1-Wire Automation Software

v1.2.0

Starter Guide

January 2016

Table of Contents

Revision History

Date	Authors	Description
2015-03-06	Peter S'heeren	Initial release.
2015-06-15	Peter S'heeren	Second release.
2016-01-07	Peter S'heeren	Third release.

1 Overview

This document presents a step-by-step guide for getting started with the 1-Wire Automation Software.

You're encouraged to read through this document first before consulting the various user manuals that come with the 1-Wire Automation Software.

2 Programs

The 1-Wire Automation Software comprises a set of programs that closely work together: server, GUI, logger, and control panel. With the exception of the control panel, these programs all run on Linux and Windows.

The server is the heart of the 1-Wire Automation Software. It manages your 1-Wire adapters and 1-Wire slaves, it can read sensors and control PIO pins, and interact with client programs. The server operates like most other Internet server software, it listens to a port and accepts incoming client connections locally and over the network.

If you're a Linux user, your operating system is one of many distributions and versions that are available. Although the software is available for many platforms, your operating system may or may not be supported. You should definitely install and run the software to make sure it works on your Linux system.

The GUI program offers, as its name suggests, a graphical user interface for working interactively with the server. The GUI is a client of the server. It visualizes the state of the 1-Wire adapters and 1-Wire slaves, and allows you to read sensor data and manipulate PIO pins. The GUI provides many buttons for executing client commands, like enumerating 1-Wire slaves. Thanks to these buttons you're saved from typing lots of client commands in a terminal program like PuTTY or netcat.

The logger is another client of the server. Based on a configuration file, it reads sensor data from various 1-Wire slaves and stores the data in a database or text file.

The control program is specifically created for Windows users. This program offers a convenient way to start and stop the server and logger. More specifically, it offers an easy and interactive way to install and run the server and logger as Windows services.

With the 1-Wire Automation Software, you're not confined to a single computer. You can deploy the software on multiple computers that are connected to a local network or even to the Internet. For example, you can run the server on a BeagleBone Black, run the logger on a Linux-based database server, and run the GUI program on a Windows PC to inspect the server.

3 Linux

Overview

This document describes commands entered at the command prompt in Linux. When entering a command in the context of your personal login, the document uses the $ notation. For example:

```
$ ./owsasgui &
```

A command entered in the context of the root login is written with the # notation. For example:

```
# ./owsas -v
```

The current directory isn't shown. Instead, the text indicates where the current directory is supposed to be.

Picking the Right Package

The Linux version of the 1-Wire Automation Software is distributed as a tar-gzip archive file, called a package. The filename of a package is structured as follows:

owsas-version**-**architecture**.tar.gz**

version	The version number of the product, formatted as a major-minor-micro triplet. For example: **1.2.0**
architecture	A name that indicates the architecture for which the package is meant.

The filename always ends in the **.tar.gz** extension.

Linux is available on a variety of so-called architectures. The architecture is about the environment programs run in. It involves the instruction set architecture (ISA) of the processor, the Application Binary Interface (ABI) for calling the Linux kernel and libraries, use or emulation of floating point unit (FPU) instructions. An ISA can have many versions that are upwards compatible, like the Intel x86 family of processors for example.

The 1-Wire Automation Software runs on several architectures. There's a package for each supported architecture. The architecture portion of the filename marks the target architecture:

armel	ARM processor, 32-bit, Little Endian, FPU emulated in software (soft-float).
armhf	ARM processor, 32-bit, Little Endian, FPU present (hard-float).
i486	Intel 80486 or later processor, 32-bit.
amd64	AMD64 and Intel 64 instruction set, 64-bit.

Other architectures may be added in the future.

This list may seem limited, nevertheless it covers a wide range of Linux-based systems including SBCs, PCs and embedded systems.

The **armel** and **armhf** architecture are very commonplace nowadays. Many single board computers (SBCs) and other Linux devices are equipped with an ARM-based application

processor. Popular SBCs include BeagleBone Black, Banana Pi, Gnublin, Raspberry Pi.

Probably the most difficult part is to determine whether the **armel** or **armhf** package must be used. This really depends on your Linux distribution. Your distribution may explicitly state the architecture. In many cases, architecture **armel** is advertised as "soft float", while **armhf** is advertised as "hard float". If this doesn't help, use the method using **gcc** as explained later.

The **i486** and **amd64** architectures are typically used on PCs. Some SBCs may also incorporate an x86 or compatible processor.

If you've difficulty determining the architecture of your particular Linux installation, you can run the following command:

```
$ gcc -dumpmachine
```

Note that **gcc** must have been installed for this to work.

The outcome of this command is the compiler's "target machine" a.k.a. "GNU triplet" which is key in determining the architecture of your Linux system. Here's a list of commonly produced results and the corresponding architecture name:

arm-linux-gnueabi	**armel**
arm-linux-gnueabihf	**armhf**
i486-linux-gnu **i586-linux-gnu** **i686-linux-gnu**	**i486**
x86_64-linux-gnu	**amd64**

Besides the architecture, the version of the Linux kernel and installed libraries plays a role in the successful deployment of a package. Usually the versions are upwards compatible. That's why the packages are built on Linux distributions aged two or more years old, so you don't have to install the latest Linux version in order to run the software.

It may be possible that your particular architecture isn't supported. If so, you may contact us, we may be able to create a package for your Linux system.

Installing the Software

Once you've picked the right package for your Linux system, you're ready to install the software. For the sake of simplicity, let's assume you're using package **owsas-1.2.0-armhf.tar.gz**.

Be sure you're logged in using your regular account. This may or may not be **root**. Download or copy the package to a local directory.

First you've to extract the package. If you're using a command line shell, run the following command:

```
$ tar -xzp -f owsas-1.2.0-armhf.tar.gz
```

The package will be extracted to a separate directory called **owsas-1.1.0-armhf**. If your working in a graphical desktop environment, you probably can extract the package by right-clicking on the file and choosing "extract" or similar.

Now it's time to install the software. This must be done from the command line shell. Change the current directory to the one where the package was extracted to:

```
$ cd owsas-1.1.0-armhf/
```

Run the installation script:

```
$ ./install.sh
```

If you're not logged in as the **root** user, the script will ask for a **sudo** password in order to run a portion of the installation with root permissions. Enter the password accordingly to bring the installation to a successful end.

Do not attempt to run **install.sh** from the graphical desktop environment. The reason is the installation script asks for a **sudo** password; this only works in a command line shell.

The script installs the following components:

- The programs and other files are placed in a directory structure with **/opt/owsas/** as the base directory.
- A desktop icon file is added to directory **Desktop/** of the user account. This only applies when a graphical desktop environment exists on your Linux installation.

Running the Software

There are a number of programs you can run: the server, the logger, and the GUI program. The programs must be started from the command line shell. The GUI program can also be started by clicking the icon on the desktop.

It's recommended to run the server with root permissions. The server has built-in drivers for 1-Wire adapters and most drivers require root permissions to access the hardware.

Log in as the **root** user (or use the **su** command) and change to the binary directory of the 1-Wire Automation Software:

```
# cd /opt/owsas/bin/
```

Run the 1-Wire server with verbose output enabled:

```
# ./owsas -v
```

Since verbose output is enabled, the server prints text describing what's going on.

The server first loads the default configuration file. This file is called **owsas.cfg**. It's located in the binary directory (that's the same directory as the server).

The configuration file refers to a command file. This file is called **owsas-startup-cmds.txt**. It's also located in the binary directory.

Press CTRL-C to quit the server. The server detects CTRL-C and exits gracefully.

4 Windows

Overview

This document describes commands entered at the command prompt in Windows with the > notation. For example:

```
> owsas.exe -v -console
```

The current directory isn't shown. Instead, the text indicates where the current directory is supposed to be.

Installing the Software

The Windows version of the 1-Wire Automation Software is packaged as a standalone installer. The filename of the installer is structured as follows:

owsas-version**-win32-install.exe**

version	The version number of the product, formatted as a major-minor-micro triplet. For example: **1.2.0**

The installer is an executable file. The filename always ends in the **.exe** extension.

Run the installer and follow the steps until installation is done. The following components will be installed:

- The programs and other files are placed in a directory structure that's based in the destination directory. You can choose the destination directory during the "Choose Users" and "Choose Install Location" steps.
- Icons for the control panel and GUI program are placed on your desktop.
- A folder is added to your start menu. You can control this in during the "Choose Start Menu Folder" step.

The programs are placed in directory **bin/** inside the destination directory. This location is called the binary directory.

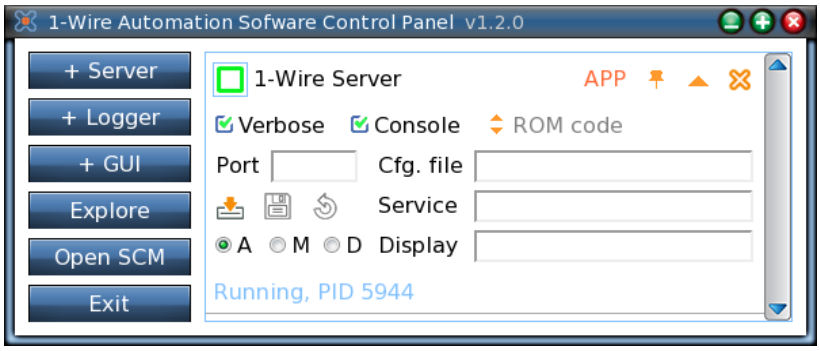
Running the Software

There are a number of programs you can run: the control panel, the server, the logger, and the GUI program.

First start the control panel by double-clicking the corresponding icon on the desktop. You can move and resize the control panel as you wish. The program will save the window settings when you click "Exit" so the window's position and size will be the same next time you start the program.

The control panel adds an icon to the tray of your desktop. You can click this icon to show and hide the control panel. As long as the tray icon is visible, the control panel is running. The only way to completely shut down the program is clicking "Exit".

In the control panel, click "+ Server". A panel is added. Change "Verbose" and "Console" to the checked state. Click the red arrow to start the server. The server appears as a console window.



Since verbose output is enabled, the server prints text describing what's going on.

The server first loads the default configuration file. This file is called **owsas.cfg**. It's located in the binary directory (that's the same directory as the server).

The configuration file refers to a command file. This file is called **owsas-startup-cmds.txt**. It's also located in the binary directory.

Click the green square in the control panel to stop the server. You can also press CTRL-C or CTRL-BREAK in the console window to quit the server. In either case the server exits gracefully.

The control panel supports the logger. Click "+ Logger" to add a panel. This is analogous to running the server.

Click "Explore" to open an explorer window for the binary directory. You usually click this button when you want to edit the configuration and command files. It's also a convenient way to get to know where the binary directory is located.

You can start the various programs from the command line. Open the command line interpreter (cmd.exe) and change the current directory to the binary directory.

For example, this command starts the 1-Wire server in a console window with verbose output enabled:

```
> owsas.exe -v -console
```

5 Configuring the Server

Overview

As explained earlier, when the server starts up, it loads a configuration file and optionally a command file.

You can use the default configuration file or use your own configuration file by specifying the **-cfg** command line parameter. The latter option is useful if you're planning to run multiple instances of the server.

The configuration file contains static settings, like the server port number, allowed IP addresses, user accounts, the command file to process.

The command file contains client commands that are executed when the server starts up. Typically, these client commands add and enable your 1-Wire adapters and tell the server how your 1-Wire networks are set up.

Note that the client commands in the command file are executed no different from client commands that are issued over a network connection. This means you don't necessarily have to put commands in the command file or even use a command file at all. You can issue these commands later when the server has been started up. The command file is a convenience, not a necessity.

The installer puts default configuration and command files in the binary directory. These files contain comments for the most part.

Setting Up the Server Connection

By default, the server is configured to listen at port 5001. You can change the server port in the configuration file.

The server offers a number of additional settings for restricting access:

- Max. number of incoming connections. This setting restricts the number of clients that may connect with the server.
- User accounts. A client must successfully authenticate with the server, else the server closes the connection.
- List of allowed client IP addresses. The server allows access based on the IP address of the client.

Here's an example configuration file:

Configuration File

```
port      5001;
allowip   localhost;
user      "johhny" "12az34qs";
maxconn   2;
cmdfile   "owsas-startup-cmds.txt";
```

This example configuration file restricts server access to two clients on the local system. The clients must authenticate with the server.

Consult the server's user manual for more information.

Setting Up 1-Wire Adapters

USB to 1-Wire Adapters

USB-based 1-Wire adapters are easy to set up. All you need to do is issue the **lu enable** client command to enable USB support in the server:

Command File

```
lu enable
```

Plug in the adapter and the server will discover it. The server automatically adds and enables the 1-Wire adapter.

Other 1-Wire Adapters

The server supports a host of 1-Wire adapters. You've to issue client command **Adapter Add** to tell the server how a 1-Wire adapter is connected to the computer. Once added, you've to tell the server to enable the adapter with client command **Adapter Enable**.

For example, suppose you're using a DS9097U adapter connected to the serial port on a Linux PC. The following commands set up your adapter:

Command File

```
adapter "ow" add serial "/dev/ttyS0" ds9097u
adapter "ow" enable
```

The same adapter connected to the first COM port on a Windows PC:

Command File

```
adapter "ow" add serial "\\.\COM1" ds9097u
adapter "ow" enable
```

Note the syntax of the serial path. Refer to chapter “Serial Paths” for more information on how to correctly format paths of serial ports.

Some adapters are susceptible to surprise-removal events. This means an adapter can disappear from the computer due to an unexpected disconnection from the computer. Network and USB are typical examples.

Even the DS9097U from our example may be subject to surprise-removal events. If the adapter is connected to a USB to serial adapter, the latter may be unplugged from the computer, taking down the DS9097U in the process.

It's possible to instruct the server to try to recover from a surprise-removal by attempting a reconnection periodically. For example, assuming the DS9097U is connected to a USB to serial adapter:

Command File

```
adapter "ow" add serial "/dev/ttyUSB0" ds9097u
adapter "ow" attr reconn=on,5000
adapter "ow" enable
```

The second command tells the server to attempt a reconnection every 5 seconds in case the adapter has been the subject of a surprise-removal.

The server supports many 1-Wire adapters. Consult the server's user manual for more

information on how to add your specific adapters.

Setting Up 1-Wire Slaves

Once you've set up your 1-Wire adapters, it's time to tell the server about your networks of 1-Wire slaves. There are various methods:

- Add and move.
- Add and detect.
- Load topology file and detect.
- Enumerate.

Whichever method or combination of methods you use, you should always give the server some time to enable your adapters. For most types of 1-Wire adapters, inserting a **Wait** client command that stalls for 2 seconds will do the job.

The choice of method depends on your situation. The speed of enumeration and detection varies a lot between types of adapters. Some adapters are very efficient with enumeration while other adapters enumerate very slowly. A similar difference in speed exists for detection. The best way to pick a method or a combination of methods that suits best is trying out the various possibilities yourself.

Add and Move

This method involves the following steps:

1. Issue client command **Dev Add** to add a 1-Wire slave. The slave is added to the non-present devices area (NPD for short). You can set attributes as well, like a description of your 1-Wire slave.
2. Issue client command **Dev Move** to relocate the slave to the correct channel or hub port. Be sure the target adapter has been enabled at this point.

For example, a DS9097U adapter is wired to a small 1-Wire network with two slaves:

Command File

```
# Add 1-Wire slaves to the NPD
dev "36-000003612CDB" add
dev "3B-00000019CCEF" add ds1825
dev "3B-00000019CCEF" attr descr="Kitchen temperature"

# Add and enable adapter
adapter "ow" add serial "/dev/ttyS0" ds9097u
adapter "ow" enable
wait 2000

# Move 1-Wire slaves to the adapter
dev "36-000003612CDB" move ch "ow":1:1
dev "3B-00000019CCEF" move ch "ow":1:1
```

This command file uses adapter names. If you happen to work with a USB-based adapter, the server picks the adapter name. In that case, you can refer to the adapter number:

Command File

```
# Add 1-Wire slaves to the NPD
dev "36-000003612CDB" add
dev "3B-00000019CCEF" add ds1825
dev "3B-00000019CCEF" attr descr="Kitchen temperature"

# Discover and enable USB adapter
lu enable
wait 2000

# Move 1-Wire slaves to the adapter
dev "36-000003612CDB" move ch 1:1:1
dev "3B-00000019CCEF" move ch 1:1:1
```

Add and Detect

This method work as follows: add your 1-Wire slaves to the NPD area, move them to one or more unallocated channels (UCH for short), and instruct the server to figure out which UCH corresponds with which channel.

For example, a DS9097U adapter is wired to a 1-Wire network containing two slaves:

Command File

```
# Add 1-Wire slaves to the NPD
dev "36-000003612CDB" add
dev "3B-00000019CCEF" add ds1825
dev "3B-00000019CCEF" attr descr="Kitchen temperature"

# Move 1-Wire slaves to a UCH
uch add
dev "36-000003612CDB" move uch 1
dev "3B-00000019CCEF" move uch 1

# Add and enable adapter, detect
adapter "ow" add serial "/dev/ttyS0" ds9097u
adapter "ow" enable
wait 2000
adapter "ow" detect
```

You can speed up this command file by enabling automatic detection on the adapter:

Command File

```
# Add 1-Wire slaves to the NPD
dev "36-000003612CDB" add
dev "3B-00000019CCEF" add ds1825
dev "3B-00000019CCEF" attr descr="Kitchen temperature"

# Move 1-Wire slaves to a UCH
uch add
dev "36-000003612CDB" move uch 1
dev "3B-00000019CCEF" move uch 1

# Add and enable adapter, detect
adapter "ow" add serial "/dev/ttyS0" ds9097u
adapter "ow" attr detect=on
adapter "ow" enable
```

As soon as the 1-Wire adapter is enabled, the detection procedure will kick in. There's no need for a waiting period in this case.

If you want to apply this method with a USB-based adapter, go ahead as follows:

Command File

```
# Add 1-Wire slaves to the NPD
dev "36-000003612CDB" add
dev "3B-00000019CCEF" add ds1825
dev "3B-00000019CCEF" attr descr="Kitchen temperature"

# Move 1-Wire slaves to a UCH
uch add
dev "36-000003612CDB" move uch 1
dev "3B-00000019CCEF" move uch 1

# Discover and enable USB adapters, turn on automatic detection
hw attr detect=on
lu enable
```

The command file will turn on automatic detection for all adapters, including the USB-based ones.

Load Topology File and Detect

A topology file contains information about 1-Wire networks and their 1-Wire slaves.

You can save the current topology of a running server by issuing a **Topo Save** client command from a terminal program like PuTTY or netcat:

Client Terminal Program

```
topo save "mytopology.txt"
```

Note that the topology file is saved on the server side, not on the client side.

In the command file, you can load one or more topology files and run the detection procedure, as in this example:

Command File

```
topo load "mytopology.txt"

# Add and enable various adapters, detect
hw attr detect=on
adapter "ow" add serial "/dev/ttyS0" ds9097u
adapter "ow" enable
lu enable
```

Enumerate

When the server is starting up, you can initiate enumeration of one or more channels or even all adapters. For example:

Command File

```
# Add and enable various adapters, detect
adapter "ow" add serial "/dev/ttyS0" ds9097u
adapter "ow" enable
lu enable
wait 2000
hw enum
```

In the above example, enumeration is requested for all channels of all adapters, thus a full enumeration. Doing so renders an up-to-date topology every time the server starts up. Be warned that a full enumeration may take a long time to complete depending on the type of adapters and the amount of slaves that are presents in your 1-Wire networks.

6 Talking with the Server

Client Connection

The 1-Wire server creates a server socket that listens at a certain port. To communicate with the server, you've to create a client connection. You can create a connection locally (on the same machine) or remotely (over the network) using a client program.

The software comes with a GUI and logger which are client programs that are designed specifically for use with the 1-Wire server.

A terminal program allows you to type in text to be sent to the server while receiving text from the server. Many terminal programs exist. PuTTY and netcat are such examples.

The server maintains information for each client connection. This information is called the client context. It includes:

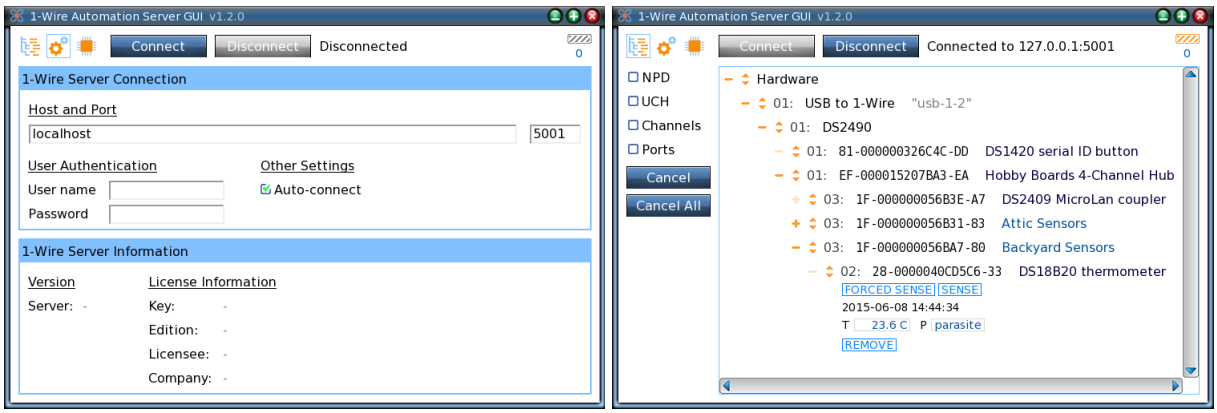
- Active commands. A client can send multiple commands. Each command may take a while to complete its task. A client has the possibility to cancel active commands.
- Temperature scale used when the server sends back temperature sensor data.
- ROM code formatting style used in client responses.
- Unsolicited responses. A client can turn on and off reporting of certain client responses in the server.

It's important to understand that this context exists on a per-connection basis, and changes made in one client context do not influence other client connections. So for example, you can have one client connection that receives temperature values in Fahrenheit and another client connection that receives temperature values in Celsius.

1-Wire GUI

This program is a very useful tool for inspecting the 1-Wire topology, setting up your 1-Wire network, reading sensor data, controlling PIOs. It runs on Linux and Windows operating systems and can connect to the server locally as well as remotely.

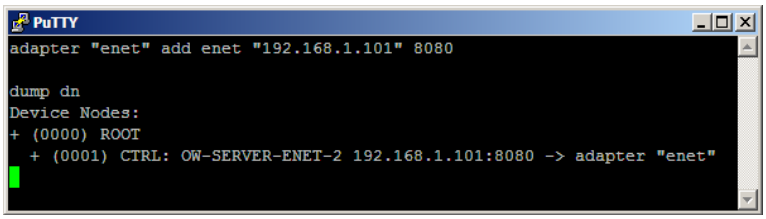
After you've installed the 1-Wire Automation Software, the program is configured for a local connection on port 5001. You can adjust these settings in the connection page (picture on the left).

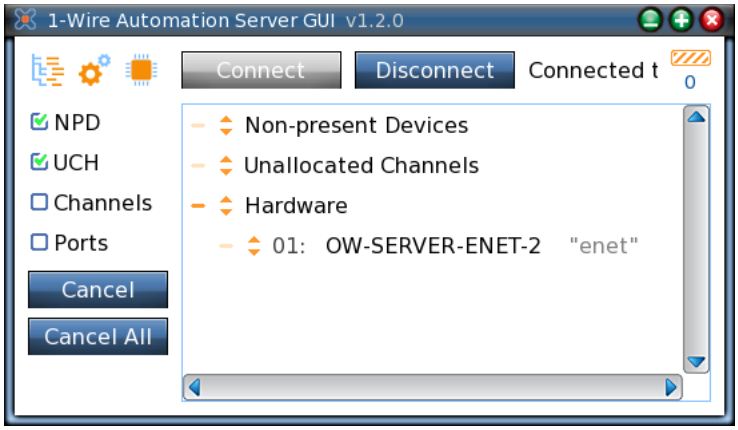


Hit the "Connect" button to establish the client connection with the server. If successful,

the program updates the contents of the "1-Wire Server Information" box. Switch to the topology page to inspect the server's 1-Wire topology (picture on the right).

The GUI program is a very convenient tool for monitoring the outcome of commands you unleash in PuTTY, netcat, or other terminal program.



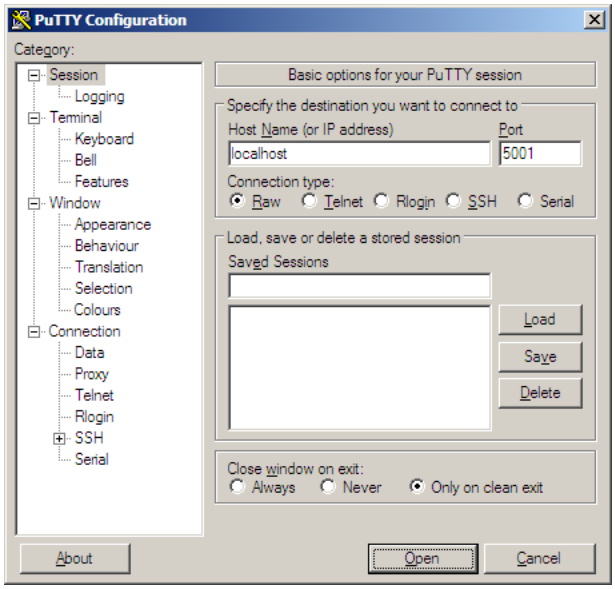


There's a good chance you'll find yourself using your terminal program and the GUI side by side. For example, command **Adapter Add** can't be issued from the GUI, so you want to enter the command in your terminal program and watch the outcome popping up in the GUI. The first picture to the right shows the manual input of an **Adapter Add** command in PuTTY. The second picture shows the result in the GUI: an adapter node has been added. From here you can use the GUI to enable the adapter, enumerate the 1-Wire slaves, and do other stuff. Note that the **Dump Device Nodes** command entered in PuTTY confirms the successful completion of the former command.

You can run multiple instances of the GUI, for example, when you want to connect from multiple machines to the same server, or when you want to inspect multiple servers from the same machine.

When you run multiple GUIs on the same machine, you're expected to start each instance with its own configuration file. Use command line parameter **-cfg** to provide a unique configuration file for each instance of the program. You can create dedicated shortcuts on your desktop, or run from the command line. The unique configuration file doesn't need to exist; when the GUI is run the first time, it'll create the configuration file when you close the program.

PuTTY



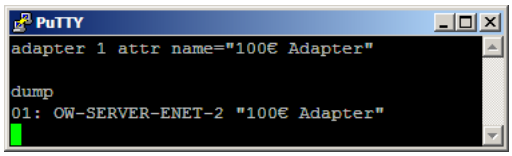
PuTTY is available for Linux and Windows. It's a widely used terminal program that interacts very well with the 1-Wire server.

The picture to the left shows how to set up a connection with the server using PuTTY's configuration window. This window pops up when you start PuTTY. The picture shows the Windows version, the Linux version is very similar.

The host name and port fields are the same as with the GUI program. Make sure the connection type is set to "Raw".

Click the "Open" button to establish the client connection. If successful, PuTTY opens up the main terminal window where you can enter client commands and receive client responses.

Unlike the GUI program, PuTTY doesn't know about the 1-Wire server. If you've enabled authentication in your server, then the very first thing you must send is the **Authentication** client command. If this step fails, the server will shut down the connection.



Normally, you don't have to change any settings in PuTTY, though it's a good idea to set the character set to UTF-8 in category "Window" → "Translation" → "Character set translation on received data". Doing so enables PuTTY to send and receive non-ASCIIZ characters that may occur in descriptive string like the name of a 1-Wire adapter.

netcat

Program netcat is available for Linux. It is designed for reading data from and writing data to network connections. The program can operate as a server or a client. For our purposes, we'll use netcat in its capacity of a network client.

Netcat needs a host name and port in order to connect to the server. If authentication is enabled, then the very first thing you must send is the **Authentication** client command.

Netcat transfers data over standard input and output. You can use netcat as a simple terminal program:

```
$ nc localhost 5001
dev "28-0000040CBBB2-C4" sense force
dev "28-0000040CBBB2" sensed ds18b20 "2014-11-12 02:44:47 0132 +019.1 C
parasite"
```

You can redirect data from and to files and other programs. For example:

```
$ echo 'dev "29-11BD2A" pio off 1' | nc localhost 5001
```

This command clears PIO pin 1 on a DS2408. There's no response. Netcat closes the connection right after sending the client command to the server.

Using netcat this way can be tricky when sending client commands like **Device Sense** that generate a response. We want to keep the connection between netcat and the server open for the duration of the command, and close the connection after the server has returned the response, ofcourse without loosing the response. This requires some fiddling on both sides of the connection. Here's how you do it:

```
$ echo 'dev "26-16B4948" sense force rsp=temp ; block ; close' | nc -q 10
localhost 5001
19.4
```

Client commands **Block** and **Close** instruct the server to close the client connection right after the **Device Sense** client command has completed and the response has been sent back. Parameter **-q 10** tells netcat to wait for at least 10 seconds before closing the client connection. This should give the server enough time to execute the command. If not, you can easily increase the value.

Reading Sensor Data

Client command **Device Sense** reads sensor data from a 1-Wire slave. For example, reading a DS18B20 temperature sensor from a terminal program:

```
dev "28-0000040CBBB2-C4" sense force
dev "28-0000040CBBB2" sensed ds18b20 "2014-11-12 02:44:47 0132 +019.1 C
parasite"
```

Date and time of accessing the 1-Wire slave and all acquired sensor data are encapsulated in the sensor data string.

You can ask the server to report a single sensor value. This is called a short response. For example, let's read the temperature of a DS2438 from a terminal program:

```
dev "26-16B4948" sense force rsp=temp
19.4
```

Note that, although a single value is returned, the server reads all sensor data from the 1-Wire slave.

Keyword "force" instructs the server to access the 1-Wire slave. If this keyword is omitted, the server will return a copy of the most recent sensor data, unless no data is present in which case the server does access the 1-Wire slave as if "force" were specified.

You can read multiple short responses from the same sensor data. Here's an example:

```
dev "26-16B4948" sense force rsp=temp
19.4
dev "26-16B4948" sense rsp=vad
2.34
dev "26-16B4948" sense rsp=vdd
4.98
dev "26-16B4948" sense rsp=cur
-2
dev "26-16B4948" sense rsp=sensedt all
2015-04-25 01:58:42
```

The first **Device Sense** client command embeds the "force" keyword, the subsequent commands don't. As a result, the first command accesses the 1-Wire slave to read and store the sensor data, while each subsequent command returns a value from the stored sensor data.

Client command **Device Sense** may fail in which case you get an error response instead of sensor values. Consult the 1-Wire server user manual for more information.

Writing PIO Pins

Client command **Device PIO** writes to the output state of one or more PIO pins. Basically, you specify which PIO pins must be set (a.k.a. turned on, written one) and which PIO pins must be cleared (a.k.a. turned off, written zero). PIO pins are numbered from 1 onwards.

```
dev "29-11BD2A" pio on 4 5 clear 110b
```

This example client command sets the 4th and 5th PIO pin, and clears the 2nd and 3rd PIO pin.

Command **Device PIO** accepts any combination of the following arguments:

- "on" followed by one or more PIO pin numbers (1..) that will be set.
- "off" followed by one or more PIO pin numbers (1..) that will be cleared.
- "set" followed by a bit-mask value denoting PIO pins that will be set.
- "clear" followed by a bit-mask value denoting PIO pins that will be cleared.

All arguments are accumulated into two pin masks, a mask for setting PIO pins, and a mask for clearing PIO pins. If the two masks overlap for some pins, then these pins are set (setting takes precedence over clearing). Unspecified PIO pins remain untouched.

7 Migrating from owfs

If you're planning to migrate your owfs-based projects to the 1-Wire Automation Software, this section provides some useful pointers to start with.

ROM Code Formatting Style

The 1-Wire server understands the ROM code formatting style employed by owfs. This means that client commands …

```
dev "26-16B4948" sense force
```

… and …

```
dev "26.48496B010000" sense force
```

… are the same.

When the server formats a client response, it formats ROM codes in native style. For example:

```
dev "28.B2BB0C040000" sense force
dev "28-0000040CBBB2" sensed ds18b20 "2014-11-12 02:44:47 0132 +019.1 C
parasite"
```

You can change the formatting style to owfs. As a result, the previous example becomes:

```
dev "28.B2BB0C040000" sense force
dev "28.B2BB0C040000" sensed ds18b20 "2014-11-12 02:44:47 0132 +019.1 C
parasite"
```

You can enable the owfs formatting style in different places:

- Run the server with command line argument **-romcode owfs**. This sets the default formatting style for all client connections.
- Specify "romcode owfs;" in the server's configuration file. This sets the default formatting style for all client connections.
- Send the following client command to change the formatting style for the active client connection:

  ```
  attr romcode=owfs
  ```

The 1-Wire GUI can be set to display owfs ROM codes.

The 1-Wire logger has options for formatting ROM code variables in owfs style.

Redirection

Since owfs exposes the 1-Wire hardware as a file system, writing to and reading from files is the standard method of interacting with the owfs software. This means redirection is often used in shell commands and scripts.

For example, the following command writes zero to the 1st PIO pin of a DS2408 (note that owfs inverts all values written to PIO channels):

```
# echo "1" > /mnt/onewire/29.2ABD11000000/PIO.0
```

Here's the equivalent command for use with the 1-Wire server:

```
$ echo 'dev "29.2ABD11000000" pio off 1' | nc localhost 5001
```

Many files in the owfs file system output a value. For example, let's read the sensed state of the 1st PIO pin of a DS2408:

```
# cat /mnt/onewire/29.2ABD11000000/sensed.0
```

Similarly, you can read a value using the 1-Wire server:

```
$ echo 'dev "29.2ABD11000000" sense force rsp=piosensed ; block ; close' |
nc -q 10 localhost 5001
```

Here's another example. It shows how to read multiple values from a DS2438 using the 1-Wire server:

```
$ echo 'dev "26.48496B010000" sense force rsp=temp ; block ; close' | nc
-q 10 localhost 5001
19.4
$ echo 'dev "26.48496B010000" sense rsp=vad ; block ; close' | nc -q 10
localhost 5001
2.34
$ echo 'dev "26.48496B010000" sense rsp=vdd ; block ; close' | nc -q 10
localhost 5001
4.98
$ echo 'dev "26.48496B010000" sense rsp=cur ; block ; close' | nc -q 10
localhost 5001
-2
$ echo 'dev "26.48496B010000" sense rsp=sensedt all ; block ; close' | nc
-q 10 localhost 5001
2015-04-25 01:58:42
```

8 Serial Paths

You need to specify a serial path in order to communicate with a serial port. The next sections explain in more detail how you specify serial paths in each supported operating system.

Linux

Serial ports are accessible in the device directory structure. A serial path starts with **/dev**. A serial path is case-sensitive.

The following table summarizes serial paths that are commonly found on Linux systems:

Serial Path	Serial Port
/dev/ttyS0	The computer's 1st on-board serial port
/dev/ttyS1	The computer's 2nd on-board serial port
/dev/ttyS2	The computer's 3rd on-board serial port
/dev/ttyS3	The computer's 4th on-board serial port
/dev/ttyUSB0	1st USB serial adapter
/dev/ttyUSB1	2nd USB serial adapter
/dev/serial/by-id/	This directory contains symbolic links to serial devices. Each symbolic link name identifies a specific device.
/dev/serial/by-path/	This directory contains symbolic links to serial devices. Each symbolic link name represents a hardware path to the device, like a specific USB port on your computer.

Here are some useful commands you can run to get information about present serial devices and their corresponding serial path. The following commands were run on a Linux system with one on-board UART and one connected USB serial adapter.

Filter information from the kernel message buffer:

```
$ dmesg | grep 'tty'
[    0.000000] console [tty0] enabled
[    0.516785] serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
[    0.517463] 00:0b: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
[    0.559097] tty tty55: hash matches
[   66.076326] usb 2-1: FTDI USB Serial Device converter now attached to ttyUSB0
```

Use the **setserial** command to produce a list of serial paths:

```
$ setserial -g /dev/ttyS* /dev/ttyUSB*
/dev/ttyS0, UART: 16550A, Port: 0x03f8, IRQ: 4
/dev/ttyS1, UART: unknown, Port: 0x02f8, IRQ: 3
/dev/ttyS2, UART: unknown, Port: 0x03e8, IRQ: 4
/dev/ttyS3, UART: unknown, Port: 0x02e8, IRQ: 3
/dev/ttyUSB0, UART: unknown, Port: 0x0000, IRQ: 0, Flags: low_latency
```

List serial devices in the device directory:

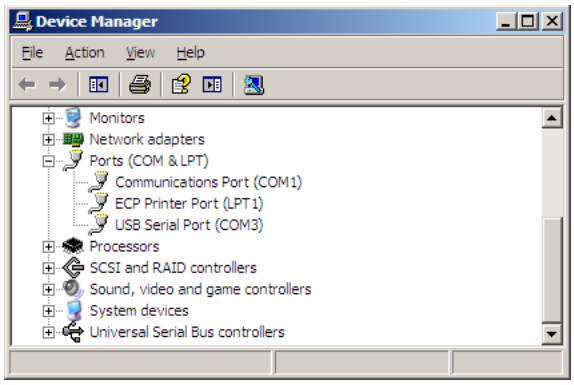
```
$ ls -l /dev/ttyS* /dev/ttyUSB*
crw-rw---- 1 root dialout   4, 64 2012-02-22 14:19 /dev/ttyS0
crw-rw---- 1 root dialout   4, 65 2012-02-22 14:19 /dev/ttyS1
crw-rw---- 1 root dialout   4, 66 2012-02-22 14:19 /dev/ttyS2
crw-rw---- 1 root dialout   4, 67 2012-02-22 14:19 /dev/ttyS3
crw-rw---- 1 root dialout 188,  0 2012-02-22 14:20 /dev/ttyUSB0
```

Windows

Serial ports are accessible through the Win32 device namespace, which is part of the NT namespace. As such a serial path starts with **\\.** followed by the device name of the serial port. A serial path is case-insensitive.

A serial port is typically named **COM<x>** where **<x>** is a number between 1 and 256. Other naming schemes may apply and aliases may exist, depending on the serial driver that controls the serial port.

You can obtain a list of available serial ports in the device manager. Open the device manager and view devices by type. The section named *Ports (COM & LPT)* contains all available serial and parallel ports.



The device name of a serial port is shown between parentheses. In the picture to the right, you can see two serial ports. COM1 is the PC's on-board serial port, COM3 represents a USB serial adapter.

The serial paths to the serial ports in the picture are:

Serial Path	Serial Port
\\.\COM1	*Communications Port (COM1)*
\\.\COM3	*USB Serial Port (COM3)*

Serial paths like **COM1** (that's without the **\\.** prefix) will work because COM1 to COM9 are reserved names in the NT namespace. COM10 to COM256 aren't reserved names and you'll have to specify the **\\.** prefix with these device names. By comparison, serial path **\\.\COM100** will work but serial path **COM100** won't work.

9 Legal Information

Disclaimer

Axiris products are not designed, authorized or warranted to be suitable for use in space, nautical, space, military, medical, life-critical or safety-critical devices or equipment.

Axiris products are not designed, authorized or warranted to be suitable for use in applications where failure or malfunction of an Axiris product can result in personal injury, death, property damage or environmental damage.

Axiris accepts no liability for inclusion or use of Axiris products in such applications and such inclusion or use is at the customer's own risk. Should the customer use Axiris products for such application, the customer shall indemnify and hold Axiris harmless against all claims and damages.

Trademarks

"Maxim Integrated" is a trademark of Maxim Integrated Products, Inc.

"1-Wire" and "iButton" are registered trademarks of Maxim Integrated Products, Inc.

"Raspberry Pi" is a trademark of the Raspberry Pi Foundation.

All product names, brands, and trademarks mentioned in this document are the property of their respective owners.

10 Contact Information

Official website: http://www.axiris.eu/