Universal transceiver control interface – TCI

Version 1.1

Introduction

This document describes TCI interface, what can you do with it and how can you do that. This document may be of interest to programmers, who will implement this interface in their software and devices.

TCI – Transceiver Control Interface is the network interface for control, data transfer and synchronization between a transceiver/receiver, logger, digital software, skimmer, other software and external power amplifier, band filters, antenna switch, radio station controller and other devices.

TCI was created as an advanced alternative to old-fashioned COM-port interface and audio cables. It utilizes full duplex web-socket protocol, which works above TCP connection providing data exchange between server and client, and cross-platform capabilities. Transceiver works as a server, all other software and devices as clients. Server and clients may work inside the same PC (SDR-software-server, logger etc. – clients) and/or in separate physical devices, connected via local network (classical transceiver, power amplifier, antenna switch, BPF-filters etc.).

TCI interface has main commands to control a transceiver (CAT-system analogue), receives CW macroses from clients and transmits them on the air, passes transceiver's IQ stream to clients, receives spots from skimmers and Internet clusters, receives/transmits audio signal for digital mode operation*.

* Will be added in the new TCI release.

TCI uses an extensible architecture; thus, it can be enriched with new functionality, preserving efficiency of old functions. You can add a certain functionality to the TCI interface, which corresponds to specific needs of any software developer or hardware manufacturer (of receivers, transceivers, power amplifiers, antenna switches etc.). TCI has a server/client identifier; thus, clients can detect an assigned name for a transceiver/receiver (server), this function will help other manufacturers to preserve names of their devices in implemented TCI.

We created TCI interface to unify data exchange between different devices and various software. Modern transceivers and software should be connected via the unified interface – TCI.

MIT license to use demo-software of the TCI client

License to use TCI interface

Description

Any command is the ASCII line, which contents name of the command and list of arguments corresponding to this command. There are some reserved characters, which cannot be part of the command name and its 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 there are no arguments in the command, then after the name of the command should go end of the command character «;». If the command is incorrect, it is ignored. Letter case (capitalization) is irrelevant.

ExpertSDR2 software works as a server, which can have several clients connected simultaneously, they will be synced between each other via server. When client is connected to the ExpertSDR2, it receives ExpertSDR2 status: firstly, it receives *Read only* commands, then different parameters like frequency, modulations, etc.

When any parameter is changed in the ExpertSDR2 (server), it informs each connected client, so clients do not have to send continuous requests to a server, any change will be reported to every client. If a client will change its state, server will set it for itself and send it to every other client, so server works as a synchronizer. All clients connected to a server will be automatically synced. This way of operation allows to minimize network load, by lessening the traffic.

TCI protocol offers CW operation with line commands.

There are two types of commands:

1. Macros;
2. Message.

Macros – this is a list of characters, which has no rules, but abides to “change of speed” and “abbreviation transfer” commands.

Message – special command, which consists of three parts:

1. Text before callsign;
2. Callsign;
3. Text after callsign.

When transmitting the message, you can transmit the rest of your callsign after transmitting the message with only part of callsign, also you can change transmitting speed in the middle of the message and use abbreviations.

If you need to insert some abbreviation inside the text, it will look like:

ANY TEXT |SK| OTHER TEXT

All characters, which are put between vertical brackets, will be transmitted together.

If you need to decrease transmitting speed, use this character «**<**», to increase speed use «**>**». Speed change step 5 wpm, e.g.

ANY TEXT>TEXT+5WPM<TEXT-5WPM>>>TEXT+15WPM

Text commands may contain reserved characters, that is why they should be replaced with the other characters, which will be converted back on the server end:

1. Character «**:**» should be replaced with «**^**»;
2. Character «**,**» should be replaced with «**~**»;
3. Character «**;**» should be replaced with «*****».

Command to send macros looks like:

cw_macros:arg1,arg2;

arg1 – periodic number of the software transceiver;

arg2 – text to be sent before the callsign;

To send this line «+5wpmTU -5wpmRA6LH +10wpm599 004 *SK*» the command will be:

cw_macros:0,>TU <RA6LH >>599 004 |SK|;

Command to send CW message with opportunity to send callsign twice or to enter full callsign in the middle of transmission:

cw_msg:arg1,arg2,arg3,arg4;

arg1 - periodic number of the software transceiver;

arg2 - text before callsign;

arg3 - callsign;

arg4 - text after callsign;

Example:

To transmit «TU RA6LH 599 004» command will be:

cw_msg:0,TU,RA6LH,599 004;

If you need to repeat the callsign «TU RA6LH RA6LH 599 004», command will be:

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

If there is no text before callsign «RA6LH RA6LH 599 004», instead of text use special character "_":

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

If while sending the message you don't know full callsign and you'll need to edit it or change, probably not once, the command will be: cw_msg:arg1;

Example:

cw_msg:0,_,RA6$2,599 004;

cw_msg:RA6L;

cw_msg:RA6LH;

If callsign editing was late, it came through after callsign transmission was finished, then this command is ignored. Editing process can be done only for "not yet sent" symbols. After the fact that callsign is transmitted, client receives the command with the final callsign sent on the air.

Example:

callsign_send:RA6LH;

If while transmitting the CW macros you need to stop transmission, use this command:

Example:

cw_macros_stop;

Also, TCI supports order of transmitted CW messages, if you'll enter several cw_msg commands one-by-one, they will be transmitted one-by-one, in this case callsign addition will be applied to currently sent callsign.

Commands is divided on three types:

- Read only;
- Read/Write;
- Write only.

List of commands:

VFO_LIMITS	Receiver's frequency tuning limits	
Set		Arguments: *arg1* — bottom frequency limit, Hz. *arg2* — top frequency limit, Hz.
Read	*To be sent after connection;*	
Answer	*VFO_LIMITS:arg1,arg2;*	
Type	*Read only*	
Example	*VFO_LIMITS:10000,30000000;*	

IF_LIMITS	IF filter frequency limits (in ESDR2 only for VFOA)	
Set		Arguments: *arg1* — bottom frequency limit, Hz. *arg2* — top frequency limit, Hz.
Read	*To be sent after connection;*	
Answer	*IF_LIMITS:arg1,arg2;*	
Type	*Read only*	
Example	*IF_LIMITS:-48000,48000;* *IF_LIMITS:-96000,96000;*	

TRX_COUNT	Number of receivers (transceivers) in the radio	
Set		Arguments: *arg1* — number of receivers/transceivers (physical or software).
Read	*To be sent after connection;*	
Answer	*TRX_COUNT:arg1;*	
Type	*Read only*	
Example	*TRX_COUNT:2;* *TRX_COUNT:8;*	

CHANNEL_COUNT	Number of additional receiver channels (slices) in one receiver (A/B/C)	
Set		Arguments: *arg1* — number of receiver channels (slices).
Read	*To be sent after connection;*	
Answer	*CHANNEL_COUNT:arg1;*	
Type	*Read only*	
Example	*CHANNEL_COUNT:2;* *CHANNEL_COUNT:3;*	

DEVICE	Name of the device	
Set		Arguments: *arg1* — name of the device.
Read	*To be sent after connection;*	
Answer	*DEVICE:arg1;*	
Type	*Read only*	
Example	*DEVICE:SunSDR2;* *DEVICE:ColibriDDC;*	

RECEIVE_ONLY	Determine device as a receiver or transceiver	
Set		Arguments: *arg1* — number of receive channels.
Read	*To be sent after connection;*	
Answer	*RECEIVE_ONLY:arg1;*	
Type	*Read only*	
Example	*RECEIVE_ONLY:true;* *RECEIVE_ONLY:false;*	

MODULATIONS_LIST	List of supported mode types	
Set		Arguments: Mode type to be sent as a name.
Read	*To be sent after connection;*	
Answer	*MODULATIONS_LIST:arg1, arg2, ... ,argN;*	
Type	*Read only*	
Example	*MODULATIONS_LIST:AM,LSB,USB,FM;* *RECEIVE_ONLY:AM,SAM,LSB,USB,CW,NFM,WFM;*	

TX_ENABLE	Permission to use transmitter	
Set		Arguments: *arg1* — periodic number of receiver/transmitter. *arg2* — transmission permitted (true)/transmission prohibited (false).
Read	*To be sent while operating device*	
Answer	*TX_ENABLE:arg1, arg2;*	
Type	*Read only*	
Example	*TX_ENABLE:0,true;*	

READY	To be sent after initialization commands while connecting	
Set		
Read	*To be sent after connection;*	
Answer	*READY;*	
Type	*Read only*	
Example		

TX_FOOTSWITCH	PTT footswitch signal	
Set	*TX_FOOTSCWITCH:arg1,arg2;*	Arguments: *arg1* — periodic number of receiver. *arg2* — footswitch state (pressed (true), not pressed (false))
Type	*Read only*	
Example	*TX_FOOTCWITCH:0,true;* *TX_FOOTCWITCH:0,false;*	

START	Start ExpertSDR2	
Set	*START;*	Arguments:
Type	*Read / Write*	
Example	*START;*	

STOP	Stop ExpertSDR2	
Set	*STOP;*	Arguments:
Type	*Read / Write*	
Example	*STOP;*	

DDS	Tuning of the RX's center frequency (center of the panorama)	
Set	DDS:*arg1,arg2;*	Arguments: *arg1* — receiver's periodic number. *arg2, arg5* — tuning frequency, Hz.
Read	DDS:*arg1;*	
Answer	DDS:*arg1,arg2;*	
Type	*Read / Write*	
Example	DDS:*0,7200050;* DDS:*1;*	

IF	IF filter tuning in panorama bandwidth	
Set	*IF:arg1,arg2,arg3;*	Arguments: *arg1* — receiver's periodic number. *arg2* — channel's periodic number (A / B). *Arg3* — new tuned frequency in respect to DDS (center of the panorama), Hz.
Read	*IF:arg1,arg2;*	
Answer	*IF:arg1,arg2,arg3;*	
Type	*Read / Write*	
Example	*IF:0,0,-12000;* *IF:0,0,23000;* *IF:1,0;*	
RIT_ENABLE	Enable RIT	
Set	*RIT_ENABLE:arg1,arg2;*	Arguments: *arg1* — receiver's periodic number. *arg2* — enable indicator.
Read	*RIT_ENABLE:arg1;*	
Answer	*RIT_ENABLE:arg1,arg2;*	
Type	*Read / Write*	
Example	*RIT_ENABLE:0,true;* *RIT_ENABLE:0,false;* *RIT_ENABLE:1;*	

MODULATION	Set mode type	
Set	*MODULATION:arg1,arg2;*	Arguments: *arg1* — receiver's periodic number. *arg2* — mode type (line). List of supported mode types: AM / SAM / DSB / LSB / USB / CW / NFM / WFM / SPEC / DIGL / DIGU / DRM
Read	*MODULATION:arg1;*	
Answer	*MODULATION:arg1,arg2;*	
Type	*Read / Write*	
Example	*MODULATION:0,LSB;* *MODULATION:0,CW;* *MODULATION:1;*	

After switching the frequency band, the ExpertSDR2 restores saved settings for the newly selected band, which includes mode and RX filter bandwidth etc. That is why when you change the band, it's required to wait until you (client software) receive MODULATION and RX_FILTER_BAND commands from ExpertSDR2, in case they were changed, if your client software hadn't received these commands while the protection interval 200 ms then send MODULATION and RX_FILTER_BAND commands.
If mode and RX filter bandwidth haven't changed in the ExpertSDR2, it will not send MODULATION and RX_FILTER_BAND commands.

RX_ENABLE	Enable software receivers	
Set	*RX_ENABLE:arg1,arg2;*	Arguments: *arg1* — receiver's periodic number. *arg2* — enable indicator.
Read	*RX_ENABLE:arg1;*	
Answer	*RX_ENABLE:arg1,arg2;*	
Type	*Read / Write*	
Example	*RX_ENABLE:1,true;*	

	RX_ENABLE:2,false; *RX_ENABLE:1;*	

XIT_ENABLE	Enable XIT	
Set	*XIT_ENABLE:arg1,arg2;*	Arguments: *arg1* — receiver's periodic number. *arg2* — enable indicator.
Read	*XIT_ENABLE:arg1;*	
Answer	*XIT_ENABLE:arg1,arg2;*	
Type	*Read / Write*	
Example	*XIT_ENABLE:0,true;* *XIT_ENABLE:0,false;* *XIT_ENABLE:1;*	

SPLIT_ENABLE	Enable SPLIT mode	
Set	*SPLIT_ENABLE:arg1,arg2;*	Arguments: *arg1* — receiver's periodic number. *arg2* — enable indicator.
Read	*SPLIT_ENABLE:arg1;*	
Answer	*SPLIT_ENABLE:arg1,arg2;*	
Type	*Read / Write*	
Example	*SPLIT_ENABLE:0,true;* *SPLIT_ENABLE:0,false;* *SPLIT_ENABLE:1;*	

RIT_OFFSET	Tune RIT offset	
Set	*RIT_OFFSET:arg1,arg2;*	Arguments: *arg1* — receiver's periodic number. *arg2* — offset frequency, Hz.
Read	*RIT_OFFSET:arg1;*	
Answer	*RIT_OFFSET:arg1,arg2;*	
Type	*Read / Write*	
Example	*RIT_OFFSET:0,500;* *RIT_OFFSET:0,-200;* *RIT_OFFSET:1;*	

XIT_OFFSET	Tune XIT offset	
Set	*XIT_OFFSET:arg1,arg2;*	Arguments: *arg1* — receiver's periodic number. *arg2* — offset frequency, Hz.
Read	*XIT_OFFSET:arg1;*	
Answer	*XIT_OFFSET:arg1,arg2;*	
Type	*Read / Write*	
Example	*XIT_OFFSET:0,500;* *XIT_OFFSET:0,-200;*	

	XIT_OFFSET:1;	

RX_CHANNEL_ENABLE	Enable additional receive channels	
Set	*RX_CHANNEL_ENABLE:arg1,arg2,arg3;*	Arguments: *arg1* — receiver's periodic number. *arg2* — channel's periodic number. *arg3* — enable indicator.
Read	*RX_CHANNEL_ENABLE:arg1,arg2;*	
Answer	*RX_CHANNEL_ENABLE:arg1,arg2,arg3;*	
Type	*Read / Write*	
Example	*RX_CHANNEL_ENABLE:0,1,true;* *RX_CHANNEL_ENABLE:0,1,false;* *RX_CHANNEL_ENABLE:1, 1;*	

RX_FILTER_BAND	Adjust IF filter width	
Set	*RX_FILTERL_BAND:arg1,arg2,arg3;*	Arguments: *arg1* — receiver's periodic number. *arg1* — bottom frequency limit, Hz. *arg2* — top frequency limit, Hz.
Read	*RX_FILTER_BAND:arg1,arg2;*	
Answer	*RX_FILTER_BAND:arg1,arg2,arg3;*	
Type	*Read / Write*	
Example	*RX_FILTER_BAND:0,30,2700;* *RX_FILTER_BAND:0,-2900,-70;* *RX_FILTER_BAND:1;*	

After switching the frequency band, the ExpertSDR2 restores saved settings for the newly selected band, which includes mode and RX filter bandwidth etc. That is why when you change the band, it's required to wait until you (client software) receive MODULATION and RX_FILTER_BAND commands from ExpertSDR2, in case they were changed, if your client software hadn't received these commands while the protection interval 200 ms then send MODULATION and RX_FILTER_BAND commands.
If mode and RX filter bandwidth haven't changed in the ExpertSDR2, it will not send MODULATION and RX_FILTER_BAND commands.

RX_SMETER	Signal level (S-Meter) in filter bandwidth	
Set	*RX_SMETER:arg1,arg2,arg3;*	Arguments: *arg1* — receiver's periodic number. *arg2* — channel's periodic number. *arg3* — signal level.
Read	*RX_SMETER:arg1,arg2;*	
Answer	*RX_SMETER:arg1,arg2,arg3;*	
Type	*Read / Write*	
Example	*RX_SMETER:0,0,-72;* *RX_SMETER:0,1,-63;* *RX_SMETER:1,0;*	

CW_MACROS_SPEED	Set CW speed for macros	
Set	*CW_MACROS_SPEED:arg1;*	Arguments: *arg1* — CW speed, WPM.
Read	*CW_MACROS_SPEED;*	
Answer	*CW_MACROS_SPEED:arg1;*	
Type	*Read / Write*	
Example	*CW_MACROS_SPEED:30;* *CW_MACROS_SPEED:42;* *CW_MACROS_SPEED;*	

CW_MACROS_DELAY	Set delay between “turn to TX” and “start of macros transmission”	
Set	*CW_MACROS_DELAY:arg1;*	Arguments: *arg1* — delay before start of transmission, ms.
Read	*CW_MACROS_DELAY;*	
Answer	*CW_MACROS_DELAY:arg1;*	
Type	*Read / Write*	
Example	*CW_MACROS_DELAY:100;* *CW_MACROS_DELAY:150;* *CW_MACROS_DELAY;*	

TRX	Switch between RX/TX modes	
Set	*TRX:arg1,arg2, arg3;*	Arguments: *arg1* — receiver’s periodic number. *arg2* — enable indicator. *arg3* — source of the signal (unnecessary) (mic – mic signal, vac – signal from VAC)
Read	*TRX:arg1;*	
Answer	*TRX:arg1,arg2;*	
Type	*Read / Write*	
Example	*TRX:0,true;* *TRX:0,true,mic;* *TRX:0,true,vac;* *TRX:0,false;* *TRX:1;*	
For common use, the TRX command might have two arguments, in this case server will automatically determine the source of the input signal for transmission. When you select DIGL/DIGU modes the transceiver will automatically enable VAC and use the signal from it. In all other cases will be used signal from the mic. If you need to manually determine the source of the signal, you should use the third argument: mic – mic signal, vac – signal from VAC.		

IQ_START	Start IQ signal output	
Set	*IQ_START:arg1;*	Arguments: *arg1* — receiver’s periodic number.
Type	*Read / Write*	
Example	*IQ_START:0;*	

IQ_STOP	Stop IQ signal output	
Set	*IQ_STOP:arg1;*	Arguments:
Type	*Read / Write*	
Example	*IQ_STOP:0;*	*arg1* — receiver's periodic number.

IQ_SAMPLERATE	Set IQ signal sample rate	
Set	*IQ_SAMPLERATE:arg1;*	
Read		Arguments:
Answer		
Type	*Read / Write*	*arg1* — sample rate, Hz.
Example	*IQ_SAMPLERATE:48000;* *IQ_SAMPLERATE:96000;* *IQ_SAMPLERATE:192000;*	Supported sample rates: 48 / 96 / 192 kHz

AUDIO_START	Start audio stream	
Set	*AUDIO_START:arg1;*	Arguments:
Type	*Read / Write*	
Example	*AUDIO_START:0;*	*arg1* — receiver's periodic number.

AUDIO_STOP	Stop audio stream	
Set	*AUDIO_STOP:arg1;*	Arguments:
Type	*Read / Write*	
Example	*AUDIO_STOP:0;*	*arg1* — receiver's periodic number.

AUDIO_SAMPLERATE	Set audio stream sample rate	
Set	*AUDIO_SAMPLERATE:arg1;*	
Read		Arguments:
Answer		
Type	*Read / Write*	*arg1* — sample rate, Hz.
Example	*AUDIO_SAMPLERATE:8000;* *AUDIO_SAMPLERATE:24000;* *AUDIO_SAMPLERATE:48000;*	Supported sample rates: 8 / 12 / 24 / 48 kHz

SPOT	Send spot to ExpertSDR2 to display	
Set	*SPOT:arg1,arg2,arg3,arg4;arg5*	Arguments: *arg1* — callsign. *Arg2* — mode. *Arg3* — frequency, Hz. *Arg4* — color ARGB. *Arg5* — additional text.
Read		
Answer		
Type	*Write only*	
Example	*SPOT:RN6LHF,CW,7100000,16711680,ANY_TEXT;*	
Color is coded with 32-bit characterless number *0x00FF0000* -> *16711680*		

SPOT_DELETE	Delete spot	
Set	*SPOT:arg1;*	Arguments: *arg1* — callsign.
Read		
Answer		
Type	*Write only*	
Example	*SPOT_DELETE:IT8TY;*	

Receiving of IQ signal and receiving and transmitting of audio stream happens via binary websocket connection, packet structure looks like this:

```
typedef struct
{
    quint32 receiver;       //!< receiver's periodic number
    quint32 sampleRate;     //!< sample rate
    quint32 format;         //!< always equals 4 (float 32 bit)
    quint32 codec;          //!< compression algorithm (not implemented yet), always 0
    quint32 crc;            //!< check sum (not implemented yet), always 0
    quint32 length;         //!< length of data field
    quint32 type;           //!< type of data stream
```

```
    quint32 reserv[9];     //!< reserved
    float   data[4096];    //!< data field
}DataStream;
```

Type of data stream is determined with the following numeration:

```
typedef enum
{
    IqStream = 0,       //!< Receiver's IQ signal stream
    RxAudioStream,      //!< Receiver's audio stream
    TxAudioStream,      //!< Transceiver's audio stream
    TxChrono,           //!< Stream of temporary markers to transmit audio signal

}StreamType;
```

Audio stream processing.

In receive mode ExpertSDR2 sends RxAudioStream packet to TCI client. In transmit mode ExpertSDR2 sends TxChrono packet (it doesn't have a data filed) to TCI client, in the packet header indicated sample rate and sample count which is required for ExpertSDR2 for transmission. When TCI client receives a TxChrono packet, it sends back TxAudioStream packet with generated signals. This signal should have parameters indicated in TxChrono packet header (sample rate and sample count). Audio stream supports several sample rates: 8 kHz, 12 kHz, 24 kHz, 48 kHz.

List of software with TCI support:

- SDC
- LogHX
- SWISSLOG
- RUMLog
- 5MContest
- MacLoggerDX

Conclusion

TCI interface will gradually develop, in time it will be added with new commands and functionality. Follow the updates of TCI interface.