"

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

2.12 hfep_set_uart_frame_len

Function prototype:

```
int HSF_API_EX hfep_set_uart_frame_len(int len);
```

Description:

If UART protocol is "Frame", set length of frame.

Parameters:

len: set the length of frame

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

2.13 hfep_set_uart_frame_time

Function prototype:

```
int HSF_API_EX hfep_set_uart_frame_time(int time);
```

Description:

If UART protocol is “Frame”, set time interval of frame sending.

Parameters:

time: set time interval of frame sending.

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

2.14 hfep_set_uart_edit

Function prototype:

```
int HSF_API_EX hfep_set_uart_edit(char *baudrate, char *databits, char *stopbits, char *parity);
```

Description:

Set UART with multiple parameters.

Parameters:

Reference to above single parameter setting.

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

2.15 hfep_set_uart_clean

Function prototype:

```
int HSF_API_EX hfep_set_uart_clean(char *sockname);
```

Description:

Clean UART session statistics.

Parameters:

sockname: sock name of the UART attached.

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

2.16 hfep_get_uart_config

Function prototype:

```
int HSF_API_EX hfep_get_uart_config(char *baudrate, char *databits, char *stopbits, char *parity, char *flowCtrl);
```

Description:

Get UART configuration.

Parameters:

Reference to above parameter setting, the pointer should be maintained by user and not be NULL.

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

3. Network Setting

3.1 hfep_set_network_rout

Function prototype:

```
int HSF_API_EX hfep_set_network_rout(char *name, char *rout);
```

Description:

Configure network route.

Parameters:

name: network name

rout: the route of the network out, should be one of following

“uart”: network data will be routed to UART

“custom”: network data will be routed to customizing interface

“log”: network data will be route to log

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

3.2 hfep_set_network_keepalive

Function prototype:

```
int HSF_API_EX hfep_set_network_keepalive(char *name, int keepalive);
```

Description:

Set time interval for keep-alive frame to send.

Parameters:

name: network name

keepalive: time interval (seconds) for keep-alive frame to send.

Return Value:

0: Success

Others: Fail

Notes:

If set to 0, or less than 0, it will be set to default value (60 seconds).

Header file:

Hfplatform_ep_api.h

3.3 hfep_set_network_timeout

Function prototype:

```
int HSF_API_EX hfep_set_network_timeout(char *name, int timeout);
```

Description:

Set timeout value for TCP connection. If no data send/receive during this period, it will be treated as disconnect. It will restart the connection.

Parameters:

name: network name

timeout: timeout (seconds) for TCP connection.

Return Value:

0: Success

Others: Fail

Notes:

If set to 0, or less than 0, it will be set to default value (0).

Header file:

Hfplatform_ep_api.h

3.4 hfep_set_network_security

Function prototype:

```
int HSF_API_EX hfep_set_network_security(char *name, char secu, char *key, char keyLen,  
char smode);
```

Description:

Set network security.

Parameters:

name: network name

secu: type of security, should be one of followings

enum EP SOCK SECURITY

{

EP SOCK SECURITY NONE = 0,

EP SOCK SECURITY AES,

EP SOCK SECURITY DES,

EP SOCK SECURITY TLS,

EP SOCK SECURITY PASSWORD,

};

key: key for security, should be less than 64B

keyLen: key length

smode: string mode of the key, 0:ascii, 1:hex

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

3.5 hfep_set_network_bufsize

Function prototype:

```
int HSF_API_EX hfep_set_network_bufsize(char *name, int bufsize);
```

Description:

Set buffer size for network.

Parameters:

name: network name
bufsize: buffer size, should be larger than 0, otherwise, it is set to default value (1024)

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

3.6 hfep_set_network_connectmode

Function prototype:

```
int HSF_API_EX hfep_set_network_connectmode(char *name, char mode, char *s_str, char s_strlen, char s_strmode);
```

Description:

Set network connection mode.

Parameters:

name: network name
mode: network mode, should be one of the followings
enum EP_SOCKET_CONNECTMODE

```

    {
        EP SOCK_CONNECTMODE_ALWAYS    = 0,
        EP SOCK_CONNECTMODE_BURST,
    };
s_str: string to indicate disconnect, used in burst mode.
s_strlen: string length
s_strmode: string mode, 0:ascii, 1:hex

```

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

3.7 hfep_set_network_active

Function prototype:

```
int HSF_API_EX hfep_set_network_active(char *name);
```

Description:

Active network.

Parameters:

name: network name

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

3.8 hfep_get_network_config

Function prototype:

```
int HSF_API_EX hfep_get_network_config(char *name, char *proto, char *next);
```

Description:

Get network configuration.

Parameters:

name: network name

proto: pointer to protocol character returned, will be one of followings

```
enum EP_SOCK_PROTO
{
    EP_SOCK_PROTO_TCPSERVER= 0,
    EP_SOCK_PROTO_TCPCLIENT,
    EP_SOCK_PROTO_UDPSERVER,
    EP_SOCK_PROTO_UDPCLIENT,
    EP_SOCK_PROTO_HTTPC,
    EP_SOCK_PROTO_TELNETD,
};
```

next: pointer of the next network name, user can get it by call this function. If it is set to NULL, only current network configuration will be returned.

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

3.9 hfep_new_tcpserver

Function prototype:

```
int HSF_API_EX hfep_new_tcpserver(char *name, unsigned short lport);
```

Description:

Create a new TCP server.

Parameters:

name: network name

lport: local port of the TCP server

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

3.10 hfep_new_tcpclient

Function prototype:

```
int HSF_API_EX hfep_new_tcpclient(char *name, char *server, unsigned short rport);
```

Description:

Create a new TCP client.

Parameters:

name: network name

server: remote TCP server address, either IP address(x.x.x.x) or domain name.

rport: remote TCP server port

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

3.11 hfep_new_udpserver

Function prototype:

```
int HSF_API_EX hfep_new_udpserver(char *name, unsigned short lport);
```

Description:

Create a new UDP server.

Parameters:

name: network name

lport: local UDP server port

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

3.12 hfep_new_udpclient

Function prototype:

```
int HSF_API_EX hfep_new_udpclient(char *name, char *server, unsigned short rport);
```

Description:

Create a new UDP client.

Parameters:

name: network name

server: remote UDP server address, either IP address(x.x.x.x) or domain name.

rport: remote UDP server port

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

3.13 hfep_del_network

Function prototype:

```
int HSF_API_EX hfep_del_network(char *name);
```

Description:

Delete a network.

Parameters:

name: network name

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

4. Customer Network&UART Setting

By default, network or UART data will be routed to destination predefined. Sometimes, user need to analysis the data from network or UART. EPORT SDK provide a method that trans the data to user predefined application (callbacks) to customize the utility.

4.1 Customer Structure

```
typedef void (* custom_net_connect_callback_t)(char *name, int socket);
typedef void (* custom_net_close_callback_t)(char *name, int socket);
typedef void (* custom_net_recv_callback_t)(char *name, int socket, char* data, int len);
typedef void (* custom_uart_recv_callback_t)(char uart, char *data, int len);

typedef struct _EP_CUSTOM_CONFIG_
{
    custom_net_connect_callback_t connect_cb;    //network connection callback
    custom_net_close_callback_t close_cb;        //network close callback
    union _recv_{
        custom_net_recv_callback_t net;         //network data receiving callback
        custom_uart_recv_callback_t uart;       //UART data receiving callback
    } recv_cb;
} strCustomConfig;
```

Header file:

Hfplatform_ep_api.h

4.2 hfep_custom_net

Function prototype:

```
int HSF_API_EX hfep_custom_net(char *name, strCustomConfig *cfg);
```

Description:

Customize existing network.

Parameters:

name: network name (already created, see network setting section)
 cfg: callback configuration, the 'socket' is the same socket in Linux network system.

Return Value:

0: Success
 Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

4.3 hfep_custom_tcpserver

Function prototype:

```
int HSF_API_EX hfep_custom_tcpserver(char *name, unsigned short lport, strCustomConfig *cfg);
```

Description:

Create a new custom TCP server.

Parameters:

name: network name of the custom TCP server

lport: local port of the TCP server.

cfg: callback configuration

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

4.4 hfep_custom_tcpclient

Function prototype:

```
int HSF_API_EX hfep_custom_tcpclient(char *name, char *server, unsigned short rport, strCustomConfig *cfg);
```

Description:

Create a new custom TCP client.

Parameters:

name: network name of the custom TCP client

server: remote TCP server address, either IP address(x.x.x.x) or domain name.

rport: remote TCP server port

cfg: callback configuration

Return Value:

0: Success

Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

4.5 hfep_custom_udpserver

Function prototype:

```
int HSF_API_EX hfep_custom_udpserver(char *name, unsigned short lport, strCustomConfig *cfg);
```

Description:

Create a new custom UDP server.

Parameters:

name: network name of the custom TCP client
lport: local UDP server port
cfg: callback configuration

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

4.6 hfep_custom_udpclient

Function prototype:

```
int HSF_API_EX hfep_custom_udpclient(char *name, char *server, unsigned short rport, strCustomConfig *cfg);
```

Description:

Create a new custom UDP client

Parameters:

name: network name of the custom UDP client
server: remote UDP server address, either IP address(x.x.x.x) or domain name.
rport: remote UDP server port
cfg: callback configuration

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

4.7 hfep_custom_uart

Function prototype:

```
int HSF_API_EX hfep_custom_uart(char uart, strCustomConfig *cfg);
```

Description:

Customize UART utility.

Parameters:

uart: uart number, for EP10/EP20/HF511A, it always be 1.
cfg: callback configuration

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

4.8 hfep_custom_net_send

Function prototype:

```
int HSF_API_EX hfep_custom_net_send(char *name, char *buf, int len, int socket);
```

Description:

Send data through predefined network.

Parameters:

name: network name.
buf: data to send
len: data length
socket: the destination socket, valid for TCP server

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

4.9 hfep_custom_uart_send

Function prototype:

```
int HSF_API_EX hfep_custom_uart_send(char uart, char *buf, int len);
```

Description:

Send data to UART.

Parameters:

uart: uart number, for EP10/EP20/HF511A, always be 1.
buf: data to send
len: data length

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

5. Custom Web

5.1 hfep_custom_web_register

Function prototype:

```
typedef int (* hfcj_set_config_callback_t)(cJSON *jCustom);  
typedef int (* hfcj_get_config_callback_t)(cJSON *jCustom);  
int HSF_API_EX hfep_custom_web_register(hfcj_set_config_callback_t set,  
hfcj_get_config_callback_t get);
```

Description:

Register custom web handler

Parameters:

set: when web is set and is to inform application, callback is called to obtain those web setting (JSON format).

get: when web is to get the setting to display, callback is called to obtain the setting from application (JSON format)

Return Value:

0: Success

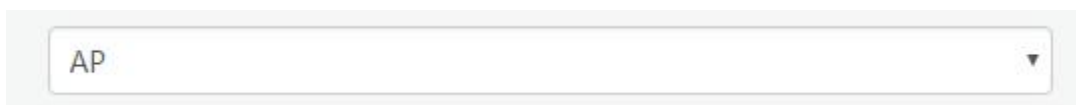
Others: Fail

Notes:

1. Data are represented by a JSON object,
2. 'description' and 'value' are mandatory
3. 'description' should be less than 30 Bytes
4. If there are more than 1 (≥ 2) options, use 'options'
5. Use '|' to separate different options;
6. For "options": "0|1", it will show switch on web page like this:



"options": "a|b|c" will show list as below:



Editable text like following will be shown if no 'options'



6. 'help' is optional, it shows the help context.
7. 'rules' is option, use it to verify the input rules

Currently, following rules are supported:

```
required: 'The %s field is required.',
matches: 'The %s field does not match the %s field.',
min_length: 'The %s field must be at least %s characters in length.',
max_length: 'The %s field must not exceed %s characters in length.',
exact_length: 'The %s field must be exactly %s characters in length.',
greater_than: 'The %s field must contain a number greater than %s.',
greater_than_or_equal: 'The %s field must contain a number greater than or
equal %s.',
less_than: 'The %s field must contain a number less than %s.',
less_than_or_equal: 'The %s field must contain a number less than or
equal %s.',
alpha: 'The %s field must only contain alphabetical characters.',
alpha_numeric: 'The %s field must only contain alpha-numeric characters.',
numeric: 'The %s field must contain only numbers.',
integer: 'The %s field must contain an integer.'
```



```
decimal: 'The %s field must contain a decimal number.',
valid_ip: 'The %s field must contain a valid IP.',
valid_ipv6: 'The %s field must contain a valid IPv6.',
valid_url: 'The %s field must contain a valid URL.',
hex: 'The %s field must contain a hex number.',
not_in: 'The %s field cann\'t contain one of values %s.',
character_hex: 'The %s field must be western characters or hex numbers.',
character_hex_max_length: 'The %s field must not exceed %s western
characters or hex numbers in length.',
character: 'The %s field must only contain western characters.',
not_matches: 'The %s field matches the %s field.',
valid_ip_domain: 'The %s field must contain a valid IP or domain.'
```

When there are more rules, separated them by '|', for example

```
"rules": "numeric|integer|greater_than_or_equal[1000]|less_than_or_equal[30000] "
```

Following example is from ep_api_test.c

```
hfep_custom_web_register(custom_web_post, custom_web_get);

int custom_web_post(cJSON *jCustom)
{
    char *key1, *key2;

    key1 = hfcj_get_string(jCustom, "key1");
    key2 = hfcj_get_string(jCustom, "key2");
    printf("get key %s:%s\n", key1, key2);
    return 0;
}

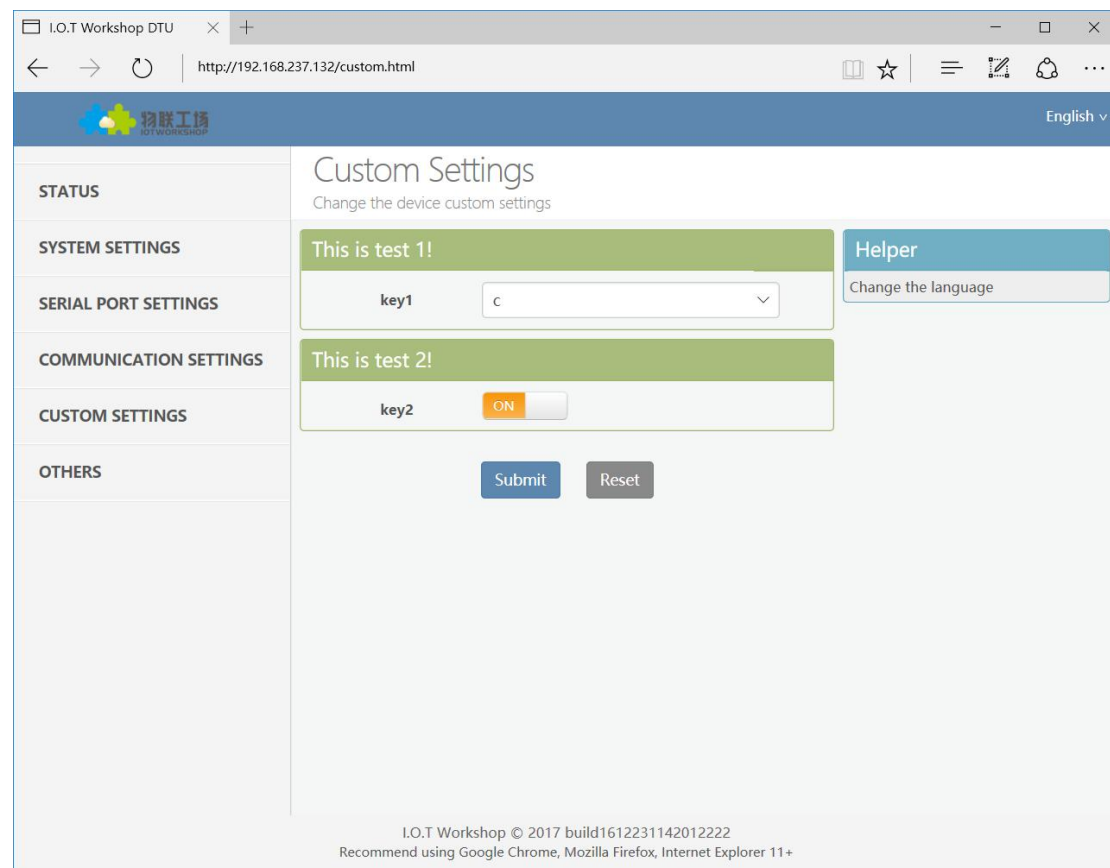
int custom_web_get(cJSON *jCustom)
{
    cJSON *tmp;
    tmp = hfcj_add_object_item(jCustom, "key1");
    hfcj_add_string(tmp, "description", "This is test 1!");
    hfcj_add_string(tmp, "value", "c");
    hfcj_add_string(tmp, "options", "a|b|c");

    tmp = hfcj_add_object_item(jCustom, "key2");
    hfcj_add_string(tmp, "description", "This is test 2!");
    hfcj_add_string(tmp, "value", "1");
    hfcj_add_string(tmp, "options", "0|1");

    return 0;
}
```

}

The final web page is like this:



Header file:

Hfplatform_ep_api.h

5.2 hfep_custom_web_unregister

Function prototype:

```
int HSF_API_EX hfep_custom_web_unregister(void);
```

Description:

Unregister custom web.

Parameters:

None.

Return Value:

0: Success
Others: Fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

Following APIs are used to get/analysis/set cJSON object.

5.3 hfcj_get_object_item

Function prototype:

```
cJSON *HSF_API_EX hfcj_get_object_item(cJSON *object, const char *str);
```

Description:

Get sub cJSON object with key 'str' from cJSON object.

Parameters:

object: input cJSON object

str: key of a JSON object

Return Value:

cJSON object pointer if success, otherwise NULL

Notes:

None.

Header file:

Hfplatform_ep_api.h

5.4 hfcj_get_string

Function prototype:

```
char *HSF_API_EX hfcj_get_string(cJSON *object, char *name);
```

Description:

Get string value from cJSON object if it's type is string.

Parameters:

object: input cJSON object

name: key of a JSON object

Return Value:

string pointer if success, otherwise NULL

Notes:

None.

Header file:

Hfplatform_ep_api.h

5.5 hfcj_get_number

Function prototype:

```
int HSF_API_EX hfcj_get_number(cJSON *object, char *name);
```

Description:

Get number value from cJSON object if it's type is number.

Parameters:

object: input cJSON object
name: key of a JSON object

Return Value:

Number value, '-1' if fail

Notes:

None.

Header file:

Hfplatform_ep_api.h

5.6 hfcj_get_bool

Function prototype:

```
bool HSF_API_EX hfcj_get_bool(cJSON *object, char *name);
```

Description:

Get bool value from cJSON object if it's type is bool.

Parameters:

object: input cJSON object
name: key of a JSON object

Return Value:

bool value

Notes:

None.

Header file:

Hfplatform_ep_api.h

5.7 hfcj_add_object_item

Function prototype:

```
cJSON * HSF_API_EX hfcj_add_object_item(cJSON *object, char *name);
```

Description:

Create sub cJSON object to cJSON object with a key.

Parameters:

object: input cJSON object
name: key of the new JSON object

Return Value:

The new created cJSON object

Notes:

None.

Header file:

Hfplatform_ep_api.h

5.8 hfcj_add_number

Function prototype:

```
void HSF_API_EX hfcj_add_number(cJSON *object, char *name, int value);
```

Description:

Add a number object with its key&value

Parameters:

object: input cJSON object
name: key of the number object
value: value of the number object

Return Value:

None.

Notes:

None.

Header file:

Hfplatform_ep_api.h

5.9 hfcj_add_string

Function prototype:

```
void HSF_API_EX hfcj_add_string(cJSON *object, char *name, char *str);
```

Description:

Add a string object with its key&value

Parameters:

object: input cJSON object
name: key of the string object
value: value of the string object

Return Value:

None.

Notes:

None.

Header file:

Hfplatform_ep_api.h

5.10 hfcj_add_bool

Function prototype:

```
void HSF_API_EX hfcj_add_bool(cJSON *object, char *name, bool value);
```

Description:

Add a bool object with its key&value

Parameters:

object: input cJSON object
name: key of the bool object
value: value of the bool object

Return Value:

None.

Notes:

None.

Header file:

Hfplatform_ep_api.h

6. Flash & GPIO

6.1 hfuflash_read

Function prototype:

```
int HSF_IAP hfuflash_read(unsigned int offset, char *data, int len);
```

Description:

Read data from user flash area

Parameters:

offset: offset for reading in user flash
data: pointer of buffer for reading
len: length of data reading

Return Value:

How many data eventually read from user flash. Less than 0 if failed.

Notes:

The flash size for customer use is 64K total. Read data within the area (offset from 0 to 65535).

Header file:

hfflash.h

6.2 hfuflash_write

Function prototype:

```
int HSF_IAP hfuflash_write(unsigned int offset, char *data, int len);
```

Description:

Write data in user flash area

Parameters:

offset: offset for writing in user flash
data: pointer of buffer for writing
len: length of data

Return Value:

How many data eventually write to user flash. Less than 0 if failed.

Notes:

The flash size for customer use is 64K total. Write data within the area (offset from 0 to 65535).

Header file:

hfflash.h

6.3 hfsys_set_gpio_value

Function prototype:

```
int HSF_API hfsys_set_gpio_value(int gpio, int dir, int val);
```

Description:

Set gpio direction and value

Parameters:

gpio: gpio number, eg, 1,2...
dir: direction of gpio, 0:in, 1:out
value: 0:low, 1: high

Return Value:

0: Success
Other: Fail

Notes:

None

Header file:

hfsys.h

6.4 hfsys_get_gpio_value

Function prototype:

```
int HSF_API hfsys_get_gpio_value(int gpio);
```

Description:

Get gpio value

Parameters:

gpio: gpio number, eg, 1,2...

Return Value:

0: Low
1: high

Notes:

None

Header file:

hfsys.h

6.5 ep10_gpio_config

Function prototype:

```
int HSF_API ep10_gpio_config(int gpio, int dir, int val)
```

Description:

Configure gpio for EP10/EP20/HF511A

Parameters:

gpio: gpio number, eg, 1,2
dir: direction of gpio, 0:in, 1:out
value: 0:low, 1: high

Return Value:

0: Success
-1: Fail

Notes:

None

Header file:

hfsys.h

6.6 ep10_gpio_set

Function prototype:

```
int HSF_API ep10_gpio_set(int gpio, int val)
```

Description:

Set gpio value for EP10/EP20/HF511A

Parameters:

gpio: gpio number, eg, 1,2
value: 0:low, 1: high

Return Value:

0: Success
-1: Fail

Notes:

None

Header file:

hfsys.h

6.7 ep10_gpio_get

Function prototype:

```
int HSF_API ep10_gpio_get(int gpio)
```

Description:

Get gpio value for EP10/EP20/HF511A

Parameters:

gpio: gpio number, eg, 1,2

Return Value:

value: 0:low, 1: high

Notes:

None

Header file:

hfsys.h