# API

## Web client side

### Api call structure

Protocol: WebSocket
Format: JSON
Property `FID`(function ID) - decimal or hexadecimal numder of API function
Property `ARG` - payload, json object
Property `SID`(session ID, responce only) - auto incrementing request counter for private subscriptions, tick counter for public subscriptions.

### Echo test (1000)

Request `{"FID":1000, "ARG":{"key":"value"}}`
Response `{"FID":"0x000003e8","SID":"0x000001eb","ARG":{"key":"value"}}`

### Private subscription or long task test (1001)

Request `{"FID":1001}`
Response `{"FID":"0x000003e9","SID":"0x00000223","ARG":{"STA":"0x00000001"}}`
Now user receiving personal data asinchroniusly
Secondary request `{"FID":1001}`
Response `{"FID":"0x000003e9","SID":"0x00000223","ARG":{"STA":"0x00000002"}}`
Task and subscription cancelled

### Public subscription test (1002)

Request `{"FID":1002}`
Response(personal) `{"FID":"0x000003ea","SID":"0x00000224","ARG":{"STA":"0x00000001"}}`
Task notifies all subscribers with the same data asinchroniusly
Response(public) `{"FID":"0x000003ea","SID":"0x0010e13b","ARG":{"data":"Async test"}}`
Response(public) `{"FID":"0x000003ea","SID":"0x0010e525","ARG":{"data":"Async test"}}`
...
Secondary request `{"FID":1002}`
Response(personal) `{"FID":"0x000003e9","SID":"0x00000224","ARG":{"STA":"0x00000002"}}`
User subscription cancelled

### Modbus master (2000)

FN {byte} - function number
ADR {byte} - modbus device address
RA(optional) {word} - register address
RVA(optional) {word} - register value or amount
CV(optional) {byte} - code value

RD(optoanal) {depends on function} - registers value(s)
// RAW(unsupported) {any bytes seq <=255 bytes len} - transfer raw data (see uart api)
AWT(optional) {dword} - awaite responce timeout ms (0: dotn't awaite, [default] >0: 100ms min)
RDL(optional) {dword} - auto repeat delay ms ([default]0: dotn't repeat, >0:(100ms min))
TIDC - cancel modbus task with TID. If set, other options ignoreg

**Example:**

Request: `{"FID":2000,"ARG":{"AWT":500,"RDL":500,"FN":3,"ADR":"0x01","RA":0,"RVC":20}}`
Response: `{"FID":"0x000007d0","SID":"0x00000006","ARG":{"TID":"0x0000000A","ADR":"0x01","FN":"0x03","CV":"0x00","RA":"0x0000","RC":"0x14","RD":["0x04d2","0x223d","0x0000","0x1165","0x0000","0x0022","0x0002","0x1d0d","0x0059","0x0162","0x18d2","0x0000","0x0022","0x0044","0x0000","0x0381","0x7eb3","0x0000","0x0024","0x0003"]}}`
Request: `{"FID":2000,"ARG":{"TIDC":10}}`
Response: `{"FID":"0x000007d0","SID":"0x00000006","ARG":{"STA":"0x00000002"}`

### UART (3000)

BR(optional) {dword} - boudrate
PAR(optional) {byte} - parity (0 - none; 1 - odd; 2 - even)
WL(optional) {byte} - word length (0 - 7bits; 1 - 8bits)
SB(optional) {byte} - stop bits (0 - 1sb; 1 - 2sb; 2 - unsupported; 3 - 1,5sb)
no arg - subscription on uart notifications (e.g. config change)\

**Example:**

Request: `{"FID":3000,"ARG":{"BR":115200}}` set boudrate and left untouched other options
Response(public): `{"FID":"0x00000bb8","SID":"0x00037053","ARG":{"BR":"0x0001c200","WL":"0x01","PAR":"0x00","SB":"0x00"}}`

## Firmware side

### Api handlers

```
/*
 * Api handler example, user defined. Invoking each time we get websocket request with FID
 * with which this handler was registered.
 * pxApiCall - API call descriptor
 * ppxContext - user context pointer (in/out): can be set by user and will be preserved for next call
 * ulPending - count of pending (uncompleted) API calls
 * pucData - pointer to current API call argument data
 * ulDataLen - size of data buffer pointed by pucData
 * Returns: true if complete
 */
```

```
uint8_t bSomeApiHandler(void *pxApiCall, void **ppxContext, uint32_t ulPending,
uint8_t *pucData, uint32_t ulDataLen);

/*
 * Registers API handler function for websocket calls
 * fHandler - API handler function pointer
 * ulFid - websocket function identifier for client-side calls
 * pxContext - initial context passed to handler's ppxContext parameter
 * Returns: true on success, false on failure
 */
uint8_t bApiCallRegister(ApiHandler_t fHandler, uint32_t ulFid, void
*pxContext);

/*
 * Marks one pending API invocation as completed
 * Decrements ulPending by 1. When ulPending becomes 0:
 *   - Handler is called final time with ulPending = 0 for cleanup
 *   - System then releases all call resources
 *
 * Use final handler call (ulPending=0) to free user-allocated resources
 */
void vApiCallComplete(void *pxApiCall);

/*
 * API Call Status Codes:
 * Normal statuses:
 */
#define API_CALL_STATUS_COMPLETE                    0x00000000  /**< Operation
completed successfully */
#define API_CALL_STATUS_EXECUTING                   0x00000001  /**< Operation
is in progress */
#define API_CALL_STATUS_CANCELED                    0x00000002  /**< Operation
was canceled by user */
#define API_CALL_STATUS_BUSY                        0x00000003  /**< System is
busy, try again later */

/*
 * Error statuses (bit 31 set):
 */
#define API_CALL_ERROR_STATUS_BAD_REQ               0x80000000  /**< Malformed
request */
#define API_CALL_ERROR_STATUS_FRAGMENTED            0x80000001  /**< Fragmented
request not supported */
#define API_CALL_ERROR_STATUS_NO_FID                0x80000002  /**< Function
ID not found */
#define API_CALL_ERROR_STATUS_BAD_ARG               0x80000003  /**< Invalid
argument provided */
#define API_CALL_ERROR_STATUS_NO_FREE_DESCRIPTORS   0x80000004  /**< No free
API descriptors available */
#define API_CALL_ERROR_STATUS_NO_MEM                0x80000006  /**< Memory
allocation failed */
#define API_CALL_ERROR_STATUS_NO_ACCESS             0x80000007  /**< Access
denied */
```

```
#define API_CALL_ERROR_STATUS_NO_HANDLER            0x8000000E  /**< No handler
registered for this FID */
#define API_CALL_ERROR_STATUS_INTERNAL              0x8000000F  /**< Internal
system error */

/**
 * Sends status update for specific API call
 *
 * Used for individual communication with a single client
 *
 * pxApiCall API call descriptor obtained in handler
 * ulSta Status code (see API_CALL_STATUS_* or API_CALL_ERROR_STATUS_* macros)
 * Returns true on success
 */
uint8_t bApiCallSendStatus(void *pxApiCall, uint32_t ulSta);

/**
 * Sends JSON data for specific API call
 *
 * Used for individual communication with a single client
 *
 * pxApiCall API call descriptor obtained in handler
 * ucJson Pointer to JSON data buffer
 * ulLen Length of JSON data in bytes
 * Returns true on success
 */
uint8_t bApiCallSendJson(void *pxApiCall, const uint8_t *ucJson, uint32_t
ulLen);

/**
 * Sends JSON data to ALL clients that called specified function ID
 *
 * Used for broadcast communication to multiple clients
 *
 * ulFid Function ID to broadcast to
 * ucData Pointer to JSON data buffer
 * ulLen Length of JSON data in bytes
 * Returns true on success
 */
uint8_t bApiCallSendJsonFidGroup(uint32_t ulFid, const uint8_t *ucData,
uint32_t ulLen);
```