

# simplyCAN

## USB-to-CAN Adapter

### USER MANUAL

4.01.0001.22001 1.0 en-US ENGLISH



---

# Important User Information

## Liability

Every care has been taken in the preparation of this document. Please inform HMS Industrial Networks of any inaccuracies or omissions. The data and illustrations found in this document are not binding. We, HMS Industrial Networks, reserve the right to modify our products in line with our policy of continuous product development. The information in this document is subject to change without notice and should not be considered as a commitment by HMS Industrial Networks. HMS Industrial Networks assumes no responsibility for any errors that may appear in this document.

There are many applications of this product. Those responsible for the use of this device must ensure that all the necessary steps have been taken to verify that the applications meet all performance and safety requirements including any applicable laws, regulations, codes, and standards.

HMS Industrial Networks will under no circumstances assume liability or responsibility for any problems that may arise as a result from the use of undocumented features, timing, or functional side effects found outside the documented scope of this product. The effects caused by any direct or indirect use of such aspects of the product are undefined, and may include e.g. compatibility issues and stability issues.

The examples and illustrations in this document are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular implementation, HMS Industrial Networks cannot assume responsibility for actual use based on these examples and illustrations.

## Intellectual Property Rights

HMS Industrial Networks has intellectual property rights relating to technology embodied in the product described in this document. These intellectual property rights may include patents and pending patent applications in the USA and other countries.

---

# Table of Contents

Page

<b>1</b>	<b>User Guide .....</b>	<b>3</b>
1.1	Target Audience.....	3
1.2	Document History .....	3
1.3	Trademark Information .....	3
1.4	Conventions .....	4
<b>2</b>	<b>Safety Instructions .....</b>	<b>5</b>
2.1	Information on EMC .....	5
2.2	General Safety Instructions .....	5
2.3	Intended Use.....	5
<b>3</b>	<b>Scope of Delivery .....</b>	<b>5</b>
<b>4</b>	<b>Product Description .....</b>	<b>6</b>
<b>5</b>	<b>Installation.....</b>	<b>7</b>
<b>6</b>	<b>Operation.....</b>	<b>8</b>
6.1	simplyCAN Bus Monitor .....	8
6.2	USB LED .....	9
6.3	CAN LED .....	9
<b>7</b>	<b>Additional Components .....</b>	<b>10</b>
7.1	CAN Bus Termination .....	10
<b>8</b>	<b>Technical Data .....</b>	<b>11</b>
<b>9</b>	<b>Troubleshooting .....</b>	<b>11</b>
<b>10</b>	<b>Cleaning .....</b>	<b>12</b>
<b>11</b>	<b>Support/Return Hardware.....</b>	<b>12</b>
11.1	Support .....	12
11.2	Return Hardware .....	12

---

<b>12 Disposal.....</b>	<b>12</b>
<b>13 API Documentation .....</b>	<b>13</b>
13.1 API Functions .....	13
13.1.1 simply_open .....	13
13.1.2 simply_close .....	13
13.1.3 simply_initialize_can.....	13
13.1.4 simply_identify .....	14
13.1.5 simply_start_can .....	14
13.1.6 simply_stop_can .....	15
13.1.7 simply_reset_can .....	15
13.1.8 simply_can_status .....	16
13.1.9 simply_set_filter .....	17
13.1.10 simply_receive.....	18
13.1.11 simply_send.....	18
13.1.12 simply_get_last_error .....	19
13.2 State Diagram.....	20
<b>A Regulatory Compliance .....</b>	<b>21</b>
A.1 EMC Compliance (CE) .....	21
A.2 Disposal and recycling.....	21

# 1 User Guide

Please read the manual carefully. Make sure you fully understand the manual before using the product.

## 1.1 Target Audience

This manual addresses trained personnel who are familiar with CAN and the applicable standards. The contents of the manual must be made available to any person authorized to use or operate the product.

## 1.2 Document History

Version	Date	Description
1.0	March 2019	First release

## 1.3 Trademark Information

Ixxat® is a registered trademark of HMS Industrial Networks AB. All other trademarks mentioned in this document are the property of their respective holders.

## 1.4 Conventions

Instructions and results are structured as follows:

- ▶ instruction 1
- ▶ instruction 2
  - result 1
  - result 2

Lists are structured as follows:

- item 1
- item 2

**Bold typeface** indicates interactive parts such as connectors and switches on the hardware, or menus and buttons in a graphical user interface.

This font is used to indicate program code and other kinds of data input/output such as configuration scripts.

This is a cross-reference within this document: [Conventions, p. 4](#)

This is an external link (URL): [www.hms-networks.com](http://www.hms-networks.com)

Safety advice is structured as follows:



Cause of the hazard!  
Consequences of not taking remediate action.  
How to avoid the hazard.

Safety signs and signalwords are used dependent on the level of the hazard.



*This is additional information which may facilitate installation and/or operation.*



This instruction must be followed to avoid a risk of reduced functionality and/or damage to the equipment, or to avoid a network security risk.



### Caution

This instruction must be followed to avoid a risk of personal injury.



### WARNING

This instruction must be followed to avoid a risk of death or serious injury.

## 2 Safety Instructions

### 2.1 Information on EMC



Risk of interference to radio and television if used in office or home environment! The product is a class B device.

Use exclusively included accessories or HMS accessories that are intended for use with the device. Use exclusively shielded cables.

Make sure that the shield of the interface is connected with the device plug and the plug on the other side.

### 2.2 General Safety Instructions

- ▶ Protect product from moisture and humidity.
- ▶ Protect product from too high or too low temperature (see [Technical Data, p. 11](#)).
- ▶ Protect product from fire.
- ▶ Do not paint the product.
- ▶ Do not modify or disassemble the product. Service must be carried out by HMS Industrial Networks.
- ▶ Store products in dry and dust-free place.

### 2.3 Intended Use

The device is used to connect computer systems to CAN networks to exchange data for example to configure a device via CAN or to read device diagnosis data. The simplyCAN is intended for the connection to a computer via the USB interface.

## 3 Scope of Delivery

Included in the scope of delivery:

- simplyCAN device

The following is available via download from [www.simplycan.info](http://www.simplycan.info):

- simplyCAN bus monitor
- installation file *setup.bat* (necessary for Windows 7)
- programming API
- programming examples
- user manual

A CAN bus termination can be ordered separately.

## 4 Product Description

The simplyCAN is an active USB adapter which enables the user to connect a computer with a CAN network to monitor the network traffic and to interact with other network devices. The simplyCAN is a plug and play device due to the easy installation and the easy-to-use CAN programming interface.

### Features

- USB 1.1 Full-Speed (12 MBit/s)
- 1 x CAN high-speed channel according to ISO 11898-2
- D-Sub 9 fieldbus connection, pin allocation according to CiA 303-1
- USB cable with plug type A



*Windows: the simplyCAN is tested on Windows 7 (32 bit and 64 bit) and Windows 10 (64 bit).*

*Linux: the simplyCAN is tested using Ubuntu 14.04 on Linux kernel version 4.4 as well as Ubuntu 18.04 on Linux kernel version 4.15.*

---



*To access the USB interface administrator rights might be necessary.*

---



## 5 Installation



**Insufficient power supply!**

Connect the device directly to the computer or to self-powered hubs to ensure sufficient power supply. Extension cables may cause connection issues.



*USB interface supports hot plug!*

*It is possible to plug or unplug the device during operation.*

On Windows 10 and Linux the USB interface is automatically installed when plugged in, without a driver installation.

- ▶ Download the simplyCAN zip file from [www.simplycan.info](http://www.simplycan.info) and unpack the file.
- ▶ On Windows 7 double-click *setup.bat* to install the driver for Windows 7.



*On Windows 10 the COM interface for the simplyCAN is showed as **USB Serial Device (COMx)** in the device manager when plugged in. To show the simplyCAN as **Ixxat simplyCAN (COMx)** in the device manager, execute the file *setup.bat* also on Windows 10.*

- ▶ Plug the USB connector in the USB port of computer.
  - Hardware is automatically found and installed.
  - USB LED is green flashing.
- ▶ If necessary install a bus termination (see [CAN Bus Termination, p. 10](#)).
- ▶ Connect the CAN fieldbus connector to the CAN fieldbus.
- ▶ Start the simplyCAN bus monitor (see [Operation, p. 8](#)).

### Connectors

The shield of the USB cable is connected to ground using a 100 nF capacitor. The shield of the CAN connector is connected to CAN ground via a 1 MΩ resistor and a 10 nF capacitor. USB\_shield is connected to CAN\_shield via a 4.7 nF capacitor.



*For best noise immunity connect the shields of the CAN cables directly to the device ground.*

#### Pin Allocation D-Sub 9

Signal	Pin No.
CAN high	7
CAN low	2
CAN GND	3, 6

## 6 Operation

### 6.1 simplyCAN Bus Monitor

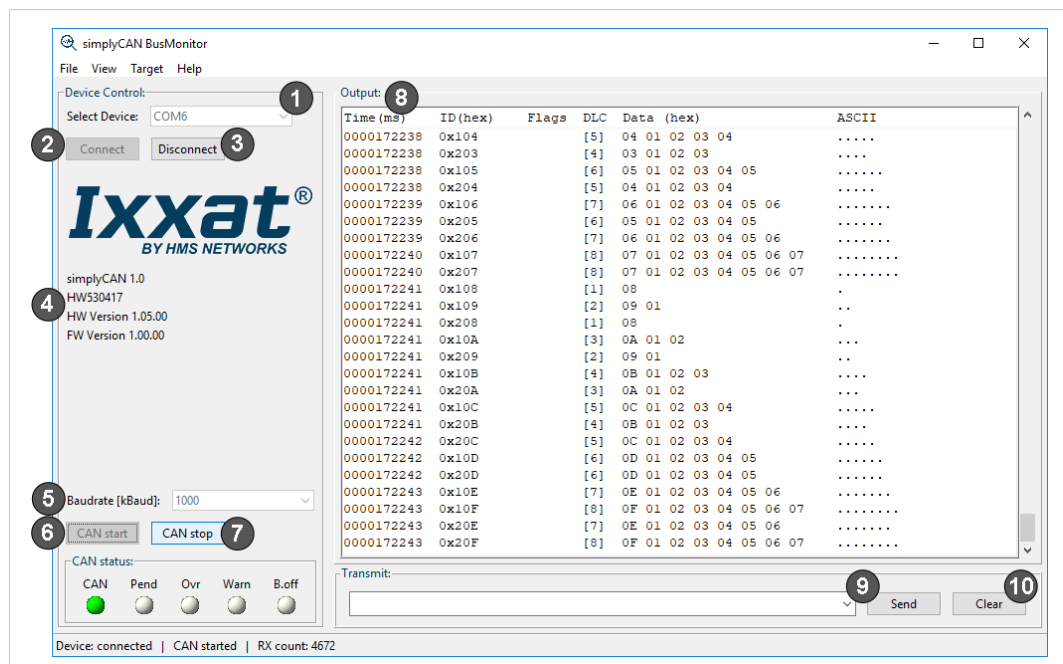


Fig. 1 simplyCAN bus monitor

- ▶ Start the simplyCAN bus monitor.
  - If one simplyCAN is connected to the computer, the device is automatically selected and connected (1).
- ▶ If several simplyCAN are connected to the computer, select the desired device (1) and click button **Connect** (2).
  - Information about the device are displayed (4).



The simplyCAN bus monitor can be opened several times to connect several simplyCAN devices simultaneously.

- ▶ To change the device, click button **Disconnect** (3), select the device in drop-down list **Select device** (1) and click button **Connect** (2).
- ▶ Select the desired CiA baudrate (5).
- ▶ To start the communication, click button **CAN start** (6).
  - CAN messages are shown in the window **Output** (8).
  - In transmitted messages the time stamp is 0 and the flag **S** is displayed.
- ▶ To send a message, enter the message in the **Transmit** line (9) (see [Transmit Messages, p. 9](#) for more information).
- ▶ Click button **Send** (9).
  - If the entered message is valid, the message is transmitted.
  - If the entered message is invalid, the error message **Syntax error** and a description of the message format is shown.
- ▶ Stop the communication with button **CAN stop** (7).

- Clear the output window with button **Clear (10)**.



For a busload higher than 65 % data loss is possible. Data loss is signalled to the application by the **Ovr** LED in the simplyCAN bus monitor.

### Transmit Messages

Syntax: **<id> [R] [E] [<data>...]**

- **id**: identifier (decimal or hexadecimal)
- **R**: remote transmit request for message
- **E**: message in Extended frame format (29 bit)
- **data**: data bytes of the message (decimal or hexadecimal), in RTR messages the first data byte contains the DLC



*If hexadecimal values are used, they must begin with 0x.*

*Example: 256 in dec is 0x100 in hex*

#### Examples

Message in simplyCAN bus monitor	Description
0x100 0x11 0x22 0x3 0x44	11 bit message with ID 100 (hex) and 4 data bytes
0x1FE1200 E 1 2 3 4 5 6 7 8	29 bit message with ID 1FE1200 (hex) and 8 data bytes
123 R 8	11 bit remote frame with ID 123 and DLC=8
0x1FE1200 R 8	29 bit remote frame with ID 1FE1200 (hex) and DLC=8

## 6.2 USB LED

The USB LED reflects the status of the USB communication.

LED state	Description	Comments
Off	Power off	No power or device defect
Green flashing	No active connection	Device ready to use, simplyCAN bus monitor or API must be started to use the device
Green	Active connection	Device in use

## 6.3 CAN LED

The CAN LED reflects the status of CAN communication.

LED state	Description	Comments
Off	No communication	No communication, device not connected to CAN
Green flashing	Communication OK	LED is triggered with each message.
Red flashing	Communication with errors	Controller is in state <i>error warning</i> or in state <i>error passive</i> , communication is possible.
Red	Bus off	Controller is in state <i>bus off</i> , no communication possible.

## 7 Additional Components

### 7.1 CAN Bus Termination

There is no bus termination resistor for the CAN bus integrated in the device. HMS Industrial Networks offers a bus termination resistor as a feed through connector.



**Fig. 2** CAN bus termination resistor

- For ordering information see [www.ixxat.com](http://www.ixxat.com).

## 8 Technical Data

USB interface	USB 1.1, Full-Speed (12 MBit/s)
CAN bitrates	10 kbit/s to 1 Mbit/s, only CiA recommended bit rates are supported: 10, 20, 50, 125, 250, 500, 800, 1000
CAN transceiver	TI SN65HVD251
CAN bus termination	None
Dimensions	80 x 50 x 22 mm
Weight	Approx. 100 g
Power supply	Via USB, 5 V DC/100 mA
Galvanic isolation	800 V DC/500 V AC for 1 min
Operating temperature	-20 to +70 °C
Storage temperature	-40 to +85 °C
Relative humidity	10 % to 95 %, non condensing
Housing material	ABS plastic
Protection class	IP40

## 9 Troubleshooting

### USB LED is off after connecting.

No power or defect device

- ▶ Make sure that the device is correctly connected to the USB port.
- ▶ Connect the device directly to the computer or to self-powered hubs.

### Extension cable is used and device is not working.

Extension cables may cause connection issues.

- ▶ Remove the extension cable.
- ▶ Connect the device directly or via an active USB hub to the computer.

## 10 Cleaning

- ▶ Disconnect the device from power supply.
- ▶ Remove dirt with a soft, chemical untreated, dry cloth.

## 11 Support/Return Hardware

Observe the following information in the support area on [www.ixxat.com](http://www.ixxat.com):

- information about products
- FAQ lists
- installation notes
- updated product versions
- updates

### 11.1 Support

- ▶ For problems or support with the product request support at [www.ixxat.com/support](http://www.ixxat.com/support).
- ▶ If required use support phone contacts on [www.ixxat.com](http://www.ixxat.com).

### 11.2 Return Hardware

- ▶ Fill in the form for warranty claims and repair on [www.ixxat.com](http://www.ixxat.com).
- ▶ Print out the Product Return Number (PRN resp. RMA).
- ▶ Pack product in a physically- and ESD-safe way, use original packaging if possible.
- ▶ Enclose PRN number.
- ▶ Observe further notes on [www.ixxat.com](http://www.ixxat.com).
- ▶ Return hardware.

## 12 Disposal

- ▶ Dispose of product according to national laws and regulations.
- ▶ Observe further notes about disposal of products on [www.ixxat.com](http://www.ixxat.com).

## 13 API Documentation

### 13.1 API Functions



To show the use of the functions, examples in C, C# and Python are available on [www.simplycan.info](http://www.simplycan.info).

#### 13.1.1 simply\_open

Opens the serial communication interface. The message filter of the CAN controller is opened for all message identifiers.

```
bool simply_open(char *serial_port);
```

##### Parameter

Parameter	Dir.	Description
<i>serial_port</i>	[in]	Name of the serial communication port (e.g. <i>COM1</i> or <i>/dev/ttyACM0</i> ). Use the simplyCAN bus monitor to detect on which serial COM port the simplyCAN is connected. With Windows it is also possible to use the device manager and with Linux the command <code>ls -l /dev/serial/by-id</code> .

##### Return Value

Return value	Description
true	Function succeeded
false	Error occurred, call <code>simply_get_last_error</code> for more information.

#### 13.1.2 simply\_close

Closes the serial communication interface and resets the CAN controller.

```
bool simply_close(void);
```

##### Return Value

Return value	Description
true	Function succeeded
false	Error occurred, call <code>simply_get_last_error</code> for more information.

#### 13.1.3 simply\_initialize\_can

Initializes the CAN controller.

```
bool simply_initialize_can(uint16_t bitrate);
```

##### Parameter

Parameter	Dir.	Description
<i>bitrate</i>	[in]	CAN bitrate as integer value, possible values: 10, 20, 50, 125, 250, 500, 800, 1000

##### Return Value

Return value	Description
true	Function succeeded
false	Error occurred, call <code>simply_get_last_error</code> for more information.

### 13.1.4 simply\_identify

Gets firmware and hardware information about the simplyCAN device.

```
bool simply_identify(identification_t *p_identification);
```

#### Parameter

Parameter	Dir.	Description
<i>p_identification</i>	[out]	Pointer to the identification structure

#### Identification Structure

```
typedef struct _identification {  
    uint8_t fw_version[8];  
        // Zero terminated firmware version string e.g. "1.00.00"  
    uint8_t hw_version[8];  
        // Zero terminated hardware version string e.g. "1.00.00"  
    uint8_t product_version[8];  
        // Zero terminated product version string e.g. "1.00.00"  
    uint8_t product_string[30];  
        // Zero terminated product string e.g. "simplyCAN 1.0"  
    uint8_t serial_number[9];  
        // Zero terminated serial number e.g. "HW123456"  
} identification_t;
```

#### Return Value

Return value	Description
true	Function succeeded
false	Error occurred, call <code>simply_get_last_error</code> for more information.

### 13.1.5 simply\_start\_can

Starts the CAN controller. Sets the CAN controller into running mode and clears the CAN message FIFOs. In running mode CAN messages can be transmitted and received.

```
bool simply_start_can(void);
```

#### Return Value

Return value	Description
true	Function succeeded
false	Error occurred, call <code>simply_get_last_error</code> for more information.



### 13.1.6 **simply\_stop\_can**

Stops the CAN controller. Sets the CAN controller into mode *init*. Does not reset the message filter of the CAN controller. Only stop the CAN controller when the `CAN_STATUS_PENDING` flag is not set.

```
bool simply_stop_can(void);
```

#### Return Value

Return value	Description
true	Function succeeded
false	Error occurred, call <code>simply_get_last_error</code> for more information.

#### Remarks

To ensure that all messages are transmitted before stopping the CAN controller, read the CAN status until the `CAN_STATUS_PENDING` flag is no longer set.

### 13.1.7 **simply\_reset\_can**

Resets the CAN controller (hardware reset) and clears the message filter (open for all message identifiers). Sets the CAN controller into mode *init*.

```
bool simply_reset_can(void);
```

#### Return Value

Return value	Description
true	Function succeeded
false	Error occurred, call <code>simply_get_last_error</code> for more information.

### 13.1.8 simply\_can\_status

Gets the status of the CAN controller.

```
bool simply_can_status(can_sts_t *can_sts);
```

#### Parameter

Parameter	Dir.	Description
<i>can_sts</i>	[out]	Status as bit coded 16 bit value (see CAN status structure)

#### CAN Status Structure

```
typedef struct _can_sts {  
    uint16_t sts;  
    // bit coded status flags (see CAN status definitions)  
    uint16_t tx_free;  
    // number of free elements in CAN message tx fifo  
} can_sts_t;
```

#### CAN Status Definitions

```
/* CAN status definitions */  
#define CAN_STATUS_RUNNING          (0x01)  
#define CAN_STATUS_RESET            (0x02)  
#define CAN_STATUS_BUSOFF           (0x04)  
#define CAN_STATUS_ERRORSTATUS      (0x08)  
#define CAN_STATUS_RXOVERRUN        (0x10)  
#define CAN_STATUS_TXOVERRUN        (0x20)  
#define CAN_STATUS_PENDING          (0x40)
```

#### Return Value

Return value	Description
true	Function succeeded
false	Error occurred, call <code>simply_get_last_error</code> for more information.

### 13.1.9 simply\_set\_filter

Sets the 11 or 29 bit message filter of the CAN controller. To set the 29 bit message filter, the MSB in parameter value must be set.

```
bool simply_set_filter(uint32_t mask, uint32_t value);
```

#### Parameter

Parameter	Dir.	Description
<i>mask</i>	[in]	11 or 29 bit CAN message identifier mask
<i>value</i>	[in]	11 or 29 bit CAN message identifier value, set MSB to set the 29 bit message filter

#### Return Value

Return value	Description
true	Function succeeded
false	Error occurred, call <code>simply_get_last_error</code> for more information.

#### Remark

With the mask/value filter (available for 11 bit and 29 bit identifiers) possible valid identifiers based on bit masks can be defined.

Binary representation of mask:

- binary positions with value 1 are relevant for the filter
- binary positions with value 0 are not relevant for the filter

Binary representation of value:

- Defines the values for the positions that are marked as relevant (1) in mask.
- Values in positions that are marked as not relevant (0) in mask are ignored.

The following formula expresses the condition under which an identifier passes the filter:

- if  $(value \& mask) == (identifier \& mask)$  then identifier is valid

#### Example 11 Bit Identifier

	hex	bin
<b>Value</b>	0x700	0111:0000:0000
<b>Mask</b>	0x700	0111:0000:0000
<b>Result</b>	0x700	0111:0000:0000
Any identifier between 0x700 and 0x7FF passes the filter, as only the first 3 bits of the mask are marked as relevant.		

#### Example 29 Bit Identifier

	hex	bin
<b>Value</b>	0x90003344	1001:0000:0000:0000:0011:0011:0100:0100
<b>Mask</b>	0x1F00FFFF	0001:1111:0000:0000:1111:1111:1111:1111
<b>Result</b>	0x10003344	0001:0000:0000:0000:0011:0011:0100:0100
256 identifier between 0x10003344 and 0x10FF3344 pass the filter, where the last two bytes are 0x3344.		

To allow 29 bit messages to pass the filter, the MSB in parameter value must be set.

#### Further Examples

Value	Mask	Valid message identifiers which pass the filter
0x100	0x7FF	0x100
0x100	0x700	0x100–0x1FF
0x000	0x000	0x000–0x7FF

### 13.1.10 `simply_receive`

Receives a single CAN message.

```
int8_t simply_receive(can_msg_t *can_msg);
```

#### Parameter

Parameter	Dir.	Description
<code>can_msg</code>	[out]	Pointer to the CAN message structure the received CAN message is copied to

#### CAN Message Structure

```
typedef struct _can_msg {  
    uint32_t timestamp;           // in milliseconds  
    uint32_t ident;               // MSB=1: extended frame  
    uint8_t dlc;                  // MSB=1: remote frame  
    uint8_t payload[8];  
} can_msg_t;
```

#### Return Value

Return value	Description
1	Message received
0	No message available in the receive queue
-1	Error occurred, call <code>simply_get_last_error</code> for more information.

### 13.1.11 `simply_send`

Writes a CAN message to the transmit FIFO. To check if the message is transmitted, request the CAN status with [`simply\_can\_status`](#).

```
bool simply_send(can_msg_t *can_msg);
```

#### Parameter

Parameter	Dir.	Description
<code>can_msg</code>	[in]	Pointer to a CAN message to be transmitted (see <a href="#">CAN Message Structure</a> )

#### Return Value

Return value	Description
true	Function succeeded
false	Error occurred, call <code>simply_get_last_error</code> for more information.

#### Remark

With `simply_send` CAN messages are not transmitted immediately but they are written to the transmit FIFO. Check if the message is transmitted with [`simply\_can\_status`](#). If the `CAN_STATUS_PENDING` flag is set, the message is not yet transmitted. Request the CAN status until the `CAN_STATUS_PENDING` flag is not set any more.

### 13.1.12 simply\_get\_last\_error

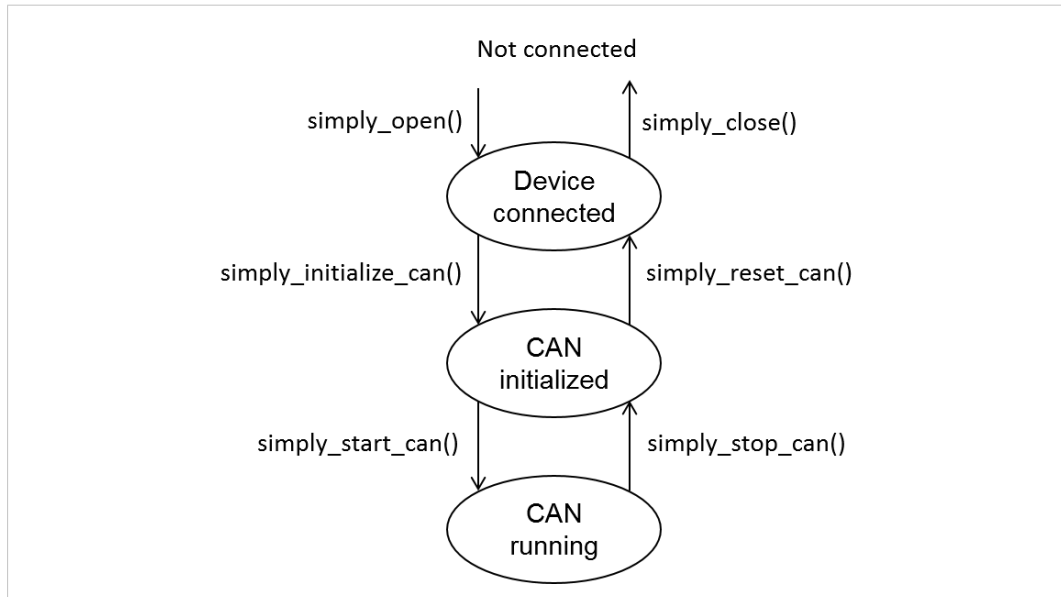
Gets the last error code. After reading the error code with `simply_get_last_error` the error code is set to 0. Each error can only be read once.

```
int16_t simply_get_last_error(void);
```

#### Return Value

Return value	Error	Description
0	<code>SIMPLY_S_NO_ERROR</code>	No error occurred
-1	<code>SIMPLY_E_SERIAL_OPEN</code>	Unable to open the serial port
-2	<code>SIMPLY_E_SERIAL_ACCESS</code>	Access on serial port denied
-3	<code>SIMPLY_E_SERIAL_CLOSED</code>	Serial communication port closed
-4	<code>SIMPLY_E_SERIAL_COMM</code>	Serial communication error
-5	<code>SIMPLY_E_CMND_REQ_UNKNOWN</code>	Command unknown on device
-6	<code>SIMPLY_E_CMND_RESP_TIMEOUT</code>	Command response timeout reached
-7	<code>SIMPLY_E_CMND_RESP_UNEXPECTED</code>	Unexpected command response received
-8	<code>SIMPLY_E_CMND_RESP_ERROR</code>	Command response error
-9	<code>SIMPLY_E_INVALID_PROTOCOL_VERSION</code>	Invalid simplyCAN protocol version
-10	<code>SIMPLY_E_INVALID_FW_VERSION</code>	Invalid device firmware version
-11	<code>SIMPLY_E_INVALID_PRODUCT_STRING</code>	Invalid simplyCAN product string
-12	<code>SIMPLY_E_CAN_INVALID_STATE</code>	Invalid CAN state
-13	<code>SIMPLY_E_CAN_INVALID_BAUDRATE</code>	Invalid CAN baudrate
-14	<code>SIMPLY_E_TX_BUSY</code>	Message not sent, Tx is busy
-15	<code>SIMPLY_E_API_BUSY</code>	API is busy.

## 13.2 State Diagram



**Fig. 3** SimplyCAN states

### Function Calls and the Corresponding Valid States

Function	Valid states
<code>simply_open()</code>	Not connected
<code>simply_close()</code>	Device connected, CAN initialized
<code>simply_initialize_can()</code>	Device connected, CAN initialized
<code>simply_reset_can()</code>	Device connected, CAN initialized, CAN running
<code>simply_start_can()</code>	CAN initialized
<code>simply_stop_can()</code>	Device connected, CAN initialized, CAN running
<code>simply_receive()</code>	CAN initialized, CAN running
<code>simply_send()</code>	CAN running
<code>simply_get_last_error()</code>	All states
<code>simply_can_status()</code>	Device connected, CAN initialized, CAN running
<code>simply_identify()</code>	Device connected, CAN initialized, CAN running
<code>simply_set_filter()</code>	CAN initialized

## A Regulatory Compliance

### A.1 EMC Compliance (CE)



The product is in compliance with the Electromagnetic Compatibility Directive. More information and the Declaration of Conformity is found at [www.ixxat.com](http://www.ixxat.com).

### A.2 Disposal and recycling



You must dispose of this product properly according to local laws and regulations. Because this product contains electronic components, it must be disposed of separately from household waste. When this product reaches its end of life, contact local authorities to learn about disposal and recycling options, or simply drop it off at your local HMS office or return it to HMS.

For more information, see [www.hms-networks.com](http://www.hms-networks.com).

