1-Wire Automation Server v1.2.0

User Manual – Part 2

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 USBMicro client commands. Added W1 client command. Added ROMCode to Attribute client command. Added TMEX to Adapter Add client command. Second release.
2016-01-07	Peter S'heeren	Added client command List. Added client command Report Add/Remove DN. Added client response Dev Sensed Error. Third release.

1 Overview

This part documents the commands and responses that are defined in the client protocol implemented by the 1-Wire Automation Server.

2 Protocol Data

Once a client is connected to the server, the client and the server transfer data bytes to each other. Data in either direction is UTF-8 encoded and represents a stream of Unicode characters.

The server processes received data bytes through a number of stages to produce a single line of text. A text line contains zero or more commands and is described by the **Command Text Line** syntax block (see section "Command and Response Line" below). The stages are:

1. UTF-8 decoder: The incoming stream of data bytes is decoded to a stream of Unicode characters. The Unicode characters are buffered. When an end-of-line (EOL) marker arrives, the buffer is passed to the next stage.
2. Tokenizer: During this stage, groups of consecutive Unicode characters are transformed into syntactical elements called tokens.
3. Parser: Tokens must be sequenced according to syntax rules so they form a valid **Command Text Line**. The parser applies these syntax rules to the stream of tokens. During this stage each command that occurs in the line is added to the command queue.

Errors can occur at each stage. If verbose output is enabled, the server prints useful information in case of an error.

The server's command processor implements this level of complexity in order to cope with all possible input. A client may send all kinds of data, including utter gibberish. In all cases, the server must persist.

A client is assumed to process incoming data bytes the same way. Therefore, the server sends responses using the same syntax rules that apply to commands. Nevertheless, the client can usually implement a less complex model of processing since the server always formats its responses deterministically. When the server sends a response, it never uses tabulation characters, always space characters. A response never contains multiple consecutive space characters, thus in places where whitespace is required the server always emits one space character.

UTF-8 Decoder

The UTF-8 decoder converts 1..4 data bytes to a Unicode character.

Note that UTF-8 is a superset of ASCII 7-bit, thus 1-byte values 0..127 are the same for UTF-8 and ASCII.

Tokenizer

The tokenizer transforms the stream of Unicode characters into elements called tokens. Each token represents one or more consecutive Unicode characters. Note that all characters are subjected to tokenization; never will a character be discarded.

Tokens are the most basic syntactical building blocks and greatly determine the overall syntax of commands and responses.

Label Token

A label is composed of one or more characters:

- First character: 'A'..'Z', 'a'..'z', or '_'.
- Consecutive characters: 'A'..'Z', 'a'..'z', '0'..'9', '_'.

Labels are case-insensitive; "Dog", "DOG", and "dog" or interpreted the same.

Example labels:

`piosensed`	`mnu3`	`_the_label`	`_45_minutes_`

Number Token

A number is composed of one or more characters:

- First character: '0'..'9'.
- Consecutive characters: '0'..'9', 'A'..'Z', 'a'..'z', '_'.

Note that not all letters produce a valid number. Underscores can be inserted to augment the readability of a number.

The last character determines the numeral system:

- Decimal: 'd' or 'D'.
- Hexadecimal: 'h' or 'H'.
- Binary: 'b' or 'B'.

The default is decimal.

Numbers are case-insensitive. Leading zeroes or allowed. The valid range of values is 0..4294967295 (FFFFFFFFh).

Examples:

`125`	`10_1011_1000_B`	`0Fc8Eh`	`1234d`

String Token

A string token represents characters between two double quote characters (**"**).

Comment Token

A comment starts with a hash character (**#**) and ends before the next end-of-line marker. Comments are always discarded during the parsing stage.

Comments come in handy when you put commands in a text file and you want to add useful remarks to the commands. For example:

```
# Add AbioWire adapter. Let the server determine which interface to use,
# i2c-dev or BSC.
adapter "ow" add bscdetect abiowire
```

Character Token

This token represents a single Unicode character. The tokenizer assigns a character token to every Unicode character that doesn't fit in any of the other tokens.

Examples:

- Space character, code 32.
- Tabulation character, code 9.
- Carriage return, code 13.
- Line feed, code 10.
- Colon, code 58.

Note that carriage return and line feed are end-of-line markers.

Parser

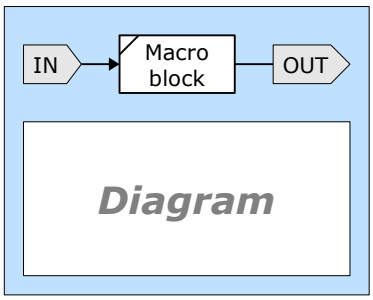
The parser checks the stream of tokens for valid syntax. Valid syntax is documented with syntax flow diagrams. If a token doesn't fit in any flow, a syntax error occurs.

Space and tabulation character tokens act as delimiters between other tokens. A command may contain multiple space and tabulation characters consecutively.

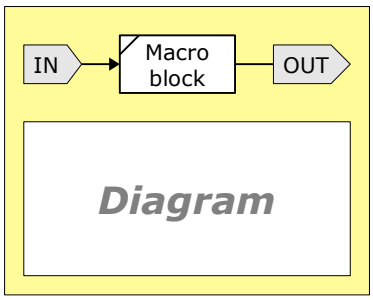
The parser always discards comment tokens. Note that a comment always concludes a text line.

3 Syntax Flow Diagrams

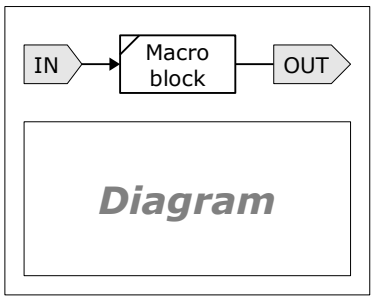
Commands and responses are represented with syntax flow diagrams. These diagrams are hierarchical in nature. The macro block element embodies the concept of hierarchy.



Client command syntax diagram



Client response syntax diagram



Client command and response syntax diagram

The background color indicates whether the diagram applies to a command, a response, or both.

The diagrams are composed of elements:

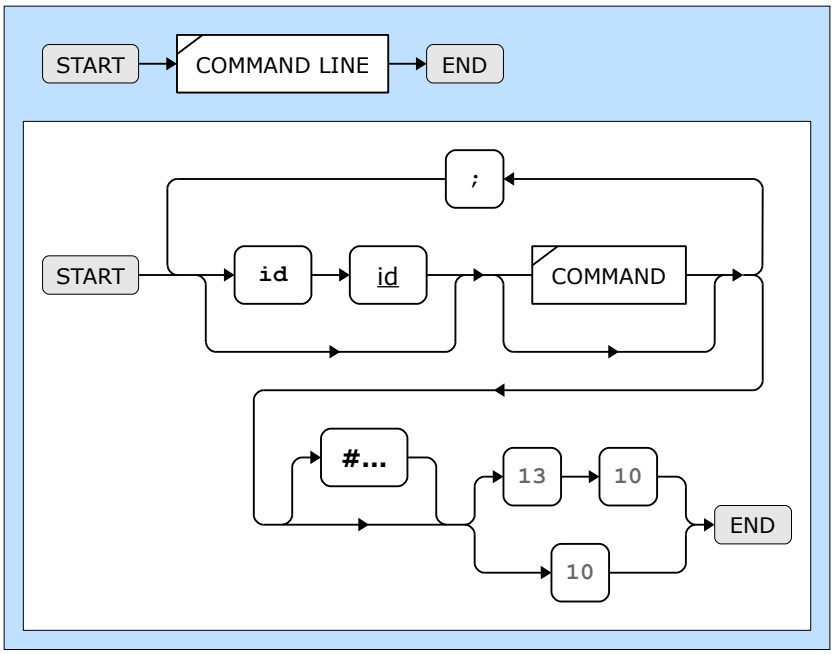
dev	Label token	START	Start of a text line
:	Character token (literal)	END	End of a text line
13	Character token (code)	IN	Entry to macro block
port	Number token	OUT	Exit from macro block
"..." Description	String token	→	Next element, one or more blank[1] characters allowed/required[2]
#...	Comment token	BL →	Next element, no blank[1] characters allowed
Adapter ID	Macro block		

[1] Space characters (32) and tabulation characters (9).

[2] This depends on which elements the arrow connects.

4 Command and Response Line

Command Text Line

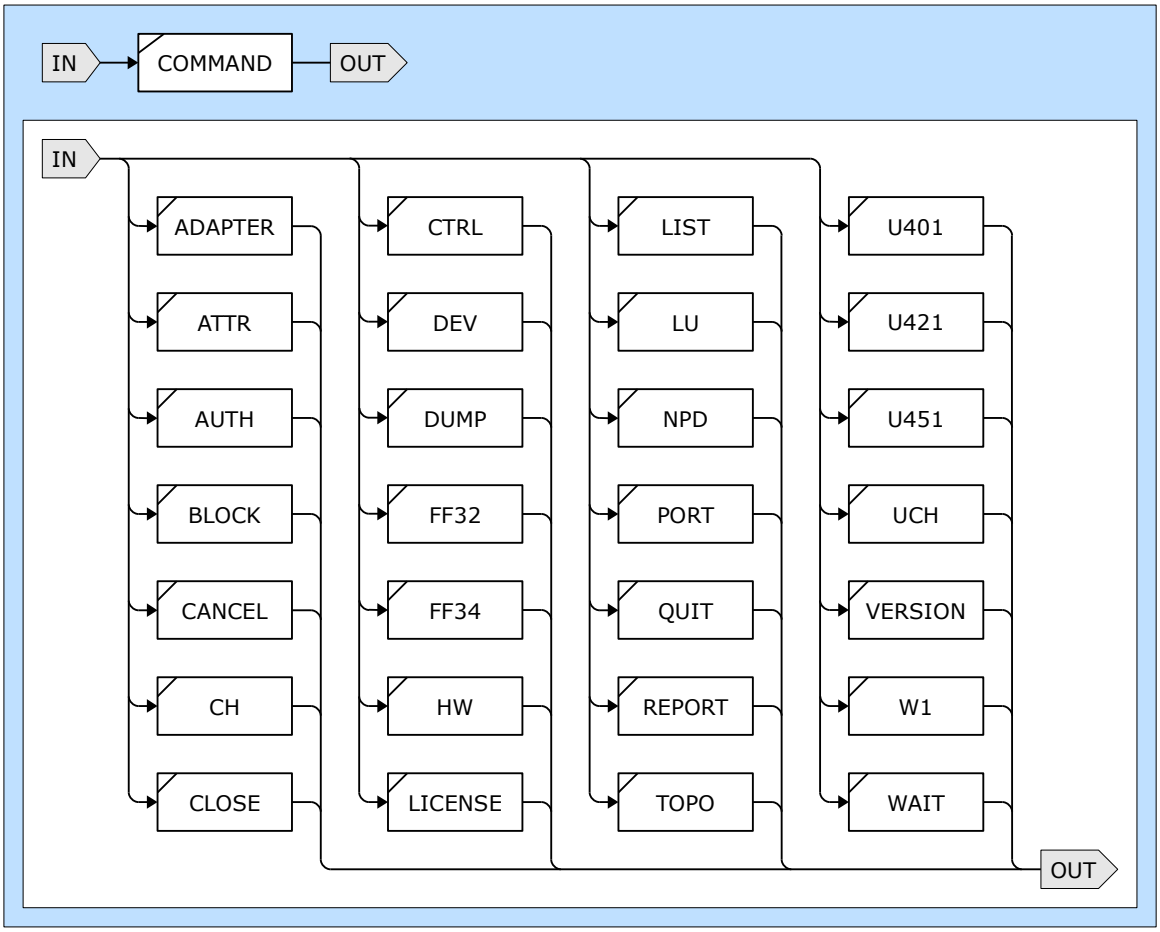


As mentioned earlier in section "Protocol Data" above, the server decodes and buffers incoming UTF-8 characters until an end-of-line marker comes in (stage 1), after which the resulting so-called command text line is tokenized (stage 2) and parsed (stage 3).

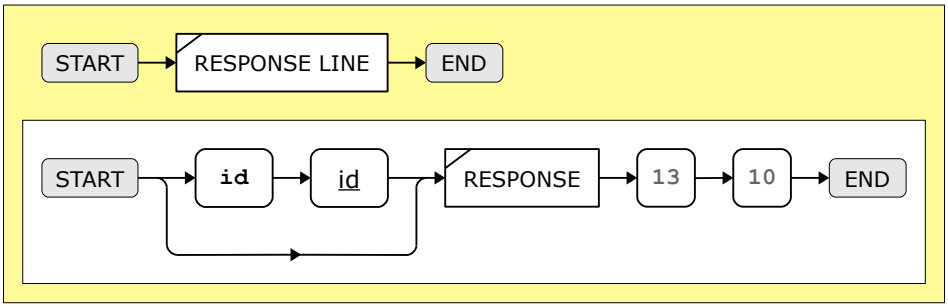
Example command text lines:

```
uch purge
adapter 1 enum              # Enumerate the first 1-Wire adapter
hw enum ; block ; dump      # Enumerate all, dump results when finished
dev "28-40CBBB2" add ; dev "28-40CBBB2" attr poll=on,60000
```

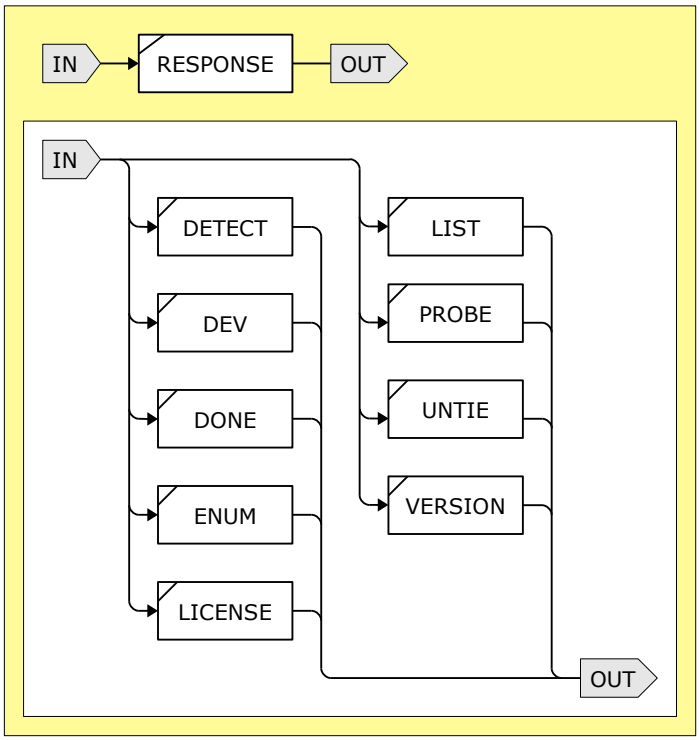
Command


IN
COMMAND
OUT
IN
ADAPTER
ATTR
AUTH
BLOCK
CANCEL
CH
CLOSE
CTRL
DEV
DUMP
FF32
FF34
HW
LICENSE
LIST
LU
NPD
PORT
QUIT
REPORT
TOPO
U401
U421
U451
UCH
VERSION
W1
WAIT
OUT

Response Text Line



Response

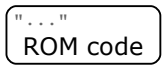


The responses the server produces.

5 Common Syntax

ROM Code

A ROM code is specified inside a string token. ROM codes occur in client commands and responses. The client protocol defines two formatting styles: native and owfs.



Native Formatting Style

When you specify a ROM code in a client command, the 8-bit family code and 48-bit serial number are mandatory, the 8-bit CRC is optional. If you do specify a CRC value, then it must be valid, else the client command is considered invalid.

The ROM code resides in a string token. The values are hexadecimal digits, leading zeroes are allowed. The values must be separated by hyphen characters. Other characters including whitespace are not permitted.

```
hw probe "20-00000014C3CF-0E"
hw probe "20-14C3CF-E"
hw probe "20-00000014C3CF"
hw probe "20-14C3CF"
hw probe "1-16707B5B"
```

When the server returns a client response, it formats ROM codes consistently as follows[1]: 2-digit family code → hyphen → 12-digit serial number. For example:

```
probe ch "usb-4-2":1:1 "20-00000014C3CF" present
```

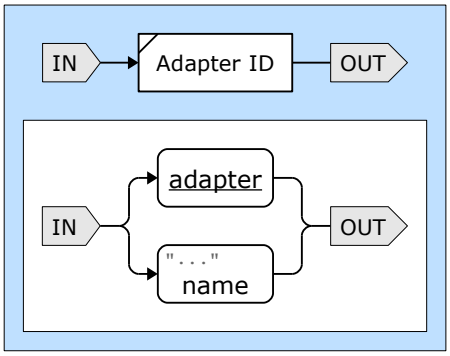
[1] The formatting of ROM codes in the responses from **Dump** commands may vary. Remember that these responses are subject to change and are not meant for processing by software.

owfs Formatting Style

This formatting style is the same for commands and responses: 2-digit family code → dot → 12-digit serial number in reversed byte-order. Example command and response:

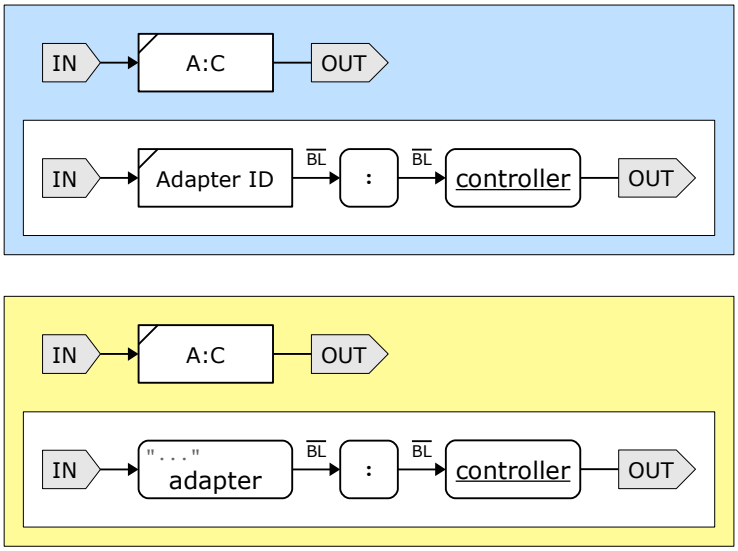
```
hw probe "26.48496B010000"
probe ch "usb-1-2":1:1 "26.48496B010000" present
```

Adapter Identifier

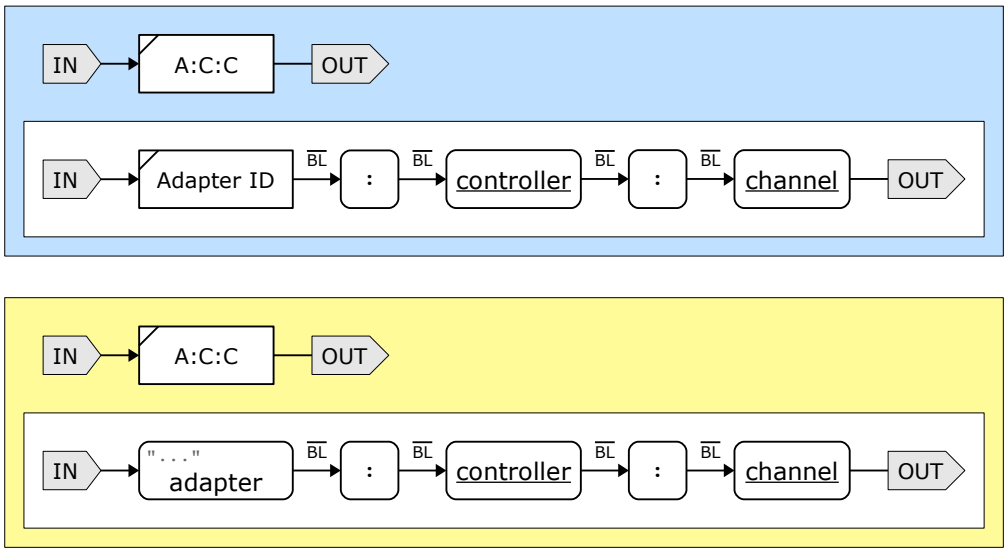


An adapter can be identified with its name or its number. If a string token is present, it represents an adapter name. The name is case-sensitive. If a number token is present, it indicates an adapter number (1..).

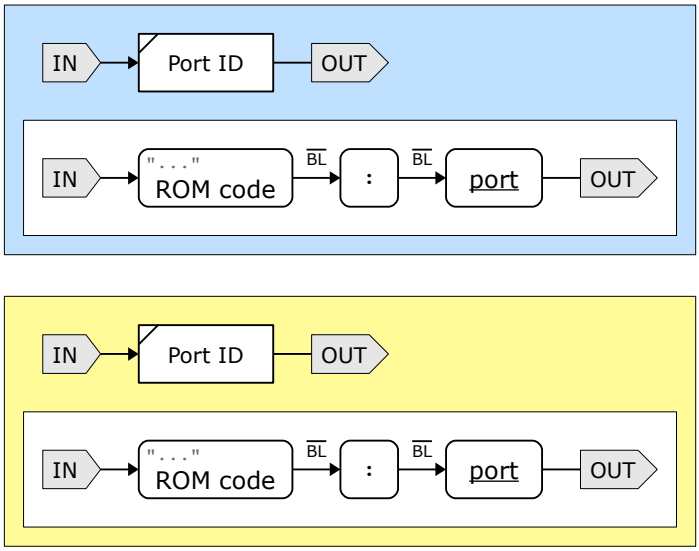
Adapter-Controller



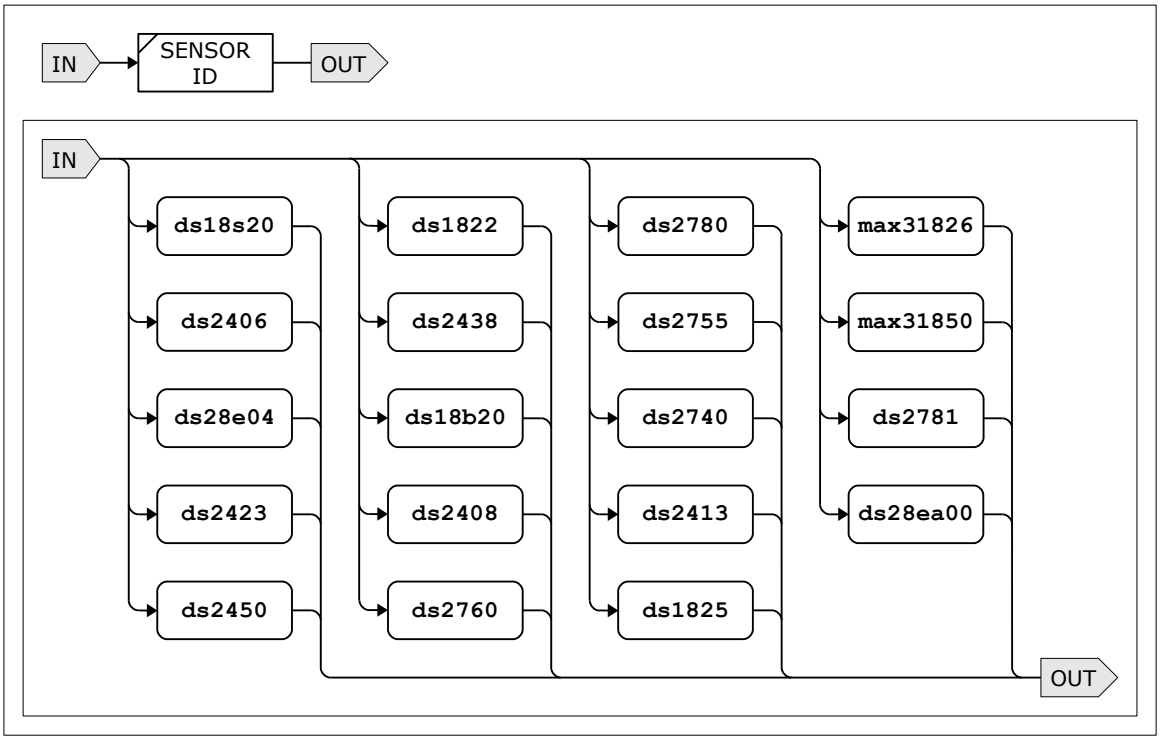
Adapter-Controller-Channel



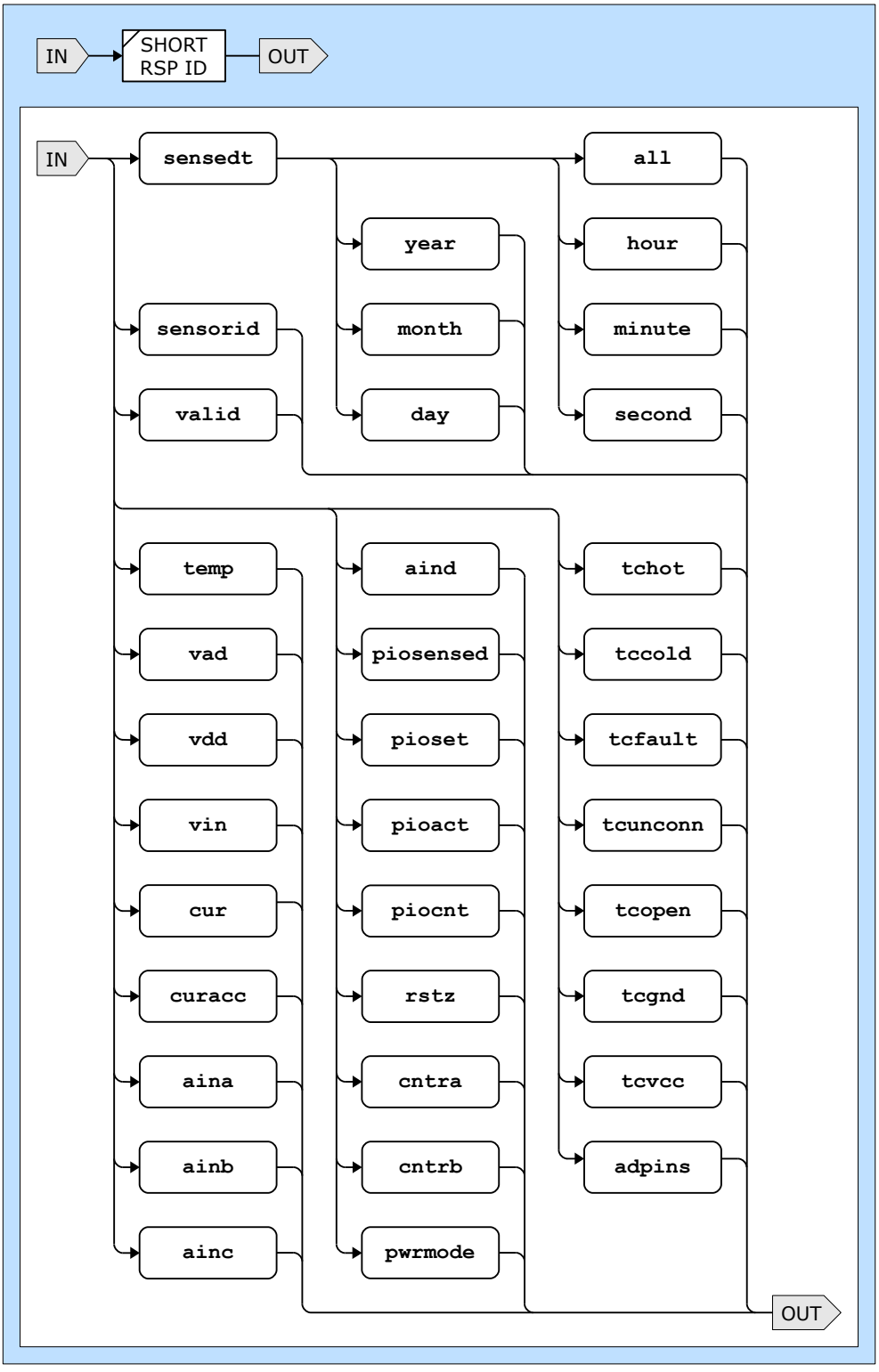
Port Identifier



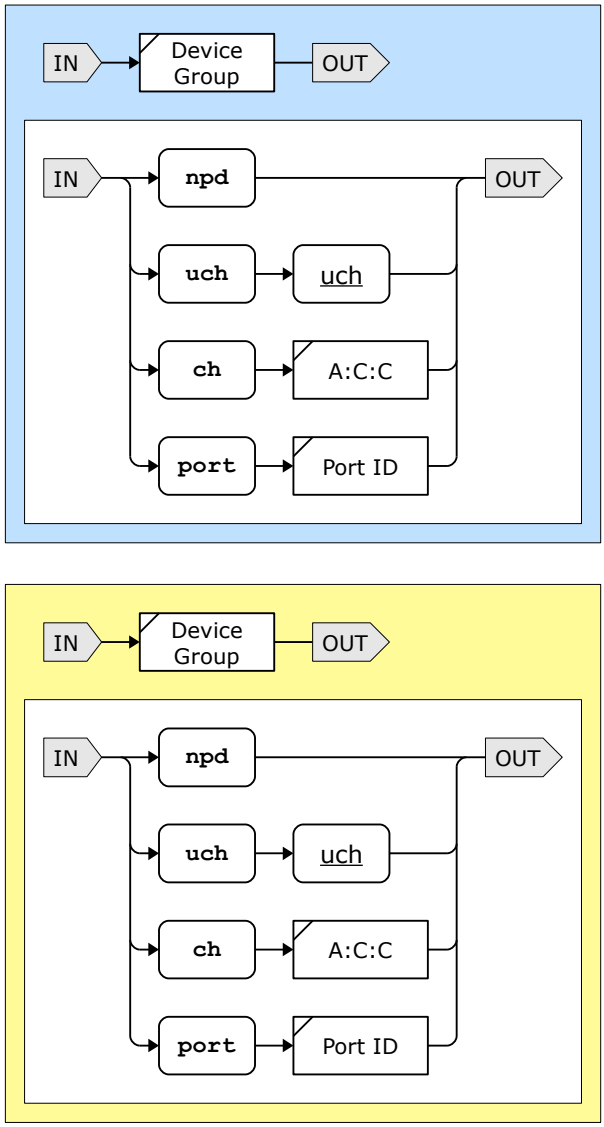
Sensor Identifier



Short Response Identifier

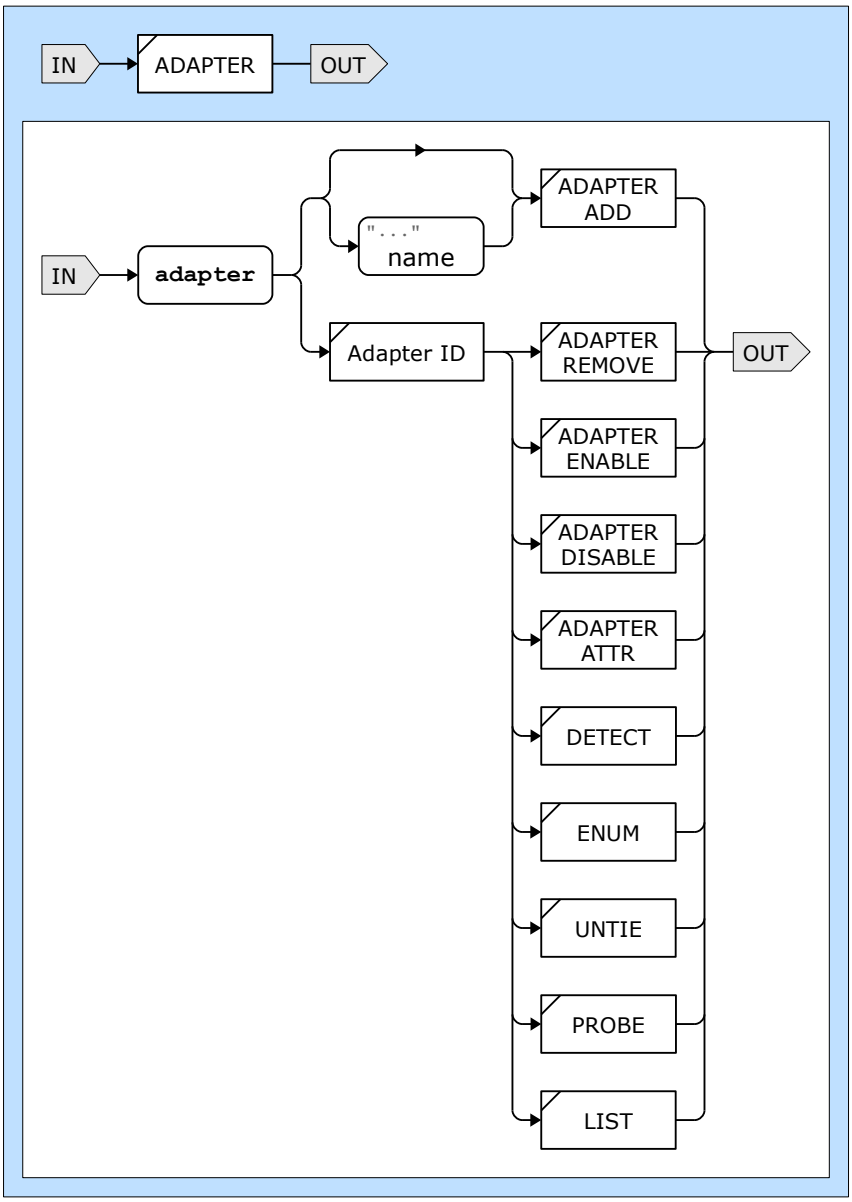


Device Group



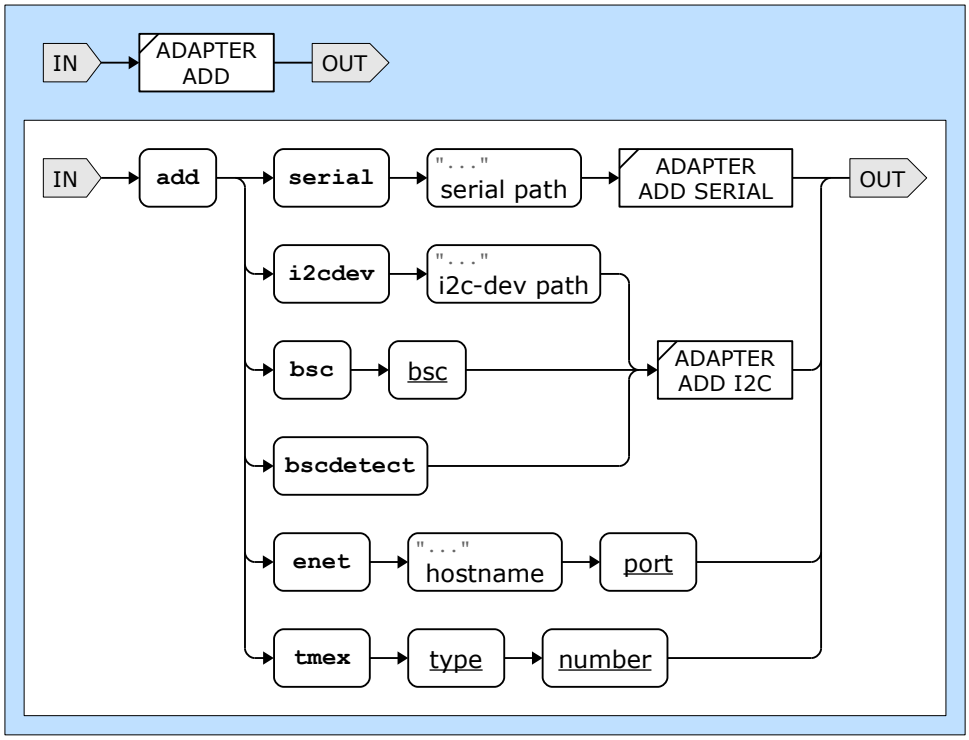
6 Commands and Responses

Adapter



Base command for performing adapter-related tasks.

Adapter Add



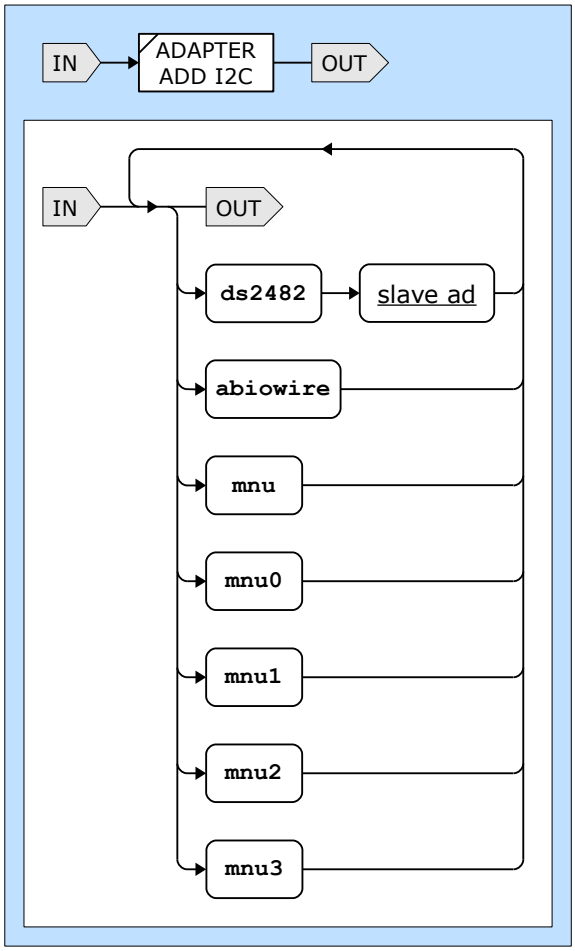
This client command adds a 1-Wire adapter. The arguments following the **add** label describe one or more device nodes that make up a 1-Wire adapter.

If an adapter name is specified, it must not match any existing adapter name.

If no adapter name is specified, the server generates a unique adapter name. The server always generates a unique name for dynamically added adapters like the DS9490.

The **Adapter Add** client command is one of a handful of commands that are executed immediately instead of being queued.

Adapter Add I2C



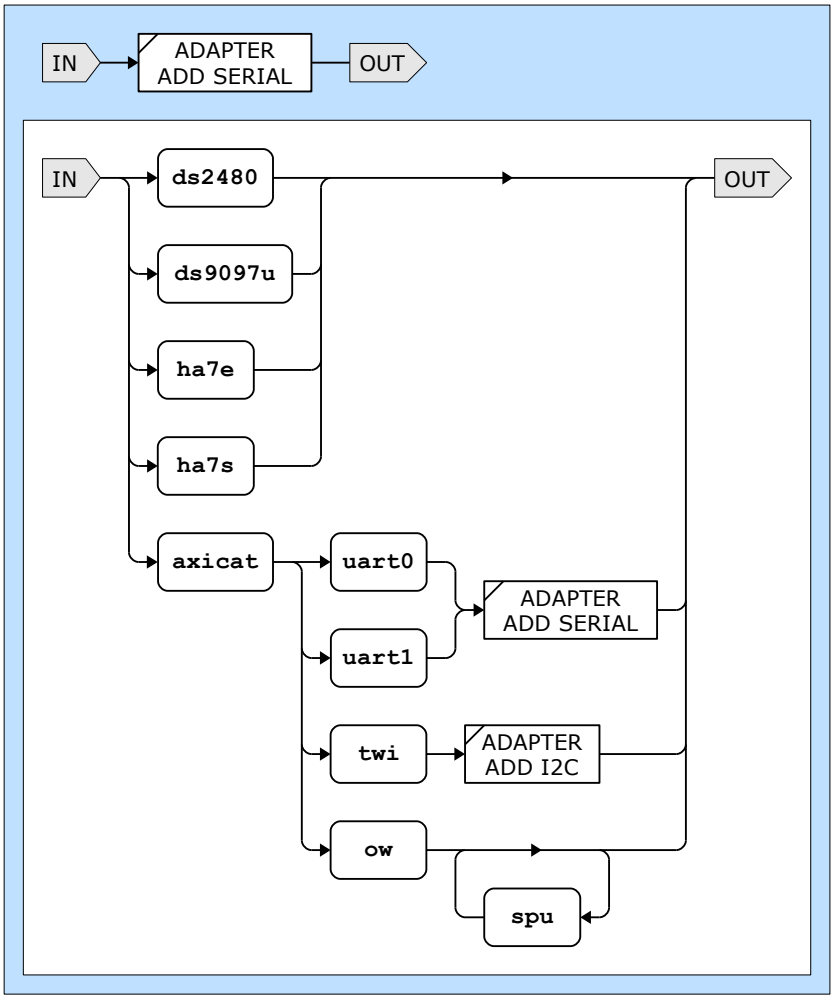
This syntax describes one or more I2C slaves that reside on the same I2C bus and make up a single adapter. Any combination is allowed, as long as no I2C slave addresses overlap with already defined I2C slaves on the same I2C bus. Each I2C slave is reflected by a device node.

The **ds2482** label is used for DS2482-100 and DS2482-800 controllers. The server distinguishes between these controllers when the built-in DS2482 driver is enabling the chip.

The **abiowire** label covers AbioWire and AbioWire+ adapters. These adapters are software-compatible. The label results in the addition of three device nodes.

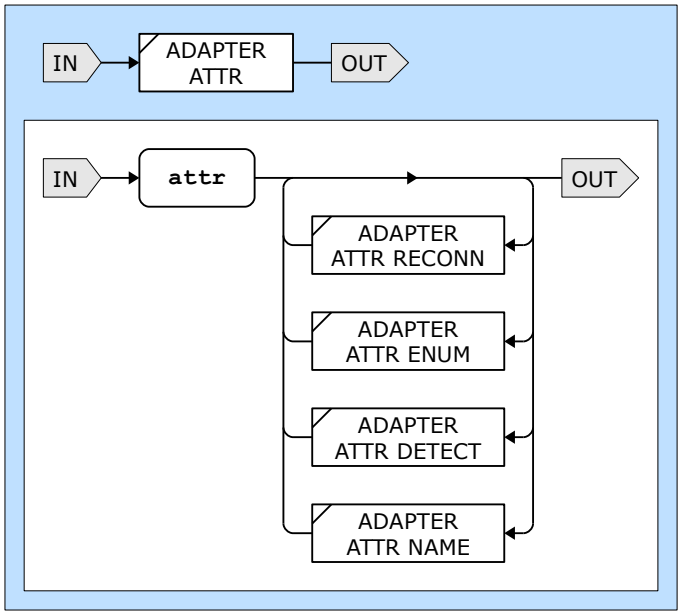
Labels **mnu**, **mnu0**, **mnu1**, **mnu2** and **mnu3** target m.nu 1-Wire adapters.

Adapter Add Serial

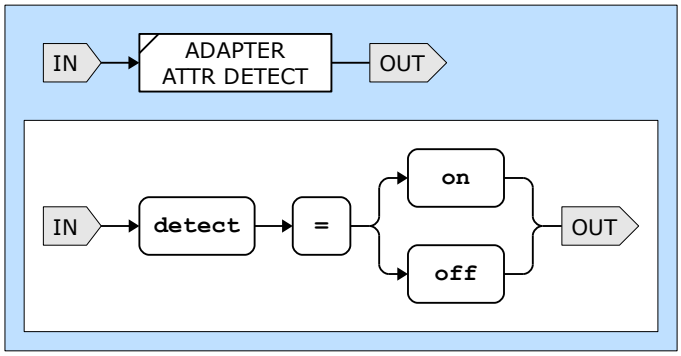


This syntax describes a 1-Wire master with serial interface.

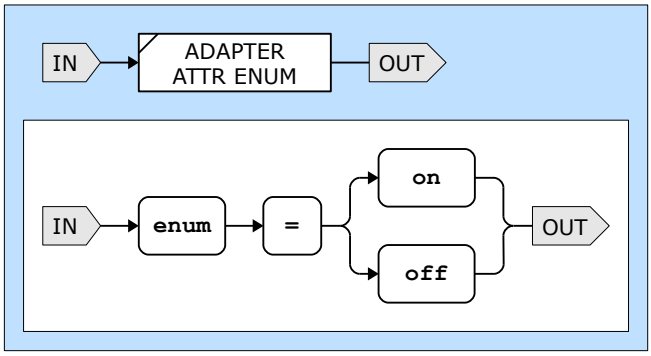
Adapter Attribute



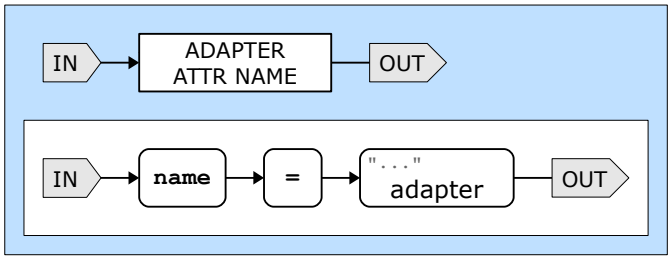
Adapter Attribute Detect



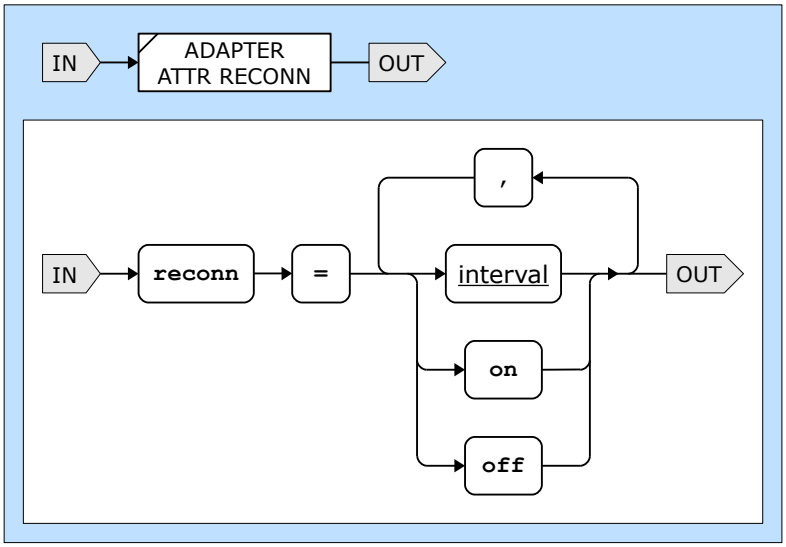
Adapter Attribute Enumerate



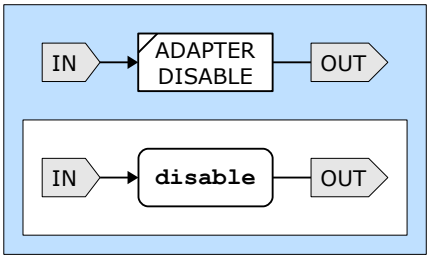
Adapter Attribute Name



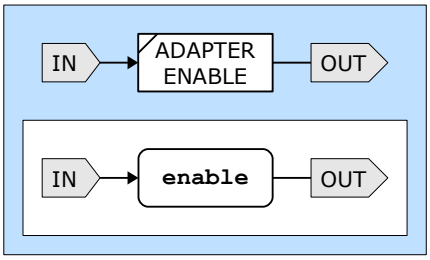
Adapter Attribute Reconnect



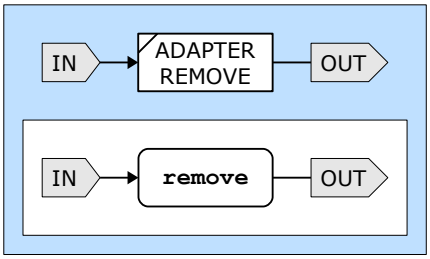
Adapter Disable



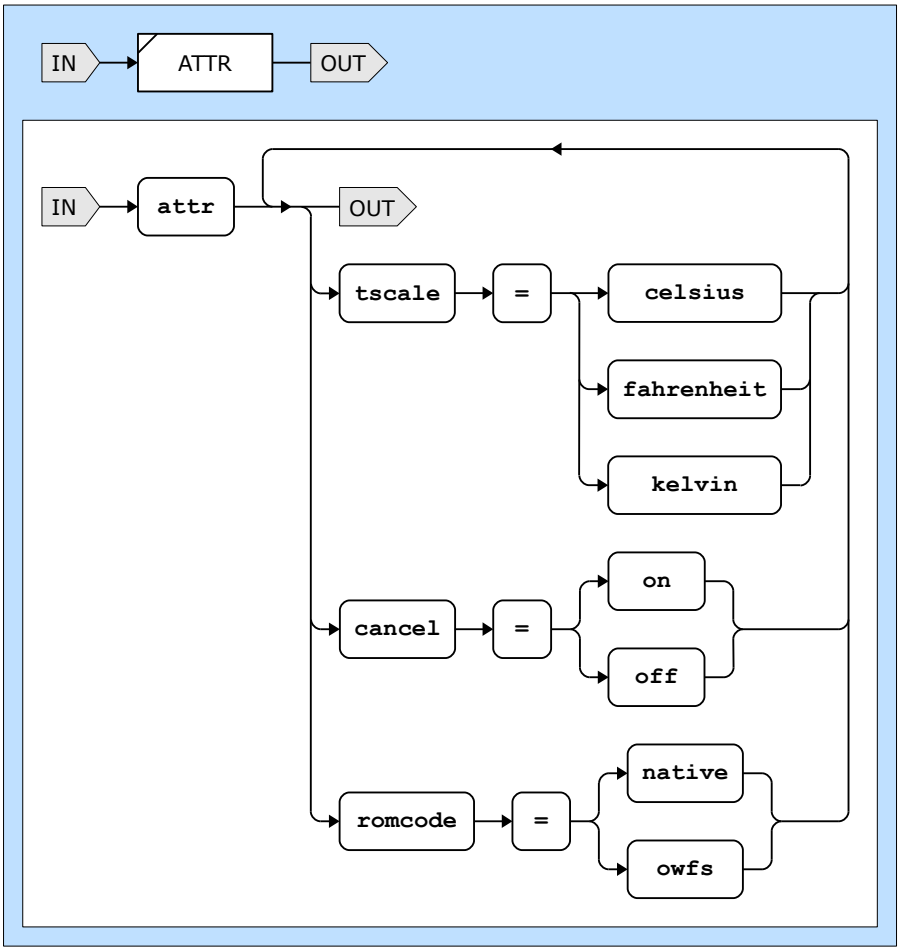
Adapter Enable



Adapter Remove



Attribute



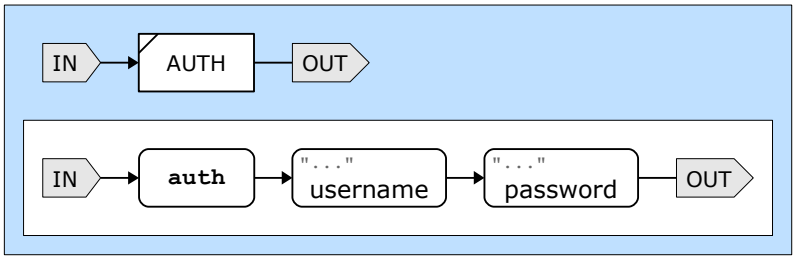
This command sets attributes of the client connection.

Label **tscale** select the temperature scale that is applied when the server formats temperature values in sensor data strings and short responses. The default is Celsius.

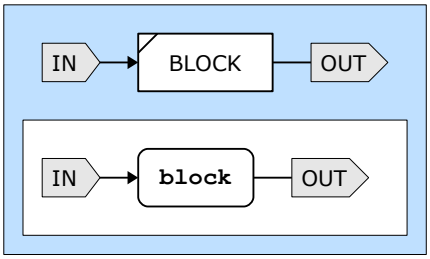
Label **cancel** influences behavior when the client connection is closed. If set to **on**, the server cancels all queued commands that belong to the client. If set to **off**, all queued commands are detached from the client. The default is off.

Label **romcode** selects the formatting of ROM codes in client responses.

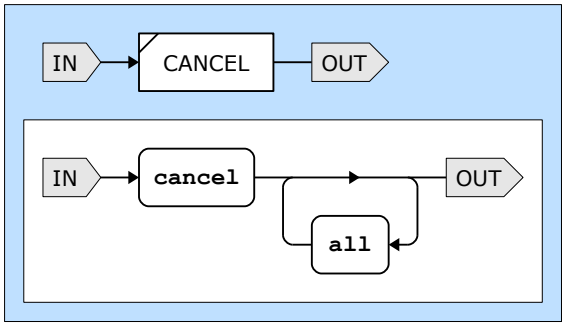
Authentication



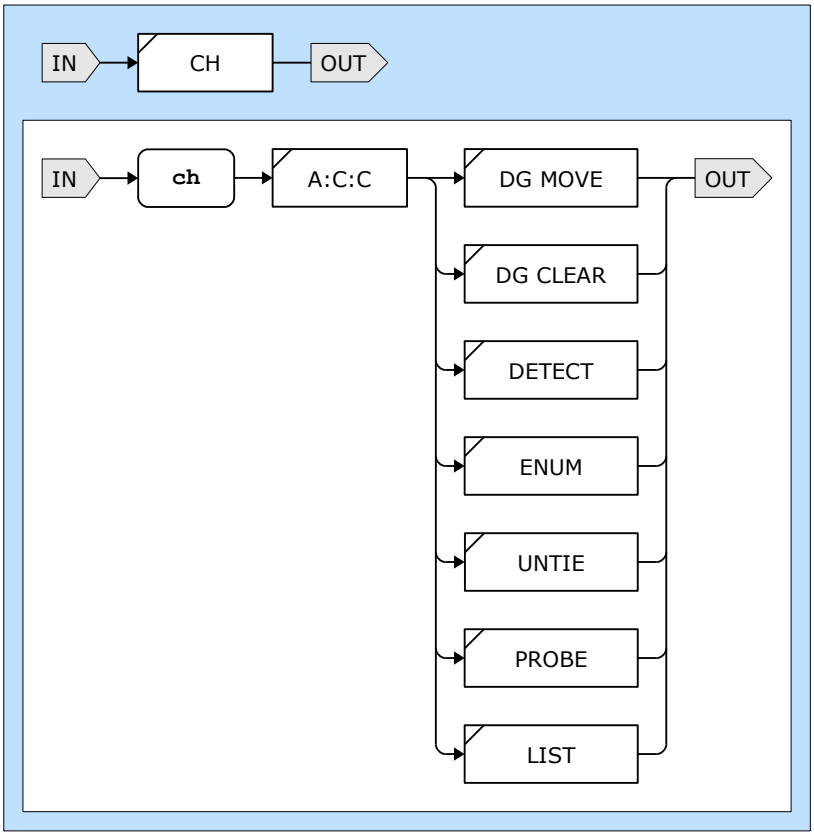
Block



Cancel

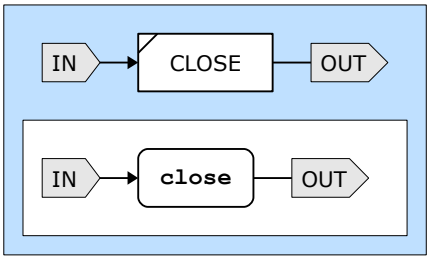


Channel



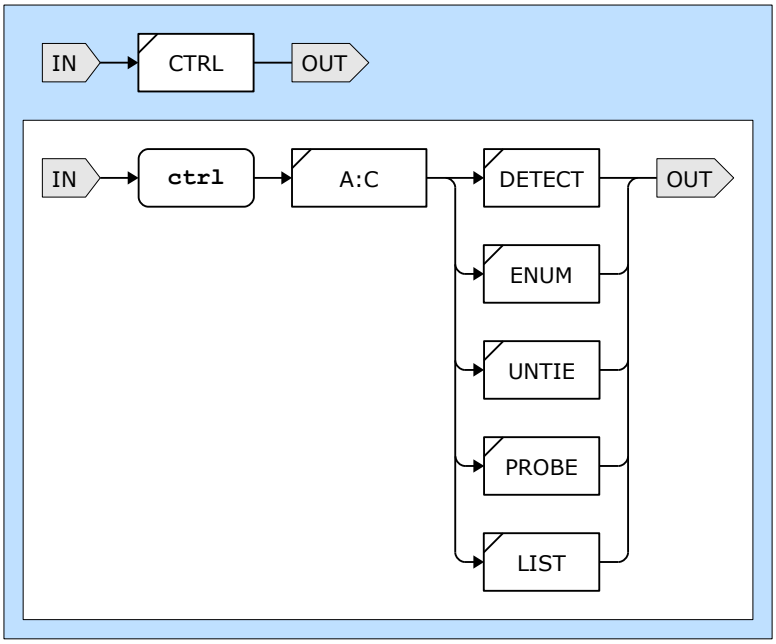
Base command for performing tasks on the specified channel.

Close

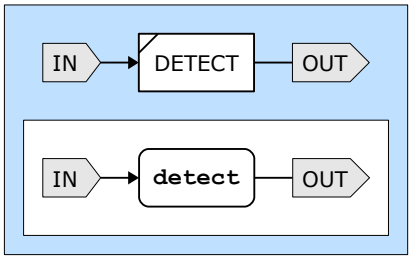


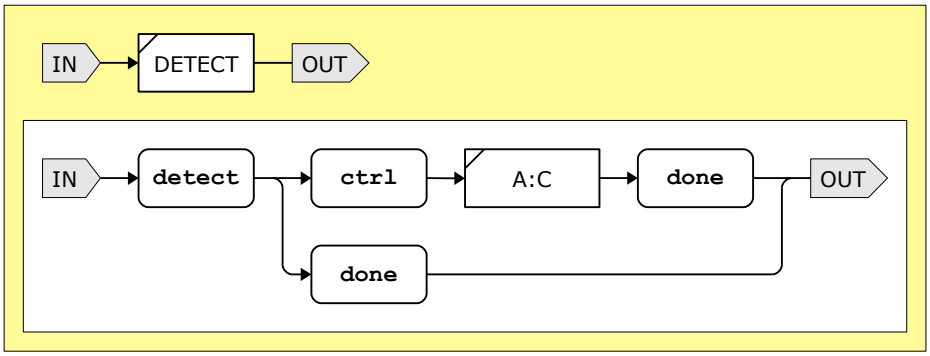
When the server executes this command, it closes the connection with the client.

Controller



Detect

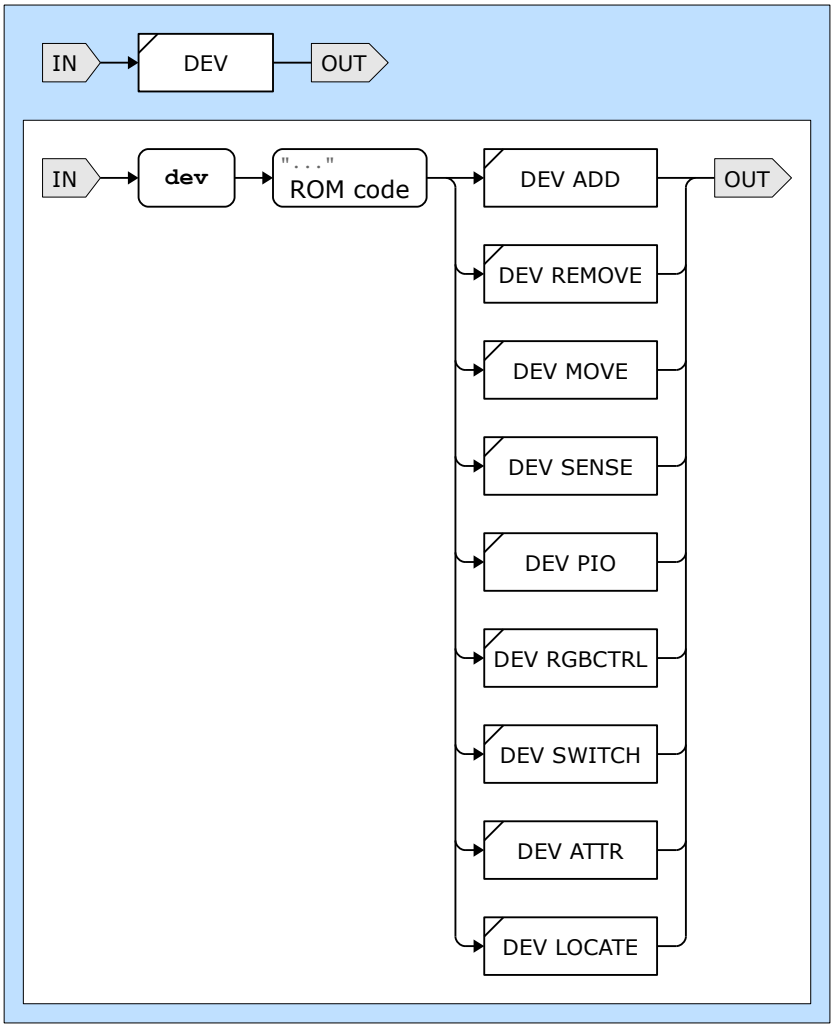




Run the detection procedure for a controller, optionally confined to a single channel.

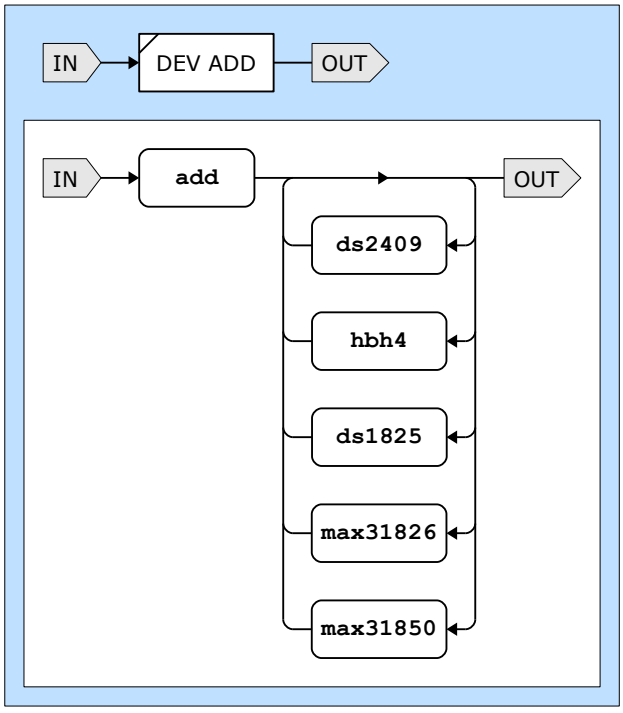
This macro command is expanded at the controller level.

Device



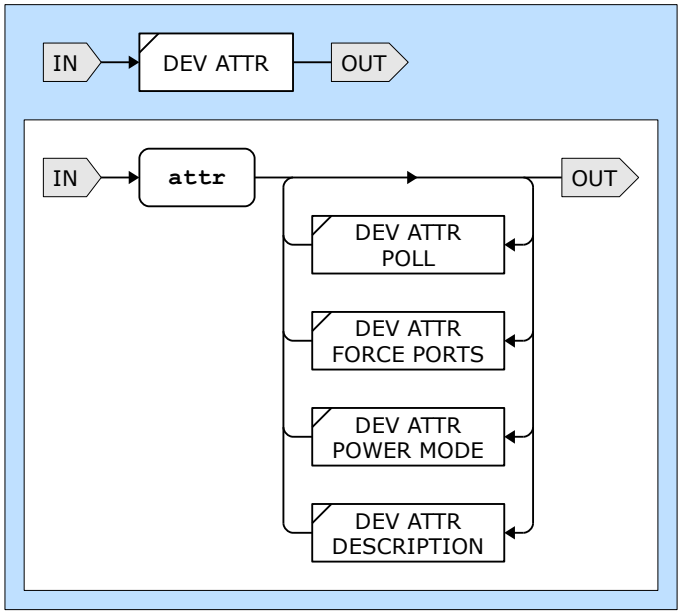
Base command for performing tasks on the specified 1-Wire slave.

Device Add



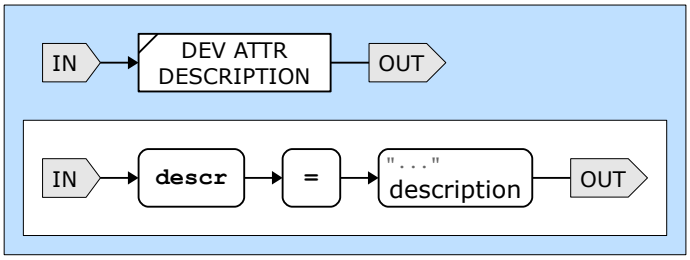
This command adds a 1-Wire slave. If the 1-Wire slave is already present in the topology, the command will be ignored.

Device Attribute

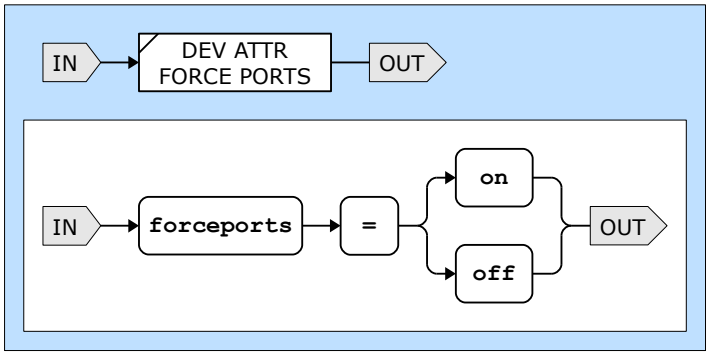


This command sets attributes of the specified 1-Wire slave.

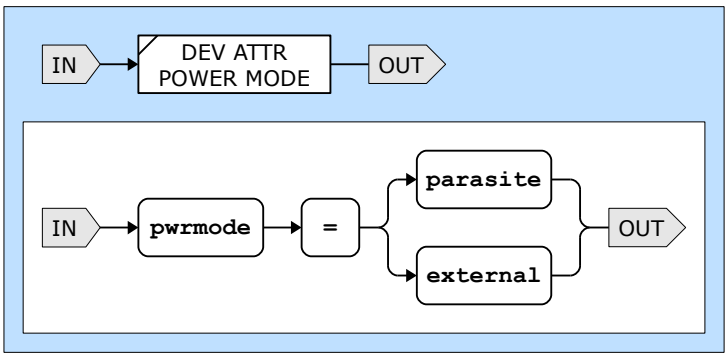
Device Attribute Description



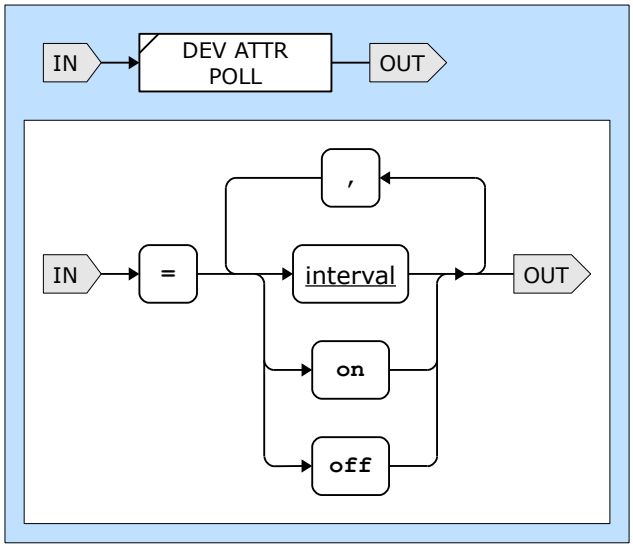
Device Attribute Force Ports



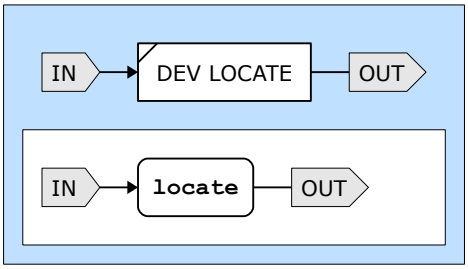
Device Attribute Power Mode



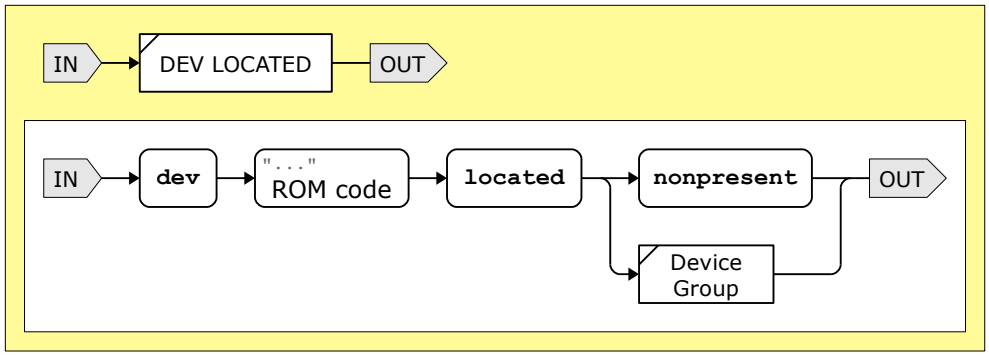
Device Attribute Poll



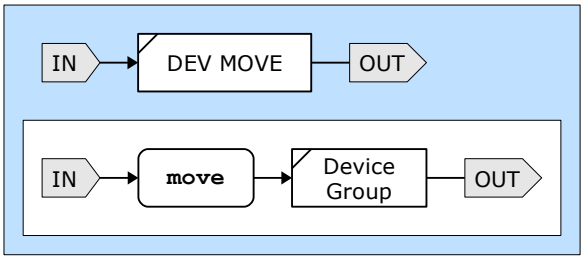
Device Locate



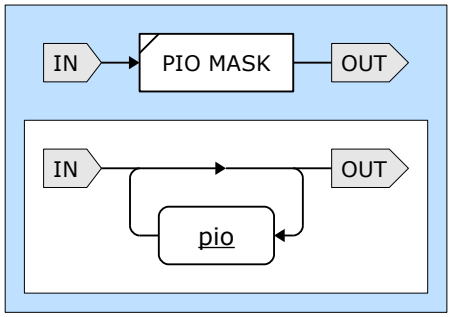
Device Located

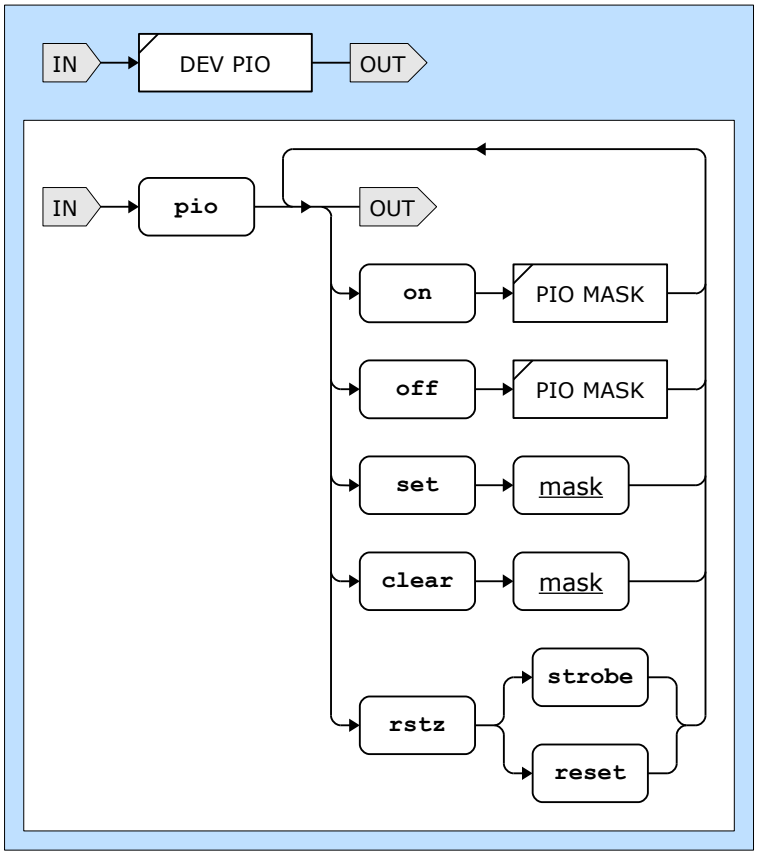


Device Move

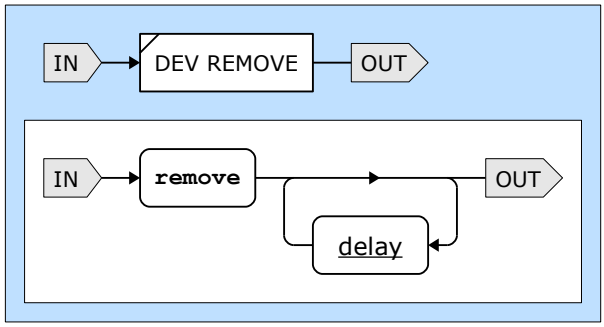


Device PIO


IN
PIO MASK
OUT
IN
OUT
pio


IN
DEV PIO
OUT
IN
pio
OUT
on
PIO MASK
off
PIO MASK
set
mask
clear
mask
strobe
rstz
reset

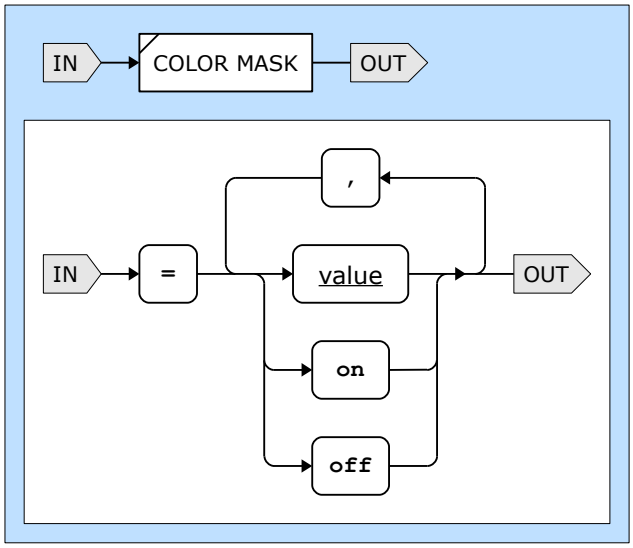
Device Remove

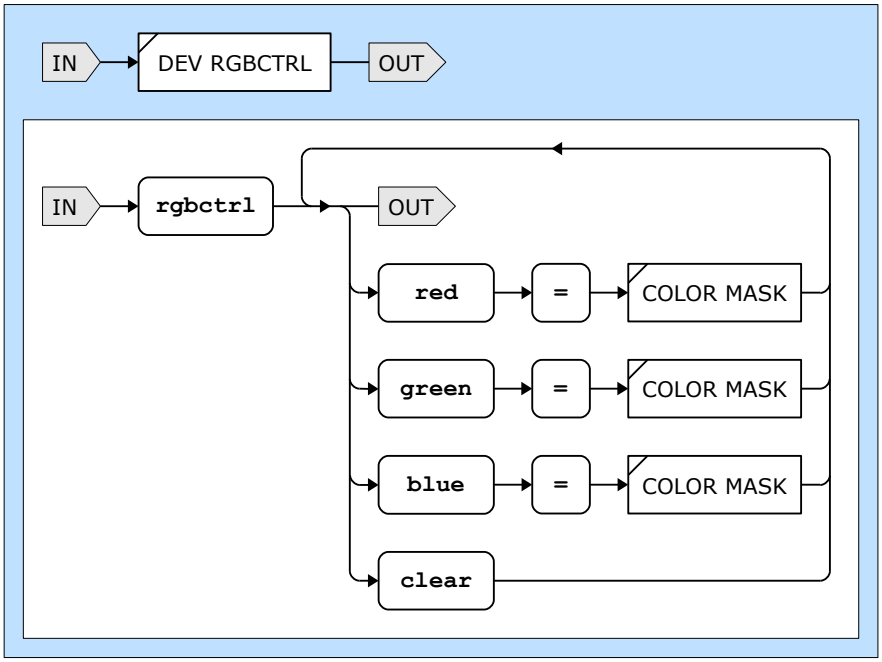


This command remove the 1-Wire slave with the given ROM code.

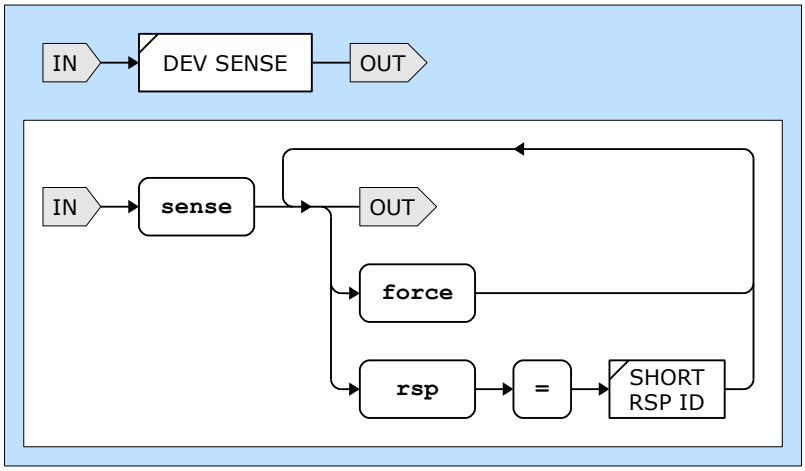
If a non-zero delay is present, the server will wait for this number of milliseconds before removing the 1-Wire slave.

Device RGB Controller

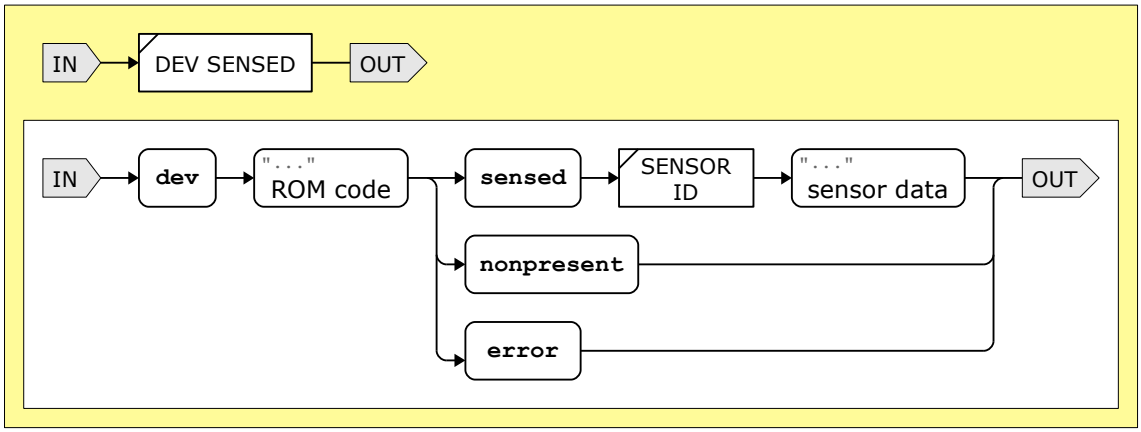




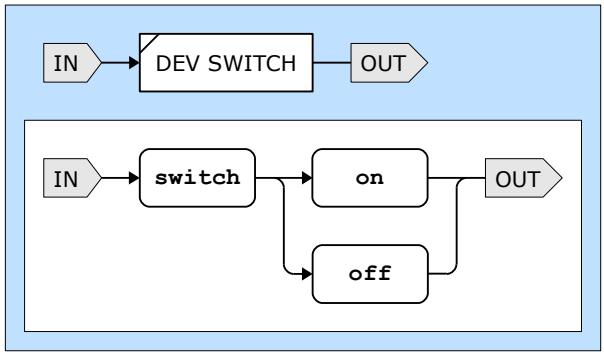
Device Sense



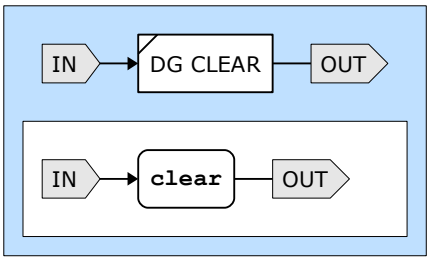
Device Sensed



Device Switch

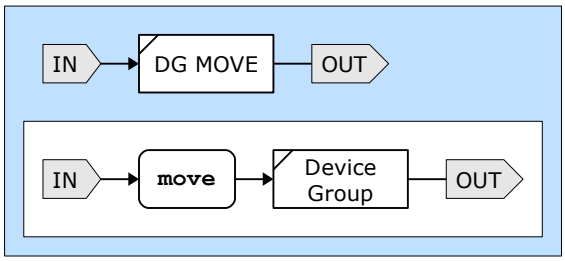


DG Clear



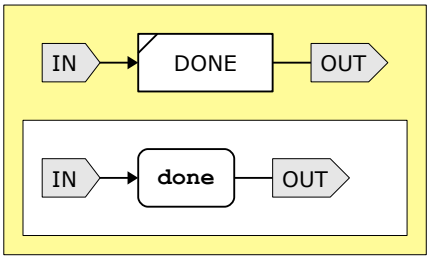
Clear a device group. All 1-Wire slaves in the device group are removed, thus become unknown to the server.

DG Move



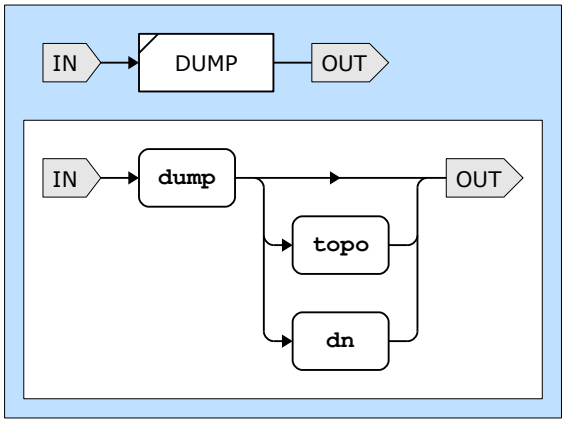
Move a device group to another location in the topology.

Done

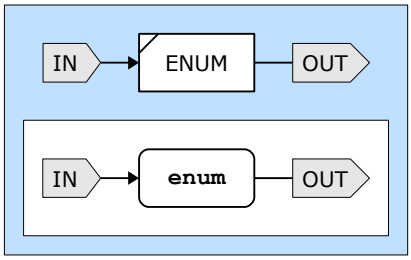


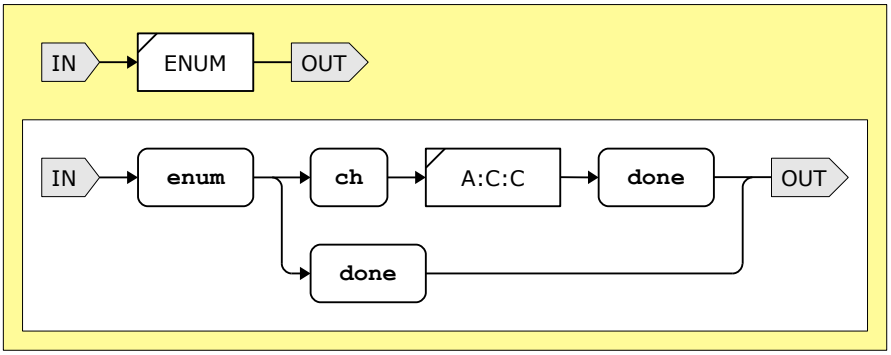
This response is generated when a command with an identifier has completed.

Dump



Enumerate

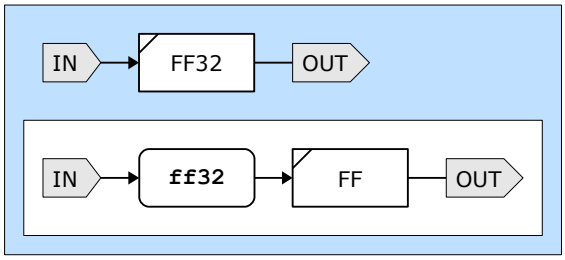




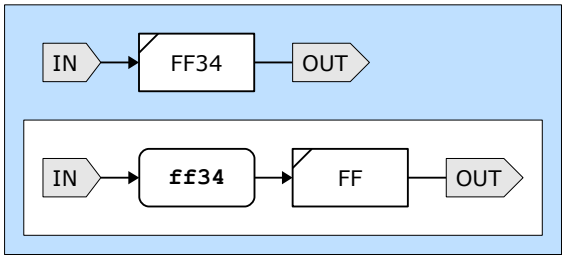
Enumerate 1-Wire slaves residing behind a channel.

This macro command is expanded at the channel level.

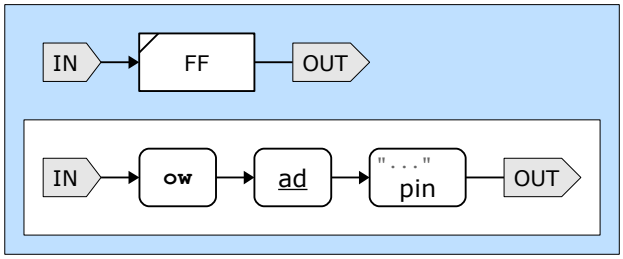
FF32


IN
FF32
OUT
IN
ff32
FF
OUT

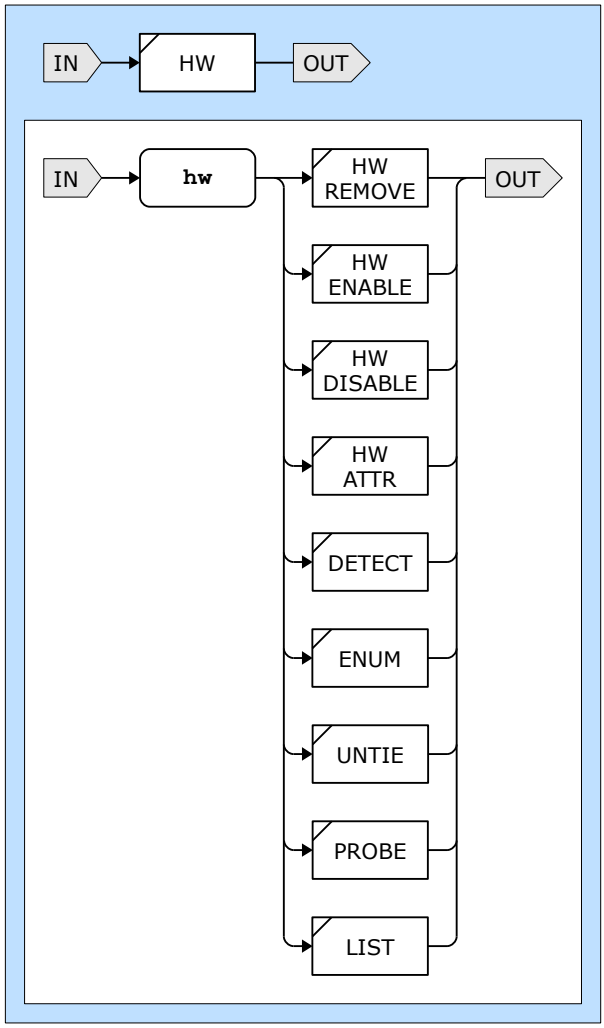
FF34


IN
FF34
OUT
IN
ff34
FF
OUT

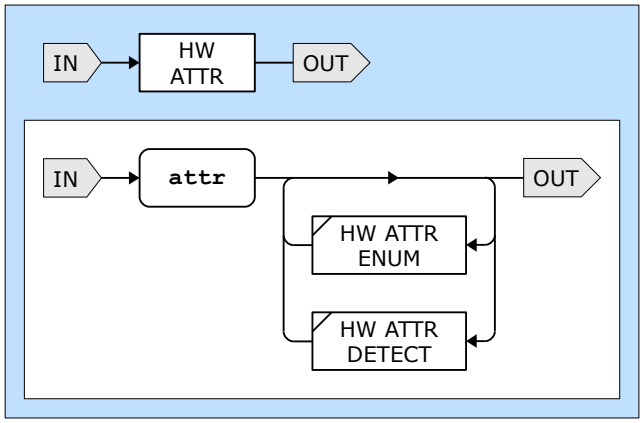
FF


IN
FF
OUT
IN
ow
ad
"..."
pin
OUT

Hardware

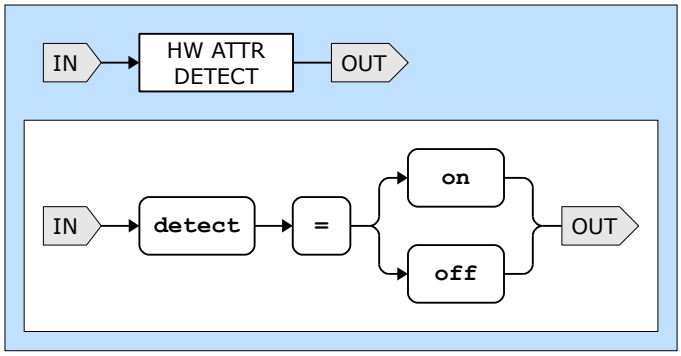

IN
HW
OUT
IN
hw
HW REMOVE
OUT
HW ENABLE
HW DISABLE
HW ATTR
DETECT
ENUM
UNTIE
PROBE
LIST

Hardware Attribute



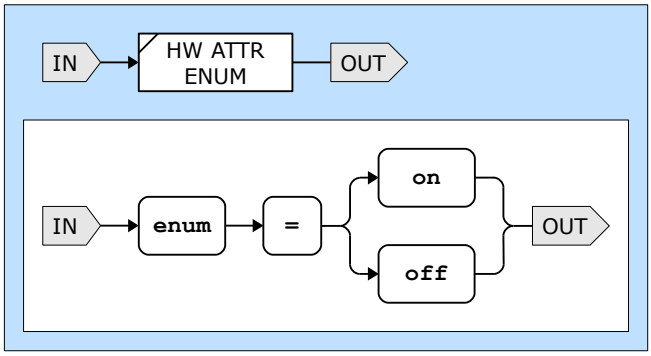
Set hardware attributes.

Hardware Attribute Detect



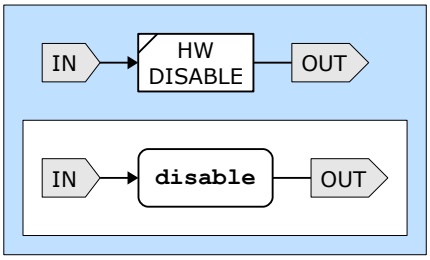
Turn on or off automatic detection for all adapters.

Hardware Attribute Enumerate



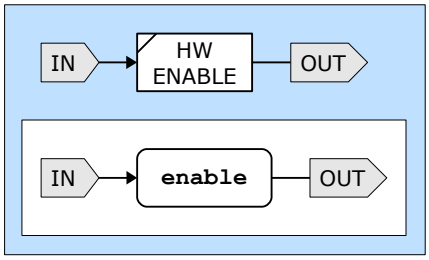
Turn on or off automatic enumeration for all adapters.

Hardware Disable



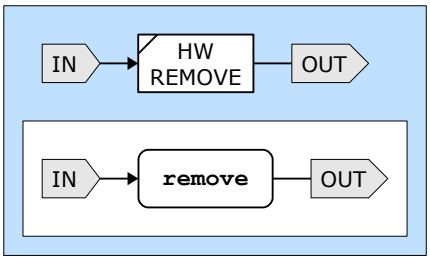
Enable all adapters. This is a macro command.

Hardware Enable



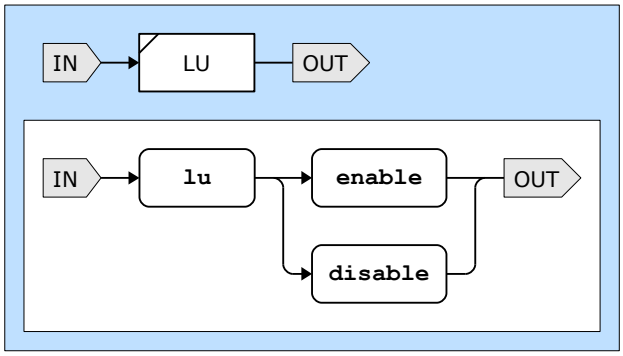
Disable all adapters. This is a macro command.

Hardware Remove

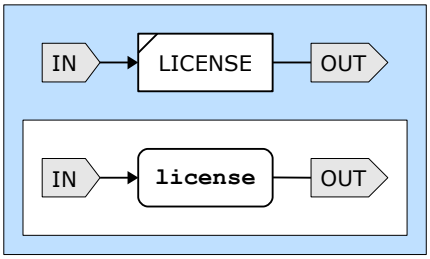


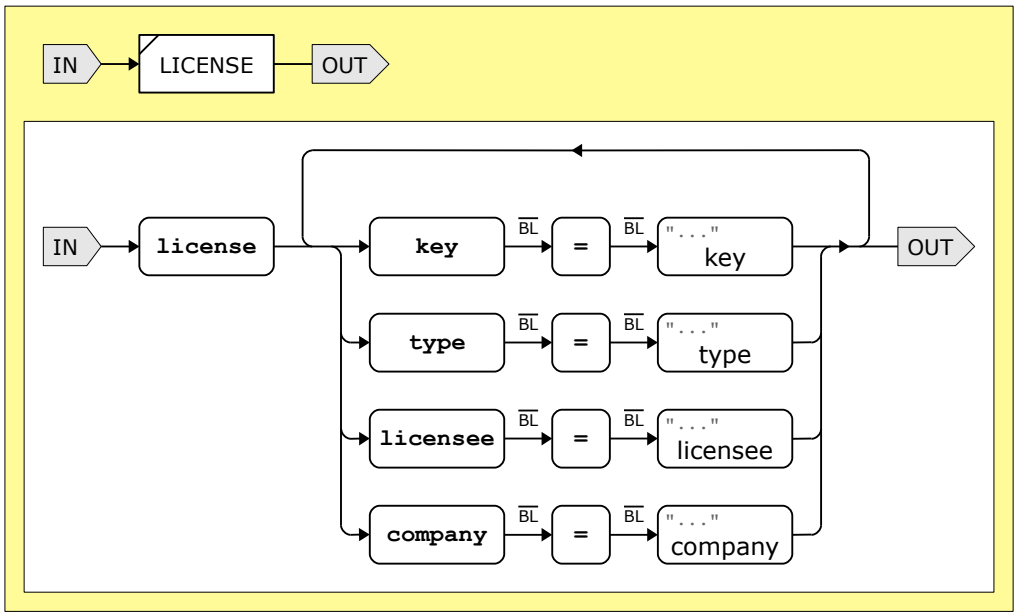
Remove all adapters. This is a macro command.

LibUSB

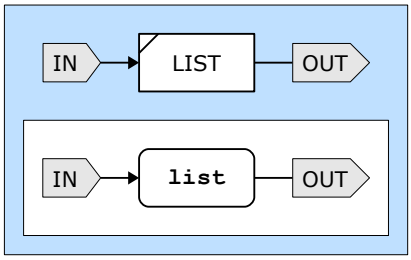


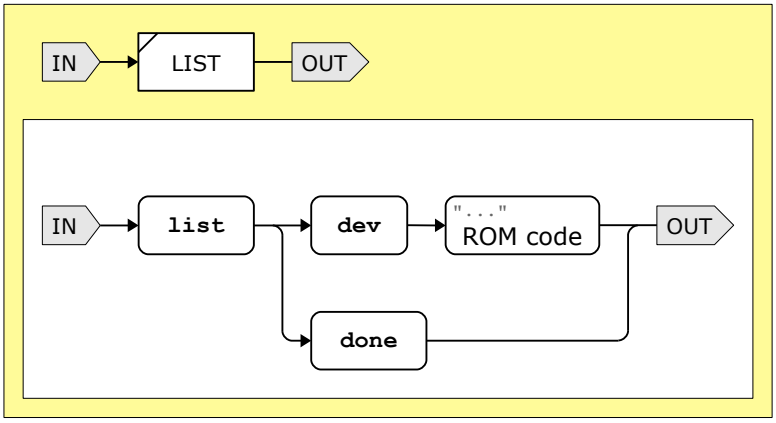
License



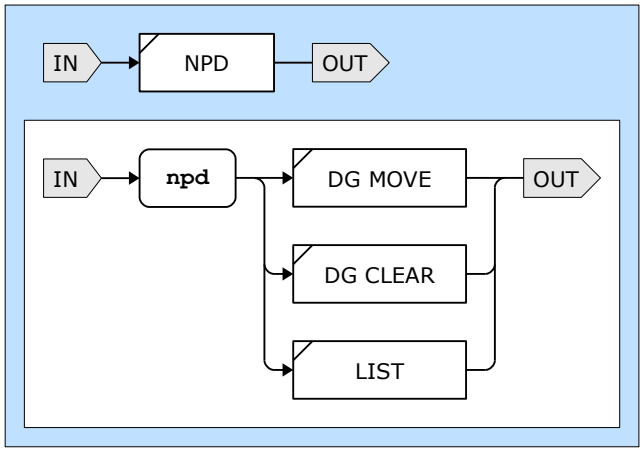


List


IN
LIST
OUT
IN
list
OUT

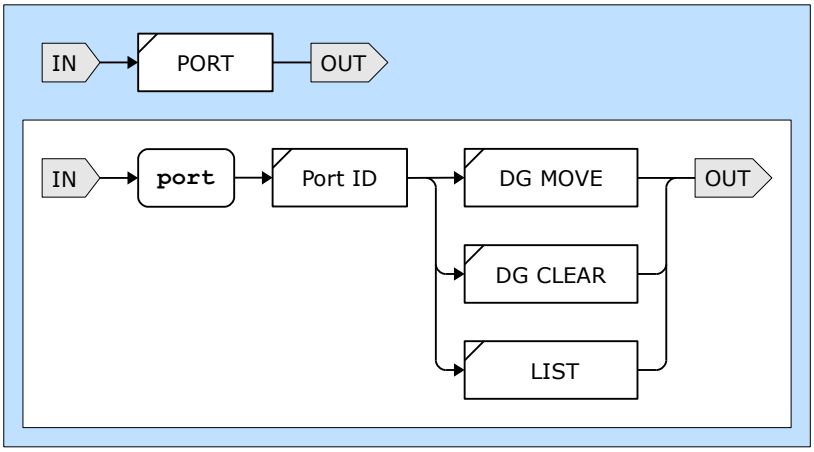

IN
LIST
OUT
IN
list
dev
"..."
ROM code
OUT
done

NPD



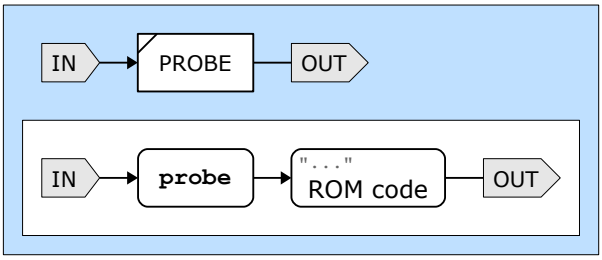
Base command for performing tasks on the non-present devices.

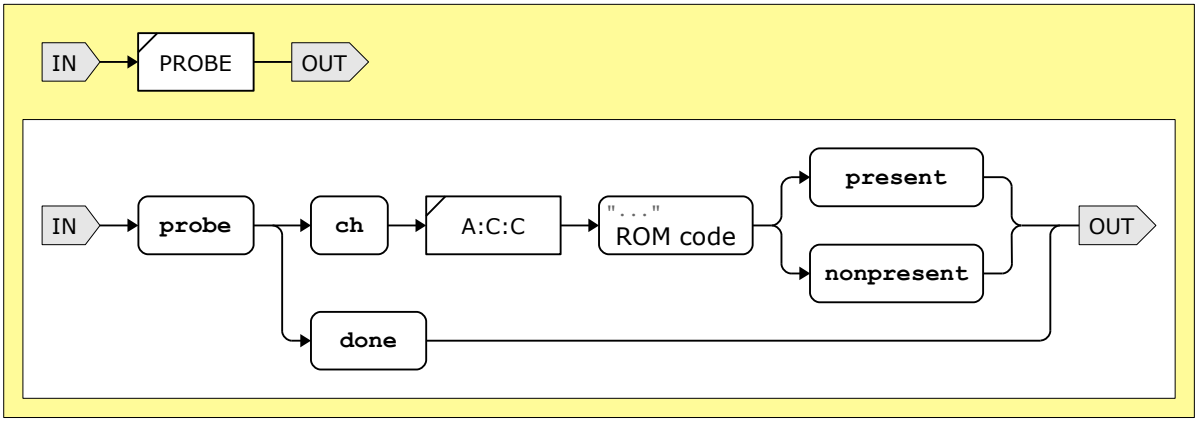
Port



Base command for performing tasks on a specified hub port.

Probe

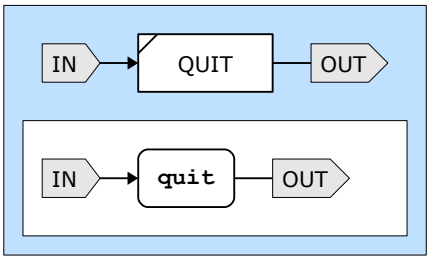




Probe the presence of a 1-Wire slave behind a channel. The ROM code represents the target 1-Wire slave.

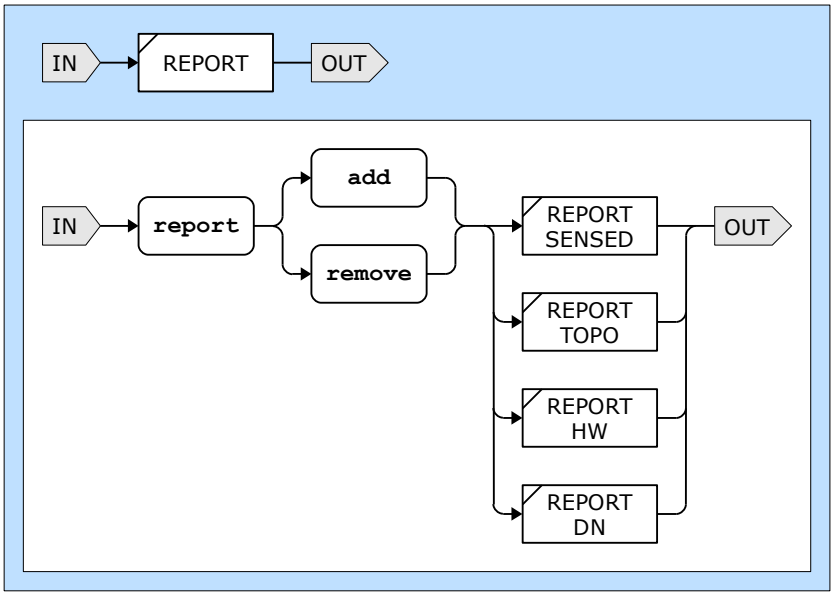
This macro command is expanded at the channel level.

Quit



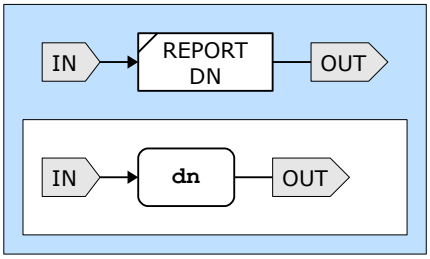
Quits the server.

Report

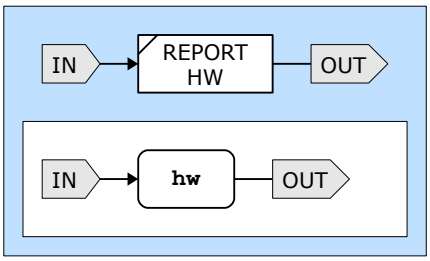


This command controls the unsolicited responses the server sends to the client. Use the command to add and remove pieces of information the client wants to receive.

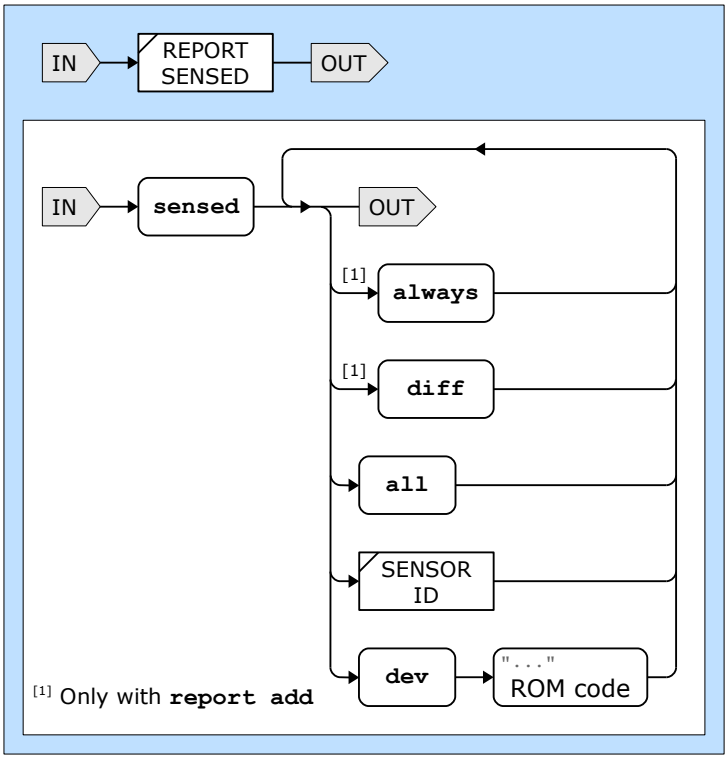
Report Device Nodes



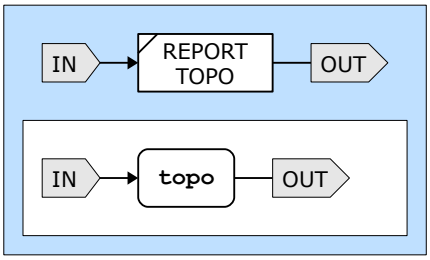
Report Hardware



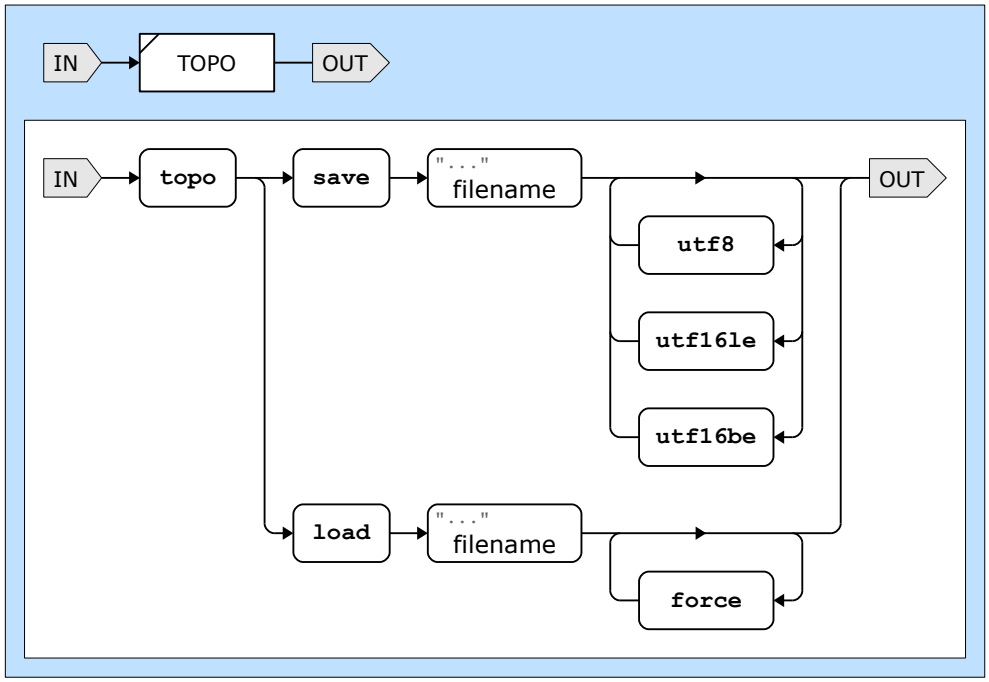
Report Sensed



Report Topology

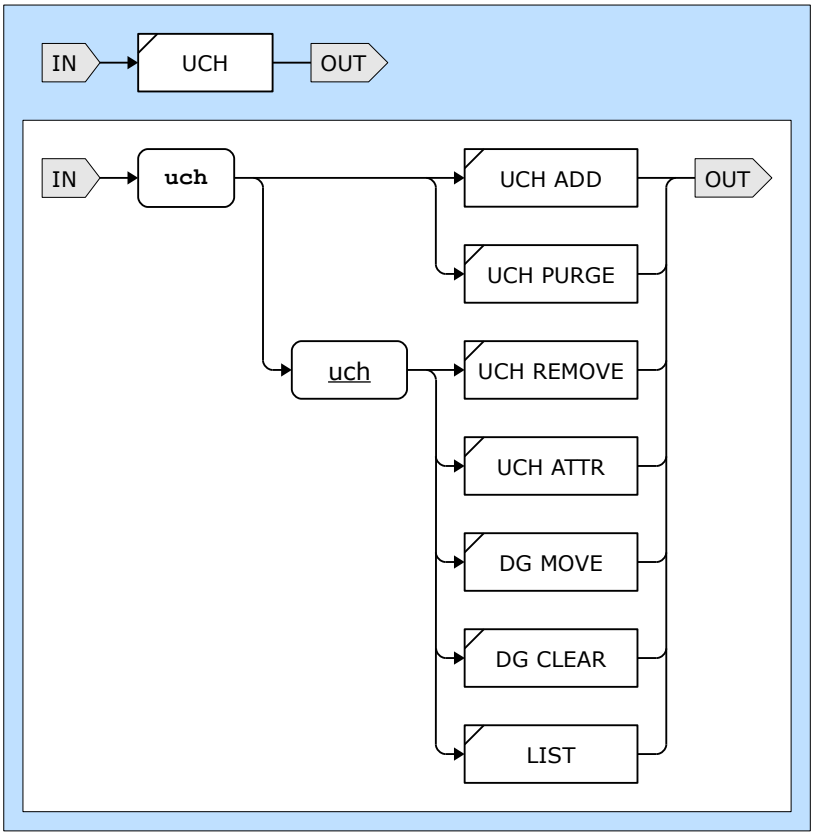


Topology



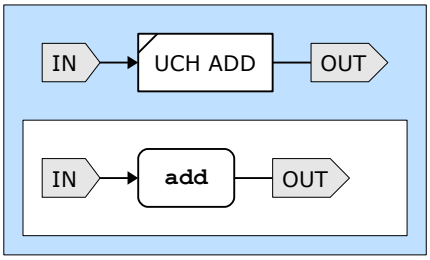
Commands for loading and saving topology files.

UCH



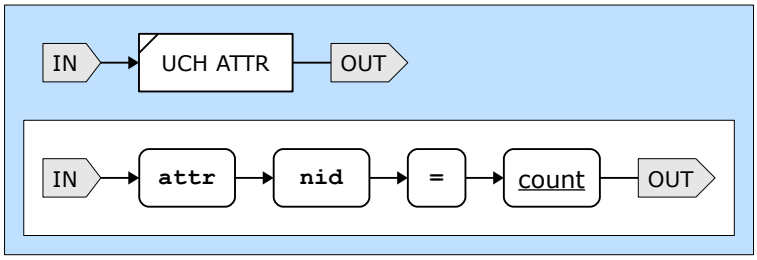
Base command for performing tasks on unallocated channels.

UCH Add



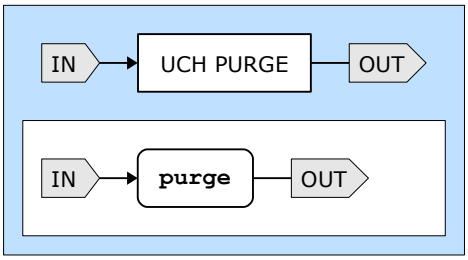
Add a new unallocated channel. The newly added unallocated channel is empty, meaning it contains no 1-Wire slaves.

UCH Attribute



Set attributes of an unallocated channel. Currently, you can set the number of non-identification slaves to be taken into account during a detection procedure.

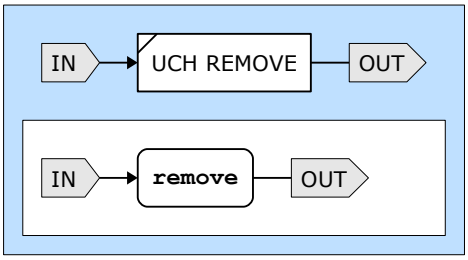
UCH Purge



Client command **UCH Purge** removes all empty unallocated channels.

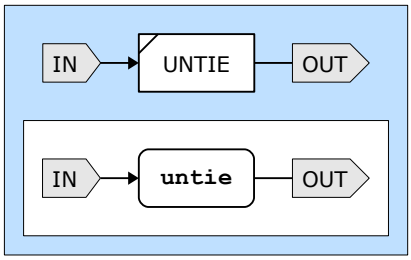
Empty unallocated channels usually pile up when a 1-Wire adapter appears and disappears regularly while it is set to enumerate automatically upon arrival. The enumeration typically finds all 1-Wire slaves that are stored in one or more unallocated channels and moves these slaves behind the adapter's channel or channels, resulting in empty unallocated channels. The idea is to execute the **UCH Purge** command periodically in such case.

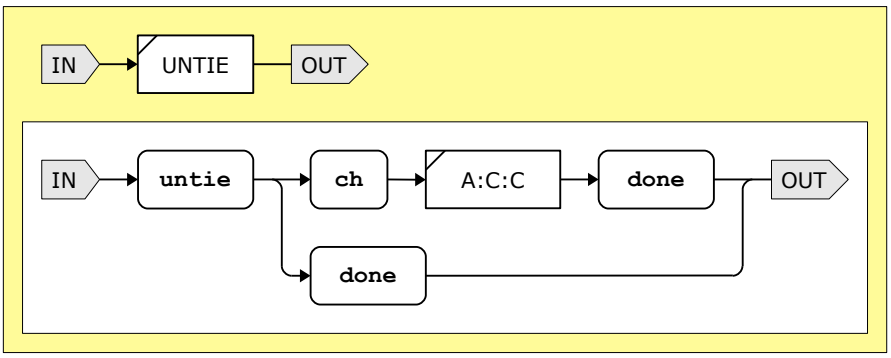
UCH Remove



Remove an unallocated channel. If 1-Wire slaves reside in the unallocated channel, the server moves them to the non-present devices first.

Untie

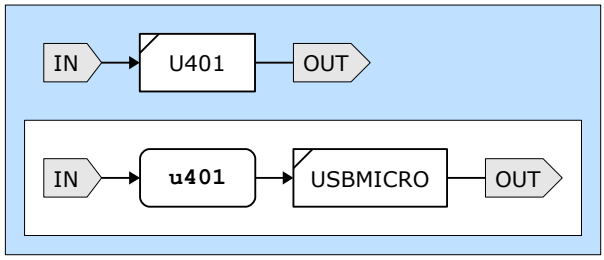




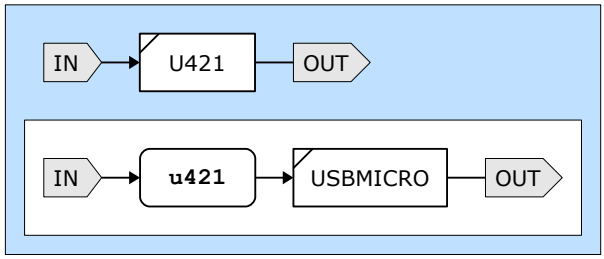
Untie all 1-Wire slaves residing behind a channel and move them to a new unallocated channel.

This macro command is expanded at the channel level.

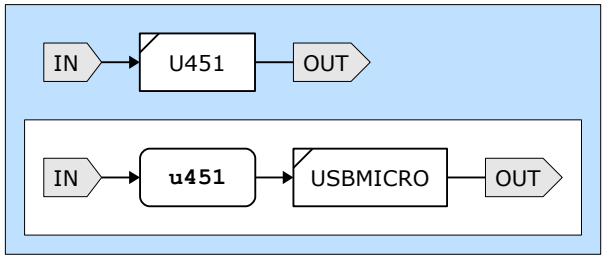
U401



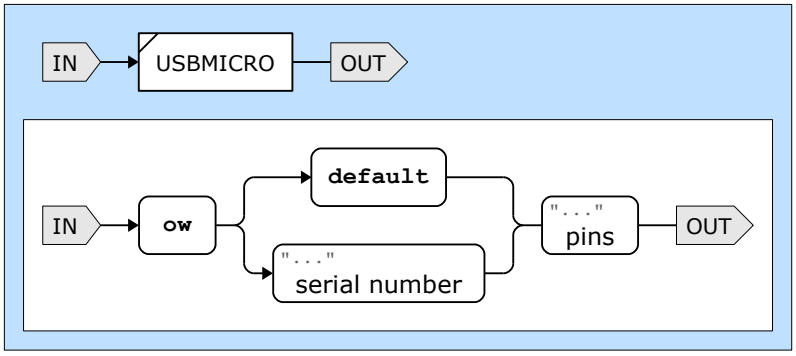
U421



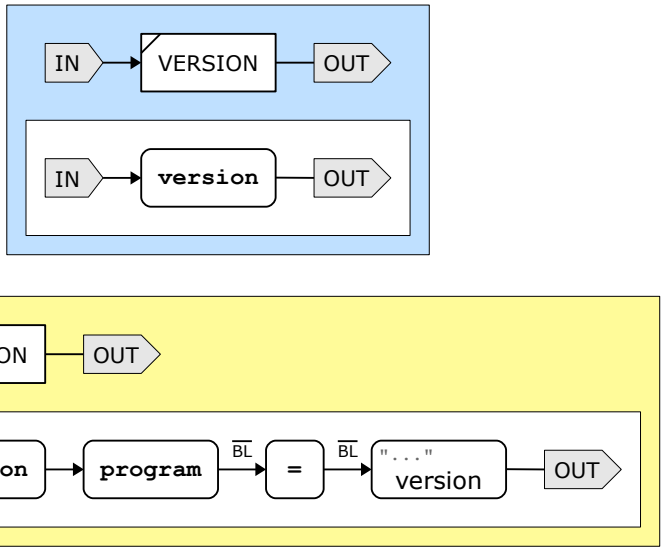
U451



USBMICRO

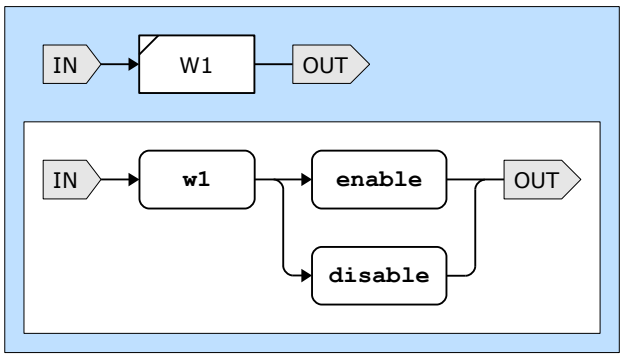


Version

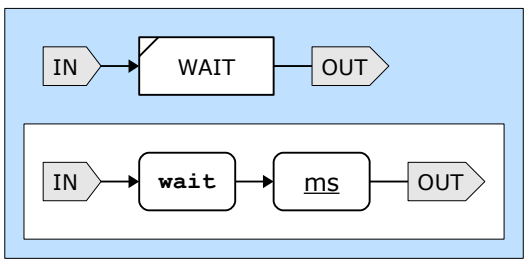


This command responds with version information. Currently, the program version of the server is returned as a major-minor-micro triplet.

W1



Wait



7 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.

8 Contact Information

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