1-Wire Automation Server Logger

v1.2.0

User Manual

January 2016

Table of Contents

Revision History

Date	Authors	Description
2015-03-06	Peter S'heeren	Initial release.
2015-06-15	Peter S'heeren	Added section ROM Code Variable. Second release.
2016-01-07	Peter S'heeren	Added sections about SQLite. Third release.

1 Overview

The 1-Wire Automation Server Logger, called the logger throughout this document, is a powerful software tool for automating data acquisition and data logging in 1-Wire-based projects.

The logger is a client of the 1-Wire Automation Server, called the 1-Wire server throughout this document.

The logger can connect with multiple 1-Wire servers and multiple databases at once, and generate multiple text files at the same time, offering great flexibility in your logging strategy.

When a database server is temporarily unreachable, the logger buffers all sensor data in memory. Once the database is up and running again, the buffered data is committed. As a result no data is lost during periods when a database is offline.

2 Third-party Resources

MySQL

If you want to log to a MySQL database, the logger must load the MySQL library. You can download the software from the MySQL website.

MySQL home page: http://www.mysql.com/

If you're running Linux, the package manager that comes with your Linux distribution usually includes the latest version of the MySQL library.

PostgreSQL

In order to log data to a PostgreSQL database, the logger must load the PostgreSQL library. The library can be downloaded from the PostgreSQL website.

PostgreSQL home page: http://www.postgresql.org/

If you're running Linux, the package manager that comes with your Linux distribution usually includes the latest version of the PostgreSQL library.

SQLite

In order to log data to an SQLite database, the logger must load the SQLite3 library. The logger uses version 3 of the library. The library can be downloaded from the SQLite website.

SQLite home page: http://www.sqlite.org/

If you're running Linux, the package manager that comes with your Linux distribution usually includes the latest version of the SQLite3 library.

3 Command Line

Parameter	Description
-service	Run the program as a service. For this to work, the program must be installed as a Windows service.
-console	Open a console. This parameter as no effect when the program is run as a service.
-v	Enable verbose output.
-h	Display help and exit.
-cfg FILE	Specify configuration file.

Parameters **-service** and **-console** are specific to the Windows version of the logger.

The **-cfg** parameter overrules the default configuration file. The given filename may include an absolute or relative path.

4 Running the Logger

Current Directory

When the logger starts up, it changes its current directory to the location where the logger's executable file resides. This is also the directory where the 1-Wire Automation Software installer puts the program files, unless of course you've moved the files to another directory.

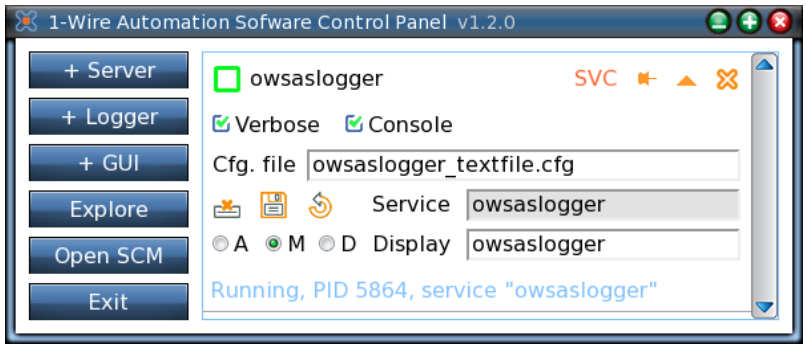
The current directory acts as the base directory for filenames that are specified with a relative path. This is true in the following places:

- Command line parameter **-cfg**.
- Configuration file keyword **textfile**.
- Configuration file keyword **library** in **mysql** block.
- Configuration file keyword **library** in **pgsql** block.
- Configuration file keyword **library** in **sqlite** block.
- Configuration file keyword **filename** in **textfile** block.

Windows

Control Panel

The control panel application offers a convenient way of configuring and running the logger (and server). The 1-Wire Automation Software installer creates a shortcut on your desktop. When the control panel is running, it shows an icon in the desktop's tray for quick access.



The graphical interface enables you to run the logger as an application or a Windows service. You can configure command line parameters, start and stop the logger as application or Windows service, install the logger as a Windows service, etc. Please read the 1-Wire Automation Software Control Panel user manual for more information.

Command Line

You can run the logger from the command line. In this environment, you run the logger as an application.

You can't run the logger as a service from the command line; specifying parameter **-service** won't work. Use the control panel application or the Windows Service Control Manager to start and stop services.

The following examples assume the current directory in the command line interpreter is

changed to the location of the logger executable file.

```
> owsaslogger.exe
```

This command runs the logger in the background, thus as an invisible program. The logger loads the default configuration file.

```
> owsaslogger.exe -console -v -cfg d:\myowsaslogger.cfg
```

This command runs the logger with a console and verbose output enabled. The specified configuration file is loaded.

```
> owsaslogger.exe -v > owsas.log
```

The logger is run in the background. The logger loads the default configuration file. Verbose output is written to the specified file.

```
> owsaslogger.exe -h
```

The logger displays the help screen and exits.

Stopping the Logger

In Windows, the logger can be stopped gracefully using the following methods:

- Stop the logger in the control panel application.
- If the logger has a console, press CTRL+C, press CTRL+BREAK, or click the console window's close button.

Linux

Required Privileges

The logger can be run with or without root privileges.

The 1-Wire Automation Software is installed as root user, hence all installed files are owned by the root user, including the provided configuration files. Most users want to use these configuration files, possibly modified, so the logger must have root privileges in order to be able to access these files.

As a rule of thumb, run the logger with root privileges.

Starting the Logger

The following examples assume the current directory in the shell is changed to the location of the logger executable file.

```
# owsaslogger -v -cfg /home/peter/myowsaslogger.cfg
```

Verbose output is enabled. The specified configuration file is loaded.

```
# owsaslogger -v > owsaslogger.log &
```

This command runs the logger in the background, detached from the console. The logger loads the default configuration file. Verbose output is written to the specified file.

```
# owsas -h
```

The logger displays the help screen and exits.

If you want to run the logger when Linux starts up, relying on **cron** is a good choice. You can edit the cron table as follows:

```
# crontab -e
```

Stopping the Logger

In Linux, the logger can be stopped gracefully using the following methods:

- Send signal SIGTERM to the logger.
- If the logger is attached to a console, press CTRL+C.

Use the kill program to send the SIGTERM signal. For example, if the process identifier of the logger is 18108, send SIGTERM from the shell:

```
# kill 18108
```

Use **killall** if you want to specify the program name rather than the process identifier:

```
# killall owsaslogger
```

Note that **killall** sends the SIGTERM signal to all processes with the specified name. If you're running multiple instances of **owsaslogger**, the **killall** command will terminate all of them.

5 Configuration File

Overview

When the logger starts up, it processes a configuration file. This file contains essential information for setting up the logging operation.

The logger treats the configuration file as read-only. The logger will never write a configuration file to disk.

The configuration file must use one of these character encoding schemes: UTF-8, UTF-16 Little Endian, or UTF-16 Big Endian. The logger recognizes the character encoding by means of the byte order mark (BOM) at the beginning of the file. If no byte order mark is present, the logger assumes the file is encoded as UTF-8.

Note that UTF-8 encoding is a super set of ASCII characters 0..127. This means you can create a configuration file with a simple ASCII editor.

Path

If command line parameter **-cfg** is specified, the given filename overrules the default configuration file. The filename may include an absolute or relative path. If the path is relative, the location of the logger's executable file acts as the base directory. The given filename must exist, else the logger stops with an error.

If command line parameter **-cfg** isn't specified, the logger assumes a default configuration file is located in the program directory. The default configuration file is called **owsaslogger.cfg**.

Contents

The configuration file is made up of statements. A statement begins with a keyword, followed by parameters.

There are two types of statements: simple statements and compound statements.

A simple statement ends with a semi-colon. For example:

```
port 3306;
```

A compound statement is parent to a so-called information block. An information block consists of statements enclosed by brace characters **{...}**. The compound statement ends where the information block closes. For example:

```
pgsql "mydb"
{
    username    "johnny";
    password    "ia45hl80";
}
```

Compound statements introduce a hierarchical structure of statements. Nested compound statements are possible. Statements that are not part of any information block reside at the top-level of the configuration file.

Note that a simple statement as well as a compound statement can span multiple lines.

When a statement begins with an unknown keyword, the logger skips the statement. The logger can skip simple statements and compound statements.

If a statement starts with a percentage character (**%**), the logger automatically skips the entire statement. This feature comes in handy, for example, if you want to disable the definition of a MySQL connection.

The hash character (**#**) starts a comment. This character can appear anywhere in a line. The comment spans all subsequent characters until the end-of-line marker. The logger filters out all comments.

The configuration file recognizes the following keywords at the top-level:

Keyword	Description
`text`	Definition of a text declaration.
`sensed`	Trigger for logging contents when a 1-Wire server reports sensor data from a 1-Wire slave.
`owsas`	Connection with 1-Wire Automation Server.
`mysql`	Connection with MySQL database.
`pgsql`	Connection with PostgreSQL database.
`sqlite`	Connection with SQLite database.
`textfile`	Text file for logging sensor data.

Logging Destination

The following destinations are available for logging:

- MySQL database.
- PostgreSQL database.
- SQLite database.
- Text file.

6 Text Composition

Overview

Various actions performed by the logger involve composition of text. For example, when the 1-Wire server reports sensor data of a DS18B20 and the sensor data is to be stored into a PostgreSQL database, then at some point the logger must compose an SQL statement for writing sensor values to the database.

Composing text is an important activity of the logger. A text declaration is defined in a top-level statement that starts with the **text** keyword. Each text declaration consists of a unique name and a set of textual elements. These elements include constant strings, character codes, and variables. Here's an example:

```
text "now" = "Current time and date: " 34 var(curdt) 34;
```

The example text declaration consists of a constant string, character 34 (double quote), a variable called **curdt** (current date and time), and another character 34. When this text declaration is composed, the result will be a line like this:

```
Current time and date: "2014-12-15 22:15:24"
```

The actual composition takes place during certain activities. Such activity is called the originator of the composition. The following originators exist:

Mnemonic	Activity
owsas-cmd	The logger sends an initial command to a 1-Wire server.
mysql-cmd	The logger sends an initial command to a MySQL database.
mysql-log	A logging action for a MySQL database is executed.
pgsql-cmd	The logger sends an initial command to a PostgreSQL database.
pgsql-log	A logging action for a PostgreSQL database is executed.
sqlite-cmd	The logger sends an initial command to an SQLite database.
sqlite-log	A logging action for an SQLite database is executed.
textfile-open	The logger creates or opens a text file.
textfile-cmd	The logger writes an initial command to a text file.
textfile-log	A logging action for a text file is executed.

The result of a composition is a Unicode text. The text can contain all Unicode characters, including control characters, but no zeroes. It's possible to generate a text that spans multiple lines. For example:

```
text "poll_ds28ea00" =
    "dev " 34 "42-00000038D0BE-A3" 34 " add" 10
    "dev " 34 "42-00000038D0BE-A3" 34 " attr poll=on,60000" 10;
```

The resulting text after composition will be:

```
dev "42-00000038D0BE-A3" add<EOL>
dev "42-00000038D0BE-A3" attr poll=on,60000<EOL>
```

… where **<EOL>** means character code 10.

Textual Element: Constant String

A constant string contains characters enclosed by double quotes. A string can't contain double quotes or control characters.

A constant string is a static textual element. It always renders the same Unicode characters when the containing text declaration is composed.

Textual Element: Character Code

There are a number of situations where you need to specify characters by their code rather than using a constant string:

- Double quote character (34): Double quotes enclose a constant string and as such can't appear inside a constant string.
- Control characters: These characters can't appear inside a constant string. Such character codes include carriage return (13) and line feed (10).
- You're editing the configuration file using an ASCII editor and you want to insert a Unicode character that's not part of ASCII, for example, the trade mark sign ™ (code 2122h).
- You're editor can't display a Unicode character. Usually you'll see a square instead. This is a good opportunity for using a literal code instead.

Valid character codes are 0001h..D7FFh and E000h..10FFFFh.

A character code is a static textual element. It always renders the same Unicode character when the containing text declaration is composed.

Textual Element: Variable

A variable is, as it's name suggests, a dynamic textual element. The formatting of the text depends on:

- The value of the variable.
- Properties of the variable. For example: variable **reg tchot hex lz** is formatted using hexadecimal digits and leading zeroes.
- The originator of the composition. For example: variable **curdt** is formatted using underscores when the originator is **textfile-open**.
- For sensor data, the sensor identifier of the 1-Wire slave.
- The temperature scale of the logging destination.

Not all variables are available for all originators. For example, variable **temp** is only available when the originator logs sensor data and the sensor data contains a temperature value.

If during composition of a text declaration an unavailable variable is encountered, the composition is aborted and the originator won't complete its task. Instead the logger prints an error message (if verbose printing is enabled).

Note that the logger doesn't check for unavailable variables in text declarations when it reads the configuration file. Doing so would require a complex checking procedure.

Temperature Variables

A number of variables denote a temperature: **temp**, **tchot**, **tccold**. The temperature is formatted as a decimal value. The logger uses the temperature scale of the logging destination, unless a temperature scale has been explicitly specified. Labels **celsius**, **fahrenheit** and **kelvin** denote temperate scales.

Examples:

```
var(temp celsius)
var(tchot fahrenheit)
var(tccold kelvin)
```

PIO State Variables

There are three variables that deal with the state of PIO pins: **piosensed**, **pioset**, **pioact**. You must specify whether the variable applies to all pins or a specific pin:

- **all**: the variable applies to all PIO pins.
- **1**, **2**, **3**,…: the variable applies to this PIO pin.

Examples:

```
var(piosensed 4)
var(pioset 5)
var(pioact all)
```

Register Variables

Register variables capture the state of registers in 1-Wire slaves. A register always is 16-bit or 32-bit wide. You can specify the radix of the formatted value and choose padding with leading zeroes. Labels **bin**, **dec** and **hex** select a radix of 2, 10, and 16 resp. Label **lz** enables leading zeroes. The default setting is radix 16 without leading zeroes.

```
var(reg temp)
var(reg temp bin lz)
var(reg temp dec)
var(reg temp lz hex)
```

ROM Code Variable

Variable **romcode** stores all elements of a 1-Wire slave's ROM code: the family code (8-bit), the serial number (48-bit), and the CRC value (8-bit). The variable has one property, the formatting style. If not specified, the logger applies formatting style **compact**.

The following table summarizes the available formatting styles and the resulting Unicode text for ROM code "28-0000040CD5C6-33":

Variable	Unicode Text
`var(romcode)` `var(romcode compact)`	`280000040CD5C6`
`var(romcode family)`	`28`
`var(romcode serial)`	`0000040CD5C6`
`var(romcode serialinverse)`	`C6D50C040000`
`var(romcode crc)`	`33`
`var(romcode native)`	`28-0000040CD5C6`
`var(romcode owfs)`	`28.C6D50C040000`

Variable: Current Date and Time

Variable	**curdt**
Originator	**all**

This variable represents the current date and time. The formatting depends on the originator.

For all originators except **textfile-open**, the logger applies dashes, a space and colons, for example "2014-12-08 02:15:20".

In case of originator **textfile-open**, all values are separated by underscores, for example "2014_12_08_02_15_20". The reason is underscores are virtually always valid characters when used in a filename, while colons and spaces are not always valid depending on the file system.

Variable: Sense Date and Time

Variable	**sensedt**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**all**

Date and time the sensor data was acquired.

The logger applies dashes, a space and colons in the formatting of the text, for example "2014-12-08 02:15:20".

Variable: Sensor Identifier

Variable	**sensorid**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**all**

The sensor identifier as it appears in the sensor data.

Variable: ROM Code

Variable	**romcode**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**all**

The ROM code that's present in the sensor data.

Variable: Temperature

Variable	**temp**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**ds18s20, ds1822, ds2438, ds18b20, ds2760, ds2780, ds2755, ds1825, max31826, ds2781, ds28ea00**

The temperature measured by the 1-Wire slave.

Variable: Temperature Register

Variable	**reg temp**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**ds18s20, ds1822, ds2438, ds18b20, ds2760, ds2780, ds2755, ds1825, max31826, ds2781, ds28ea00**
Bits	**16**

The value of the temperature register of the 1-Wire slave.

Variable: Voltage V_{AD} Pin

Variable	**vad**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**ds2438**

The voltage measured on pin V_{AD}.

Variable: Voltage Register V_{AD} Pin

Variable	**reg vad**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**ds2438**
Bits	**16**

The value of the voltage register after sampling pin V_{AD}.

Variable: Voltage V_{DD} Pin

Variable	vdd
Originator	mysql-log, pgsql-log, textfile-log
Sensor ID	ds2438

The voltage measured on pin V_{DD}.

Variable: Voltage Register V_{DD} Pin

Variable	reg vdd
Originator	mysql-log, pgsql-log, textfile-log
Sensor ID	ds2438
Bits	16

The value of the voltage register after sampling pin V_{DD}.

Variable: Voltage V_{IN} Pin

Variable	vin
Originator	mysql-log, pgsql-log, textfile-log
Sensor ID	ds2760, ds2780, ds2755, ds2781

The voltage measured on pin V_{IN}.

Variable: Voltage Register V_{IN} Pin

Variable	reg vin
Originator	mysql-log, pgsql-log, textfile-log
Sensor ID	ds2760, ds2780, ds2755, ds2781
Bits	16

The value of the voltage register after sampling pin V_{IN}.

Variable: Current

Variable	cur
Originator	mysql-log, pgsql-log, textfile-log
Sensor ID	ds2438, ds2760, ds2780, ds2755, ds2740, ds2781

The current measured by the 1-Wire slave.

Variable: Current Register

Variable	**reg cur**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**ds2438, ds2760, ds2780, ds2755, ds2740, ds2781**
Bits	**16**

The value of the current register.

Variable: Current Accumulator

Variable	**curacc**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**ds2760, ds2780, ds2755, ds2740, ds2781**

The accumulated current measured by the 1-Wire slave.

Variable: Current Accumulator Register

Variable	**reg curacc**	
Originator	**mysql-log, pgsql-log, textfile-log**	
Sensor ID	**ds2760, ds2780, ds2755, ds2740, ds2781**	
Bits	**16**	**ds2760, ds2755, ds2740**
	32	**ds2780, ds2781**

The value of the current accumulator register.

Variable: ADC Inputs

Variable	**aina** **ainb** **ainc** **aind**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**ds2450**

The voltage measured on the ADC pin.

Variable: ADC Input Registers

Variable	**reg aina** **reg ainb** **reg ainc** **reg aind**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**ds2450**
Bits	**16**

ADC voltage registers.

Variable: PIO Sensed

Variable	piosensed
Originator	mysql-log, pgsql-log, textfile-log
Sensor ID	ds2406, ds28e04, ds2408, ds2760, ds2780, ds2755, ds2740, ds2413, ds2781, ds28ea00

Sensed (input) state of one or all PIO pins. The variable is formatted as a decimal value. If one PIO pin is reported, the resulting value is **0** or **1**. If all PIO pins are reported, the value is **0..2^{pins}-1** and represents a bit mask.

Variable: PIO Output

Variable	pioset
Originator	mysql-log, pgsql-log, textfile-log
Sensor ID	ds2406, ds28e04, ds2408, ds2413, ds28ea00

Output state of one or all PIO pins. The variable is formatted as a decimal value. If one PIO pin is reported, the resulting value is **0** or **1**. If all PIO pins are reported, the value is **0..2^{pins}-1** and represents a bit mask.

Variable: PIO Activity

Variable	pioset
Originator	mysql-log, pgsql-log, textfile-log
Sensor ID	ds2406, ds28e04, ds2408

Activity state of one or all PIO pins. The variable is formatted as a decimal value. If one PIO pin is reported, the resulting value is **0** or **1**. If all PIO pins are reported, the value is **0..2^{pins}-1** and represents a bit mask.

Variable: PIO Count

Variable	piocnt
Originator	mysql-log, pgsql-log, textfile-log
Sensor ID	ds2406

The number of available PIO pins.

Variable: RSTZ Configuration

Variable	rstz
Originator	mysql-log, pgsql-log, textfile-log
Sensor ID	ds2408

This variable denotes the configuration of the RSTZ pin:

- 0: the pin is configured as $\overline{\text{RST}}$ input.
- 1: the pin is configured as $\overline{\text{STRB}}$ output.

Variable: Counter

Variable	**cntra** **cntrb**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**ds2423**

Value of counters A and B.

Variable: Power Mode

Variable	**pwrmode**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**ds18s20, ds2406, ds28e04, ds2450, ds1822, ds18b20, ds2408, ds1825, max31826, max31850, ds28ea00**

Power mode of the 1-Wire slave:

- 0: external power mode.
- 1: parasite power mode.

Variable: Thermocouple Hot Temperature

Variable	**tchot**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**max31850**

Thermocouple hot temperature.

Variable: Thermocouple Hot Temperature Register

Variable	**reg tchot**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**max31850**
Bits	**16**

Value of the thermocouple hot temperature register.

Bits 1..0 are always zero.

Variable: Thermocouple Cold Temperature

Variable	**tccold**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**max31850**

Thermocouple cold temperature.

Variable: Thermocouple Cold Temperature Register

Variable	**reg tccold**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**max31850**
Bits	**16**

Value of the thermocouple cold temperature register.

Bits 3..0 are always zero.

Variable: Thermocouple Fault

Variable	**tcfault**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**max31850**

This variable says whether a fault in the thermocouple is detected:

- 0: No fault detected.
- 1: Fault detected.

Variable: Thermocouple Unconnected

Variable	**tcunconn**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**max31850**

This variable indicates whether the thermocouple is connected or not:

- 0: Connected.
- 1: Unconnected.

Variable: Thermocouple Open Circuit

Variable	**tcopen**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**max31850**

This variable says whether an open circuit is detected:

- 0: No open circuit detected.
- 1: Open circuit detected.

Variable: Thermocouple Short to GND

Variable	**tcgnd**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**max31850**

This variable indicates whether a short to the GND pin is detected:

- 0: No short detected.
- 1: Short detected.

Variable: Thermocouple Short to VCC

Variable	**tcvcc**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**max31850**

This variable indicates whether a short to the VCC pin is detected:

- 0: No short detected.
- 1: Short detected.

Variable: Address Pins

Variable	**adpins**
Originator	**mysql-log, pgsql-log, textfile-log**
Sensor ID	**ds1825, max31826, max31850**

This variable contains the state of the address pins.

7 Trigger

Overview

Keyword **sensed** starts a compound statement that defines a trigger for logging sensor data.

The logger recognizes the following keywords in the information block:

Keyword	Description
`match`	Matching criterion.
`on`	Logging action to take when a match has occurred.

Both **match** and **on** can occur multiple times in the information block.

The keyword **sensed** is derived from the 1-Wire server's client response **Device Sensed**.

Matching Criterion

A matching criterion contains the following information:

1. Either a 1-Wire slave's ROM code or a sensor identifier. A ROM code targets a specific 1-Wire slave, a sensor identifier targets an entire family of 1-Wire slaves.
2. A flag indicating how sensor data for the ROM code or sensor identifier must be reported: always, or differentially (that is, when the sensor data differs from the last reading). Keyword **always** selects always reporting, keyword **diff** selects differential reporting. The default is **always**.

Examples:

```
match always dev "10-802A49A17";
```

Match for specific 1-Wire slave, the 1-Wire server reports all sensor data.

```
match ds18b20;
```

Match for all DS18B20 slaves, the 1-Wire server reports all sensor data.

```
match diff dev "28-40CBBB2";
```

Match for specific 1-Wire slave, the 1-Wire server reports sensor data differentially.

As soon as the logger connects to a 1-Wire server, it sends a **Report Add Sensed** client command for each **match** statement. If you specify a ROM code or sensor identifier multiple times throughout the configuration file, make sure that you consistently select the same method of reporting (always or differentially), else the effective method of reporting may become unclear to you.

Logging Action

A logging action contains the following information:

1. A success/error flag indicating whether successfully or unsuccessfully acquired sensor data should be processed. Keywords **success** and **error** determine the state of this flag.
2. Logging destination: MySQL database, PostgreSQL database, text file.

Examples:

`on success mysql "mydb" "add_temp";`	MySQL destination, success flag set.
`on success pgsql "pgdb" "add_temp";`	PostgreSQL destination, success flag set.
`on error textfile "mytf" "log error";`	Text file destination, error flag set.

Each logging destination is indicated by a keyword:

- MySQL: keyword **mysql**.
- PostgreSQL: keyword **pgsql**.
- Text file: keyword **textfile**.

The keyword is followed by two strings. The first string contains the name of the definition of the target destination. The second string contains the name of the text declaration that represents the actual logging command.

The contents of the logging command depend on the logging destination. For a database, the logging command is usually an SQL statement that starts with an "INSERT" keyword. For a text file destination, the logging command is literal text that's written to the text file.

The text declaration of the logging command usually contains various variables. You should make sure that you only use those variables that exist for all 1-Wire slaves that occur in the matching criteria. For example:

```
sensed
{
    match ds2450;    # A/D converter
    match ds1822;    # Temperature sensor

    on success pgsql "pgdb" "log_data";
}
```

There are two matching criteria, one for DS2450 slaves and another for DS1822 slaves. Text declaration "log_data" can contain variable **pwrmode** since both the DS250 and the DS1822 have a power mode configuration, but the text declaration shouldn't contain variable **temp** since only the DS1822 incorporates a temperature sensor.

Processing Sensor Data

When the logger receives a **Device Sensed** client response from a 1-Wire server, a process of matching and logging is set in motion. The logger iterates through all triggers in the order they appear in the configuration file. In each trigger, the logger iterates through the matching criteria in the order they appear in the information block. Each match criterion is checked with the sensor data until a match occurs.

In case of a match, the logger iterates through the logging actions in the order they appear in the information block. A logging action is executed if its success/error flag corresponds with the success/error state of the sensor data. The logger doesn't iterate through the remaining matching criteria.

8 Connection with 1-Wire Automation Server

Keyword **owsas** starts a compound statement that defines a connection with a 1-Wire server. Each **owsas** definition must have a unique name.

The logger recognizes the following keywords in the information block:

Keyword	Description
`host`	Hostname or IP address of the 1-Wire server (ASCII).
`port`	Port of the 1-Wire server (1..65535).
`username`	1-Wire username (Unicode).
`password`	1-Wire password for username (Unicode).
`retry`	Connection retry wait time in milliseconds (0..).
`cmd`	Command (text declaration).

Keywords **host** and **port** are mandatory. These values indicate the network location of the 1-Wire server.

Keywords **username** and **password** are required if the 1-Wire server is configured for user authentication.

When the connection with the 1-wire server is broken, the logger waits a while before it tries to reconnect. Use keyword **retry** to specify the wait time. The default value is 5000 milliseconds.

Keyword **cmd** stands for initial command and refers to a text declaration. Each initial command represents one or more command text lines. Multiple initial commands can be specified. As soon as a connection is established, the logger sends the initial commands to the 1-Wire server in the order they appear in the information block. As such the initial commands must represent client commands.

Note that there's no keyword for selecting the temperature scale. The logger doesn't use the temperature values in the sensor data strings the 1-Wire server reports. Instead, the logger uses the temperature register values from which it derives the temperature values.

When the connection with the 1-Wire server is established, the logger sends a number of client commands in the following order:

1. If a username is specified, the logger sends the **Authentication** command.
2. The logger sends the text composed from the initial commands.
3. The logger sends **Report Add Sensed** commands for all 1-Wire slaves and sensor identifiers that are specified in the matching criteria in the triggers.

Example:

```
owsas "myowsas"
{
    host        "localhost";
    port        5001;
    username    "Gregory";
    password    "kl45gh78";
    retry       2000;
}
```

Example of initial command:

```
# Add 1-Wire slave and enable polling
text "poll_ds28ea00" =
    "dev " 34 "42-00000038D0BE-A3" 34 " add" 10
    "dev " 34 "42-00000038D0BE-A3" 34 " attr poll=on,60000" 10;

owsas "myowsas"
{
    host        "localhost";
    port        5001;

    cmd         "poll_ds28ea00";
}
```

Note that the logger never enables polling of 1-Wire slaves. You've to send the required commands to the 1-Wire server. You can use initial commands as in the last example, you can add client commands to the start-up command file of the 1-Wire server, you can put the client commands in a text file and stream them to the 1-Wire server using **netcat**, etc.

9 Connection with MySQL Database

Keyword **mysql** starts a compound statement that defines a connection with a MySQL server. Each **mysql** definition must have a unique name.

The logger recognizes the following keywords in the information block:

Keyword	Description
`host`	Hostname or IP address of the MySQL server (ASCII).
`port`	Port of the MySQL server (1..65535).
`username`	MySQL username (Unicode).
`password`	MySQL password for username (Unicode).
`database`	MySQL database (Unicode).
`retry`	Connection retry delay time in milliseconds (0..).
`tscale`	Temperature scale: **celsius**, **fahrenheit**, **kelvin**.
`cmd`	Command (text declaration).
`library`	Path and filename of the MySQL library (Unicode).

Keywords **host**, **username** and **password** are mandatory. If no database is specified, the SQL statements must contain explicit references to database(s). If no port is specified, the MySQL library uses the default port.

When the connection with the database server is broken, the logger waits a while before it tries to reconnect. Use keyword **retry** to specify the wait time. The default value is 5000 milliseconds.

Keyword **cmd** stands for initial command and refers to a text declaration. Each initial command represents an SQL statement. Multiple initial commands can be specified. As soon as a connection is established, the logger sends the initial commands to the MySQL server in the order they appear in the information block. You can optionally specify that an initial command must be successfully executed in the MySQL server or else the connection is broken.

The default temperature scale is Celsius. Use keyword **tscale** to select a scale. The temperature scale is applied whenever the logger formats a temperature that's not attributed with a specific scale.

Keyword **library**, if specified, points to the location and name of the MySQL library. This string may include a relative or absolute path.

Example:

```
mysql "mydb"
{
    host        "localhost";
    port        3306;
    username    "johnny";
    password    "ia45hl80";
    database    "sensors";
    retry       10000;
    tscale      fahrenheit;
    library     "C:\xampp\mysql\lib\libmysql.dll";
}
```

Example of initial commands:

```
text "mydb init cmd 1" = composition of SQL statement(s);
text "mydb init cmd 2" = composition of SQL statement(s);

mysql "mydb"
{
    host        "192.168.1.115";
    username    "johnny";
    password    "ia45hl80";

    # Execute command, disregard success or failure result
    cmd         "mydb init cmd 1";

    # Execute command, break connection if the SQL statement fails
    cmd         "mydb init cmd 2" succeed;
}
```

Let's have a look at an example logging action:

```
text "log_temp" =
    "INSERT INTO sensors.temperatures ( timestamp, romcode, temp_f) "
    "VALUES ("
    "'" var(sensedt)          "',"
    "'" var(romcode)          "',"
    "'" var(temp fahrenheit) "')";

sensed
{
    match   ds2438;
    on success mysql "mydb" "log_temp";
}
```

Text declaration "log_temp" must represent an SQL statement. In the example, the text is composed to an INSERT command that adds sensor data to a table called "temperatures". The database "sensors" is explicitly referenced, as is permitted by MySQL. The sensor data comes from DS2438 slaves.

10 Connection with PostgreSQL Database

Keyword **pgsql** starts a compound statement that defines a connection with a PostgreSQL server. Each **pgsql** definition must have a unique name.

The logger recognizes the following keywords in the information block:

Keyword	Description
`host`	Hostname of the PostgreSQL server (ASCII).
`hostaddr`	Host IP address of the PostgreSQL server (ASCII).
`port`	Port of the PostgreSQL server (1..65535).
`username`	PostgreSQL username (Unicode).
`password`	PostgreSQL password for username (Unicode).
`database`	PostgreSQL database (Unicode).
`retry`	Connection retry wait time in milliseconds (0..).
`tscale`	Temperature scale: **celsius**, **fahrenheit**, **kelvin**.
`cmd`	Command (text declaration).

Use keyword **host** if you want to specify a hostname for locating the PostgreSQL server. The PostgreSQL library will look up the hostname.

Use keyword **hostaddr** if you want to specify the IP address of the PostgreSQL server.

Keyword **password** is mandatory. It's recommended to specify keywords **host** or **hostaddr**, **username** and **database** as well, else the PostgreSQL library picks some settings you probably don't want to use.

If no port is specified, the PostgreSQL library uses the default port.

When the connection with the database server is broken, the logger waits a while before it tries to reconnect. Use keyword **retry** to specify the wait time. The default value is 5000 milliseconds.

The default temperature scale is Celsius. Use keyword **tscale** to select a scale. The temperature scale is applied whenever the logger formats a temperature that's not attributed with a specific scale.

Keyword **cmd** stands for initial command and refers to a text declaration. Each initial command represents an SQL statement. Multiple initial commands can be specified. As soon as a connection is established, the logger sends the initial commands to the PostgreSQL server in the order they appear in the information block. You can optionally specify that an initial command must be successfully executed in the PostgreSQL server or else the connection is broken.

Note that there's no **library** keyword. The PostgreSQL library imports other libraries which the system can't locae based on the **library** keyword. Instead you should make sure the system can load the PostrgreSQL libraries.

Example:

```
pgsql "mydb"
{
    host        "localhost";
    %hostaddr    "127.0.0.1";
    port        5432;
    username    "johnny";
    password    "ia45hl80";
    database    "sensors";
    retry       10000;
    tscale      kelvin;
}
```

Example of initial commands:

```
text "mydb init cmd 1" = composition of SQL statement(s);
text "mydb init cmd 2" = composition of SQL statement(s);

pgsql "mydb"
{
    hostaddr    "192.168.1.120";
    username    "johnny";
    password    "ia45hl80";
    database    "sensors";

    # Execute command, disregard success or failure result
    cmd         "mydb init cmd 1";

    # Execute command, break connection if the SQL statement fails
    cmd         "mydb init cmd 2" succeed;
}
```

Let's have a look at an example logging action:

```
text "log_volt" =
    "INSERT INTO voltages ( timestamp, romcode, voltage) VALUES ("
    "'" var(sensedt) "',"
    "'" var(romcode) "',"
    "'" var(vin)     "')";

sensed
{
    match   ds2755;
    match   ds2760;
    on success pgsql "mydb" "log_volt";
}
```

Text declaration “log_volt” must represent an SQL statement. In the example, the text is composed to an INSERT command that adds sensor data to a table called “voltages”. The sensor data comes from DS2755 and DS2760 slaves.

11 Connection with SQLite Database

Keyword **sqlite** starts a compound statement that defines a connection with an SQLite database. Each **sqlite** definition must have a unique name.

The logger recognizes the following keywords in the information block:

Keyword	Description
`dbfilename`	Path and filename of the SQLite database (Unicode).
`retry`	Connection retry delay time in milliseconds (0..).
`tscale`	Temperature scale: **celsius**, **fahrenheit**, **kelvin**.
`cmd`	Command (text declaration).
`library`	Path and filename of the SQLite3 library (Unicode).

Keyword **dbfilename** is mandatory.

When the connection with the database file is broken, the logger waits a while before it tries to reconnect. Use keyword **retry** to specify the wait time. The default value is 5000 milliseconds.

Keyword **cmd** stands for initial command and refers to a text declaration. Each initial command represents an SQL statement. Multiple initial commands can be specified. As soon as a connection is established, the logger sends the initial commands to the SQLite database in the order they appear in the information block. You can optionally specify that an initial command must be successfully executed in the SQLite database or else the connection is broken.

The default temperature scale is Celsius. Use keyword **tscale** to select a scale. The temperature scale is applied whenever the logger formats a temperature that's not attributed with a specific scale.

Keyword **library**, if specified, points to the location and name of the SQLite3 library. This string may include a relative or absolute path.

Example:

```
sqlite "mydb"
{
    dbfilename  "sensors.db";
    retry       10000;
    tscale      fahrenheit;
    library     "D:\sqlite3.dll";
}
```

Example of initial commands:

```
text "mydb init cmd 1" = composition of SQL statement(s);
text "mydb init cmd 2" = composition of SQL statement(s);

sqlite "mydb"
{
    dbfilename  "sensors.db";

    # Execute command, disregard success or failure result
    cmd         "mydb init cmd 1";

    # Execute command, break connection if the SQL statement fails
    cmd         "mydb init cmd 2" succeed;
}
```

Let's have a look at an example logging action:

```
text "log_temp" =
    "INSERT INTO temperatures ( timestamp, romcode, temp_f) "
    "VALUES ("
    "'" var(sensedt)          "',"
    "'" var(romcode)          "',"
    "'" var(temp fahrenheit) "')";

sensed
{
    match   ds2438;
    on success mysql "mydb" "log_temp";
}
```

Text declaration "log_temp" must represent an SQL statement. In the example, the text is composed to an INSERT command that adds sensor data to a table called "temperatures". The sensor data comes from DS2438 slaves.

12 Text File

Keyword **textfile** starts a compound statement that defines a text file for writing sensor data to. Each **textfile** definition must have a unique name.

The logger recognizes the following keywords in the information block:

Keyword	Description
`filename`	Path and name of the text file (text declaration).
`enc`	Byte encoding: **utf8**, **utf16le**, **utf16be**.
`mode`	Mode of opening file: **overwrite**, **append**.
`retry`	Connection retry delay time in milliseconds (0..).
`tscale`	Temperature scale: **celsius**, **fahrenheit**, **kelvin**.
`cmd`	Command (text declaration).

Keyword **filename** refers to a text declaration. It represents the path and filename of the text file.

When the logger creates a new text file (because it doesn't exist or it's to be overwritten), the program selects a byte encoding for writing Unicode characters to the text file. The default encoding is UTF-8. Use keyword **enc** to enforce a specific encoding.

Keyword **mode** decides whether the text file is appended or overwritten (created from scratch). The default mode is overwrite. Note that if the text file doesn't exist, the mode automatically switches to overwrite.

If an I/O error occurs while the logger is writing data to the text file, the logger closes the file handle and waits a while before it tries to reopen the text file. Use keyword **retry** to specify the wait time. The default value is 5000 milliseconds. This setting is very useful in case the text file resides on a network file system that may become temporarily unreachable due to network problems.

The default temperature scale is Celsius. Use keyword **tscale** to select a scale. The temperature scale is applied whenever the logger formats a temperature that's not attributed with a specific scale.

Keyword **cmd** stands for initial command and refers to a text declaration. Each initial command represents text. Multiple initial commands can be specified. As soon as a connection is established, the logger writes the initial commands to the text file in the order they appear in the information block.

You can optionally specify that an initial command is only written when the text file has been created. This way you can write a header to a newly created text file.

Example timestamped text files on network file system:

```
text "tf_filename" = "\\ServerPC\data\" var(curdt) "_sensor_data.log";

textfile "tf"
{
    filename    "tf_filename";
    enc         utf16le;
    mode        overwrite;
    retry       30000;
}
```

Example of initial commands:

```
text "tf_filename" = "sensors.log";

text "tf_header" =
    "#" 13 10
    "# Sensor data" 13 10
    "#" 13 10
    13 10;

text "tf_entry" = "# Entry: " var(curdt) 13 10;

textfile "tf"
{
    filename    "tf_filename";
    mode        append;
    cmd         "tf_header" create;
    cmd         "tf_entry";
}
```

When the text file is first created, text declaration "tf_filename" is composed and written to the text file. The text file begins with a nice header made up of a number of text lines.

Text declaration "tf_entry" is composed and written every time the text file is opened. The composed text describes the date and time of the moment the file is opened.

Example logging action:

```
text "log_ds1825" = var(sensedt) "  " var(romcode) "  "
                    var(temp celsius) " C" 13 10;

sensed
{
    match       ds1825;
    on success  textfile "tf" "log_ds1825";
}
```

In the example, text declaration "log_ds1825" is composed to Unicode text and written to the text file in the effective encoding. The sensor data comes from DS1825 slaves.

13 Software Revision History

Version	Description
1.0.0	▪ Initial release.
1.1.0	▪ Removed keyword library from postgresql block. ▪ Added formatting style property for variable romcode.
1.2.0	▪ Added support for SQLite.

14 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

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

15 Contact Information

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