

## ADVANCE INFORMATION

ALL INFORMATION IN THIS USER MANUAL IS PRELIMINARY AND SUBJECT TO CHANGE.

# FanMagic – *Fault Tolerant Intelligent Fan Controller and WatchDog Timer*

R 2.0 (July 2006)

1999 by DV "Industrial Computer"®

## Introduction

This device should be attached and tuned by specialized personnel only after familiarizing themselves with current manual!

This manual is intended for firmware release **2.0**. The firmware release number is written on the label stick on the CPU inside the device. Also firmware release number can be obtained by a proper command sent through the serial line.

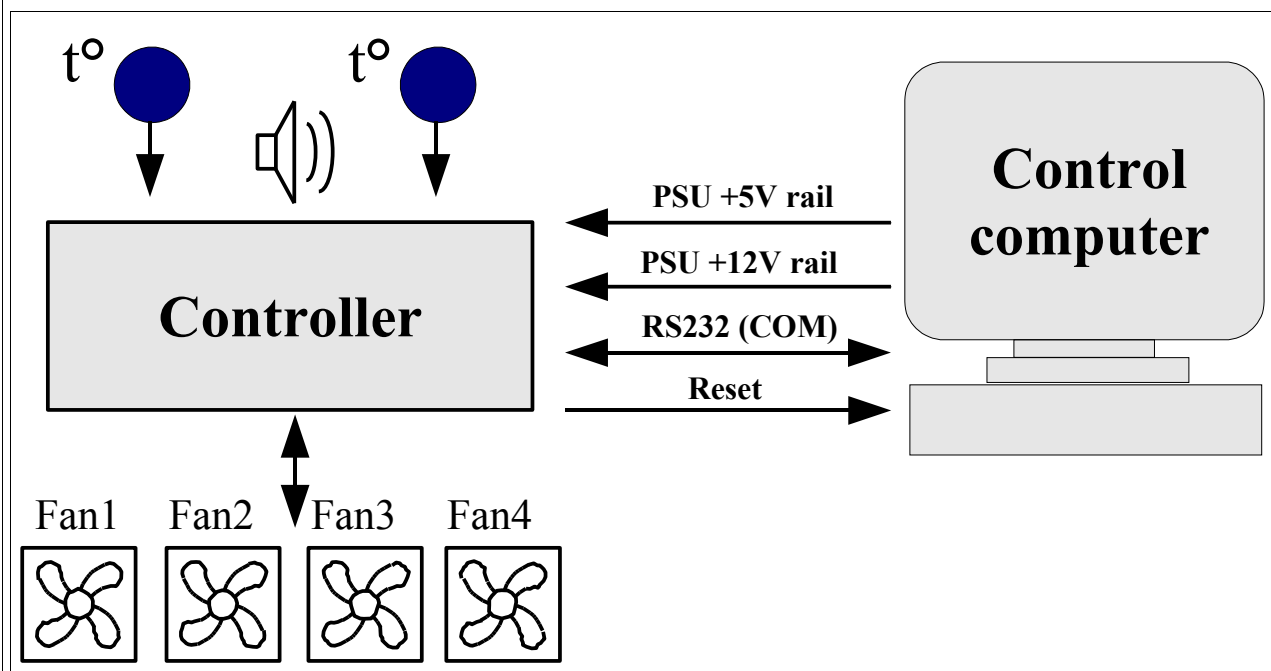
All registered trademarks are the property of their respective owners.

## Overview

**FanMagic** (further controller) is a hardware-firmware device that realizes a temperature-based automatic-fan-speed control and serial line communication with computer via COM-port. The controller operates completely independent of computer's software once the

control loop is configured through serial line. Monitoring is done through serial line also. In the un-fortunate event of some system software failure, thermal management will still continue. The controller also includes a WatchDog to supervise the computer. The controller uses pre-charted matrix to make linear control. Linear control varies the voltage applied to the fan to adjust its speed. Fan revolution varies depending on measured temperature of airflow according to matrix response. Reducing of the fan speed as the temperature drops minimizes the system acoustic noise, prolongs fans service life period, diminishes power consumption. The controller assures early diagnostics of fan failures and increases reliability of hardware functioning due to the overall monitoring of cooling system and signalling in case of failures. The use of controller combined with the system with redundant cooling facilities allows to design fault-tolerant system with long-life performance.

There are two modification of FanMagic controller:  
1) FanMagic - without serial line and WatchDog timer.  
2) FanMagicX - with serial line and WatchDog timer.



FanMagic features include:

- ◆ Controller-regulator of 4 12VDC fans. The fan revolution measuring and slide control of fans depending on measured temperature that uses matrix with variable parameters. The stand-alone monitoring of cooling system with output of results to serial line.
- ◆ 3 channels of voltage measuring (+5V, +12V rails and fan voltage)
- ◆ 2 channels of temperature remote sensing (using remote thermistors)
- ◆ Chime and warning light (buzzer and “alarm/error” red LED)
- ◆ EEPROM to store data, matrix and configuration

FanMagicX features include (extra to above listed):

- ◆ Fixed 9600, 8 bit, no parity, 1 stop bit operation
- ◆ RTS/CTS hardware handshaking
- ◆ Controllable counter that acts as a WatchDog timer for the (connected) control computer

## Fan Control

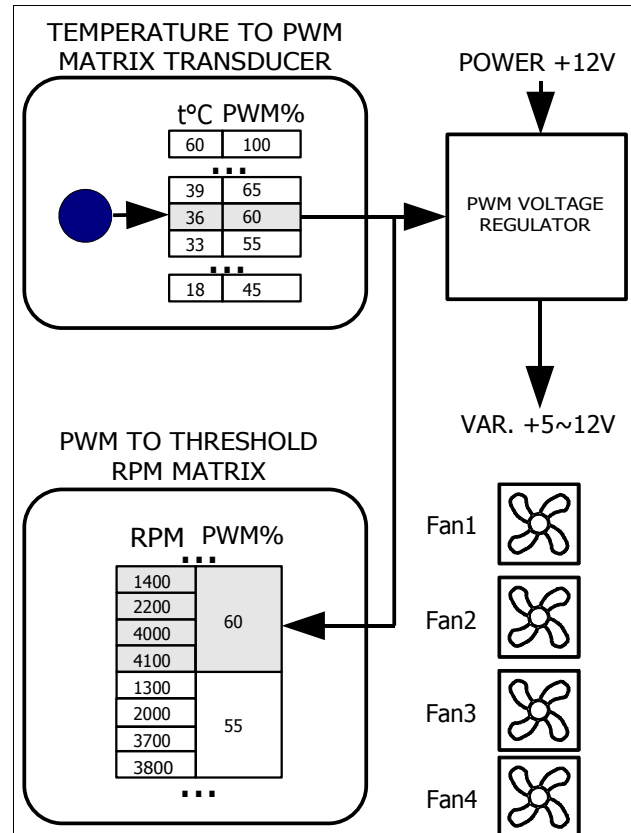
The principle of controller operation is frequent measuring (3 times per second) and analysing of airflow temperature and rotation speed of attached fans as well as generating of the voltage applied to the fan according to preliminary charted matrix. A typical 12VDC fan has an effective operating-voltage range of about 7 to 12V. The lowest voltage at which the fan will operate is about 5 to 7V, but this may be the lowest voltage at which the fan will run once already spinning. The controller varies this voltage by step-down converter with pulse width modulation (PWM) to realize linear control. Up to 4 fans with nominal voltage 12VDC and with three wires (third wire is the fan's tachometer signal) could be attached to the controller. The controller can detect the quantity of attached fans by means of own special procedure that will be described further. The range of measured temperature from 18 to 60°C is quantized by 3°C intervals. Each interval is associated with PWM value of step-down converter. The value of PWM that is associated with temperature interval could be changed and regulation curve could be mould individually. The content of such matrix with pre-charted by producer default values is given below:

Table 1

T°C	<18	<21	<24	<27	<30	<33	<36	<39	<42	<45	<48	<51	<54	<57	<60
PWM%	45	45	45	45	50	55	60	65	70	75	80	85	90	95	100

The figure below that explains fan control shows how the

measured value of temperature (36°C) addresses the row of matrix with PWM 60%.



PWM value determines fan voltage and finally the rotation speed.

Measured values are compared with tabulated points (individually for each fan). Normally functioning fan should not have the rotation speeds lower than threshold matrix values. In presented example the measured value of temperature (36°C) and PWM value 60% determine the threshold rotation values for 1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> fan as 1400, 2200, 4000 and 4100 correspondingly. Even if one of fans has rotation speed lower than pre-charted threshold value, the controller detects it as a fault, chimes, lights the LED and defines PWM 100%. Thus, early diagnostics of fan failure is assured at the time when rotation (and cooling) ability still is not lost but fault has been detected and signalized. If quantity of fans and air performance are redundant, the early diagnostics allows to increase reliability of cooling system essentially, while fan speed-down at low temperature increases fan life and decreases acoustic noise of the most unreliable and noisy element of modern electronics – fan.

Both tables of matrixes are placed in nonvolatile memory and there are some ways for their corrections if needed.

The controller (matrix) can be tuned up to determine individual handling characteristic both for PWM temperature characteristic and for attached fan behaviour in real use environment.

# Connection

Figure below shows pinouts and profiles for the controller connectors. **NOTE!** The controller takes power voltage from +5V rail. Fan's driver (PWM step-down convertor) takes voltage from +12V rail.

J1, J2, J3 and J4 are connectors that are used to connect 3-wire 12VDC fans (third wire is the fan's tachometer signal).

J5 (a 4-pin polarized connector) should be connected to power supply unit of control computer.

J7 is a 10-pin polarized connector that is used for connection via serial line (RS-232) to the COM-port of the computer. **NOTE!** J7 is present on FanMagicX version only. The controller uses the RTS/CTS hardware handshaking of serial line flow control. However, the controller ignores signal on RTS line assuming that control computer is always ready to receive data.

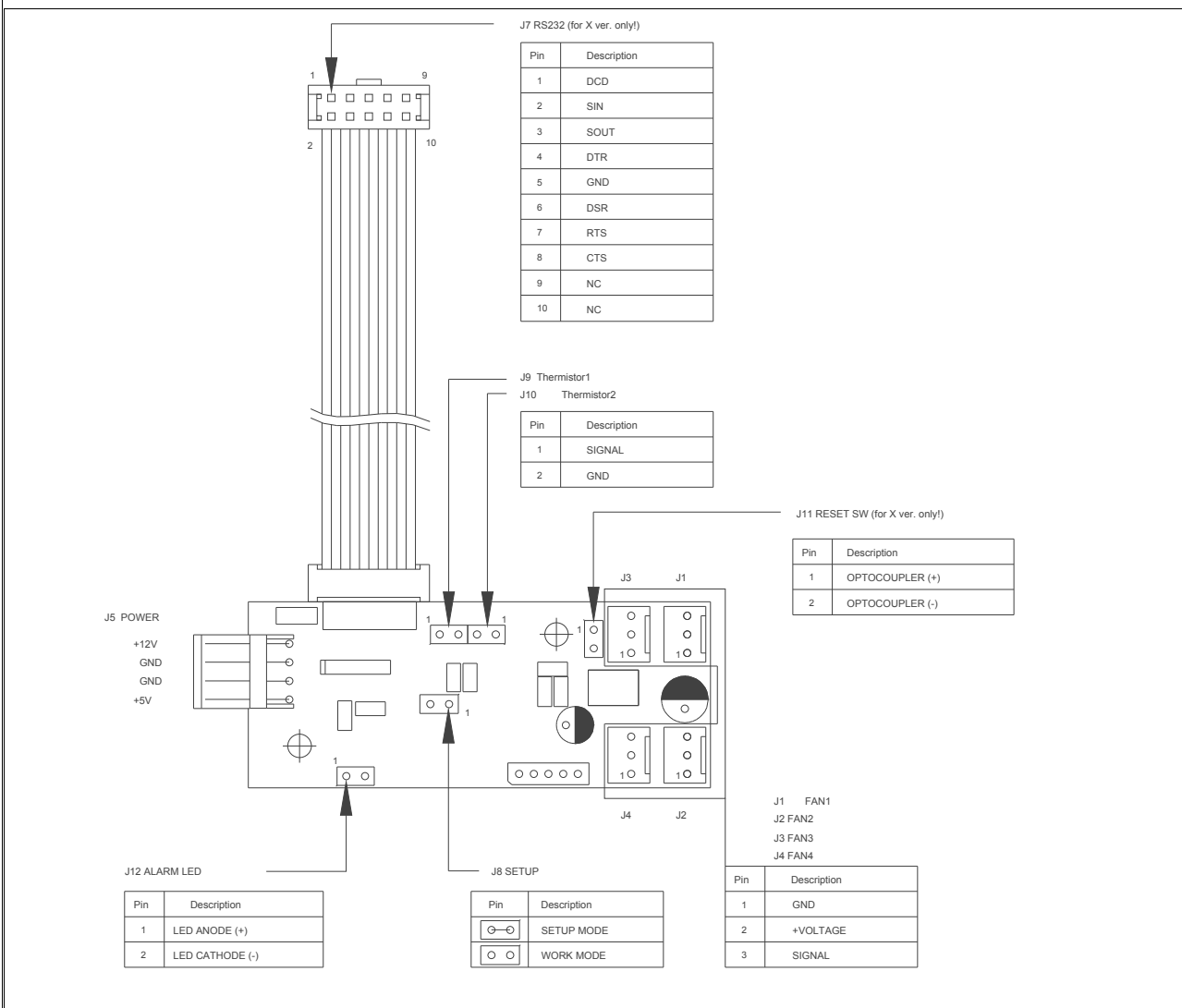
Therefore, really only four-wire link is required – SG(Signal Ground), SIN(RD)(Serial Input/Read of data), SOUT(TR)(Serial Output/Transmitt of data), CTS(Clear To Send). **NOTE!** J7 is present on FanMagicX version only.

J8 is reserved for next releases and for release V2.0 is used for EEPROM defaults setting (setup mode).

J9 and J10 (2-pin connectors) are used to connect remote (0.4m) temperature sensors (thermistors).

J11 should be connected to “Front Panel” connector of computer's motherboard that is usually connected to “Reset” button located on PC front panel. I.e. the controller acts as this button with the difference that the polarity should be kept. The controller can be connected instead of or in parallel to “Reset” button. **NOTE!** J11 is present on FanMagicX version only.

J12 should be connected to “Alarm LED”.



## Serial Line (FanMagicX only)

The controller that is connected to control computer by serial line, operates as a terminal device that communicates via fixed 9600, 8 bit, no parity, 1 stop bit connection. Computer acts as initiator of data communication. The controller responds by data block to the most commands sent from computer. The controller realizes a likeness of ANSI/VT100 terminal by means of subset of escape commands with extensions for WatchDog timer, temperature and voltage measurement channel and so on. Input from serial line is latched by incoming buffer of 23 bytes size. String with 24 bytes and more will be wrapped. The controller starts parsing after receiving of CR(0x0d) or LF(0x0a) symbol or after overflowing of input buffer (23 bytes). Note, that parameters, such as the cursor position, are in ASCII digits, so 12 represents the character "1" and the character "2", not a single byte with the value of 12. Also, characters are case sensitive, so Esc[@W and Esc[@w are two different commands. There are two sets of commands: «Terminal» and «Monitor». The controller starts with «Monitor» set of commands after power on and can be switched to «Terminal» set and back to «Monitor» set via serial line. Commands and examples of communication via serial line are described in further sections.

## Operation

The controller starts up immediately after arising of voltage on +5V power rail. There are two operation modes: main «Monitor» mode and auxiliary «Setup» mode. The controller provides supervision and fan speed rotation control in «Monitor» mode only. The controller switches to «Monitor» mode when +5V power rail voltage arises and the J8 has been opened. Otherwise, if the J8 has been closed when +5V power rail voltage arises, the controller switches to «Setup» mode. In «Setup» mode the controller executes the *fan examination procedure*. Also, this procedure could be invoked if the controller has been in «Monitor» mode. The procedure can be invoked by sending of corresponding command from computer via serial line (FanMagicX only). In «Monitor» mode, the controller

provides supervision and fan rotation speed control in spite of serial line connection. If serial line has been connected, then input from the serial line (computer) is parsed, decoded and executed (if it is correct command sequence). With «Monitor» set of commands the controller echoes with serial line input data bytes. Thus, if simple TTY-terminal is connected via COM-port of PC to FanMagicX controller, it is possible to realize dialog with firmware of FanMagicX controller, to see voltages, fan rotation speed or fan examination and so on. It is not necessary to install any software on PC as simple TTY-terminal is usually preinstalled with another utilities for all known OS. In «Monitor» mode the controller runs the program of temperature and voltage monitoring, measures fan revolution speed and smoothly controls of fan depending on measured temperature based upon algorithm that uses matrix with variable parameters. Thus, it acts as autonomous controller of cooling system.

## Connected fan examination procedure

When, the *fan examination procedure* starts, the controller recognizes connected fans and emits short beeps. The quantity of beeps are equal to quantity of detected fans. Then the controller varies PWM value from 100% to 40% with 5% step and varies testing voltage from max to min values every 3sec. The procedure starts with 100% value of PWM. Later, the rest of 12 values of PWM from 95% to 40% are proceeded. Outcome of procedure is the update of fan threshold revolution matrix for really connected fans. Procedure lasts for about 1 minute. At that time the controller traces over the fan voltage and fan behaviour. The controller determines the number of connected fans and their nominal revolution values for all values of PWM and for all corresponding fans voltages. If bugs (errors) occur or if abnormal behaviour of fans is detected, then the default values are being loaded to the fan threshold revolution matrix (factory pre-charted values). The content of fan threshold revolution matrix with pre-charted by producer default values is shown below:

Table 2

PWM% \ RPM	100	95	90	85	80	75	70	65	60	55	50	45	40
<b>Fan1</b>	1900	1800	1700	1600	1500	1400	1300	1200	1100	1000	900	800	700
<b>Fan2</b>	1900	1800	1700	1600	1500	1400	1300	1200	1100	1000	900	800	700
<b>Fan3</b>	1900	1800	1700	1600	1500	1400	1300	1200	1100	1000	900	800	700
<b>Fan4</b>	1900	1800	1700	1600	1500	1400	1300	1200	1100	1000	900	800	700

## WatchDog timer (FanMagicX only)

The WatchDog timer is designed to overpass a computer hanging up. Usually computer almost never hangs up. However, if it hangs up and gets stuck, usually there is nobody at the site to press the reset or nobody knows where the stuck computer is because there were no problem with it for a long time. The WatchDog timer is used just for such situations. It automatically restarts the computer that is hung up. To restart computer the WatchDog timer closes circuit between pins 1 and 2 of J11 connector. The WatchDog timer is off after starting of the controller. To switch it on the command "Esc[WW" should be sent and timeout (number of ticks, one tick is 0.3 seconds) should be set by indicating required W, which is a value between 0 and 255. The driver program needs to set the timeout periodically to avoid the WatchDog hits and following "reset" contacts closing. Detailed information about WatchDog is described in further sections.

## Setup mode

The controller switches to «Setup» mode when +5V power rail voltage arises and the J8 has been closed by shorting 1-2 pin. At the start of operating in «Setup» mode the controller executes the *fan examination procedure*. Then the controller emits beeps to indicate temperature value. Long beeps mean decades while short beeps mean ones of double-digit value of temperature in degree Celsius. For example the temperature of 25°C will cause the controller to sound with two long beeps and five short beeps. **Note!** Jumper J8 should be closed all the time during of «Setup» mode. If status of jumper J8 is changing from closed to open while the *fan examination procedure* is executed, then the controller does not emit beeps to indicate temperature value. In this case, the controller emits endless beep that means that PWM values matrix for temperature intervals and fan threshold revolution matrix have been pre-charted by default values for four fans (tables 1 and 2).

## Commands that are Supported by the Controller via Serial Line ("Terminal" Set)

**Note!** Symbols of commands are case sensitive and are in ASCII digits. Many commands require numeric or symbols arguments. These are indicated below as underlined. So, "Esc[FF@P=R" is telling you to replace the E, P and R with decimal numbers (for example, "Esc[F1@40=1000").

### The Controller Release Request – Esc [ c

The controller responds to this code with "V2.0" string to indicate its firmware release.

### Fix Threshold Value of Rotation Speed (RPM) for Indicated Fan and PWM Value – Esc [ F E @ P = R

New threshold value of rotation speed R (revolutions per minute RPM – 0~25000) of fan E (allowed value from 1 to 4) for indicated value of pulse width modulation (allowed value of PWM from 40% to 100%) should be saved in matrix in EEPROM. The controller checks whether command parameters meet allowed values. The controller ignores a command if discrepancy occurs. Also, the controller makes quantization of parameters if needed. The value of R should be quantized to the nearest smallest integer value down to 100rpm. The value of P should be quantized to the nearest smallest integer value down to 5%. For example, "Esc[F1@42=1055" command fixes new threshold value 1000rpm (1055 is quantized to 1000) of fan 1 for PWM 40% (42 is quantized to 40).

### Get Threshold Value of Rotation Speed (RPM) for Indicated Fan and PWM Value – Esc [ F E @ P = R

Threshold value of rotation speed (revolutions per minute) of fan E (allowed value from 1 to 4) for indicated value of pulse width modulation (allowed value of PWM from 40% to 100%) should be read from EEPROM and sent to the computer. The controller checks whether command parameters meet allowed values. The controller ignores a command if discrepancy occurs. Also, the controller makes quantization of parameters if needed. The value of P should be quantized to the nearest smallest integer value down to 5%. For example, "Esc[f2@99" command returns matrix threshold value of fan 2 for PWM 95% (99 is quantized to 95).

### Switch the Controller to "Monitor" set of commands – Esc [ m

The controller switches to «Monitor» set of commands and sends the string "\n\rMonitor mode\n\r>" to serial link as a prompt to dialog after receiving this command. Symbols \n and \r are CR(0x0d) and LF(0x0a) accordingly.

### Assign PWM Value for Indicated Temperature Interval – Esc [ P P @ T

New PWM value P (allowed value of PWM from 40% to 100%) for indicated temperature interval T (allowed value from 18 to 60°C) should be saved in matrix in EEPROM. The controller checks whether command parameters meet allowed values. The controller ignores a command if discrepancy occurs. Also, the controller

makes quantization of parameters if needed. The value of P should be quantized to the nearest smallest integer value down to 5%. The value of T should be quantized to the nearest smallest integer value down to 3°C. For example, "Esc[P63@38" command assigns new PWM value 60% (63 is quantized to 60) for temperature interval 36–39°C (38 is quantized to 36, for temperature series from 18 to 60 with step 3°C).

### Get PWM Value for Indicated Temperature Interval – Esc [ p @ T

PWM value for indicated temperature interval T (allowed value from 18 to 60°C) should be read from EEPROM and sent to the computer. The controller ignores a command if discrepancy occurs. Also, the controller makes quantization of parameters if needed. The value of T should be quantized to the nearest smallest integer value down to 3°C. For example, "Esc[p@20" command returns matrix value of PWM for temperature interval 18°C (20 is quantized to 18°C, for temperature series from 18 to 60 with step 3°C).

### Get Current Value of Rotation Speed for Indicated Fan – Esc [ r F

Current value of rotation speed (revolutions per minute – RPM) that has been measured by the controller for the indicated fan F (0 or 1) is sent to serial line as a response to received command.

### Get Current Temperature Value – Esc [ t C

Current temperature value that is measured for selected channel C (0 or 1) should be sent to serial line as a response to received command.

### Get Current Voltage Value – Esc [ v C

Current voltage value (in millivolts) that is measured for selected channel C (0 corresponds to +5V, 1 corresponds to +12V and 2 corresponds to fan voltage) should be sent to serial line as a response to received command.

### Get Current Value of WatchDog Timeout Counter – Esc [ w

Current value of WatchDog timeout counter should be sent to serial line as a response to received command.

### Load Value of WatchDog Timeout Counter – Esc [ W W

New value W (number of ticks from 0 to 255, one tick is 0.3 seconds) should be loaded into WatchDog timeout counter. For example "Esc[W100" loads timeout counter to 30sec.

## Using of the Controller with “Monitor” Set of Commands in MS Windows (MS Windows XP for example)

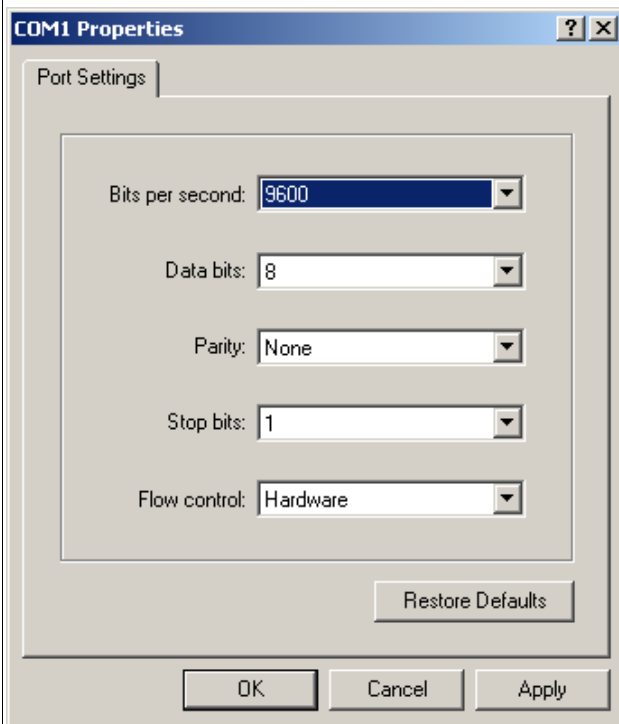
Assume that computer has available COM1 port that should be set as 9600, 8 bit, no parity, 1 stop bit operation. As a dialog application let us use *Hyper Terminal* that is included in MS Windows distributive. To get it ready, open the folder **Communications** and run **Hyper Terminal** (click on “Start” button and then **Programs - Accessories – Communications**. Select **Connection Description**). The following dialog window appears on the screen:



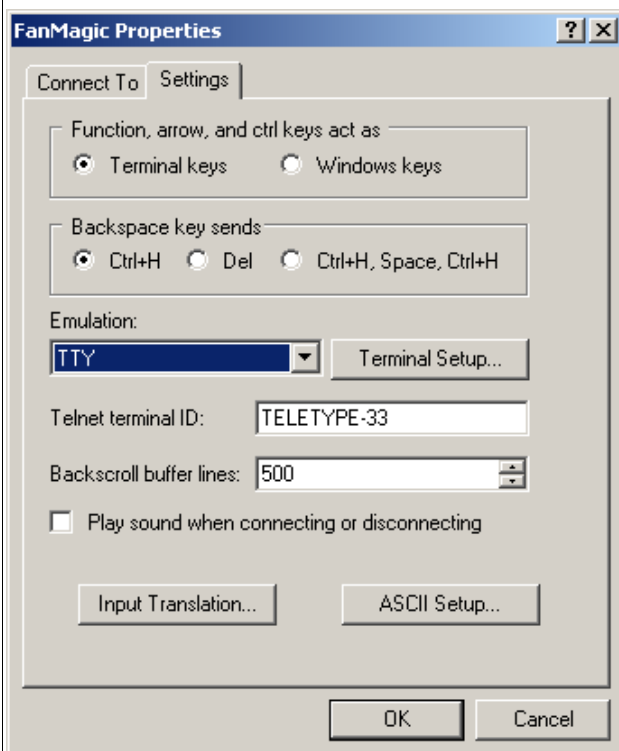
Let us name our connection as **FanMagic** (arbitrary named). Then select connection via serial port COM1 in **Connect To** dialog window:



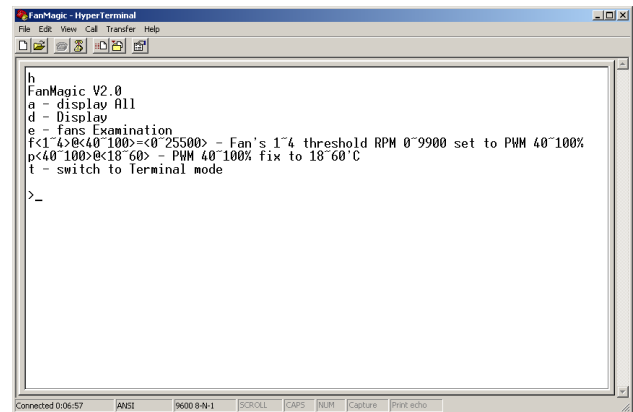
And set up COM1 properties:



Click on **File** button in main menu of *Hyper Terminal* program and click on item **Properties** in pop up menu. In dialog window **FanMagic Properties** select bookmark **Settings** and select the following properties:



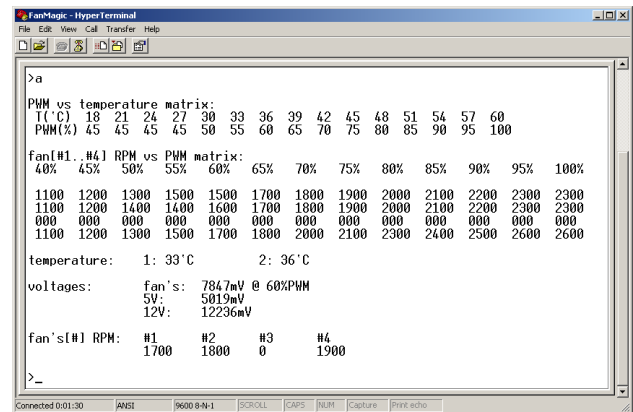
When **Hyper Terminal** configuration has been finished everything is ready to dialog with the controller. Commands that are entered on terminal keyboard are sent to the controller **FanMagic** via serial port while input data flow is displayed on terminal window. In the clear terminal window enter the first command **h (help)** – press keys **h** and then **Enter**. As a response the controller sends firmware release number and list of commands that are supported in «Monitor» mode:



If you see this list, all settings are made correct and you can continue with study of commands.

**- a (display all)**

Press keys **a** and **Enter**. As a response the controller sends matrix of PWM values, fan threshold revolution matrix, current values of temperature, voltage and fan rotation speed at the command sending:



**- d (display)**

Press keys d and Enter. As a response the controller sends current values of temperature, voltage and fan rotation speed at the command sending:

```

1100 1200 1300 1500 1600 1700 1800 1900 2000 2200 2200 2400 2400
000 000 000 000 000 000 000 000 000 000 000 000 000
1800 1200 1300 1400 1600 1800 1900 2100 2200 2400 2500 2600 2600

temperature: 1: 33°C 2: 35°C

voltages: fan's: 7828mV @ 60%PWM
          5V: 5012mV
          12V: 12255mV

fan's[#] RPM: #1 #2 #3 #4
              1700 1700 0 1900

>d
temperature: 1: 33°C 2: 36°C

voltages: fan's: 7923mV @ 60%PWM
          5V: 5012mV
          12V: 12255mV

fan's[#] RPM: #1 #2 #3 #4
              1700 1800 0 1900

>_

```

**- e (fans examination)**

Press keys e and Enter. As a response the controller starts examination procedure of connected fans applying the testing voltage every 3sec. varying from minimal to maximal values and varying PWM value from 100% to 40% with 5% step. All 13 steps of fans examination procedure are displayed in **Hyper Terminal** window step by step with displaying of measured values of fan rotation speed and fans voltage. Outcome of procedure is the update of fan threshold revolution matrix for really connected fans. Procedure lasts for about 1 minute. At that time the controller traces over the fan voltage and fan behaviour. The controller determines the number of connected fans and their nominal revolution values for all values of PWM and for all corresponding fans voltages. If bugs (errors) occurs or if abnormal behaviour of fans is detected, then the default values are being loaded to the fan threshold revolution matrix (factory pre-charted values). A screenshot of finished procedure is shown below:

```

1700 1800 0 1900

>e
fan examination:
PWM fan voltage fan's RPM
% mV #1 #2 #3 #4
100 11818 2500 2500 000 2900
95 11913 2500 2500 000 2900
90 11476 2400 2400 000 2800
85 10849 2300 2300 000 2700
80 10260 2200 2200 000 2600
75 9386 2100 2100 000 2400
70 9025 2000 2000 000 2300
65 8417 1900 1900 000 2100
60 7961 1700 1800 000 2000
55 7239 1700 1600 000 1800
50 6688 1500 1600 000 1600
45 6023 1400 1400 000 1500
40 5719 1300 1300 000 1400

threshold: 200 200 000 300
Matrix have updated in EEPROM

>_

```

Given example shows that three fans with nominal rotation speed 2500 and 2500 and 2900 RPM (for PWM 100%) are connected to the controller. At the PWM 40%

the fans voltage decreases to 5,719V. At the same time the fan rotation speed values decrease to 1300 and 1300 and 1400 RPM correspondingly. The controller measured threshold values of rotation speed that are lower than nominal values by 200 and 300 RPM and wrote these values into EEPROM. These values of 200 and 300 RPM the controller calculates using its own internal algorithm.

**- f (fan's threshold RPM set to PWM)**

Given screenshot for examination procedure of connected fans shows the value of 1400 RPM for PWM 40% and fan number 4. Hence, the threshold value of rotation speed  $1400-300=1100$  RPM is written in EEPROM. One can check this using command **a** (in your case particular value could be different since it depends on fan characteristics). To modify this threshold value to 1000 RPM the following command should be entered:

f4@40=1000 and Enter

Similarly, for fan number 1 and for PWM 60% you can modify threshold value to 1600 RPM by the following command:

f1@60=1600 and Enter.

It is convenient to control the accuracy of fixing of threshold values by submitting command **a** from time to time.

**- p (PWM fix to temperature)**

Given screenshot for command **a** shows default PWM matrix for temperature range with PWM value 45% for range from 18 to 21°C. To modify regulation curve to PWM value of 40% and to reduce rotation speed for temperature range from 18 to 21°C you should enter the following command:

p40@18 and Enter

Similarly, to modify regulation curve to PWM value of 45% and to reduce rotation speed for temperature range from 30 to 33°C you should enter the following command:

p45@30 and Enter

It is convenient to control the accuracy of fixing of threshold values by submitting command **a** from time to time.

**- t (switch to terminal set of commands)**

Command **t** swithes the controller to «Terminal» set of commands by program that is run on control computer. It is possible to switch the controller manually to continue dialog with **Hyper Terminal**, however it is very unfriendly dialog and it has sense only for study of the controller set of commands «Terminal». «Terminal» set of commands is established to interact with special software that is dedicated for the controller **FanMagic**.

## Using of the Controller with “Monitor” Set of Commands in OS Linux

Let us use **minicom** – menu-driven communications program to demonstrate functionality of the controller in OS Linux. Minicom is a program with friendly interface that allows to read data from serial port and to write to this port (COM port, in Windows terminology). This program is part of any distributive. Install **minicom** if this program has not been installed yet. To set it up run it as root:

```
minicom -s
```

The configuration menu appears after starting:



Use “Up” and “Down” keys to make navigation over menu items and select item to change configuration. In **Serial port setup** dialog window set up serial line as following:



In **Modem and dialing**, delete initialization strings: **Init string** and **Reset string**. After finishing return to configuration menu and select **Save setup as dfl** to save configurations in file `/etc/minirc.dfl`. To finish configuration and proceed directly to terminal select item

**Exit**. Now, commands that are entered on terminal keyboard are sent to the controller **TeleServ** via serial port while input data flow is displayed on terminal window. In terminal window enter the first command **h** (**help**) – press keys **h** and then **Enter**. As a response the controller sends firmware release number and list of commands that are supported in «*Monitor*» mode (similarly to previous described section for MS Windows). Similarly for MS Windows, it is possible to dialog by supported commands **a**, **d**, **e**, **f**, **p**, that are described in previous section. Behaviour of the controller on entered commands is absolutely the same because the dialog is provided by firmware of the controller and is OS or terminal software independent. Type Ctrl+A and then Q to exit from program **minicom**. Of course, it is not recommend to run program **minicom** as a root. One should preferably create user account with permission to write to `/dev/ttyS0` and `/dev/ttyS1`. This is usually achieved by adding a new user to `dialout` group.

## Some Examples of Using of the Controller

All examples of using controller are realized on Perl scripts. Operation of these scripts is tested in OS SuSE Linux, however they should operate in any other Unix like OS. Interface with the controller is allocated in special module **FanMagic.pm**. You should copy this file in any directory and you should define environment variable PERLLIB. The easiest way is to add to the file `/etc/profile` the following string:

```
PERLLIB="the name of your directory"
export PERLLIB.
```

You can indicate directory where modules would be searched in the script adding at the beginning of the script the following directive:

```
use libs "the name of your directory"
```

This way is more preferable if the script is run as a CGI program under web server Apache (will be described later). The source code of module **FanMagic.pm** is shown below:

```
package FanMagic;

use strict;
use vars qw(@ISA @EXPORT $VERSION $KEYBUF $DELAY);
use Fcntl qw(:DEFAULT :flock);
use Carp;
use Time::HiRes qw(usleep);
```

```
use Exporter;
@ISA = qw(Exporter);
@EXPORT = qw(tsOpen tsRead tsReadAll tsWrite tsCommand tsTemperature tsVoltage tsRpms tsWatchDog
tsPwm tsVersion tsKey tsWaitKey tsPrint tsSetTerminalMode);
$VERSION = '0.01';

$KEYBUF = "";
$DELAY = 0.1;

#Open device FanMagic (the controller)
sub tsOpen {
    my $name = shift;
    $name = "/dev/ttyS0" unless ($name);
    sysopen(FANMAGIC, $name, O_RDWR, 0666) || croak "Can't open $name";
    flock(FANMAGIC, LOCK_EX);
}

#Read string from the controller
sub tsRead {
    my $rin;
    my $str = "";
    vec($rin = "", fileno(FANMAGIC), 1) = 1;
    sysread(FANMAGIC, $str, 32) if( select($rin, undef, undef, $DELAY) > 0 );
    return $str;
}

#Read all strings from the controller
sub tsReadAll {
    my ($rin, $str);
    while (1) {
        vec($rin = "", fileno(FANMAGIC), 1) = 1;
        last if( select($rin, undef, undef, $DELAY) <= 0 );
        sysread(FANMAGIC, $str, 32);
    }
    return $str;
}

#Write string to the controller
sub tsWrite {
    my $str = shift;
    return unless ($str);
    syswrite(FANMAGIC, $str);
}

#Send command to the controller (valid for "terminal" mode)
sub tsCommand {
    my $comm = shift;
    tsWrite(chr(0x1b)."[".$comm."\n");
}

#Send some commands to the controller (parameters should be placed in array)
sub tsGroupCommand {
    my ($comm, @in) = @_;
    my @out = ();
    foreach (@in) {
        tsCommand($comm.$_);
        push(@out, tsRead());
    }
    return @out;
}
```

```
#Get temperature array
sub tsTemperature {
  return tsGroupCommand("t", qw(0 1));
}

#Get voltage array
sub tsVoltage {
  return tsGroupCommand("v", qw(0 1 2));
}

#Get fan rotation speed array (RPM)
sub tsRpms {
  return tsGroupCommand("r", qw(0 1 2 3));
}

#Get / Set value of counter of WatchDog timer
sub tsWatchDog {
  my $watch_dog = shift;
  if ($watch_dog ne "") {
    $watch_dog = int($watch_dog / 0.3);
    $watch_dog = 255 if $watch_dog > 255;
    tsCommand("W$watch_dog");
    usleep(100000);
    tsRead();
  }
  tsCommand("w");
  usleep(100000);
  my $curr_val = tsRead();
  return (0.3 * $curr_val);
}

1;
```

The first example of the controller *FanMagic* is remote monitoring. Information about current values of fan rotation speed, values of voltage and temperature of remote host could be obtained in your computer browser window. To get it, both web server and script *monitoring.cgi* that provides such service should be installed and run. This script is tested with installed web server Apache on controlled (remote) host. Execution of CGI programs should be enabled in Apache settings (refer to documentation of web server for instructions). Script *monitoring.cgi* should be copied to controlled host in directory where CGI program are located. To test it you can run web browser and enter local URL of script in the address string, for example:

<http://127.0.0.1/cgi-bin/monitoring>

The following page with table should appear:

TeleServ V1.3	
Mar. 2, 2006	
14:40:55	
Temperature	
1	22°C
2	21°C
PWM	
45%	
Voltage	
Fan's	7828 mV
+5V	4870 mV
+12V	12236 mV
Fun's	
1	0 RPM
2	0 RPM
3	0 RPM
4	1900 RPM
WatchDog	
0	

If the host is connected to the net, URL of script should be entered in the address string of your browser. Then you can watch the same page. The script updates information on the screen of browser automatically. The frequency of updating each 3sec is defined by variable `$delay_refresh` and could be changed.

The source code of script *monitoring.cgi* is shown below:

The second example is a program that uses the WatchDog timer of the controller. The WatchDog timer is designed to overpass a computer hanging up. Usually computer almost never hangs up. However, if it hangs up and gets stuck, usually there is nobody at the site to press the reset or nobody knows where the stuck computer is because there were no problem with it for a long time. The WatchDog timer is used just for such situations. It automatically restarts the computer that is hung up. The WatchDog timer is off after starting of the controller. To switch it on, the command "Esc[WW]" should be sent and timeout (number of ticks, one tick is 0.3 seconds) should be set by indicating required W, which is a value between 0 and 255. To switch the timer off, timeout should be set to 0 by "Esc[W0]" command. The special driver is used to load and to control timeout counter. To ensure non-stop running of the computer, the driver should set the timeout periodically to avoid that the WatchDog hits with following "reset" contacts closing. If your computer ever locks up, the driver no longer sets the timeout and the WatchDog hits – "reset" contacts are closed. The WatchDog is designed in such way that it hits only once. This allows to avoid it hitting again during the file system check which will probably follow after the reset (or after power on start). When the computer comes up again, the driver program needs to re-enable it (to send the command Esc[WW with needed timeout). The WatchDog guarantees that the system is always able to execute programs. It does not guarantee that application is still running and responding. To check such things, you can use a *crontab* entry for Linux/UNIX or other programs. You can be confident that the *crontab* will be working because the WatchDog ensures that software in general is still executing. For example, you can design a script that is triggered by a *cronjob* and downloads a webpage from your webserver every 50 seconds. However, you have to be careful with that: a webserver can get heavily loaded with a lot of requests and it is normal if it does not answer all of them. For our example we have created a script *watchdog.cgi* to learn the WatchDog timer facilities in interactive mode. This script, as the first example, is CGI program with web interface and you can use it for remote host as well. To install script refer to instructions for the first example (script *monitoring.cgi*, that is given above). To learn the WatchDog timer on your local computer, enter URL of script in the address string of your browser, for example:

<http://127.0.0.1/cgi-bin/watchdog>

The following page with table should appear:  
Fill in the **Refresh time** field with the time of automatic information update on browser screen. The comfortable time is 2-3 seconds. Fill in the **WatchDog timeout** field with the initial value of counter (for example, 50

seconds). Fill in the **Update time** field with the time of update of initial value of counter (for example, 30 seconds). Clicking on button **start** runs page with continuous updating (time of update is indicated in **Refresh time**), takes up the counter of timer into initial value and allows to watch the dncounting and then taking up the counter of timer. Do not connect J13 of the controller to "reset" connector of motherboard when you play with refresh time and update time. Otherwise every time when counter value is zeroizing the computer resets. If the host is connected to the net, URL of script should be entered in the address string of your browser. Then you can watch the same page.

The source code of script *watchdog.cgi* is shown below:

```
package FanMagic;

use strict;
use vars qw(@ISA @EXPORT $VERSION $KEYBUF $DELAY);
use Fcntl qw(:DEFAULT :flock);
use Carp;
use Time::HiRes qw(usleep);

use Exporter;
@ISA = qw(Exporter);
@EXPORT = qw(tsOpen tsRead tsReadAll tsWrite tsCommand tsTemperature tsVoltage tsRpms tsWatchDog
             tsPwm tsVersion tsKey tsWaitKey tsPrint tsSetTerminalMode tsCursor);
$VERSION = '0.01';

$KEYBUF = "";
$DELAY = 0.1;

#Open device TeleServ (the controller)
sub tsOpen {
    my $name = shift;
    $name = "/dev/ttyS0" unless ($name);
    sysopen(TELESERV, $name, O_RDWR, 0666) || croak "Can't open $name";
    flock(TELESERV, LOCK_EX);
}

#Read string from the controller
sub tsRead {
    my $rin;
    my $str = "";
    vec($rin = "", fileno(TELESERV), 1) = 1;
    sysread(TELESERV, $str, 32) if( select($rin, undef, undef, $DELAY) > 0 );
    return $str;
}
```

```
#Read all strings from the controller
sub tsReadAll {
    my ($rin, $str);
    while (1) {
        vec($rin = "", fileno(TELESERV), 1) = 1;
        last if( select($rin, undef, undef, $DELAY) <= 0 );
        sysread(TELESERV, $str, 32);
    }
    return $str;
}

#Write string to the controller
sub tsWrite {
    my $str = shift;
    return unless ($str);
    syswrite(TELESERV, $str);
}

#Send command to the controller (valid for "terminal" mode)
sub tsCommand {
    my $comm = shift;
    tsWrite(chr(0x1b)."[".$comm."\n");
}

#Send some commands to the controller (parameters should be placed in array)
sub tsGroupCommand {
    my ($comm, @in) = @_ ;
    my @out = ();
    foreach (@in) {
        tsCommand($comm.$_);
        push(@out, tsRead());
    }
    return @out;
}

#Get temperature array
sub tsTemperature {
    return tsGroupCommand("t", qw(0 1));
}

#Get voltage array
sub tsVoltage {
    return tsGroupCommand("v", qw(0 1 2));
}

#Get fan rotation speed array (RPM)
sub tsRpms {
    return tsGroupCommand("r", qw(0 1 2 3));
}
```

```

#Get / Set value of counter of WatchDog timer
sub tsWatchDog {
  my $watch_dog = shift;
  if ($watch_dog ne "") {
    $watch_dog = int($watch_dog / 0.3);
    $watch_dog = 255 if $watch_dog > 255;
    tsCommand("W$watch_dog");
    usleep(100000);
    tsRead();
  }
  tsCommand("w");
  usleep(100000);
  my $curr_val = tsRead();
  return (0.3 * $curr_val);
}

#Set cursor position
sub tsCursor {
  my ($x, $y) = @_;
  tsCommand("$x."."$y."H");
}

1;

```

## TeleServ V1.3

Mar. 2, 2006

14:40:55

Temperature	
1	22 °C
2	21 °C

PWM	
45%	

Voltage	
Fan's	7828 mV
+5V	4870 mV
+12V	12236 mV

Fun's	
1	0 RPM
2	0 RPM
3	0 RPM
4	1900 RPM

WatchDog	
0	

```
#!/usr/bin/perl -w

#use lib "";
use CGI qw(:standard);
use POSIX;
use TeleServ;

#Set 3 sec delay for page refresh
$delay_refresh = 3;
#TeleServ controller serial line port
$device_name = "/dev/ttyS1";

tsOpen($device_name);
tsSetTerminalMode;

#Get measured values from the controller
$version = tsVersion();
($temp_1, $temp_2) = tsTemperature();
$_pwm = tsPwm($temp_1);
($vol_5, $vol_12, $vol_f) = tsVoltage();
($fun_1, $fun_2, $fun_3, $fun_4) = tsRpms();
$watch_dog = tsWatchDog();

#Get data and time
POSIX::setlocale( &POSIX::LC_TIME, "en_US" );
$curr_time = POSIX::strftime("%H:%M:%S", localtime());
$curr_date = POSIX::strftime("%b. %e, %Y", localtime());

#HTML page refresh
$java_script=<<END;
function load()
{ \nsetTimeout( "refresh()", $delay_refresh*1000 );\n}
function refresh()
{ \nwindow.location.href = unescape(window.location.pathname);\n}
END

print header();
print start_html(
    -title=>"TeleServ $version",
    -script=>$java_script,
    -onload=>"load()"
);

print <<EOF;
<div align="center"><h2>TeleServ $version</h2>
<p>${curr_date}<br>${curr_time}</p></div>
<table width="30%" border="0" cellpadding="5" align="center">
<tr>
<td>
        <table width="100%" border="1" cellspacing="0" cellpadding="0">
        <tr>
                <td align="center" colspan="2">Temperature</td>
        </tr>

```

```

        <tr>
            <td align="center">1</td>
            <td align="center">${temp_1}&#176;C</td>
        </tr>
        <tr>
            <td align="center">2</td>
            <td align="center">${temp_1}&#176;C</td>
        </tr>
    </table>
</td>
</tr>
<tr>
<td>
        <table width="100%" border="1" cellspacing="0" cellpadding="0">
            <tr>
                <td align="center">PWM</td>
            </tr>
            <tr>
                <td align="center">${pwm}%</td>
            </tr>
        </table>
</td>
</tr>
<tr>
<td>
        <table width="100%" border="1" cellspacing="0" cellpadding="0">
            <tr>
                <td align="center" colspan="2">Voltage</td>
            </tr>
            <tr>
                <td align="center" width="25%">Fan's</td>
                <td align="center">${vol_f} mV</td>
            </tr>
            <tr>
                <td align="center" width="25%">+5V</td>
                <td align="center">${vol_5} mV</td>
            </tr>
            <tr>
                <td align="center" width="25%">+12V</td>
                <td align="center">${vol_12} mV</td>
            </tr>
        </table>
</td>
</tr>
<tr>
<td>
        <table width="100%" border="1" cellspacing="0" cellpadding="0">
            <tr>
                <td align="center" colspan="2">Fun's</td>
            </tr>
            <tr>
                <td align="center" width="25%">1</td>
                <td align="center">${fun_1} RPM</td>
            </tr>
        </table>

```

```
<tr>
    <td align="center" width="25%">2</td>
    <td align="center">${fun_2} RPM</td>
</tr>
<tr>
    <td align="center" width="25%">3</td>
    <td align="center">${fun_3} RPM</td>
</tr>
<tr>
    <td align="center" width="25%">4</td>
    <td align="center">${fun_4} RPM</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
    <table width="100%" border="1" cellspacing="0" cellpadding="0">
    <tr>
        <td align="center">WatchDog</td>
    </tr>
    <tr>
        <td align="center">${watch_dog}</td>
    </tr>
    </table>
</td>
</tr>
</table>
EOF
print end_html();
```

Current value of WatchDog timer (sec): 0

Refresh time (sec)	<input type="text"/>
WatchDog timeout (sec)	0
Update time (sec)	<input type="text"/>

```
#!/usr/bin/perl -w

#use lib "";
use TeleServ;
use CGI qw(:standard);

sub isInt {
    my $str = shift;
    return ($str =~ /^(\d+)?$/);
}

$device_name = "/dev/ttyS1";

tsOpen($device_name);
tsSetTerminalMode;
$version = tsVersion();

%val = ();
%cookie = cookie('watchdog');
@true_keys = qw(update wd wdtick wdtickfirst);

#Check cookie
foreach (@true_keys) {
    $val{$_} = isInt($cookie{$_});
}

if (param()) {
    my $tmp;
    if (param('start')) {
        $val{update} = isInt(param('update'));
        $val{wd} = isInt(param('wd'));
        $val{wdtick} = 0 if ($val{wdtickfirst} = isInt(param('wdtick')));
    }
    #Stop refresh if button "stop" pressed or invalid values are entered
    if (param('stop') || ($val{update} eq "") || ($val{wd} eq "") || ($val{wdtickfirst} eq ""))
    {
        foreach (keys %val) {
            $val{$_} = "";
        }
        $watch_dog = tsWatchDog(0);
        goto EXIT;
    }
}
}
```

```
$val{wd} = 0 unless ($val{wd});

if ($val{wdtick} ne "") {
    $val{wdtick}--;
    if ($val{wdtick} < 0) {
        #Number of refresh cycles when WatchDog timeout should be uploaded
        $val{wdtick} = int($val{wdtickfirst}/$val{update});
        $watch_dog = tsWatchDog($val{wd});
        goto EXIT;
    }
}
$watch_dog = tsWatchDog;
EXIT:

$cookie = cookie( -name=>'watchdog',
    -value=>\%val );

print header(-cookie=>$cookie);

$html_name = "TeleServ $version";

if ($val{update}) {
    #java script that should refresh web page
    $java_script=<<JAVA;
    function load()
    {\nsetTimeout( "refresh()", $val{update}*1000 );\n}
    function refresh()
    {\nwindow.location.href = unescape(window.location.pathname);\n}
    JAVA
    print start_html(
        -title=>$html_name,
        -script=>$java_script,
        -onload=>'load()'
    );
} else {
    print start_html(
        -title=>$html_name
    );
}

print "Current value of WatchDog timer in ticks (sec): $watch_dog<br>";

print start_form();
print table({-border=>undef},
    Tr({-align=>CENTER},td(["Refresh time (sec)", textfield(-name=>'update', -default=>$val{update})])),
    Tr({-align=>CENTER},td(["WatchDog timeout (sec)", textfield(-name=>'wd', -default=>$val{wd})])),
    Tr({-align=>CENTER},td(["Update time (sec)", textfield(-name=>'wdtick', -default=>$val{wdtickfirst})]))
);
print submit(-name=>'start');
print submit(-name=>'stop');
print end_form();

print end_html();
```