

# Cybro Edge Toolkit

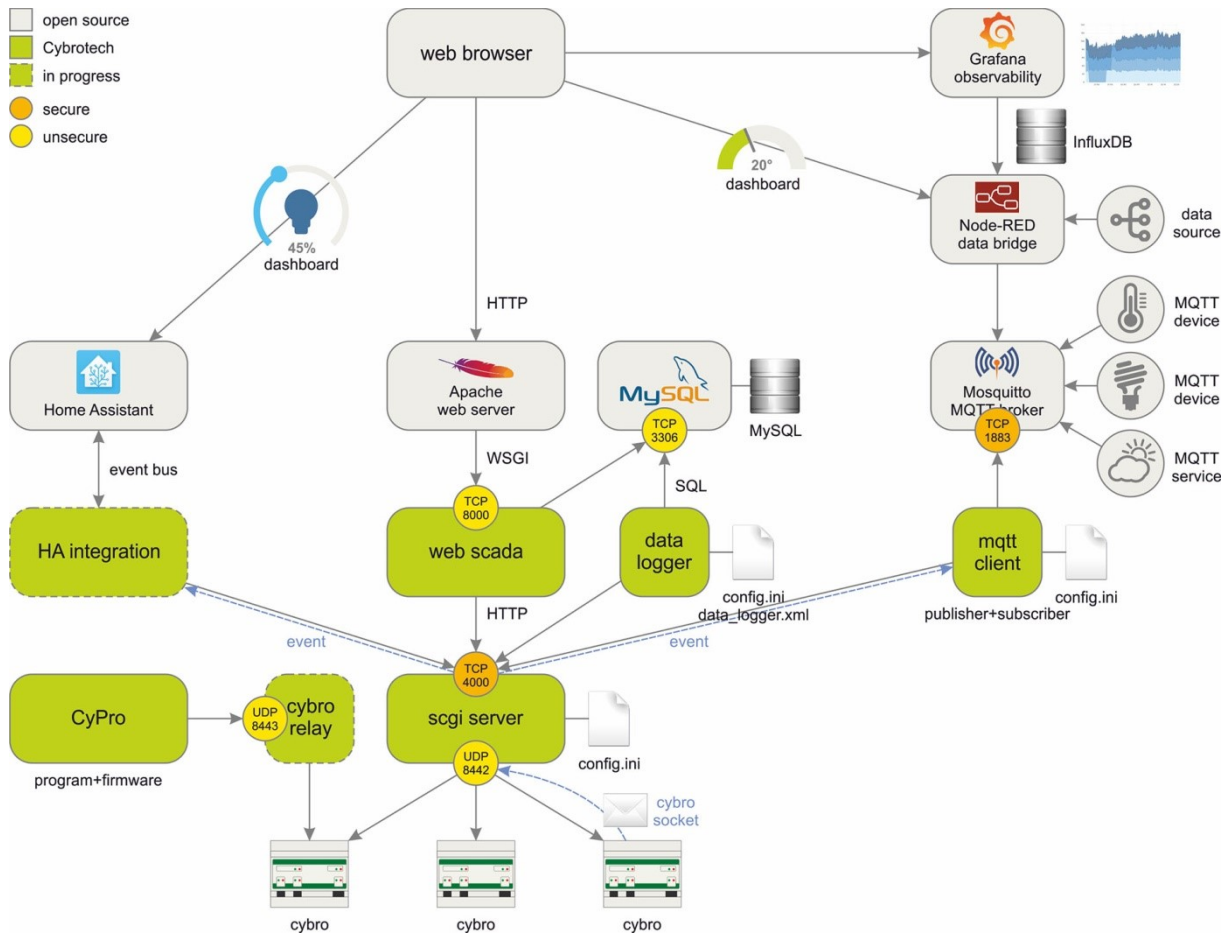
User Manual v17



© 2009-2025 Cybrotech Ltd.

# Overview

CybroEdgeToolkit is a set of applications for network connectivity, data collection and visualization. Applications are written in Python and run on various operating systems and hardware platforms.



Each connection moves data in both directions. The arrow indicates which side initiates the connection. The blue dashed line indicates propagation of controller-based events.

Applications can be executed on a single computer or distributed across multiple devices. Small footprint allows them to run on almost any hardware platform, including single-board computers.

All applications are located in a single directory:

/app/scgi_server/	read and write controller variables using HTTP requests
/mqtt_client/	publish and subscribe controller variables to MQTT
/data_logger/	periodically read from controller and store in database
/web_scada/	web-based interface for online monitoring and control
/lib/	common library functions
/tls/	security certificate
config.ini	shared configuration file
data_logger.xml	logger configuration file
requirements.txt	list of python dependencies

Applications are configured by editing the [config.ini](#) file. When applications run on the same computer, they use the same file. If the file is modified while the application is running, the application will automatically reload the file and restart.

The default [config.ini](#) is prepared to run on a single computer, without encryption.

## Installation

Applications run on Python v3.8 or later. If Python is not installed, download it from [python.org](https://python.org). Make sure to check the option "Add Python to PATH". Download the package from [cybrotech.com](https://cybrotech.com) and unzip into the recommended directory:

Linux:            /usr/bin/cybrotech/cybro\_edge\_toolkit/  
Windows:        c:\Program Files (x86)\Cybrotech\CybroEdgeToolkit\

Change directory to /app and install dependencies:

```
pip install -r requirements.txt
```

Edit config.ini to match your configuration, then start the applications you need:

```
/scgi_server/start.py  
/mqtt_client/start.py
```

Additional steps are required for the installation of web scada and data logger.

## Non-stop operation

To ensure that server is running 24/7, configure the system to execute [start.py](#) once every minute. If the server hangs up for any reason, the script will terminate the process and restart the server.

In Linux, this can be done using the [cronjob](#) service:

```
crontab * * * * * /usr/bin/cybrotech/cybro_edge_toolkit/app/scgi_server/start.py
```

The same can be done for each application.

A more advanced solution is to use [systemd](#), service manager for Linux, which provides automatic restart, logging and resource management.

In Windows, similar function can be performed with the Task Scheduler ([taskschd.msc](#)).

## Virtual environment

To avoid pollution of global scope, it's recommended to install dependencies inside the virtual environment. For more info, open [docs.python.org/3/library/venv.html](https://docs.python.org/3/library/venv.html). Commands must be executed in the /app directory.

Create virtual environment:

```
python -m venv venv
```

Activate the virtual environment:

```
./venv/Scripts/activate
```

Once virtual environment is activated, you can install dependencies and run the applications. In case you don't use them any more, deactivate the virtual environment:

```
./venv/Scripts/deactivate
```

# SCGI server

CybroScgiServer is a communication interface for reading and writing controller variables by name.

- server listens for HTTP requests on a specified port (4000)
- request can involve multiple reads and writes on multiple controllers
- server accepts connections from multiple clients at the same time
- server receives event driven sockets and push data to clients
- client/server connection can be encrypted for internet use
- configured by editing [config.ini](#) file
- runs as a background service

The controller can be connected in the following ways:

- local network, where controller's ip address is detected automatically
- over the internet, where controller must send push messages to the server
- over CAN bus, using Raspberry Pi and PiCAN2 interface (Cybro-Pi4)

The connection between controller and server is not encrypted.

## Reading and writing

To read and write data from the controller, open one of these URLs in your browser:

```
http://localhost:4000/?c20000.rtc_sec
http://localhost:4000/?c20000.cybro_power_supply&c20000.cybro_temperature
http://localhost:4000/?c20000.cybro_qx00=1
```

Replace 20000 with the serial number of your controller. The result is returned in XML format:

```
<data>
  <var>
    <name>c20000.rtc_sec</name>
    <value>59</value>
    <description>Seconds of internal real-time clock (0..59).</description>
  </var>
</data>
```

Float variables always return value with a decimal point. If an error occurs during operation, the response will contain an error code:

```
<data>
  <var>
    <name>c20000.rtc_xxx</name>
    <value>?</value>
    <description/>
    <error_code>2</error_code>
  </var>
</data>
```

The error\_code indicates which error occurred:

- 1: communication timeout
- 2: variable not found
- 3: controller not found
- 4: no plc program
- 5: no alc file

## Event-driven data

Polling sends messages in regular intervals, regardless of whether the data has changed. In contrast, event-driven communication works by having the controller detect a change and send it to the server. This minimizes delays and reduces network traffic.

The event uses the already established connection between client and server, sending data in the opposite direction.

To send an event from Cybro controller to MQTT system, follow these steps:

- create on-request socket with variables that contain event data
- write plc program that detects the event and sends the socket
- config.ini, section ETH, add socket description to enable receiving
- config.ini, section PUBLISHER, create a topic

A program that performs the task might look like this:

```
if fp(cybro_ix00) then
  event_data:=1; // pushbutton pressed
  socket_req:=1; // send socket
elseif fn(cybro_ix00) then
  event_data:=2; // pushbutton released
  socket_req:=1; // send socket
end_if;
```

The configuration file might look like this:

```
[ETH]
socket = 1; socket_req; event_data;

[PUBLISHER]
var = c20000.event_data, myhome/events, 0
```

For more details, check MQTT section.

## Encrypted connection

Encryption is process of encoding that ensures privacy (not readable to anyone else), identity (both parties confirmed), authenticity (message not modified) and reliability (message not damaged).

To create an encrypted connection, follow these steps:

- open [config.ini](#), set [tls\\_enabled=true](#)
- create a token of 64 random characters
- generate a new security certificate

Token provides an additional layer of security, allowing separate authentication for different projects, and easy revocation of access rights.

```
[SCGI]
tls_enabled = false
token = PWFChKsSB4DJoe4WC09hBaOKRDq47hMYpFqfFyHda35TdRo3yOTGAWPmWjuOry47
```

A security certificate contains a public key that is used to encrypt and decrypt messages between the client and the server. The default certificate, located in the [/app/tls](#) directory, is provided as example only and offers no security. The simplest solution is to generate a self-signed certificate.

To do this, install the last version of SSL binaries from [wiki.openssl.org](http://wiki.openssl.org).

Generate a private key (.key):

```
openssl genrsa -out private.key 4096
```

Generate a certificate signing request (.csr):

```
openssl req -new -sha256 -config ssl.cnf -key private.key -out private.csr
```

Generate the certificate (.crt):

```
openssl x509 -req -sha256 -days 3600 -in private.csr -signkey private.key -out private.crt -extensions req_ext -extfile ssl.cnf
```

Copy all generated files into [/app/tls](#) and restart server.

To check that certificate is properly installed, add generated [private.crt](#) to your browser trusted root certification authorities (Settings/Security), and try secure connection:

```
https://localhost:4000/?c20000.rtc_sec
```

Alternatively, you can purchase a certificate from an authorized authority, like [letsencrypt.org](http://letsencrypt.org).

Encryption adds a certain amount of processing to each request; therefore, it may increase server load to some extent.

## Controller alias

To simplify system maintenance, each controller can be assigned an alias. The name can include alphanumeric characters and underscore, not case sensitive. Alias is specified in the [config.ini](#) file:

```
[ALIAS]
c20000 = alpha
```

The controller is accessed the same way, except that the alias is used instead of the serial number.

```
http://localhost:4000/?alpha.rtc_sec
```

Once the alias is set, controller can no longer be accessed by the serial number.

## Data cache

The cache is used to improve server performance when handling a large number of requests.



The [valid\\_period\\_s](#) is duration, in seconds, for which the cached value remains valid. When the period is exceeded, a read from the controller is performed, delaying answer until the response is received.

The [request\\_period\\_s](#) is the amount of time in which the cached value is still returned, while a read operation is simultaneously initiated to refresh the cache. Must be shorter than the valid period.

The [cleanup\\_period\\_s](#) is the interval, in seconds, after which the expired items are removed from the cache. Set to a few minutes or more, to avoid unnecessary consumption of resources.

## System variables

System variables indicate status of the server and associated controllers. They are read in the same way as controller variables.

```
http://localhost:4000/?sys.scgi_status
```

### Server-related variables

<code>sys.scgi_status</code>	server status: <ul style="list-style-type: none"><li>• "active", server is up and running</li><li>• no response, the server is down</li></ul>
<code>sys.server_version</code>	software version in format "major.minor.release"
<code>sys.server_uptime</code>	server uptime in format "dd days, hh:mm:ss"
<code>sys.scgi_request_count</code>	total number of requests since the server is started
<code>sys.push_port_status</code>	<ul style="list-style-type: none"><li>• "active", push server is up and running</li><li>• "inactive", push disabled in configuration file</li><li>• "error", port already used by another application</li></ul>
<code>sys.push_count</code>	total number of push messages received from controllers
<code>sys.push_ack_errors</code>	total number of acknowledge errors
<code>sys.nad_list</code>	active controllers, including static, push and autodetect
<code>sys.push_list</code>	formatted list of active controllers, including their status
<code>sys.push_list_count</code>	total number of controllers in the push list
<code>sys.cache_valid</code>	configured cache validity time [s]
<code>sys.cache_request</code>	configured cache request time [s]
<code>sys.udp_rx_count</code>	total number of A-bus messages received
<code>sys.udp_tx_count</code>	total number of A-bus messages transmitted

### Controller-related variables

<code>c20000.sys.plc_status</code>	status of the controller at the last data reading: <ul style="list-style-type: none"><li>• "ok", controller is running</li><li>• "pgm missing", controller has no plc program</li><li>• "alc missing", controller has no alc file</li><li>• "offline", controller does not respond</li><li>• "?", status of the controller is unknown</li></ul>
<code>c20000.sys.ip_port</code>	controller ip address and port in format "xxx.xxx.xxx.xxx:yyyy"
<code>c20000.sys.timestamp</code>	time and date when the program is sent to the controller
<code>c20000.sys.response_time</code>	number of milliseconds between request and response
<code>c20000.sys.bytes_transferred</code>	number of bytes transfered between server and controller
<code>c20000.sys.com_error_count</code>	number of communication errors encountered by controller
<code>c20000.sys.variables</code>	list of available controller variables in XML format
<code>c20000.sys.alc_file</code>	allocation file downloaded from the controller, plain text

Replace 20000 with serial number of your controller. The response is formatted as XML or plain text. To check the format of a specific variable, create request in your browser.

## Common configurations

LAN with ip autodetect  
controllers and server are in the local network

```
[ETH]
enabled = true
autodetect_enabled = true
[PUSH]
enabled = false
[CAN]
enabled = false
```

WAN using push  
controllers send push, server has a known ip address

```
[ETH]
enabled = true
autodetect_enabled = false
[PUSH]
enabled = true
[CAN]
enabled = false
```

CAN interface  
Raspberry Pi, PiCAN2 and a single Cybro controller

```
[ETH]
enabled = false
autodetect_enabled = false
[PUSH]
enabled = false
[CAN]
enabled = true
```

## CAN interface

To use CAN interface, open Raspberry Pi [/boot/config.txt](#) and add this to the end of the file:

```
dtoverlay=spi=on
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25
dtoverlay=spi-bcm2835-overlay
```

Configure [can0](#) interface:

```
sudo /sbin/ip link set can0 up type can bitrate 100000
sudo ifconfig can0 txqueuelen 1000
sudo ifconfig can0 up
```

## Exit code

When the process is terminated, the following codes are returned:

- 1: general error, program stopped unexpectedly
- 2: file not found, can't read the configuration file
- 5: access denied, can't open A-bus or SCGI port
- 130: interrupt signal, program terminated with Ctrl-C



# MQTT client

The MQTT client acts as a connection between controllers and MQTT system.

- connects Cybro with MQTT system using scgi server
- both publisher and subscriber roles are supported
- configured by editing [config.ini](#) file

The client is designed to run as a service, ensuring continuous operation without manual intervention. Both publisher and subscriber roles are supported, enabling data to flow in both directions. The client sends Cybro variables to the broker and writes published data back to Cybro.

Client is configured by editing [config.ini](#) file. If file is modified while client is running, it will automatically reload the file and restart.

MQTT client can use encrypted connection. Open [config.ini](#), section MQTT, enable TLS, generate token and provide a security certificate.

## Publisher

The publisher role of MQTT client sends data from Cybro controller to MQTT broker. Select the variables to be published, MQTT topic and the method of operation.

Read from controller every 10 seconds and publish to MQTT:

```
[PUBLISHER]
var = c20000.cybro_temperature, myhome/system/temperature, 10
```

Read from controller every 10 seconds, publish on MQTT when value is changed:

```
[PUBLISHER]
var = c20000.cybro_temperature, myhome/system/temperature, 10, on-change
```

Publishing can also be triggered by an event. To do this, set period to zero and configure on-request socket. When socket is sent, data is pushed to the MQTT client, which then publishes it to MQTT. Periodic and event-driven publishing can be used together, allowing data to be sent at regular intervals and immediately when a change occurs.

```
[PUBLISHER]
var = c20000.event_data, myhome/events, 0
```

## Subscriber

In the subscriber role, the MQTT client receives data from MQTT broker and writes it to the controller. Multiple controllers can be subscribed to the same topic.

```
[SUBSCRIBER]
var = c20000.cybro_qx00, myhome/livingroom/output
```

To check how that works, install Mosquitto MQTT broker and start in verbose mode:

```
mosquitto -v
```

Turn the cybro output on and off:

```
mosquitto_pub -t myhome/livingroom/output -m 1
mosquitto_pub -t myhome/livingroom/output -m 0
```

# Data logger

The data logger reads data from controller and writes to database.

- periodically reads cybro variables and writes them to database
- manage alarms and events and writes them to database
- configured by editing the [config.ini](#) and [data\\_logger.xml](#) files

The logger uses scgi server for reading and writing to controller. It is designed to run as a service, ensuring operation without manual intervention. It can be executed on the same or separate computer from scgi server. When connected over the internet, encryption should be used.

The logger uses MySQL/MariaDB database. If dbase doesn't exist, data logger (v3.2.2 or later) will automatically create required tables on the first run.

## Configuration

The [config.ini](#) file contain parameters needed to access scgi server and database.

```
[SCGI]
server_address = localhost
port = 4000

[DBASE]
host = localhost
port = 3306
name = cybro
user = root
password = root
max_query_size = 100000
```

The [data\\_logger.xml](#) file defines what will be written to the database.

Variables in <sample> section are periodically read and stored into the [measurements](#) table. The period is defined for each task. Available units are: s (seconds), min (minutes), h (hours) and d (days).

Variables in <event> and <alarm> sections are read periodically, and stored into the [alarm](#) table when condition is satisfied. Events are stateless, while alarms can be active, inactive and acknowledged.

The <class>, <priority> and <message> tags have no function in the program, they are written to the database so that they can be used for data analysis.

The <list> tag can be used to automatically produce multiple variables. For example, this two groups

```
<list>
  <group>
    <name>cybro</name>
    <item>c20000</item>
    <item>c20001</item>
  </group>
  <group>
    <name>array</name>
    <item>[0]</item>
    <item>[1]</item>
    <item>[2]</item>
  </group>
</list>
```

with this line

```
<variable>{cybro}.myvar{array}</variable>
```

will produce the following output:

```
<variable>c20000.myvar[0]</variable>
<variable>c20000.myvar[1]</variable>
<variable>c20000.myvar[2]</variable>
<variable>c20001.myvar[0]</variable>
<variable>c20001.myvar[1]</variable>
<variable>c20001.myvar[2]</variable>
```

## Alarm type

### 1. Binary alarm

When the variable is set, it will create a new alarm. To create the next alarm, the variable must return to zero for at least one minute (sample period).

```
<task>
  <period>1min</period>
  <class>system</class>
  <message>Alarm requested by controller</message>
  <variable>c20001.alarm_request</variable>
  <enabled>true</enabled>
</task>
```

### 2. High limit alarm

When measured value is higher than hilimit ( $>28V$ ), it will create a new alarm. To create the next alarm, variable must fall below hilimit minus hysteresis ( $28V-2V=26V$ ).

```
<task>
  <period>1min</period>
  <class>system</class>
  <message>Power supply too high</message>
  <variable>c20000.cybro_power_supply</variable>
  <hilimit>280</hilimit>
  <hysteresis>20</hysteresis>
  <enabled>true</enabled>
</task>
```

### 3. Low limit alarm

When measured value is lower than lolimit ( $<18V$ ), it will create a new alarm. To create the next alarm, variable must rise above lolimit plus hysteresis ( $18V+2V=20V$ ).

```
<task>
  <period>1min</period>
  <class>system</class>
  <message>Power supply too low</message>
  <variable>c20000.cybro_power_supply</variable>
  <lolimit>180</lolimit>
  <hysteresis>20</hysteresis>
  <enabled>true</enabled>
</task>
```

#### 4. Out of range alarm

When measured value is outside of 18..28V range, a new alarm is triggered. To inactivate the alarm, variable must return within the 20..26V range.

```
<task>
  <period>1min</period>
  <class>system</class>
  <message>Power supply out of range</message>
  <variable>c20000.cybro_power_supply</variable>
  <hilimit>280</hilimit>
  <lolimit>180</lolimit>
  <hysteresis>20</hysteresis>
  <enabled>true</enabled>
</task>
```

## Installation

These instructions apply exclusively to data logger. When used in conjunction with web scada, database must be created and managed using Django commands.

Install database (Linux)

```
sudo apt update
sudo apt install mariadb-server -y
```

Install database (Windows)

Download the latest stable version from [mariadb.org](https://mariadb.org). Make sure the executable is included in PATH environment variable.

Configure database (Linux and Windows)

Log into database shell:

```
mysql -u <user> -p
```

Enter password and create a new database:

```
CREATE DATABASE cybro;
USE cybro;
SOURCE db_create.sql
```

Create user and grant privileges to the database:


```
CREATE USER '<user>'@'localhost' IDENTIFIED BY '<password>';
GRANT ALL PRIVILEGES ON cybro.* TO '<user>'@'localhost';
```

When finished, enter chosen user and password to the [config.ini](#) file.

# Web scada

Web-based interface for Cybro controllers that features:

- real-time monitoring
- timeplots and tables
- energy yield report
- pages built in HTML
- custom tags for dynamic content
- data export option



**Operation parameters**

<input type="text" value="1"/>	<input type="text"/>	<input type="button" value="Submit"/>	section number (0-no sections, 1-A, 2-B)
<input type="text" value="250"/>	<input type="text"/>	<input type="button" value="Submit"/>	inverter nominal power [kW]
<input type="text" value="1000"/>	<input type="text"/>	<input type="button" value="Submit"/>	nominal range of PV reference cell [W/m2]
<input type="text" value="1260"/>	<input type="text"/>	<input type="button" value="Submit"/>	total surface of photovoltaic panels [m2]

Web scada is a content management system (CMS) that manages users, controllers, pages, and plants (projects). Static content is created with HTML, dynamic content is created with custom tags.

User account and permissions are created by the administrator:

home page..... enable user to view their private plants  
plant administrator ..... allow user to change the content of their pages

A administrator..... complete control of users, pages and plants  
S site manager..... add/edit/delete static pages  
U user manager ..... add/edit/delete user accounts  
C controller manager..... add/edit/delete controllers  
T template manager..... add/edit/delete templates  
M media manager..... add/delete media files (images, video, documents)  
P plant manager..... add/edit/delete plants (must be plant administrator)  
W write permission..... edit cybro settings, acknowledge alarms (must be plant administrator)

Create a new project

1. plants, create a new plant
2. plant properties, add administrator
3. add controllers

Add controller to the project

1. configure cybro push, check acknowledge
2. open server status, check push list
3. controllers, add cybro to the list, set owner
4. plants, edit properties, add plant controller

Add pages to the project

1. create a template for each section
2. insert search/replace pair for plant-specific data

Web scada supports time zones. Timestamps (samples, events, alarms) are stored as UTC. When timestamp is displayed, time difference is calculated. Server log is recorded in server time.

## Installation

Create a new database:

```
python manage.py migrate
mysql -u user db -p < db_initial_data.sql
```

Upgrade an existing database (created with manage.py):

```
python manage.py migrate
```

Upgrade an existing database (not created with manage.py):

```
mysql -u user db -p < db_upgrade_v1111_to_v120.sql
mysql -u user db -p < db_upgrade_v114_to_v115.sql
```

After completing these steps, subsequent upgrades should be performed as if the database is created with manage.py.

This is recommended Django method for creating and migrating database. Any other approach may trigger warnings during startup and cause conflicts during upgrades.

### Setup e-mail reports

E-mail reporting system provides scheduled updates, delivering summaries and critical data directly to users' inboxes. It can generate daily performance metrics, system alerts, usage statistics, ensuring users stay informed without the need to log in to the platform.

1. Open [/app/web\\_scada/settings/settings\\_local.py](#), fill SMTP and EMAIL fields.
2. Edit user, email, save. Test. If not received, check SMTP log.
3. Add plant to user home page, subscribe to alarms and events.
4. Add crontab entries to run the reporting engine once an hour:

```
crontab 0 * * * * /usr/bin/.../app/web_scada/project/run_create_reports.py
crontab 0 * * * * /usr/bin/.../app/web_scada/project/run_send_reports.py
```

Use full path. Make sure both python files are set as executable and user rights are set accordingly.

### Run a test server

The Django test server allows local testing of your application without installing the Apache server.

```
cd /app/web_scada/
python manage.py runserver localhost:8080
```

It is recommended to use test server in virtual environment.

### Collect static files

The `collectstatic` command gathers static files from specified directories and copies them to the central directory defined by `STATIC_ROOT`. Django development server use static data without collecting. In production, it's recommended to use Apache or NGINX web server to serve static files efficiently.

```
cd /app/web_scada/
python manage.py collectstatic
```

Files will be bundled into [/app/web\\_scada/staticroot](#) directory, used by the Apache to serve static files.

More installation instructions and scripts in [/app/web\\_scada/doc/](#) directory.

## Object reference

### decimal

Temperature: 24.7 °C

Decimal number, displayed in current text style using HTML <span> element.

- var (cybro tag to read value from, e.g. c20000.cybro\_temperature)
- decimals (number of digits after decimal point, default is 0)
- groupdigits (use comma to separate digits in groups by three: 12,345,678.99, default is 0)
- zeroblanking (0-show leading zeros, 1-clear leading zeros, default is 1)
- digits (field width in digits, meaningful when zeroblanking is 0, default is 4)

```
<cybro>
  <type>decimal</type>
  <var>c20000.cybro_temperature</var>
  <decimals>1</decimals>
  <groupdigits>1</groupdigits>
  <zeroblanking>1</zeroblanking>
  <digits>4</digits>
</cybro>
```

Value is displayed with a specified number of decimal places. For real variables this is straightforward, while for integer variables decimal point is shifted by the specified number of places.

### textlist

Setpoint: High

Displays a string for each value. The tipping point is in the middle of the given values.

If action is enabled, it can set a value on click. If a single value is given, the action is write, otherwise it toggles between given values.

- var (cybro tag to read value, e.g. c20000.setpoint)
- text (texts to replace on values, alternate is <texts>Off,Low,High</texts>)
- value (specific values for texts, alternate is <values>0,50,100</values>, default is 0,1,2...)
- action (0-display only, 1-action enabled, default is 0)

```
<cybro>
  <type>textlist</type>
  <var>c20000.setpoint</var>
  <text>Off</text>
  <text>Low</text>
  <text>High</text>
  <value>0</value>
  <value>1</value>
  <value>2</value>
  <action>0</action>
</cybro>
```

### bitlist

RF signal: 

Displays an image for each value. Breakpoint is halfway between the values.

If action is enabled, it can set a value on click. If a single value is given, the action is write, otherwise it toggles between values.

- var (cybro tag to read value, e.g. c20000.selection)
- file (image files to show on values, alternate is <files>test0.png,test1.png,test2.png</files>)
- value (specific values for bitmaps, alternate is <values>0,50,100</values>, default is 0,1,2...)
- action (0-display only, 1-action enabled, default is 0)

```
<cybro>
  <type>bitlist</type>
  <var>c20000.rf_signal</var>
  <file>signal_0.png</file>
  <file>signal_1.png</file>
  <file>signal_2.png</file>
  <file>signal_3.png</file>
  <value>0</value>
  <value>1</value>
  <value>2</value>
  <value>3</value>
  <action>0</action>
</cybro>
```

### bargraph

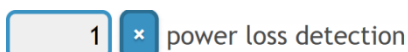


Horizontal or vertical bar, graphically representing the value.

- var (cybro tag to read value from, e.g. c20000.cybro\_temperature)
- barstyle (horizontal or vertical, default is horizontal)
- min (minimum value, default is 0)
- max (maximum value, default is 100)
- size (graph width/height in pixels, default is 100)
- thickness (graph height/width in pixels, default is 10)
- color (#rrggbb, bar color in hexadecimal RGB notation)
- bgcolor (#rrggbb, background color in hexadecimal RGB notation)

```
<cybro>
  <type>bargraph</type>
  <var>c20000.line_voltage</var>
  <barstyle>horizontal</barstyle>
  <min>0</min>
  <max>400</max>
  <size>250</size>
  <thickness>24</thickness>
  <color>#3A92C4</color>
  <bgcolor>#C0C0C0</bgcolor>
</cybro>
```

### toggle



Numeric field with toggle button. It can operate as binary (0/1), or cycle through a predefined set of values (0/25/50/75/100).

- var (cybro variable, e.g. c20000.setpoint)
- max (maximum value, default is 1)
- value (predefined set of values, alternate is <values>0,50,100</values>, default is 0,1,2...)



- digits (field width in digits, default is 4)
- color (#rrggbb, background color in hexadecimal RGB notation)

```
<cybro>
  <type>toggle</type>
  <var>c20000.power_loss</var>
  <digits>4</digits>
</cybro>
```

#### incdec

acceptable current

Numeric field with +/- buttons.

- var (cybro tag to read and write value, e.g. c20000.setpoint)
- min (minimum value, default is 0)
- max (maximum value, default is 9999)
- step (increment/decrement value, default is 1)
- decimals (number of digits after decimal point, default is 0)
- digits (field width in digits, default is 4)
- color (#rrggbb, background color in hexadecimal RGB notation)

```
<cybro>
  <type>incdec</type>
  <var>c20000.current_limit</var>
  <min>0</min>
  <max>250</max>
  <step>0.5</step>
  <decimals>1</decimals>
</cybro>
```

#### submit

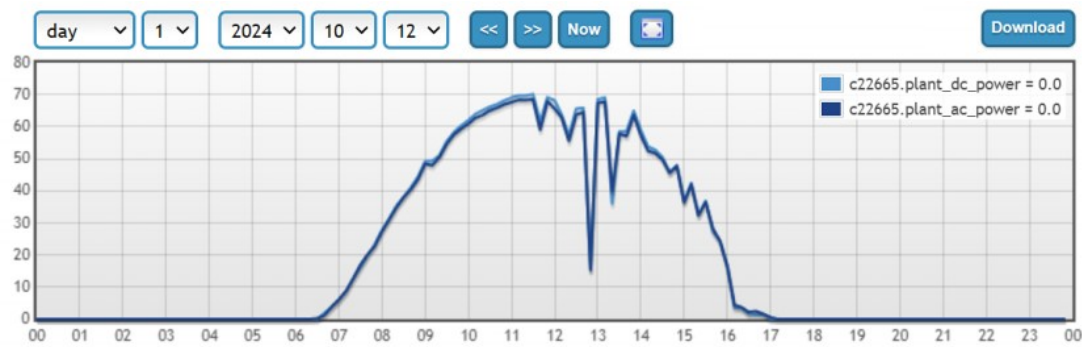
write magic

Field for display value, field for entering value, submit button.

- var (cybro tag to read and write value, e.g. c20000.setpoint)
- min (minimum accepted value, default is 0)
- max (maximum accepted value, default is 9999)
- decimals (number of digits after decimal point, default is 0)
- digits (field width in digits, default is 4)
- color (#rrggbb, background color in hexadecimal RGB notation)

```
<cybro>
  <type>submit</type>
  <var>c20000.write_magic</var>
  <min>0</min>
  <max>9999</max>
  <decimals>1</decimals>
</cybro>
```

## timeplot



Shows historical data from database, as connected line or series of bars. Supports multiple tags.

Timeplot can display multiple tags. If more than one tag needs to be displayed, `<var>` should contain name list separated by comma, and `<color>` should contain list of colors separated by comma. If not defined, default colors are used for each tag.

Available colors: aqua, azure, beige, black, blue, brown, cyan, darkblue, darkcyan, darkgrey, darkgreen, darkkhaki, darkmagenta, darkolivegreen, darkorange, darkorchid, darkred, darksalmon, darkviolet, fuchsia, gold, green, indigo, khaki, lightblue, lightcyan, lightgreen, lightgrey, lightpink, lightyellow, lime, magenta, maroon, navy, olive, orange, pink, purple, violet, red, silver, white, yellow.

- `var` (cybro variables to plot, comma separated)
- `graphstyle` (line|bars, default is line)
- `span` (15min|hour|day|month|year, default time unit)
- `spancount` (default graph width in time units)
- `spanmin` (15min|hour|day|month|year, lower limit of the user's selection range)
- `spanmax` (15min|hour|day|month|year, upper limit of the user's selection range)
- `cumulative` (graphstyle=bars, each bar shows difference to the previous one)
- `resolution` (number of columns per `<span>` unit, default is hour 6, day 24, month 31, year 12)
- `datetime` (yyyy-mm-dd hh:00, default time, "now" if empty)
- `timeskip` (specifies whether the time skip buttons are visible)
- `download` (specifies whether the download button is visible or not, default is 1)
- `min` (minimum y value or auto for autoscaling, default is 0)
- `max` (maximum y value or auto for autoscaling, default is auto)
- `decimals` (number of digits after decimal point, default is 0)
- `color` (#rrggbb, line color in hexadecimal RGB notation or one of the predefined colors)
- `width` (timeplot width in pixels, default is 750)
- `height` (timeplot height in pixels, without buttons, default is 250)
- `showlegend` (if true the legend is shown, anything else it is not, default is true)
- `legendlabel` (comma separated list of labels for tags in the legend)
- `legendunit` (a suffix for all labels in the legend)

```
<cybro>
  <type>timeplot</type>
  <var>c20000.voltage</var>
  <graphstyle>line</graphstyle>

  <span>hour</span>
  <spancount>1</spancount>
```

```

<spanmin>hour</spanmin>
<spanmax>day</spanmax>

<datetime></datetime>

<timeskip>1</timeskip>
<download>1</download>

<min>0</min>
<max>auto</max>
<decimals>0</decimals>

<cumulative>0</cumulative>
<resolution></resolution>

<color>#EDC240</color>
<width>750</width>
<height>250</height>

<showlegend>true</showlegend>
<legendlabel>Power</legendlabel>
<legendunit>kW</legendunit>
</cybro>

```

## alarm\_list

Event or alarm list for the specified controller or the whole plant.

- controller (cybro name, empty for whole plant, default is empty)
- filter (all/alarms/events, default is all)
- items (item count per page, default is 10)

```

<cybro>
  <type>alarm_list</type>
  <controller>c20000</controller>
  <filter>alarms</filter>
  <items>25</items>
</cybro>

```

## template

Container for a piece of HTML code that can be reused multiple times. Before being sent to the browser, the template is processed by a series of search/replace operations. This allows the same template to be used on various pages displaying different content.

- name: template name
- search: text to be replaced
- replace: text to replace with

```

<template>
  <name>string header</name>

  <search>xxx-project-name-xxx</search>
  <replace>Primel plant</replace>

  <search>cXXXX</search>
  <replace>c20000</replace>
</template>

```

## Troubleshooting

### SCGI server not running

1. Ensure that the SCGI port (TCP 4000) is available.
2. Ensure that the A-bus port (UDP 8442) is available.
3. Check log file for error messages.

### Apache started, no pages displayed

1. Verify Django version
2. Check `/app/web_scada/settings/settings_local.py`
3. Ensure that wsgi add-on is installed and properly configured in `httpd.conf`
4. Check that MySQL is running and the database is created, with all tables and fields

### Cybro controller does not respond

1. Verify that cybro is running
2. Manually set [push\\_req](#) and check for the acknowledge
3. Ensure the push address and port are entered correctly
4. Ping the destination server to confirm its availability
5. Try using the server's ip address instead of domain name
6. Check that the server is not being blocked by a firewall

### Cybro returns error code 2 (variable not found)

1. Ensure the program contains the variable you are trying to read
2. Check that cybro has valid IP address and correct network settings
3. Check that alc table is sent to cybro (CyPro/Configuration/General/Send allocation table to PLC)
4. Check that [config.ini](#) option [only\\_user\\_variables](#) is false