



## Command and Control

The USB I/O Expander acts as a COM port and is controlled using commands and data in ASCII format. All data is transmitted as hexadecimal bytes except for the response that is returned by the USB I/O Expander after executing a command. A command is executed after detecting a carriage return or a line feed.

The COM port settings are (although not really relevant for USB):

- Baudrate 115200 baud
- 8 bits, 1 stop bit, no parity
- RTS/CTS flow control must be enabled

The advantage of using this device as a COM port is that you can control it using a terminal emulation program on your computer by typing commands. There is no need for special libraries to control the USB I/O Expander.

## Command response

After executing a command the following response is given as ASCII character:

- 0 = Command OK
- 1 = Error in command or its parameters
- 2 = Unknown command

## Command format and answer format

The general format of a command is as follows:

```
!<command><data><cr><lf>
```

All <data> must be given in ASCII in hexadecimal format (00 . . FF). Spaces in the data are ignored. Command and data are case insensitive.

If the command is executed (or incorrect) the response as mentioned earlier is given. One command including its data may not be longer than 64 characters.

When data is returned, e.g. when reading data via the IIC interface the answer format is:

```
?<data><cr><lf>
```

All data is returned in hexadecimal format. No spaces are given between the bytes in the data.

## Command overview

The following commands are supported

Command	Function	Parameters / Answer
!RES	Resets the device.	The COM port will disconnect shortly.
!PING	Returns 0 when alive.	-
!PID<pin><direction>	Set pin direction.	<pin>: 00..07 <direction>: 00 = output 01 = input
!PIM<pin><mode>	Set pin mode.	<pin>: 00, 01, 02, 03, 06 <mode>: 00 = digital 01 = analog
!PIP<pin><pull-up>	Set pin internal weak pull-up resistor.	<pin>: 06 or 07 <pull-up>: 00 = disabled 01 = enabled
!PIW<pin><data>	Write data to the pin.	<pin>: 00..07 <data>: 00 = pin low 01 = pin high
!PIR<pin>	Read data from the pin.	<pin>: 00..07 <answer>: 00 = pin low 01 = pin high
!PYD<direction>	Set the direction of all pins at once (pin 0..7).	<direction>: 00..FF where bit 0 is the LSB. bit is 0 = output bit is 1 = input
!PYW<data>	Write data to all pins at once (pin 0..7).	<data>: 00..FF where bit 0 is the LSB. bit is 0 = pin low bit is 1 = pin high
!PYR	Read the value of all pins at once (pin 0..7).	<answer>: 00..FF where bit 0 is the LSB. bit is 0 = pin low bit is 1 = pin high
!IICI<speed>	Initialize the IIC interface using the given speed.	<speed>: 00 = 96 kHz 01 = 100 kHz 04 = 400 kHz 0A = 1 MHz
!IICW<data>	Write the given data via the IIC interface.	<data>: data to be written
!IICR<nr_of_bytes>	Read the given number of bytes via the IIC interface.	<nr_of_bytes>: number of bytes that has to be read. <answer>: data read
!DACI<output>	Initialize the DAC using the given output. VDD is used as reference voltage. DAC is disabled.	<output>: 01 or 02
!DACE	Enable the DAC.	-
!DACD	Disable the DAC.	-
!DACW<data>	Write the given data to the DAC.	<data>: 00..1F (0..31)
!SPII<mode><rate>	Initialize the SPI interface	<mode>: 00..03, where 00 = CKP=1 & CKE = 1 01 = CKP=0 & CKE = 0

Command	Function	Parameters / Answer
	using the given mode and rate. See the datasheet for more information on CKP and CKE settings. Fosc is 48 MHz.	02 = CKP=1 & CKE = 1 03 = CKP=1 & CKE = 0 <rate>: 00..02, where 00 = Fosc / 4, 01 = Fosc / 16 02 = Fosc / 64
!SPIW<data>	Write the given data via the SPI interface.	<data>: data to be written
!SPIR<nr_of_bytes>	Read the given number of bytes via the SPI interface.	<nr_of_bytes>: number of bytes that has to be read. <answer>: data read
!ADCI<channel>	Initialize the ADC for the given channel. VDD is used as reference voltage.	<channel>: 03, 04, 05, 06, 07
!ADCE	Enable the ADC.	-
!ADCD	Disable the ADC.	-
!ADCC<channel>	Select the given ADC channel.	<channel>: 03, 04, 05, 06, 07
!ADCR	Read the ADC value of the last selected channel.	<answer>: 0000..03FF
!PWMI<channel>	Initialize the PWM hardware for the given channel. The duty cycle is set to 50%.	<channel>: 01 or 02
!PWME<channel>	Enable the given PWM channel.	<channel>: 01 or 02
!PWMD<channel>	Disable the given PWM channel.	<channel>: 01 or 02
!PWMF<frequency>	Set the PWM frequency. This is the same for both channels.	<frequency>: 02EE..AFC8 (750Hz..45kHz)
!PWMC<channel><duty_cycle>	Set the duty cycle in percentage on the last selected channel.	<channel>: 01 or 02 <duty_cycle>: 00..64 (0%..100%)

When executing commands, certain preconditions must be satisfied, otherwise the response will return as 1. For example:

- When reading a pin, the direction must have been set to input
- When using the IIC interface or SPI interface, the interface must have been initialized.

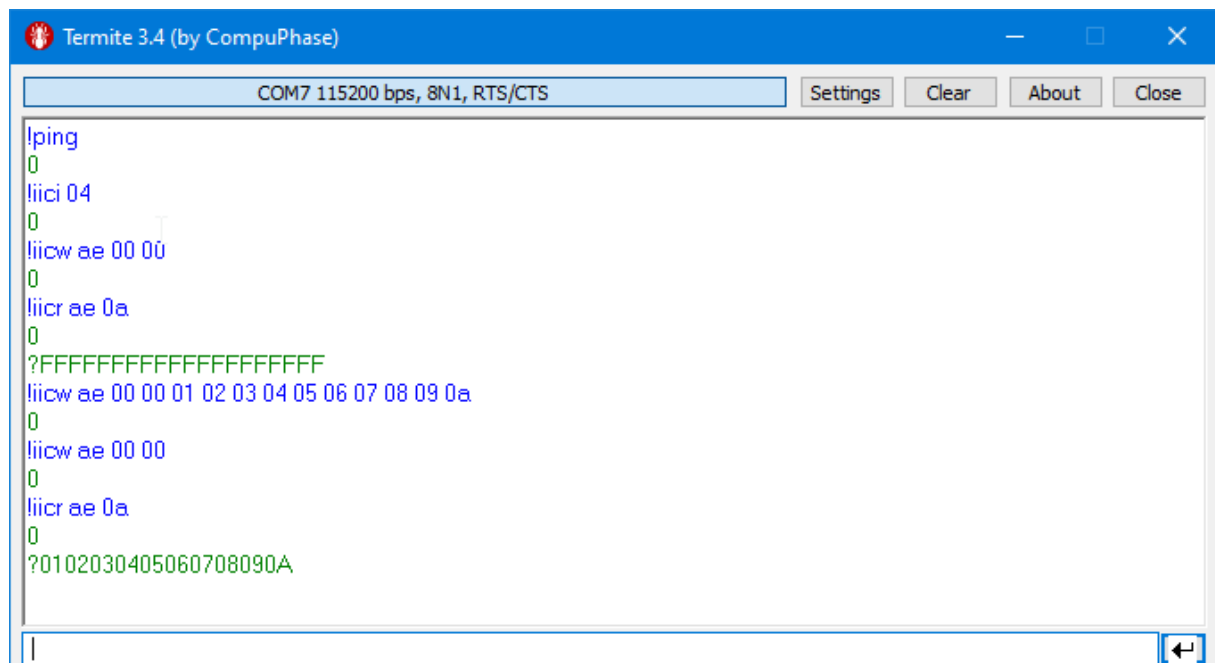
## Command examples

In the following pictures some commands are shown including the response of the USB I/O Expander.

The demo shows the reading data and writing data to an IIC EEPROM having IIC address 0xAE. The steps shown are:

1. Ping the device. This is optional, just checking if the device is alive.
2. Initialize the IIC interface at 400 kHz.
3. Reset the register – word - address of the EEPROM to 0 before reading.
4. Read 10 bytes from the EEPROM (starting at word address 0). EEPROM is empty (0xFF).
5. Write 10 bytes of data starting at word address 0.
6. Reset the register – word - address of the EEPROM to 0 before reading.
7. Read 10 bytes from the EEPROM (starting at word address 0)

Commands are given in blue, the response (0) and the answers (?) in green. Note that all data must be given as hexadecimal numbers.



The screenshot shows a terminal window titled "Termite 3.4 (by CompuPhase)" with a status bar indicating "COM7 115200 bps, 8N1, RTS/CTS". The terminal contains the following commands and responses:

```
!ping
0
!iic i 04
0
!iic w ae 00 00
0
!iic r ae 0a
0
?FFFFFFFFFFFFFFFFFFFFFFF
!iic w ae 00 00 01 02 03 04 05 06 07 08 09 0a
0
!iic w ae 00 00
0
!iic r ae 0a
0
?0102030405060708090A
```

## Python library and examples

In order to make life easier a Python library 'usb\_io\_expander.py' was created including several examples that were created for testing the USB IO Expander. The following Python examples are provided:

- Test\_Pins\_Output.py – Controlling the I/O pins as output.
- Test\_Pins\_Input.py – Controlling the I/O pins as input.
- Test\_IIC\_EEPROM.py – Reading and writing data to an EEPROM via IIC.
- Test\_IIC\_MCP23008.py – Controlling I/O pins using an I/O expander, controlled via IIC.
- Test\_IIC\_MCP23S08.py – Controlling I/O pins using an I/O expander, controlled via SPI.
- Test\_ADC.py – Converting an analog input signal to a digital value
- Test\_DAC.py – Converting a digital value to an analog output signal
- Test\_PWM.py – Generating a Pulse Width Modulation signal

## Video

If you want to see the device with the Python examples in action, have a look at this video:

## References

If you want to know more about the JAL programming language visit the JAL website at:

<http://justanotherlanguage.org/>