# CTRL.BA 0.4

THIS IS DOCUMENT FROM ANOTHER VERSION OF SERVER, IT WILL BE UPDATED SOON.

## Base - Server Binary Protocol

Base communicates with Server over custom Binary Protocol. Every message sent is acknowledged back by the receiver with another message using the same binary protocol (if don't ack is not set). In case of acknowledgment message, TXSENDER field contains the value of message it is acknowledging to and ACK bit is set in the HEADER field. Special situation where acknowledgment is not sent back from Base to Server is a confirmation of successful (or unsuccessful) authentication of Base, which Server sends back to Base. Base shouldn't ACK back on result of authentication!

Messages are pushed from transmitting side to receiving side. In case ACK is lost or message doesn't get through, the nature of TCP is to kill a connection (and flag a timeout or a connection error of some sort). After the Base re-connects and authenticates, either side that has pending messages in queue (database) will try to flush them to receiver, one by one. In case previous message did go through but connection broke while waiting for an ACK, the message will be re-transmitted and received by the receiver (and acknowledged) . In this case receiver will know that it is a re-transmission and will not process the message again. The receiver knows this because there is a TXsender field in message which is in sync with the transmitter. Base owns and handles TXbase value, and Server owns and handles TXserver value. They are managed individually and not linked to each other in any way. Communication between sender and receiver goes in asynchronous fashion.

There can be a situation where a command is transmitted and processed on the receiving side and before the ACK gets back to the transmitter, connection breaks. In this case, once connection is established again the receiver might first send a result of that previous command (Reply Command) to the sender, even before the sender re-sends the command again (because he didn't get the ACK). Sender must be able to receive that reply and process it just as if it was received after sending the Command first. Re-sent command will not be processed by the original receiver because of the TXsender field (not to be worried about).

Message consists of these fields:

<MESSAGE-LENGTH><HEADER><TXSENDER><DATA>

Messages do not have special prefix or suffix so if a receiver takes a message and starts parsing from the middle, unexpected results will occur.

### Fields

<MESSAGE-LENGTH> [2 bytes] = length of the entire message (this field not included into the length of Message).

<HEADER> [1 byte] = contains meta-data about this message. Fields are:

|  |  |  |
| --- | --- | --- |
| Name | Bit position | Description |
| SYNC | 0000 00X0 | This message tells the received to SYNC „0“. This bit is used **only** during authentication process. |
| ACK | 0000 000X | This message is an acknowledgment. |
| OUT\_OF\_SYNC | 000X 0000 | Tells that receiver is out-of-sync with the transmitted message. Currently not handled in both Server and Base. This is actually a non-recoverably sync situation that probably required flushing entire „pending TX queue“ and starting from scratch. This bit is used only if this message is an ACK. |
| PROCESSED | 0000 0X00 | Tells whether the receiving side processed this command. (It is not processed only if it was a re-transmission). This bit is used only if this message is an ACK. |
| BACKOFF | 00X0 0000 | Used with acknowledgment to inform sender to delay sending more data because this command is not buffered in the receiver and should be re-transmitted from the transmitting side again later! Backoff ACKs are not being sent from Server, because server has enough resources to accept massive amounts of data. This means that it is not required to process BACKOFF ACKs in the Base as they will never arrive.  This bit is used only if this message is an ACK. |
| NOTIFICATION | ? | Used with messages that don’t need to be acknowledged back from the receiver. |
| SYSTEM\_MESSAGE | ? | Tells to receiving side that this message is a private message between connected party and the server (not forwarded to/from either Base or Client). |

*Header bits in Message for Binary Protocol*

<TXSENDER> [4 bytes] = sequence ID of the transmitter, unsigned integer. Increments from 1 to 2^32 and can only reset to 1 during authentication process. This means that each connection can transfer 2^32 messages until it rolls over to 1. Rollover is not handled in protocol, so transmitting side should re-connect if this number gets near maximum. However, in practice this will not be required because transmitter will increment to maximum value after 136 years of sending one message per second. Socket connection will naturally break many times in that period, so this limitation is nothing to worry about.

<DATA>\* [65535-Header-TXsender = 65530 bytes] = Payload data.

*\* - this parameter is optional, meaning it can be omitted when sending or missing from received message.*

### Handling the TXbase value in Base (TXsender field for messages in Base to Server direction)

When Base starts a TCP connection with Server (or re-starts – it is the same routine), it checks whether there are pending Messages to deliver to Server. In case there are no pending Messages, it internally resets TXbase value to 1 (not zero, but one). It also sets the SYNC bit in Header field of the Authentication command so that Server resets its own TXbase to 0 (not one, but zero!). Next Message that Base should send to Server will have TXsender value „1“ (which is +1 of what Server currently has), so Server can recognize that this is the next command it should in fact receive from Base. This way it knows that it is not a re-transmission. After the Server receives a Message from Base, it sends back an ACK with that exact TXsender value back, but with ACK bit set in Header field and PROCESSED bit also set in the Header field. It also increments its TXbase value by one to expect next Message from Base. When Base receives that ACK, it removes that message from its queue. Next message that Base is going to send will have TXsender value TXbase+1. Even when queue gets empty during a connection, Base must not restart TXbase! TXbase is only restarted during authentication process after connection is established, and in case the pending queue is in fact empty.

In situations where ACK is lost, a Base will re-send the same Message after re-connecting and authenticating, but when Server receives it, it will compare its TXbase value with the TXsender value received from Base. They will match and Server will know that it is a duplicate command (re-transmission) and will not process it. It will of course acknowledge, but it will also not set the bit PROCESSED in Header field. That field is just information for the sender to know whether his command was processed by receiver or not.

The same procedure is carried on Server, when sending Messages from Server to Base. Server has its own TXserver value (not as a variable in memory, because sockets die when connection breaks, but actually TXserver is calculated from MySQL database, table *txserver2base*).

### Authentication

After socket is connected, authentication procedure must take place. Authentication packet is the only expected packet after receiving data on unauthorized socket connection.

*Authentication procedure*

**Base**

**Server**

*connect*

If baseid found in DB, connection is authorized.  
If „SYNC“ set in header field: set TXbase = 0 (next message we will receive will have TXbase = 1

If Server doesn’t have any undelivered (unacknowledged) transmissions for Base, in response command the “SYNC” bit will be set in header field. That will tell Base to re-sync its TXserver value to 0, because next data we will send will have value = 1.

Authentication message (flagged as SYSTEM MESSAGE), carrying *baseid* data (*baseid* is 16 bytes long).

Authentication **reply** message flagged as SYSTEM MESSAGE, carrying result data 0x00 – OK, or 0x01 – Error.

### System Messages (between Server and Base)

These are special messages which when received on Server from Base are not forwarded to Clients. There are also messages which will be received by Base from Server.

Currently supported system messages are: none

## Client - Server JSON Protocol

Client communicates with Server by using JSON Messages. Each JSON Message is terminated with new-line '\n' character, so this character must not be found anywhere within a Message!

Each transmission is acknowledged just like in communication between Base and Server, so the same procedures apply here, including the Authentication procedure and handling TXclient value (the same idea as with TXbase).

There is a special situation though, when notifications and TXsender synchronization values are not checked. That is when *notification* property is set to true in Header property. Notifications are sent directly to socket without re-transmissions and without any other checks! These are low priority messages such as when Client authorizes to Server, and then gets a notification about registered mobile device, and information about the online/offline status of his Base.

Backoff in Client-Server communication is not implemented because both Server and Client can (should) be able to accept „unlimited number” of Messages.

JSON Message has these objects/properties:

|  |
| --- |
| {  „header“: {  „sync“: true/false,  „ack“: true/false,  „processed“: true/false,  „out\_of\_sync“: true/false,  „notification“: true/false,  „system\_message“: true/false,  },  „TXsender“: 123456789,  „data“: „HEXADECIMAL\_ASCII\_ENCODED\_DATA“  } |

*JSON Message format*

### Properties

**Header** = header is an object containing multiple properties:

|  |  |  |
| --- | --- | --- |
| Property | Value | Description |
| sync | true/false | Tells the received of this acknowledgment to sync to „0“. Used only in authentication procedure. |
| ack | true/false | Means that this message is an acknowledgment of previous command with the same TXsender value provided. |
| out\_of\_sync | true/false | Tells that receiver is out-of-sync with the transmitted message. Currently not handled in both Server and Client. This is actually a non-recoverably sync situation that probably required flushing entire „pending TX queue“ and starting from scratch. This bit is used only if this message is an ACK. |
| processed | true/false | Tells whether the receiving side processed this command. (It is not processed only if it was a re-transmission). This bit is used only if this message is an ACK. |
| notification | true/false | Low priority messages that don't get ACKs back, and no re-transmissions in case of failure. Also TXsender field is not checked for re-transmissions. |
| system\_message | true/false | Tells to receiving side that this message is a private message between connected party and the server (not forwarded to/from either Base or Client). |

*Header properties*

**TXsender** = sequence ID of the transmitter, unsigned integer. Its value increments from 1 to 2^32 and can only be reset to 1 during authentication procedure. This means that each connection can transfer 2^32 messages until it rolls over to 1. Rollover is not handled in protocol, so transmitting side should re-connect if this number gets near maximum. However, in practice this will not be required because transmitter will increment to maximum value after 136 years of sending one message per second. Socket connection will break millions of times in that period of time, so this limitation is not to be worried about.

**data** = data payload is encoded in hexadecimal ascii format. Length of payload between Client and Server is limited to length of Base<->Server payload binary size \* 2. [*CHECK THIS STATEMENT*] Any larger data will be truncated to whatever Base Socket can accept.

### System Messages (between Server and Client)

These are special messages which when received on Server from Client are not forwarded to Base. There are also messages which will be received by Client from Server.

Currently supported system messages are:

1. Base connection status notification for Clients (sent from Server -> Client) which must be implemented on Client (watch for “system\_message”: true in “HEADER”)