

## The **trx-control** System Architecture

At the core of **trx-control** is `trxd(8)`, a highly efficient daemon running on Linux (or on Windows using WSL, Windows Services for Linux). `trxd(8)` is written in the C language and uses a modern multi-threaded approach.

On the other hand, transceiver drivers, drivers for other devices and extensions are written in the Lua scripting language, making it easy to change the behaviour of `trxd(8)` or add new functionality even for those users who are not proficient in the C language. Furthermore, adding new drivers does not need a recompilation of the software.

Each communication channel (TCP/IP socket, WebSocket) is handled by its own listener thread that will create a service thread for each client that connects to `trxd(8)`.

Each client is served by its own service thread and access to all resources, be it transceivers, devices, or, something else, is carefully synchronised to ensure not two clients access the same resource at the same time.

Transceivers can automatically be queried at a specified interval for status updates (polling) or if they support automatic status updates by themselves that mechanism can be used without the need for a polling thread,

Communication is not only be initiated by a client, but also by `trxd(8)`, e.g. to send automatic status updates.

### Why `trxd(8)`?

Linux comes with a comprehensive set of manual pages that are organized in sections that are traditionally numbered.

Section 8 of the manual is the “System Manager’s Manual”.

It is not uncommon to refer to commands indicating in brackets the manual section where it is documented.

So `trxd(8)` is documented in section 8 of the manual, see `$ man trxd`.

## One Solution to Automate Your Shack

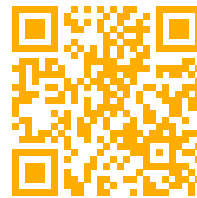
**trx-control** is a modular and highly extensible software system covering the following use cases:

- Controlling your transceivers.
- Controlling other hardware, e.g. GPIO devices.
- Connecting to databases, e.g. PostgreSQL.
- Communicating between different clients.
- Querying third party databases or systems like QRZ.com, DXCluster etc.

## Advantages Of An Open Solution

**trx-control** has a completely open design and is open source under the very liberal MIT license. Thanks to its modular design, it is easily extensible, your imagination is the only limit. `trxd(8)`, the daemon that runs at the core of the system is highly efficient and by its multi-threaded architecture it makes use of all available CPU cores.

Integrating **trx-control** into existing client software, e.g. logbooks or contesting software is straight forward thanks to a well documented JSON based client/server protocol that can be used over plain TCP/IP sockets or WebSockets.



trx-control website

**micro systems**  
Radio Communications  
Landstrasse 66  
CH-5073 Gipf-Oberfrick

<https://trx-control.msys.ch>  
[info@hb9ssb.ch](mailto:info@hb9ssb.ch)  
[#trx-control:matrix.org](https://matrix.org)  
062 871 45 65



A modern software system for Linux to control transceivers and other devices over the network.

**trx-control** is an extensible software system to control amateur radio transceivers and related hardware like relays, GPIO-pins and to integrate clients with third-party software using application specific extensions. It comes with comprehensive and complete documentation.

## Project Goals

- Provide a one-stop solution to automate your shack.
- Make it easy to add support for new transceivers or other devices.
- Make the system extensible by means of Lua, an easy to learn and easy to use scripting language.
- Use a well defined protocol based on the exchange of JSON-formatted data to communicate between **trx-control** and client software.
- Use a modern multithreaded and asynchronous approach at the core of the software.

**trx-control** is made in Switzerland.



## trx-control for End Users

### Installation

**trx-control** can be installed and built from source code or installed using prebuilt ready-to-go binaries.

### Configuration

Configuration is as easy as editing the `/etc/trxd.yaml` text file in the human readable YAML format.

### Usage

After editing the `/etc/trxd.yaml` config file, fire up `trxd(8)` using `systemctl`:

```
# systemctl start trxd
```

Also, check the `trxd(8)`, `trxctl(1)`, and `trx-control(7)` man pages.

### The `trxctl(1)` command

To test your setup or operate your transceiver, use the `trxctl(1)` command which connects to `trxd(8)` and allows you to send commands or receive and display automatic status updates.

### Spread the Word

If you like **trx-control** then please spread the word. Let others know about the project, present it in your local hamradio society etc. Ask us for ready made presentations and flyers.



User guide  
<https://trx-control.msys.ch/trx-control-user-guide.html>

## trx-control for System Integrators

### Connecting to `trxd(8)`

Your software connects to `trxd(8)` either using a plain TCP/IP socket or a WebSocket. The connection is bi-directional.

### The Client/Server Protocol

The client/server protocol is based on the exchange of JSON-formatted messages. In the case of a plain socket connection, the NDJSON format must be used.

The protocol is extensible, as is the software itself. This means that new extensions can define their own requests and replies.

### Destinations

Every device, extension etc. is a destination. You can address requests to a specific destination and you can query a list of destinations. The names of destinations are defined in the `/etc/trxd.yaml` config file.

### Automatic Status Updates

If you enable automatic status updates or listen to events of an extension, be ready to receive corresponding JSON packages at any time.

### Let Us Know

If you are adding support for **trx-control** to your software, then let us know. We will add your project to the list of programs supporting **trx-control**.



Integration guide  
<https://trx-control.msys.ch/trx-control-integration-guide.html>

## trx-control for Developers

### Overview

**trx-control** is highly extensible using the Lua programming language which is easy to learn and easy to use, yet very powerful and performant. Visit <https://lua.org> for details.

### Device Drivers

Device drivers interface `trxd(8)` with real hardware. This is where requests in the common JSON format are converted to commands for the target device and device specific replies are again converted to a common format. E.g. frequencies are always in hertz, regardless what a transceiver uses.

### Extensions

Extensions add custom functionality to a **trx-control** and make it available to clients. They define their own request and replies and support automatic status updates.

### Lua Modules

**trx-control** comes packed with Lua modules ready for use: `yaml`, `json`, `postgresql`, `curl`, `sqlite`, `linux`, `expat`. More can easily be added, even your own ones.

### Getting Help

There is the Issue Tracker on GitHub at <https://github.com/hb9ssb/trx-control>. You can also join the Matrix room `#trx-control:matrix.org`.



Developer guide  
<https://trx-control.msys.ch/trx-control-developer-guide.html>