# TCI protocol

Version 1.9

## Introduction

This document describes the TCI protocol, what it's for and how to use it. The target audience of this document is programmers who implement TCI protocol in their programs and devices.

TCI (Transceiver Control Interface) is a network interface for control, data transfer and synchronization between transceiver/receiver, contest loggers, digital mode software, skimmers and other software, as well as external power amplifiers, bandpass filter units, antenna switches, radio controllers and other devices.

TCI was created as a modern alternative to the outdated COM port and audio cable interfaces, it uses a full duplex web socket protocol that runs on top of a TCP connection and serves for server-client communications, providing cross-platform connectivity. Transceiver works as a server, all other software and devices as clients. The server and clients can be inside the same computer (program-server, hardware log, etc.-clients) and/or in separate physical devices connected through the local network (classical transceiver, power amplifier, antenna switch, FFT unit, etc.).

The TCI interface contains basic transceiver control commands (analog of CAT system), receives CW macros from clients and broadcasts them, outputs transceiver IQ stream to clients, receives spots from skimmers and Internet clusters, receives/outputs audio signal to work in digital modes.

The TCI uses an extensible architecture and can be supplemented with new functions and commands, while keeping the old ones operational. Thus, the TCI interface can be extended and supplemented to meet the specific needs of any software manufacturer and/or device manufacturer (receivers, transceivers, power amplifiers, switches, etc.). The presence of a device identifier allows the manufacturers of transceivers and receivers to switch to the TCI interface while maintaining the device model designation. The extensibility of the TCI interface allows you to create an individual set of commands and functions for each device model, while maintaining the basic command set inherent to all transceivers.

Our company advocates universal unification of data exchange between devices and software by creating the TCI interface for this purpose. Modern transceivers and software must communicate using one protocol - the TCI protocol.

**TCI interface license**

**Interface description**

Any command represents an ASCII string that contains a command name and a list of arguments corresponding to this command. There are reserved characters that cannot be included in the command name and command arguments.

List of reserved characters: «:», «,», «;».

Command structure:
1. Name of the command;
2. Separating character between command name and arguments «:»;
3. Separating character between arguments «,»;
4. End of the command character «;».

If a command has no arguments, an end of command symbol is placed after the command name. If the command is invalid, it is ignored. The case of letters does not matter.

The ExpertSDR3 program acts as a server, which can have several client connections at the same time, they will be synchronized with each other by the server. When connecting to the ExpertSDR3, the client receives the current status of the ExpertSDR3, first sending initialization commands, then parameters to set the status, such as frequency, modulation, etc.

When a parameter change occurs in the ExpertSDR3 (server) program, the server notifies all connected clients, i.e., clients do not need to poll the server constantly, any change of state will be sent in time to all clients. If the client sends a new state, the server will set it to itself, as well as send it to all clients, that is, the server acts as a synchronizer. All clients connected to the server will be automatically synchronized. This way of work allows to minimize network load, reducing traffic.

The TCI protocol implements the transmission of receiver IQ stream to clients, which is necessary for the work of special skimmer software, they automatically find the station and decode it throughout the band, and it also allows you to record radio signals in the file.

TCI is also used to transmit audio signals of the receiver to clients and to receive audio signals from clients, i.e., the client can transmit audio signals to ExpertSDR3 for radio transmission. The audio stream exchange is designed to work with digital modes, where encoding and decoding is performed by third-party software, as well as in voice modes, where audio macros can be broadcasted, which is very much in demand in contest loggers.

When working in contests, it is important to record all on the air operation, for this purpose the audio stream from the line-output is sent to all clients. The resulting audio stream can be recorded to a file or played back with a PC sound card.

**Working in CW mode**

Through the TCI protocol CW can be generated from string commands. The commands are divided into two types:

1. CW Macros;
2. CW Message.

**A CW macro is a** set of characters that has no rules, but still can include commands to change the CW speed and or to send CW abbreviations. CW macros can be used in CW terminal mode.

The command to send a macro has the form: cw_macros:arg1,arg2;

arg1 - is the periodic number of the software transceiver;
arg2 - text message.

Example: cw_macros:0,TU RA6LH 599;

To put a CW abbreviation into the text (to combine several letters together), you should put the characters between the vertical brackets: TEXT |SK| TEXT.

To change the CW speed within a text, "< " decreases the speed, and "> " increases the speed respectively. The speed step is 5 wpm, for example: ANY TEXT > TEXT+5WPM >>TEXT+15WPM

To pass the string «+5wpmTU -5wpmRA6LH +10wpm599 004 *SK*» the command would have the form: cw_macros:0,>TU >599 004 |SK|;

As text commands may contain characters which are reserved by the protocol, they are replaced by other characters and converted back in the TCI server:

1. The character «:» is replaced with «^»;
2. The character «,» is replaced with «~»;
3. The character «;» is replaced with «*».

When transmitting macros, it is possible to operate in terminal mode, i.e. the transceiver remains in transmission after the macro transmission is completed until terminal mode is disabled. The following command enables/disables the terminal mode:

- cw_terminal:true; - enable;
- cw_terminal:false; - disable.

In terminal mode, a command is sent to the client as soon as the last letter in the queue start to be transmitted: cw_macros_empty;

If the terminal mode is disabled during macro playback, the transceiver will automatically switch to receive mode when the macro is finished.

A **CW message** is a special command that has a more complex structure and consists of three parts:

1. The prefix is the text before the callsign;
2. The callsign;
3. The suffix is the text after the callsign.

The command sends a CW message and offers the possibility of repeating the callsign:

cw_msg:arg1,arg2,arg3,arg4;

- arg1 - is the number of the software transceiver;
- arg2 - prefix;
- arg3 - callsign;
- arg4 - suffix.

To pass the string «TU RA6LH 599 004» the command would have the form:

cw_msg:0,TU,RA6LH,599 004;

If the callsign is to be repeated twice «TU RA6LH RA6LH 599 004», the command will have the form:

cw_msg:0,TU,RA6LH$2,599 004;

When a CW message is already sent to TCI but the callsign is not sent yet, the callsign can be corrected using the command:

cw_msg:arg1;

An example of a callsign correction sequence:

1. cw_msg:0,_,RA6$2,599 004;
2. cw_msg:RA6L;
3. cw_msg:RA6LH;

If editing of the callsign was performed after the callsign transmission was completed, the callsign correction command is ignored. The process of editing the callsign is performed for characters that have not yet been transmitted. After the transmission of the callsign is completed, a command is sent by the TCI server to the client, containing the final version of the callsign transmitted:

callsign_send:RA6LH;

The transmission of CW macros or messages can be immediately stopped with the command:

cw_macros_stop;

### Types of TCI commands

Several types of commands are provided in the protocol:

- Initialization commands;
- Bi-directional control commands;
- Unidirectional control commands;
- Notification Commands.

Initialization commands are sent to the client upon connection, they inform the client of the basic device parameters such as operating frequency range, supported modes, etc.

Bi-directional control commands are needed to control the basic parameters of the device and ExpertSDR3. Since the commands are bidirectional, the server will synchronize the parameters of all connected clients.

Unidirectional control commands are used to control parameters unique to each client, such as enabling audio streaming, sending spots to be displayed on an ExpertSDR3 panorama, etc.

Notification commands are sent to clients at some intervals, these can be readings from various sensors and signal meters.

**Initialization commands**

| VFO_LIMITS | Device operating frequency range. | |
|---|---|---|
| Reply | VFO_LIMITS:arg1,arg2; | arg1 — lower frequency limit, Hz.<br>arg2 — top frequency limit, Hz. |
| Type | Initialization | |
| Example | VFO_LIMITS:10000,30000000; | |

| IF_LIMITS | Frequency limit values of the IF filter (for VFOA only). | |
|---|---|---|
| Reply | IF_LIMITS:arg1,arg2; | arg1 — lower frequency limit, Hz.<br>arg2 — top frequency limit, Hz. |
| Type | Initialization | |
| Example | IF_LIMITS:-48000,48000; | |
| Note | Sent when you connect a device or change the sample rate. | |

| TRX_COUNT | Number of receivers (transceivers) in the radio. | |
|---|---|---|
| Reply | TRX_COUNT:arg1; | arg1 — number of receivers/transceivers (physical or software). |
| Type | Initialization | |
| Example | TRX_COUNT:2; | |

| CHANNEL_COUNT | Number of receiving channels in one receiver (A/B/C). | |
|---|---|---|
| Reply | CHANNEL_COUNT:arg1; | arg1 — number of receive channels. |
| Type | Initialization | |
| Example | CHANNEL_COUNT:2; | |

| DEVICE | Device Name. | |
|---|---|---|
| Reply | DEVICE:arg1; | arg1 — device name. |
| Type | Initialization | |
| Example | DEVICE:SunSDR2DX; | |

| RECEIVE_ONLY | Identifies the device as a receiver or transceiver. | |
|---|---|---|
| Reply | RECEIVE_ONLY:arg1; | arg1 — receiver only (true), transceiver (false). |
| Type | Initialization | |
| Example | RECEIVE_ONLY:true; | |

| MODULATIONS_LIST | List of supported modulations. | |
|---|---|---|
| Reply | MODULATIONS_LIST:arg1,arg2, … ,argN; | The modulation is conveyed by the name. |
| Type | Initialization | |
| Example | MODULATIONS_LIST:AM,LSB,USB,FM; | |

| PROTOCOL | TCI protocol version. | |
|---|---|---|
| Reply | PROTOCOL:arg1,arg2; | arg1 — program name.<br><br>arg2 — version of the protocol. |
| Type | Initialization | |
| Example | PROTOCOL:ExpertSDR3,1.7; | |

| READY | Sent after the initialization commands. | |
|---|---|---|
| Reply | READY; | |
| Type | Initialization | |

**Bidirectional control commands**

| START | Device start. | |
|---|---|---|
| Set | START; | |
| Type | Bidirectional control | |
| Example | START; | |

| STOP | Device stop. | |
|---|---|---|
| Set | STOP; | |
| Type | Bidirectional control | |
| Example | STOP; | |

| DDS | Receiver center frequency control. | |
|---|---|---|
| Set | DDS:arg1,arg2; | arg1 — receiver periodic number;<br>arg2 — tuning frequency, Hz. |
| Read | DDS:arg1; | |
| Reply | DDS:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | DDS:0;<br>DDS:0,7100000; | |

| IF | Frequency control of the IF tuning filter within a panorama. | |
|---|---|---|
| Set | IF:arg1,arg2,arg3; | arg1 — receiver periodic number;<br>arg2 — channel number (A / B);<br>arg3 — tuning frequency, Hz. |
| Read | IF:arg1,arg2; | |
| Reply | IF:arg1,arg2,arg3; | |
| Type | Bidirectional control | |
| Example | IF:0,1;<br>IF:0,1,12500;<br>IF:0,1,-17550; | |

| VFO | Receiver frequency control. | |
|---|---|---|
| Set | VFO:arg1,arg2,arg3; | arg1 — receiver periodic number;<br>arg2 — channel number (A/B);<br>arg3 — tuning frequency, Hz. |
| Read | VFO:arg1,arg2; | |
| Reply | VFO:arg1,arg2,arg3; | |
| Type | Bidirectional control | |
| Example | VFO:0,1,7100000;<br>VFO:1,0,14250000;<br>VFO:0,1; | |

| MODULATION | Switching modes. | |
|---|---|---|
| Set | MODULATION:arg1,arg2; | arg1 — receiver periodic number;<br>arg2 — mode (string). |
| Read | MODULATION:arg1; | |
| Reply | MODULATION:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | MODULATION:0,LSB;<br>MODULATION:1;<br>MODULATION:1,NFM; | |
| Note | A list of supported modes is sent to the client when connecting to ExpertSDR3. | |

| TRX | Switching RX/TX modes. | |
|---|---|---|
| Set | TRX:arg1,arg2,arg3; | arg1 — transceiver periodic number;<br><br>arg2 — status indicator (enable – true, disable – false).<br><br>arg3 — signal source (optional) (TCI - take signal from TCI audio stream). If an argument is absent, the signal is taken from the microphone. |
| Read | TRX:arg1; | |
| Reply | TRX:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | TRX:0,true;<br>TRX:0,true,tci;<br>TRX:0,false;<br>TRX:1; | |
| Note | The signal for transmitting is always taken from the microphone selected in the ExpertSDR3. If a third-party software connected via TCI wants to transmit its audio signal, you must specify the third argument - TCI. This works if the TCI audio stream is enabled, otherwise the microphone signal is taken. | |

| TUNE | Switching between receive and Tune modes. | |
|---|---|---|
| Set | TUNE:arg1,arg2; | arg1 — transceiver periodic number;<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | TUNE:arg1; | |
| Reply | TUNE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | TUNE:0,true;<br>TUNE:0,false;<br>TUNE:1; | |

| DRIVE | Output power control. | |
|---|---|---|
| Set | DRIVE:arg1,arg2; | arg1 — transceiver periodic number;<br><br>arg2 — output power value from 0 to 100. |
| Read | DRIVE:arg1; | |
| Reply | DRIVE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | DRIVE:0,30;<br>DRIVE:0,75;<br>DRIVE:1; | |

| TUNE_DRIVE | Output power control in Tune mode. | |
|---|---|---|
| Set | TUNE_DRIVE:arg1,arg2; | arg1 — transceiver periodic number;<br><br>arg2 — output power value from 0 to 100. |
| Read | TUNE_DRIVE:arg1; | |
| Reply | TUNE_DRIVE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | TUNE_DRIVE:0,30;<br>TUNE_DRIVE:0,75;<br>TUNE_DRIVE:1; | |

| RIT_ENABLE | Enable RIT. | |
|---|---|---|
| Set | RIT_ENABLE:arg1,arg2; | arg1 — receiver periodic number;<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | RIT_ENABLE:arg1; | |
| Reply | RIT_ENABLE:arg1,arg2; | |
| Type | bidirectional control | |
| Example | RIT_ENABLE:0,true;<br>RIT_ENABLE:1; | |

| XIT_ENABLE | Enable XIT. | |
|---|---|---|
| Set | XIT_ENABLE:arg1,arg2; | arg1 — transceiver periodic number;<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | XIT_ENABLE:arg1; | |
| Reply | XIT_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | XIT_ENABLE:0,true;<br>XIT_ENABLE:1; | |

| SPLIT_ENABLE | Enable Split operation | |
|---|---|---|
| Set | SPLIT_ENABLE:arg1,arg2; | arg1 — transceiver periodic number;<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | SPLIT_ENABLE:arg1; | |
| Reply | SPLIT_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | SPLIT_ENABLE:0,true;<br>SPLIT_ENABLE:1; | |

| RIT_OFFSET | Adjust RIT offset. | |
|---|---|---|
| Set | RIT_OFFSET:arg1,arg2; | arg1 — receiver periodic number;<br><br>arg2 — offset frequency, Hz. |
| Read | RIT_OFFSET:arg1; | |
| Reply | RIT_OFFSET:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | RIT_OFFSET:0,500;<br>RIT_OFFSET:1; | |

| XIT_OFFSET | Adjust XIT offset. | |
|---|---|---|
| Set | XIT_OFFSET:arg1,arg2; | arg1 — transceiver periodic number;<br><br>arg2 — offset frequency, Hz. |
| Read | XIT_OFFSET:arg1; | |
| Reply | XIT_OFFSET:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | XIT_OFFSET:0,-350;<br>XIT_OFFSET:1; | |

| RX_CHANNEL_ENABLE | Enable additional receive channel (VFO B). | |
|---|---|---|
| Set | RX_CHANNEL_ENABLE:arg1,arg2,arg3; | arg1 — receiver periodic number;<br><br>arg2 — channel periodic number.<br><br>arg3 — status indicator (enable – true, disable – false). |
| Read | RX_CHANNEL_ENABLE:arg1,arg2; | |
| Reply | RX_CHANNEL_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | RX_CHANNEL_ENABLE:0,1,true;<br>RX_CHANNEL_ENABLE:0,1; | |
| Note | Channel (VFO) A is always on, only channel (VFO) B can be controlled. | |

| RX_FILTER_BAND | Adjust RX filter width. | |
|---|---|---|
| Set | RX_FILTER_BAND:arg1,arg2,arg3; | arg1 — receiver periodic number.<br><br>arg2 — lower frequency limit, Hz.<br><br>arg3 — top frequency limit, Hz. |
| Read | RX_FILTER_BAND:arg1; | |
| Reply | RX_FILTER_BAND:arg1,arg2,arg3; | |
| Type | Bidirectional control | |
| Example | RX_FILTER_BAND:0,30,2700;<br>RX_FILTER_BAND:1,-2900,-70;<br>RX_FILTER_BAND:0; | |

| CW_MACROS_SPEED | Control CW macros speed (WPM). | |
|---|---|---|
| Set | CW_MACROS_SPEED:arg1; | arg1 — CW speed, WPM. |
| Read | CW_MACROS_SPEED; | |
| Reply | CW_MACROS_SPEED:arg1; | |
| Type | Bidirectional control | |
| Example | CW_MACROS_SPEED:42;<br>CW_MACROS_SPEED; | |

| CW_MACROS_DELAY | CW macros TX delay | |
|---|---|---|
| Set | CW_MACROS_DELAY:arg1; | arg1 — delay before the start of CW transmission, ms. |
| Read | CW_MACROS_DELAY; | |
| Reply | CW_MACROS_DELAY:arg1; | |
| Type | Bidirectional control | |
| Example | CW_MACROS_DELAY:100;<br>CW_MACROS_DELAY; | |
| Note | Adjust the delay between the moment you've initiated TXing and when you actually TX. | |

| CW_KEYER_SPEED | Control CW keyer speed (WPM). | |
|---|---|---|
| Set | CW_KEYER_SPEED:arg1; | arg1 — CW speed, WPM. |
| Type | Bidirectional control | |
| Example | CW_KEYER_SPEED:35;<br>CW_KEYER_SPEED:42; | |
| Note | Sent only by TCI client. | |

| VOLUME | Main volume control. | |
|---|---|---|
| Set | VOLUME:arg1; | arg1 — volume value, dB.<br><br>The range of values is from -60 to 0 dB, at a value of -60 dB there is no sound. |
| Read | VOLUME; | |
| Reply | VOLUME:arg1; | |
| Type | Bidirectional control | |
| Example | VOLUME:-12;<br>VOLUME; | |

| MUTE | Mute – disable/enable main volume. | |
|---|---|---|
| Set | MUTE:arg1; | arg1 — status indicator (enable – true, disable – false). |
| Read | MUTE; | |
| Reply | MUTE:arg1; | |
| Type | Bidirectional control | |
| Example | MUTE:true;<br>MUTE:false;<br>MUTE; | |

| RX_MUTE | Mute a certain receiver. | |
|---|---|---|
| Set | RX_MUTE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | RX_MUTE:arg1; | |
| Reply | RX_MUTE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | RX_MUTE:0,true;<br>RX_MUTE:1,false;<br>RX_MUTE:0; | |

| RX_VOLUME | Volume control for each channel (VFO) in a software receiver. | |
|---|---|---|
| Set | RX_VOLUME:arg1,arg2,arg3; | arg1 — receiver periodic number.<br><br>arg2 — channel periodic number.<br><br>arg3 — volume level, dB.<br><br>The range of values is from -60 to 0 dB, at a value of -60 dB there is no sound. |
| Read | RX_VOLUME:arg1,arg2; | |
| Reply | RX_VOLUME:arg1,arg2,arg3; | |
| Type | Bidirectional control | |
| Example | RX_VOLUME:0,1,-6;<br>RX_VOLUME:0,0; | |

| RX_BALANCE | Volume balance for each channel (VFO) in a software receiver. | |
|---|---|---|
| Set | RX_BALANCE:arg1,arg2,arg3; | arg1 — receiver periodic number.<br><br>arg2 — channel periodic number.<br><br>arg3 — volume level, dB.<br><br>Volume level, dB (-40…40). Negative values decrease volume in left channel, but the right channel stays the same. Positive values do the opposite. |
| Read | RX_BALANCE:arg1,arg2; | |
| Reply | RX_BALANCE:arg1,arg2,arg3; | |
| Type | Bidirectional control | |
| Example | RX_BALANCE:0,1,-6;<br>RX_BALANCE:0,0,12;<br>RX_BALANCE:0,1; | |

| MON_VOLUME | Control monitor volume in TX mode. | |
|---|---|---|
| Set | MON_VOLUME:arg1; | arg1 — volume value, dB.<br><br>The range of values is from -60 to 0 dB, at a value of -60 dB there is no sound. |
| Read | MON_VOLUME; | |
| Reply | MON_VOLUME:arg1; | |
| Type | Bidirectional control | |
| Example | MON_VOLUME:-12;<br>MON_VOLUME; | |

| MON_ENABLE | Enable/disable monitoring in TX mode. | |
|---|---|---|
| Set | MON_ENABLE:arg1; | arg1 — status indicator (enable – true, disable – false). |
| Read | MON_ENABLE; | |
| Reply | MON_ENABLE:arg1; | |
| Type | Bidirectional control | |
| Example | MON_ENABLE:true;<br>MON_ENABLE:false;<br>MON_ENABLE; | |

| AGC_MODE | Control receiver AGC mode. | |
|---|---|---|
| Set | AGC_MODE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — operation mode.<br><br>List of supported modes:<br>• normal;<br>• fast;<br>• off. |
| Read | AGC_MODE:arg1; | |
| Reply | AGC_MODE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | AGC_MODE:0,normal;<br>AGC_MODE:1; | |

| AGC_GAIN | Control receiver AGC gain. | |
|---|---|---|
| Set | AGC_GAIN:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — gain value, dB.<br><br>The range of values is from -20 to 120 dB. |
| Read | AGC_GAIN:arg1; | |
| Reply | AGC_GAIN:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | AGC_GAIN:0,87;<br>AGC_GAIN:1; | |

| RX_NB_ENABLE | Enable/disable receiver Noise Blanker (NB) which removes impulse interferences. | |
|---|---|---|
| Set | RX_NB_ENABLE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | RX_NB_ENABLE:arg1; | |
| Reply | RX_NB_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | RX_NB_ENABLE:0,true;<br>RX_NB_ENABLE:1; | |

| RX_NB_PARAM | Adjust Noise Blanker (NB) parameters. | |
|---|---|---|
| Set | RX_NB_PARAM:arg1,arg2,arg3; | arg1 — receiver periodic number.<br><br>arg2 - triggering threshold, value range: 1 ... 100.<br><br>arg3 - pulse duration, value range: 1 ... 300. |
| Read | RX_NB_PARAM:arg1; | |
| Reply | RX_NB_PARAM:arg1,arg2,arg3; | |
| Type | Bidirectional control | |
| Example | RX_NB_PARAM:0,70,25;<br>RX_NB_PARAM:1; | |

| RX_BIN_ENABLE | Enable/disable pseudo stereo (binaural - BIN). | |
|---|---|---|
| Set | RX_BIN_ENABLE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | RX_BIN_ENABLE:arg1; | |
| Reply | RX_BIN_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | RX_BIN_ENABLE:0,true;<br>RX_BIN_ENABLE:1; | |

| RX_NR_ENABLE | Enable/disable Noise Reduction (NR) filter which removes constant noise from audio signal. | |
|---|---|---|
| Set | RX_NR_ENABLE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | RX_NR_ENABLE:arg1; | |
| Reply | RX_NR_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | RX_NR_ENABLE:0,true;<br>RX_NR_ENABLE:1; | |

| RX_ANC_ENABLE | Enable/disable Adaptive Noise Cancellation (ANC) processor which emphasizes voice signal over noise of the receiver. | |
|---|---|---|
| Set | RX_ANC_ENABLE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | RX_ANC_ENABLE:arg1; | |
| Reply | RX_ANC_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | RX_ANC_ENABLE:0,true;<br>RX_ANC_ENABLE:1; | |

| RX_ANF_ENABLE | Enable/disable Automatic Notch Filter which removes tone signals from audio signal. | |
|---|---|---|
| Set | RX_ANF_ENABLE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | RX_ANF_ENABLE:arg1; | |
| Reply | RX_ANF_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | RX_ANF_ENABLE:0,true;<br>RX_ANF_ENABLE:1; | |

| RX_APF_ENABLE | Enable/disable Analog Peak Filter which changes AFC that voice sounds more prominent over noises. | |
|---|---|---|
| Set | RX_APF_ENABLE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | RX_APF_ENABLE:arg1; | |
| Reply | RX_APF_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | RX_APF_ENABLE:0,true;<br>RX_APF_ENABLE:1; | |

| RX_DSE_ENABLE | Enable/disable Digital Surround Effect for CW signals, it provides space orientation of the CW signals in the filter bandwidth, depending on their position in the filter. | |
|---|---|---|
| Set | RX_DSE_ENABLE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | RX_DSE_ENABLE:arg1; | |
| Reply | RX_DSE_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | RX_DSE_ENABLE:0,true;<br>RX_DSE_ENABLE:1; | |

| RX_NF_ENABLE | Enable/disable Notch Filters (NF) module. | |
|---|---|---|
| Set | RX_NF_ENABLE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | RX_NF_ENABLE:arg1; | |
| Reply | RX_NF_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | RX_NF_ENABLE:0,true;<br>RX_NF_ENABLE:1; | |

| LOCK | Tuning frequency lock. | |
|---|---|---|
| Set | LOCK:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | LOCK:arg1; | |
| Reply | LOCK:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | LOCK:0,true;<br>LOCK:1; | |

| SQL_ENABLE | Enable/disable squelch. | |
|---|---|---|
| Set | SQL_ENABLE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — status indicator (enable – true, disable – false). |
| Read | SQL_ENABLE:arg1; | |
| Reply | SQL_ENABLE:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | SQL_ENABLE:0,true;<br>SQL_ENABLE:1; | |

| SQL_LEVEL | Adjust squelch threshold. | |
|---|---|---|
| Set | SQL_LEVEL:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — triggering threshold, dB.<br><br>The range of values is from -140 to 0 dB. |
| Read | SQL_LEVEL:arg1; | |
| Reply | SQL_LEVEL:arg1,arg2; | |
| Type | Bidirectional control | |
| Example | SQL_LEVEL:0,-83;<br>SQL_LEVEL:1; | |

**Unidirectional control commands**

| TX_ENABLE | Informs clients that TX is enabled or disabled for current transceiver | |
|---|---|---|
| Reply | TX_ENABLE:arg1,arg2; | arg1 — transceiver periodic number;<br><br>arg2 — transmission allowed (true) / transmission denied (false). |
| Type | Unidirectional control | |
| Example | TX_ENABLE:0,true; | |
| Note | Sent to the client when connected and when the band is changed, in case transmitter permission was changed. | |

| CW_MACROS_SPEED_UP | CW macros speed increase (WPM). | |
|---|---|---|
| Set | CW_MACROS_SPEED_UP:arg1; | arg1 — the number of WPM by which to increase the CW speed. |
| Type | Unidirectional control | |
| Example | CW_MACROS_SPEED_UP:7;<br>CW_MACROS_SPEED_UP:2; | |
| Type | Sent only by the TCI client. | |

| CW_MACROS_SPEED_DOWN | CW macros speed decrease (WPM). | |
|---|---|---|
| Set | CW_MACROS_SPEED_DOWN:arg1; | arg1 — the number of WPM by which to decrease the CW speed. |
| Type | Unidirectional control | |
| Example | CW_MACROS_SPEED_DOWN:7;<br>CW_MACROS_SPEED_DOWN:2; | |
| Type | Sent only by the TCI client. | |

| SPOT | Spot transfer for display on panorama. | |
|---|---|---|
| Set | SPOT:arg1,arg2,arg3,arg4,arg5; | arg1 - callsign.<br>arg2 - mode type.<br>arg3 - frequency, Hz.<br>arg4 - ARGB color.<br>arg5 - additional text. |
| Type | Unidirectional control | |
| Example | SPOT:RN6LHF,CW,7100000,16711680,ANY_TEXT; | |
| Note | Sent only by the TCI client. | |

| SPOT_DELETE | Delete spot. | |
|---|---|---|
| Set | SPOT_DELETE:arg1; | arg1 — callsign. |
| Type | Unidirectional control | |
| Example | SPOT_DELETE:RN6LHF; | |
| Note | Sent only by the TCI client. | |

| IQ_SAMPLERATE | IQ stream sample rate | |
|---|---|---|
| Set | IQ_SAMPLERATE:arg1; | arg1 — sample rate, (Hz).<br>Supported Sample rates: 48/96/192/384 kHz |
| Reply | IQ_SAMPLERATE:arg1; | |
| Type | Unidirectional control | |
| Example | IQ_SAMPLERATE:48000; | |
| Note | Sent only by the TCI client. | |

| AUDIO_SAMPLERATE | LF audio stream sample rate. | |
|---|---|---|
| Set | AUDIO_SAMPLERATE:arg1; | arg1 — sample rate, (Hz).<br><br>Supported Sample rates: 8/12/24/48 kHz |
| Reply | AUDIO_SAMPLERATE:arg1; | |
| Type | Unidirectional control | |
| Example | AUDIO_SAMPLERATE:12000; | |
| Note | Sent only by the TCI client. | |

| IQ_START | Start IQ stream. | |
|---|---|---|
| Set | IQ_START:arg1; | arg1 — receiver periodic number. |
| Type | Unidirectional control | |
| Example | IQ_START:0; | |
| Note | Sent only by the TCI client. | |

| IQ_STOP | Stop IQ stream. | |
|---|---|---|
| Set | IQ_STOP:arg1; | arg1 — receiver periodic number. |
| Type | Unidirectional control | |
| Example | IQ_STOP:0; | |
| Note | Sent only by the TCI client. | |

| AUDIO_START | Start LF audio stream. | |
|---|---|---|
| Set | AUDIO_START:arg1; | arg1 — receiver periodic number. |
| Type | Unidirectional control | |
| Example | AUDIO_START:0; | |
| Note | Sent only by the TCI client. | |

| AUDIO_STOP | Stop LF audio stream. | |
|---|---|---|
| Set | AUDIO_STOP:arg1; | arg1 — receiver periodic number. |
| Type | Unidirectional control | |
| Example | AUDIO_STOP:0; | |
| Note | Sent only by the TCI client. | |

| LINE_OUT_START | Start audio stream from Line Out. | |
|---|---|---|
| Set | LINE_OUT_START:arg1; | arg1 — receiver periodic number. |
| Type | Unidirectional control | |
| Example | LINE_OUT_START:0; | |
| Note | Sent only by the TCI client. | |

| LINE_OUT_STOP | Stop audio stream from Line Out. | |
|---|---|---|
| Set | LINE_OUT_STOP:arg1; | arg1 — receiver periodic number. |
| Type | Unidirectional control | |
| Example | LINE_OUT_STOP:0; | |
| Note | Sent only by the TCI client. | |

| LINE_OUT_RECORDER_START | Start recording of the audio stream from Line Out. | |
|---|---|---|
| Set | LINE_OUT_RECORDER_START:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — maximum recording time (not more than 300 sec). |
| Type | Unidirectional control | |
| Example | LINE_OUT_RECORDER_START:0,250; | |
| Note | Sent only by the TCI client.<br>When the set recording time expires the recording is deleted, to save the recording into a file, send the LINE_OUT_RECORDER_SAVE command. | |

| LINE_OUT_RECORDER_SAVE | Save audio stream recording from Line Out into a file. | |
|---|---|---|
| Set | LINE_OUT_RECORDER_SAVE:arg1,arg2; | arg1 — receiver periodic number.<br><br>arg2 — full name of the file. |
| Type | Unidirectional control | |
| Example | LINE_OUT_RECORDER_SAVE:0,home/user_name/record_dir/file_name.wav;<br><br>LINE_OUT_RECORDER_SAVE:0,home/user_name/record_dir/file_name.mp3;<br><br>LINE_OUT_RECORDER_SAVE:0,D\|\record_dir\file_name.mp3; | |
| Note | Sent only by the TCI client.<br>Supported recording formats: WAVE and MP3.<br>For Windows OS the path contains the forbidden character ":", it is replaced by "\|", also you can use forward and backslash in the path to the file. | |

| LINE_OUT_RECORDER_BREAK | Stop audio stream recording from Line Out and delete the recording. | |
|---|---|---|
| Set | LINE_OUT_RECORDER_BREAK:arg1; | arg1 — receiver periodic number. |
| Type | Unidirectional control | |
| Example | LINE_OUT_RECORDER_BREAK:0; | |
| Note | Sent only by the TCI client. | |

| SPOT_CLEAR | Delete all spots from panorama. | |
|---|---|---|
| Set | SPOT_CLEAR; | |
| Type | Unidirectional control | |
| Example | SPOT_CLEAR; | |
| Note | Sent only by the TCI client. | |

**Notification commands**

| CLICKED_ON_SPOT | Informs about click on the spot on the ExpertSDR3 panorama. (Legacy command) | |
|---|---|---|
| Set | CLICKED_ON_SPOT:arg1,arg2; | arg1 — callsign.<br><br>arg2 — frequency, Hz. |
| Type | Unidirectional control | |
| Example | CLICKED_ON_SPOT:RN6LHF, 7147500; | |
| Note | Sent only by the TCI server. | |

| RX_CLICKED_ON_SPOT | Informs about click on the spot on the ExpertSDR3 panorama. | |
|---|---|---|
| Set | RX_CLICKED_ON_SPOT:arg1, arg2,arg3,arg4; | arg1 — receiver periodic number.<br><br>arg2 — channel periodic number (A/B).<br><br>arg3 — callsign.<br><br>arg4 — frequency, Hz. |
| Type | Unidirectional control | |
| Example | RX_CLICKED_ON_SPOT:0,1,RN6LHF,7147500; | |
| Note | Sent only by the TCI server. | |

| TX_FOOTSWITCH | Informs about PTT footswitch state. | |
|---|---|---|
| Set | TX_FOOTCWITCH:arg1,arg2; | arg1 — transceiver periodic number.<br><br>arg2 — footswitch state (pressed - true, not pressed - false) |
| Type | Unidirectional control | |
| Example | TX_FOOTCWITCH:0,true; | |
| Note | Sent only by the TCI server. | |

| TX_FREQUENCY | Informs about current transmitter frequency. | |
|---|---|---|
| Set | TX_FREQUENCY:arg1; | arg1 — transmission frequency, Hz. |
| Type | Unidirectional control | |

| | | |
|---|---|---|
| Example | TX_FREQUENCY:7140000; | |
| Note | Sent only by the TCI server. | |

| APP_FOCUS | Informs about the state of the ExpertSDR3 window (in focus or not). | |
|---|---|---|
| Set | APP_FOCUS:arg1; | arg1 — focus state.<br><br>(in focus - true, out of focus - false) |
| Type | Unidirectional control | |
| Example | APP_FOCUS:true; | |
| Note | Sent only by the TCI server. | |

| SET_IN_FOCUS | Make the main ExpertSDR3 window active (in focus). | |
|---|---|---|
| Set | SET_IN_FOCUS; | |
| Type | Unidirectional control | |
| Example | APP_FOCUS:true; | |
| Note | Sent only by the TCI client. | |

| KEYER | Informs about CW Key state (not automatic). | |
|---|---|---|
| Install | KEYER:arg1,arg2; | arg1 – transceiver periodic number.<br><br>arg2 – CW key state (pressed - true, not pressed – false). |
| Type | Unidirectional control | |
| Example | KEYER:0,true; | |
| Note | Sent only by the TCI client. | |

| RX_SENSORS_ENABLE | Enable informing about signal level in RX filter bandwidth | |
|---|---|---|
| Install | RX_SENSORS_ENABLE:arg1, arg2; | arg1 – status indicator (enable – true, disable – false).<br><br>arg2 – sending interval, ms (30-1000 ms, optional) |
| Type | Unidirectional control | |
| Example | RX_SENSORS_ENABLE:true;<br>RX_SENSORS_ENABLE:true,200; | |
| Note | Sent only by the TCI client. | |

| TX_SENSORS_ENABLE | Enable informing about TX signal parameters | |
|---|---|---|
| Install | TX_SENSORS_ENABLE:arg1,arg2; | arg1 – status indicator (enable – true, disable – false).<br><br>arg2 – sending interval, ms (30-1000 ms, optional) |
| Type | Unidirectional control | |
| Example | TX_SENSORS_ENABLE:true;<br>TX_SENSORS_ENABLE:true,200; | |
| Addendum | Sent only by the TCI client. | |

| RX_SENSORS | Inform about signal level in RX filter bandwidth | |
|---|---|---|
| Install | RX_SENSORS:arg1,arg2; | arg1 – receiver periodic number.<br><br>arg2 – signal level, dBm |
| Type | Unidirectional control | |
| Example | RX_SENSORS:0, -71.5;<br>RX_SENSORS:1,-112.7; | |
| Addendum | Sent only by the TCI client. | |

| TX_SENSORS | Inform about TX signal parameters | |
|---|---|---|
| Install | TX_SENSORS:arg1,arg2,arg3,arg4,arg5; | arg1 - transceiver periodic number<br><br>arg2 - microphone signal level, dBm<br><br>arg3 - signal power at the transmitter output, W (RMS)<br><br>arg4 - peak signal power at the transmitter output, W<br><br>arg5 - SWR |
| Type | Unidirectional control | |
| Example | TX_SENSORS:0, -27.2,47.4.67.5,1.7; | |
| Addendum | Sent only by the TCI server. | |

**Commands added in 1.9 version**

| AUDIO_STREAM_SAMPLE_TYPE | Set sample format for audio stream. | |
|---|---|---|
| Install | AUDIO_SAMPLE_TYPE:arg1; | arg1 — sample format identifier<br><br>Supported formats:<br>• int16<br>• int24<br>• int32<br>• float32 |
| Type | Unidirectional control | |
| Example | AUDIO_SAMPLE_TYPE:int24;<br>AUDIO_SAMPLE_TYPE:float32; | |
| Addendum | Being sent only by the client.<br>By default, sample format is float32. | |

| AUDIO_STREAM_CHANNELS | Set the number of channels for an audio stream. | |
|---|---|---|
| Install | AUDIO_STREAM_CHANNELS:arg1; | arg1 — channels number<br><br>Supported 1 or 2 channels. |
| Type | Unidirectional control | |
| Example | AUDIO_STREAM_CHANNELS:1;<br>AUDIO_STREAM_CHANNELS:2; | |
| Addendum | Being sent only by the client.<br>By default, the number of channels equals 2. | |

<table>
<tr><td>AUDIO_STREAM_SAMPLES</td><td colspan="2">Set the amount of samples sent in a single packet.</td></tr>
<tr><td>Install</td><td>AUDIO_STREAM_CHANNELS:arg1;</td><td rowspan="3">arg1 — sample amount indicated in a Stream.length field.<br><br>The value can vary from 100 up to 2048 samples.</td></tr>
<tr><td>Type</td><td>Unidirectional control</td></tr>
<tr><td>Example</td><td>AUDIO_STREAM_SAMPLES:200;<br>AUDIO_STREAM_SAMPLES:512;</td></tr>
<tr><td>Addendum</td><td colspan="2">Being sent only by the client.<br><br>For every sample rate value there is a default sample amount (indicated in Stream.length):<br>Sample rate 48 kHz - 2048 samples<br>Sample rate 24 kHz - 1024 samples<br>Sample rate 12 kHz - 512 samples<br>Sample rate 8 kHz - 256 samples<br><br>After you set the sample amount this value will be applied to every sample rate.<br><br>It is recommended to set the sample amount so that the length of playback duration is not less than 10 ms, otherwise the signal which is going to be transmitted may be corrupted.<br>Recommended minimal sample amount:<br>48 kHz - 512 samples<br>24 kHz - 256 samples<br>12 kHz - 128 samples<br>8 kHz - 100 samples</td></tr>
</table>

### Working with audio streams via TCI

Thanks to the WebSocket protocol, commands and audio streams are separated, commands are transmitted as strings and audio streams are transmitted as byte streams. Audio streams are transmitted in blocks of bytes, a block has a header and a data field.

Structure of a block in C language:

```
struct Stream
{
    uint32_t receiver;          // receiver number
    uint32_t sample_rate;       // sampling rate
    uint32_t format;            // always equal to 4 (float 32 bit)
    uint32_t codec;             // compression algorithm (not implemented), always 0
    uint32_t crc;               // checksum (not implemented), always 0
    uint32_t length;            // number of sample
    uint32_t type;              // stream type
    uint32_t channels;          // number of channels
    uint32_t reserv[8];         // reserved
    uint8_t  data[16384];       // samples
};
```

The flow type is defined by an enumeration:

```
enum StreamType
{
    IQ_STREAM               = 0,   // Receiver IQ signal stream
    RX_AUDIO_STREAM         = 1,   // Receiver audio stream
    TX_AUDIO_STREAM         = 2,   // Audio stream for transmitter
    TX_CHRONO               = 3,   // Time marker flow for audio signal transmission
    LINEOUT_STREAM          = 4    // Linear audio output audio stream
}

enum SampleType
{
    INT16           = 0,   // 16-bit integral character type
    INT24           = 1,   // 24-bit integral character type
    INT32           = 2,   // 32-bit integral character type
    FLOAT32         = 3,   // 32-bit type with a floating point
}
```

After connection, the client can enable receiving the IQ stream with the IQ_START command, after sending this command ExpertSDR3 starts sending the IQ stream. In the Stream.length field specified the number of real samples indicated in the uint8_t data[] field, since the these are complex samples, so the number of complex samples should be calculated as Stream.length/Stream.channels.

The receiver audio stream completely duplicates the IQ stream, but the difference is that you can control certain parameters, such as:

- Channels number,
- Samples format,
- Amount of samples sent in a single packet.

When DIGL/DIGU modes are selected, a complex signal will be transmitted if the number of channels is 2, but if the number of channels is 1 then it's going to be a real signal. Audio stream of the Line Out duplicates the regular audio stream, but you cannot change its parameters.

Sending an audio stream to a transmitter has certain peculiarities: ExpertSDR3 sends a TX_CHRONO timestamp that notifies the client to send an audio signal marked as

TX_AUDIO_STREAM with the specified number of samples in the Stream.length. Timestamps are sent without waiting for a response from the client. If the signal is not ready to be sent to ExpertSDR3, the client may not send a response or may send a signal with zero counts, which corresponds to no signal - this option is preferable.

**Particularities of TCI server operation in ExpertSDR3**

Server syncs all the clients connected to it – this is a difficult task, that is why there are certain particularities in TCI server operation, which you need to consider developing your client software.

There are a lot of commands and the list grows with each new ExpertSDR3 version, but it doesn't prevent effective sync process between clients. The correct hierarchy is the basics, which definitely defines master and slave software. If several clients will try to change the same parameter at the same time, the priority is behind the first one, but the highest priority is behind the ExpertSDR3. Even if a certain client controls some parameter, ExpertSDR3 operator will always be able to take control of this.

It's important to know, that in ExpertSDR3 we've implemented saving of certain parameters by band and mode. When a band has been changed all clients get new parameters in a certain sequence. For example, if a client sets frequency, which is different from the current band, ExpertSDR3 will set this frequency and change the band, then ExpertSDR3 will reinstate the settings saved for this band and send them to connected clients. When ExpertSDR3 initiates a change of any parameter, it monopolizes it for 200 ms and clients can't change it for this time period. The same rule is applied if one of the clients controls a parameter. At this moment other clients can only get notified about the change, but cannot control it until 200 ms will pass after the moment the first client stopped changing this parameter.

CW macros operation also has its own peculiarities, since it was designed especially for contests, but of course you can use them in your everyday operation. There are two commands for sending a text message via CW in TCI. The first and simplest is the CW_MACROS command, it just sends a text on the air. During the transmission of the CW macros, you can add more macros in line so all of them will be transmitted on the air in their turn. The second command is CW_MSG, it has a more complicated structure, prefix, callsign and suffix. Until the callsign is fully transmitted on the air it can be edited, but if during the transmission of such message you'll send another CW_MSG with prefix, callsign and suffix, then the current transmission will stop and the new one will start immediately. If during the transmission of the CW_MSG you'll send CW_MACROS it will be added in line after the active transmission. If during the transmission of the CW_MACROS you'll send CW_MSG, it immediately stops the transmission of CW macros and starts transmission of message, because message has a higher priority.

## Conclusion

The TCI protocol is constantly evolving. With the release of each new version of ExpertSDR3, the number of commands expands to increase the functionality of the programs and devices connected to ExpertSDR3.

The development team always listens to the opinion of ExpertSDR3 users. We try to make our product user-friendly both for software developers, allowing them to spend less time on integration of their programs and devices, and for common users, who are interested not in technical aspects but in convenience and ease of use.

Each of you can contribute to the development of the program. Your suggestions, wishes and comments will help us to make ExpertSDR3 better.

2022
Expert Electronics, Taganrog