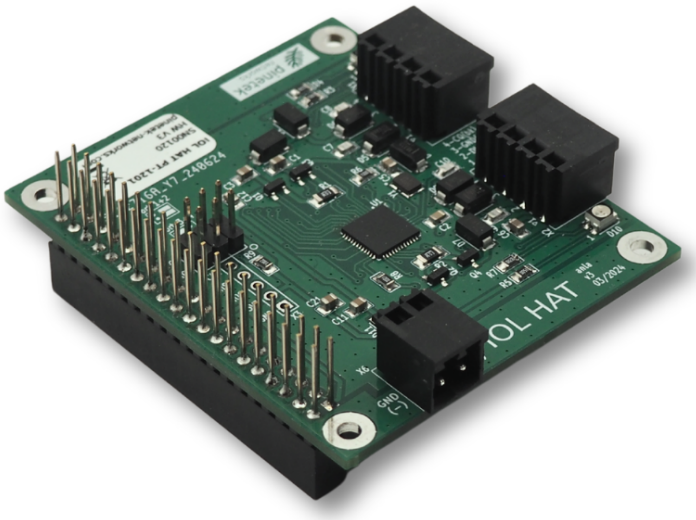
# IOL HAT Master Application Software

## Application Manual

www.pinetek-networks.com



# Table of Contents

# 1 Introduction

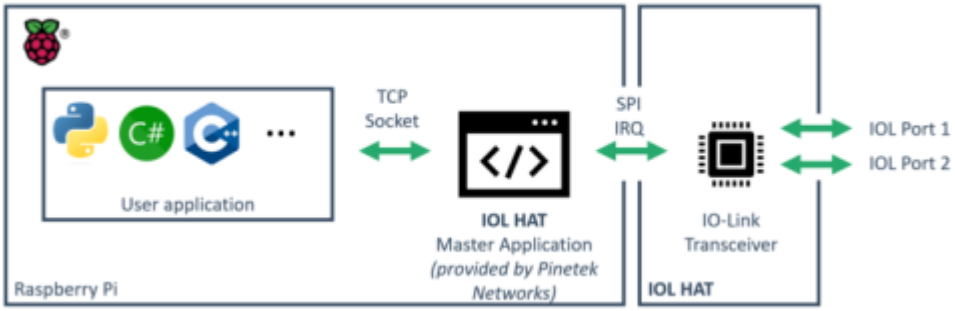
This section describes the methods and procedures that are used in connection with the open source software for the IOL HAT (“IOL master application”) which is available in the Github repository: https://github.com/Pinetek-Networks/iol-hat

The software is based on the I-Link stack by RT-Labs. Please note that the software has been proven to work in different applications, however, it comes without guarantees of specific functions or functionality.

Information on how to obtain the software and how to build it for specific targets can be found in the GitHub. The following sections describe the application of the software on the Raspberry Pi or other Single Board Computers.

## Theory of operation

The master application handles the raw communication to the MAX14819 transceiver on the IOL HAT or IOL HAT Pro over SPI and provides the stack routines to operate the IO-Link communication. The master application offers a TCP socket interface to the user application for easy integration. The master application and user application need to be run independently.




## Hints and Warnings

The following signs will guide you for dedicated areas of interest. The content is relevant for safe operation of the IOL HAT in combination with the software.

This symbol is used to show information that is relevant for the operation of the IOL HAT

This symbol is used to show areas of special attention that need to be considered for the safe operation of the IOL HAT

# 1.1 License and Disclaimer

## Software License Notice

This software is released under the GNU General Public License v3.0 (GPL-3.0). License Terms

You have the freedom to use, modify, and distribute this software. Any derivative works must also be distributed under the GPL-3.0. The complete license text is available at: https://www.gnu.org/licenses/gpl-3.0.html

## Disclaimer of Liability

The software is provided “as is” without warranty of any kind, either expressed or implied. The authors or copyright holders shall not be liable for any claims, damages, or other liability arising from the use of the software. By using this software, you acknowledge that you have read and understood these terms.

# 2 IOL Master Application

To operate the IOL HAT on a Raspberry Pi, the master application binary from the IOL HAT repository is used: https://github.com/Pinetek-Networks/iol-hat/bin

Important note on the startup order: To work properly, the master application needs to be started AFTER 24V power is applied. Failure to do so will result in broken IO-Link communication.

The suitable master application file for the used Linux distribution shall be used. The Linux version can be determined with

```
uname -a
```

If the Linux distribution is not available as binary, please refer to the building instructrions on the GitHub page to compile a matching version: https://github.com/Pinetek-Networks/iol-hat/tree/main/src-master-application

# 2.1 Prepare the Target

The IOL HAT can be used with the binary iol-master-appl that is provided in the /bin/ folder of the GitHub. This binary has been compiled with the aarch-linux toolchain for 64-bit version. For other versions and targets, please follow the instructions to build:
https://github.com/Pinetek-Networks/iol-hat/blob/main/src-master-application/README.md

The following instructions apply to Raspberry Pi, for other targets, please check with the corresponding instructions how to enable SPI and GPIO.

## Enable SPI communication

The communication between Master Application iol-master-appl and the IOL HAT is done over SPI, so SPI communication needs to be enabled on the host. To enable SPI for Raspberry Pi hosts, the following instructions can be used:

```
-sudo raspi-config
-go to: Interface Options
-go to: SPI
-select: Enable SPI
```

If you use another target, please verify with the documentation how to enable the SPI

## GPIO

The interrupt line on between host and IOL HAT is a GPIO line. To operate, **gpiod** is used as library. To install **gpiod**, please use the following command:

```
sudo apt install gpiod
```

# 2.2 Arguments and Timing

## Command line arguments

| | |
|---|---|
| -h, –help | shows help message and exits |
| -v, –version | prints version information and exits |
| -m0, –mode0 | Operating Mode Port 0 (X1), Possible values 0: IOL, 1: DI, 2: DO, 3: OFF |
| -m1, –mode1 | Operating Mode Port 1 (X2), Possible values 0: IOL, 1: DI, 2: DO, 3: OFF |
| -e, –extclock | Use clock for MAX14819 from ext source (not required for the IOL HAT) |
| -d, –delaycurrentlimit | Delay time in ms for current limit enabling after port enable, range [1..1500]. Please see below for further details |
| -b, –blankingtime | Blanking time setting 0=short…3=long for capacitive loads. Please see below for further details |
| -r, –realtime | Run realtime on given kernel. Requires root rights, see below for details |
| -i, –iolport | This specifies the IOL port (see jumper settings on the IOL HAT). Standard is port 1/2. See hardware manual for the SPI and GPIOs in use. Possible values 12 (default) and 34 |
| -t, –tcpport | Specify the TCP port for communication. If not specified here, TCP port is 12010 for IOL port 1/2 and 12011 if –iolport 34 is set. |

## IOL Port modes

The operation mode of the ports are defined with -m0 and -m1 (respectively –mode0 and –mode1) for the IOL HAT ports.

- **0 (Default) - SDCI/IO-Link operation**
  - Parameters (CMD_READ, CMD_WRITE) and Process Data (CMD_PD) as provided by the connected device
- **1 - DI: Input mode**
  - No parameters (CMD_READ, CMD_WRITE) available
  - Process data: 1 byte IN/ 0 byte OUT (CMD_PD)
    - CQ-Line against GND: Process data IN = 0x00
    - CQ-Line against +24V: Process data IN = 0x01
    - Power needs to be enabled (CMD_PWR) for operation
- **2 - DO: Output mode**
  - No parameters (CMD_READ, CMD_WRITE) available
  - Process data: 0 byte IN/1 byte OUT (CMD_PD)
    - Process data OUT = 0x01: CQ-Line driven to 24V
    - Process data OUT = 0x00: CQ-Line driven to GND
    - Power needs to be enabled (CMD_PWR) for operation
- **3 - OFF**: Port off/disabled

The port mode can only be set at program start.

## Delay current limit / blanking time

The -d (–delaycurrentlimit, default is 0) parameter allows to disable the current limit of 500mA after port enable after powering the port (CMD_PWR). This is needed for some devices that have a higher current demand at device start (note: often, the specifications for those devices are <500mA).

Use the **-d** parameter with caution, to much current on L+ may damage the IOL HAT circuit. Limit the current at the IOL HAT input ≤1A for testing and iterate to the required time with this limit. Damages to the IOL HAT hardware that are caused by the use of this are not covered by the IOL HAT product warranty. The usage is on your own risk.

The blanking time parameter -b (–blankingtime, default is 0) sets the current-limit blanking time for the L+ sensor supply. Longer blanking times allow for charging of larger capacitive loads.

| **Blanking time parameter** | 0 (default) | 1 | 2 | 3 |
|---|---|---|---|---|
| **Blanking time** | 5,5ms | 16,5ms | 55ms | 165ms |

## Timing considerations

IO-Link is a real-time protocol that requires fast response and cycle times to not run into timeouts. For running with sensors etc. that are queried 1-2 times per second, those timeouts are not crititcal. The iol-master-applcan run with standard user rights without further considerations.

If timeouts are critical (e.g., when HMI devices are connected that would change screen when timeouts occur), it is recommended to run the iol-master-appl in realtime mode. The solution is based on this description:

https://forums.raspberrypi.com/viewtopic.php?t=228727

As preparation, one core needs to be reserved. This is done by adding the this argument to /boot/firmware/cmdline.txt (for older versions of Raspberry OS, the file is /boot/cmdline.txt):

```
isolcpus=3
```

where 3 is the core you want to reserve (can be 0..3).

All arguments in the cmdline.txt need to be in ONE line, i.e. you need to add the isolcpus=3 at the end of the line. Otherwise the core reservation as described above will not work.

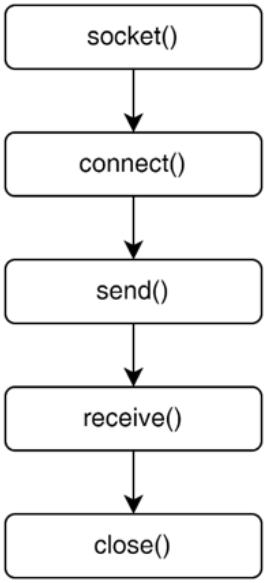
The core needs to be matching with the option that you are giving when starting the iol-master-appl, e.g., for core 3 (matching with the above example of the cmdline.txt) it would be

```
iol-master-appl -r 3
```

Each core needs to be explicitly stated. To reserve e.g. core 2 and 3 (the last two cores), the statement would be isolcpus=2,3. In this case, two iol-master-appl can be executed: iol-master-appl -r 2 and iol-master-appl -r 3, both running in realtime mode.

# 3 Binary protocol

The sequence for communication over TCP is as follows:




The commands and responses are exchanged in the send() and receive() part of the sequence. The first octet in the command structure always defines the command ID.

If two IOL HAT are stacked, they share the same SPI data lines but different CS and IRQ lines. Which lines are used is set by jumpers on the hardware. The Master Application uses different TCP ports for adressing the port 1+3 setup and the port 2+4 setup (i.e. it controls different CS and IRQ lines).

Many commands use a port index (either 0 or 1). Port index 0 refers to the IOL HAT X1 connector, Port index 1 to IOL HAT X2 connector. Which IOL HAT module is addressed is determined through the TCP port as described above.

The commands have a common structure in the first two octets:

| Octet # | Usage |
|---|---|
| 0 | Command ID |
| 1 | Length |

In case of a command success, the return message as described in the commands is returned by the TCP server.

## Error handling

In case of an error, the error message is returned:

| Octet # | Usage |
|---|---|
| 0 | Error message ID = 0xFF |

| Octet # | Usage |
|---|---|
| 1 | Error code |

The following error codes are defined:

| Error code | Usage |
|---|---|
| 0x01 | Message Length Error |
| 0x02 | Function ID unknown |
| 0x03 | Port power error, e.g. read while port disabled |
| 0x04 | Port ID error, i.e. port ID >1 called |
| 0x05 | Internal error, in this case octets 2..3 define the specific error code from the I-Link stack |
| 0x06 | Wrong status, e.g. data exchange when power off or no connection to the device |

The error codes are listed in Appendix-A.

# 3.1 CMD_PWR

The power command switches the L+ power on the corresponding port. After switching on, the communication speed on the SDCI (COM1, COM2, COM3) is automatically detected and cyclic exchange of data is started.

**Command:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x01 |
| 1 | Port index 0x00 or 0x01 |
| 3 | Power status<br>0x00 = OFF<br>0x01..0xFF=ON |

**Return Success:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x01 |
| 1 | Port index 0x00 or 0x01 |
| 3 | Power Status |

# 3.2 CMD_LED

The LED command switches the LED on the corresponding port.

**Command:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x02 |
| 1 | Port index 0x00 or 0x01 |
| 3 | LED Status (see below) |

The LED status is 0x01 for the green LED and 0x02 for the red LED. With LED status 0x03, both LEDs are activated.

**Return Success:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x02 |
| 1 | Port index 0x00 or 0x01 |
| 3 | LED Status |

# 3.3 CMD_PD

This command exchanges the process data with the connected device. Process data is not validated in length or content.

**Command:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x03 |
| 1 | Port index 0x00 or 0x01 |
| 2 | Length Out |
| 3 | Length In |
| 4.. | Data Out |

The ***Length Out*** field command is length for the output process data. The structure and length of the output data can be found in the SDCI device's documentation. The ***Length In*** field needs to be set according to the device's documentation. If the length fields are not correctly set, this may lead to invalid data or loss of communication.

**Return Success:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x03 |
| 1 | Port index 0x00 or 0x01 |
| 2 | Length Out (as given in the command) |
| 3 | Length In (as given in the command) |
| 4.. | Data In |

# 3.4 CMD_READ

This command reads a parameter with the index and subindex as given on the corresponding port.

**Command:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x04 |
| 1 | Port index 0x00 or 0x01 |
| 2..3 | Index, 16-bit value |
| 4 | Subindex |
| 5 | Length |

The ***Length*** field is the desired/maximum length for the attribute. The structure of the attribute can be found in the SDCI device's documentation.

**Return Success:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x04 |
| 1 | Port index 0x00 or 0x01 |
| 2..3 | Index, 16-bit value |
| 4 | Subindex |
| 5 | Read Length |
| 6.. | Read Data |

The structure for the Read Data can be found in the SDCI device's documentation.

# 3.5 CMD_WRITE

This command writes a parameter with the index and subindex as given on the addressed port.

**Command:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x05 |
| 1 | Port index 0x00 or 0x01 |
| 2..3 | Index, 16-bit value |
| 4 | Subindex |
| 5 | Length |
| 6.. | Write Data |

The structure of the ***Write Data*** can be found in the SDCI device's documentation.

**Return Success:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x05 |
| 1 | Port index 0x00 or 0x01 |
| 2..3 | Index, 16-bit value |
| 4 | Subindex |
| 5 | Length (as given in the command) |

To verify the write operation, a read operation (Command ID 0x04) can be used on the same ***Index*** and ***Subindex***.

# 3.6 CMD_STATUS

The status command returns the status of the corresponding port.

**Command:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x06 |
| 1 | Port index 0x00 or 0x01 |

**Return Success:**

| Octet # | Usage | Comment |
|---|---|---|
| 0 | Command ID = 0x06 | |
| 1 | Port index 0x00 or 0x01 | |
| 2 | Process data IN valid | 0x00 = Not valid 0x01 = Valid |
| 3 | Process data OUT valid | |
| 4 | Transmission rate | 0x00 = Detection Failure 0x01 = COM1 (4.8 kbps) 0x02 = COM2 (38.4 kbps) 0x03 = COM3 (230.4 kbps) |
| 5 | Process data IN length | |
| 6 | Cycle time | |
| 7 | Process data OUT length | |
| 8..9 | Vendor ID | 16-bit value |
| 10..13 | Device ID | 32-bit value |
| 14 | Power | 0x00 = Power OFF 0x01 = Power ON |

# 3.7 CMD_STATUS2

The status2 command returns the status of the corresponding port, including an additional hardware error byte from the MAX14819 channel status register. It is an extended version of CMD_STATUS (0x06).

**Command:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x08 |
| 1 | Port index 0x00 or 0x01 |

**Return Success:**

| Octet # | Usage | Comment |
|---|---|---|
| 0 | Command ID = 0x08 | |
| 1 | Port index 0x00 or 0x01 | |
| 2 | Process data IN valid | 0x00 = Not valid 0x01 = Valid |
| 3 | Process data OUT valid | |
| 4 | Transmission rate | 0x00 = Detection Failure 0x01 = COM1 (4.8 kbps) 0x02 = COM2 (38.4 kbps) 0x03 = COM3 (230.4 kbps) |
| 5 | Process data IN length | |
| 6 | Cycle time | |
| 7 | Process data OUT length | |
| 8..9 | Vendor ID | 16-bit value |
| 10..13 | Device ID | 32-bit value |
| 14 | Power | 0x00 = Power OFF 0x01 = Power ON |
| 15 | Hardware error | Lower 3 bits of MAX14819 REG_ChanStatA/B. 0x00 = No error |

# 3.8 CMD_STATUS3

The status3 command returns the chip-level status register of the MAX14819 transceiver. Unlike CMD_STATUS and CMD_STATUS2, this command is not port-specific and requires only a single command byte (no port index). The returned value is the content of the MAX14819 REG_Status register (0x1E) OR'd with any error bits that were latched since the last call to CMD_STATUS3. Reading this command clears the latch.

**Command:**

| Octet # | Usage |
|---|---|
| 0 | Command ID = 0x09 |

**Return Success:**

| Octet # | Usage | Comment |
|---|---|---|
| 0 | Command ID = 0x09 | |
| 1 | Chip status | MAX14819 REG_Status (0x1E) OR'd with latched error events. Upper nibble (bits 7..4) contains chip-level diagnostic flags; lower nibble (bits 3..0) contains channel error flags. 0x00 = No errors |

# Appendix-A Error messages

The following extended Error Messages are defined (see iolink.h in application source code):

```
  IOLINK_SMI_ERRORTYPE_NONE = 0x0000,

  /* Table C.1 ErrorTypes */
  IOLINK_SMI_ERRORTYPE_APP_DEV               = 0x8000,
  IOLINK_SMI_ERRORTYPE_IDX_NOTAVAIL          = 0x8011,
  IOLINK_SMI_ERRORTYPE_SUBIDX_NOTAVAIL       = 0x8012,
  IOLINK_SMI_ERRORTYPE_SERV_NOTAVAIL         = 0x8020,
  IOLINK_SMI_ERRORTYPE_SERV_NOTAVAIL_LOCCTRL = 0x8021,
  IOLINK_SMI_ERRORTYPE_SERV_NOTAVAIL_DEVCTRL = 0x8022,
  IOLINK_SMI_ERRORTYPE_IDX_NOT_ACCESSIBLE    = 0x8023,
  IOLINK_SMI_ERRORTYPE_PAR_VALOUTOFRNG       = 0x8030,
  IOLINK_SMI_ERRORTYPE_PAR_VALGTLIM          = 0x8031,
  IOLINK_SMI_ERRORTYPE_PAR_VALLTLIM          = 0x8032,
  IOLINK_SMI_ERRORTYPE_VAL_LENOVRRUN         = 0x8033, /* Also in C.2 */
  IOLINK_SMI_ERRORTYPE_VAL_LENUNDRUN         = 0x8034,
  IOLINK_SMI_ERRORTYPE_FUNC_NOTAVAIL         = 0x8035,
  IOLINK_SMI_ERRORTYPE_FUNC_UNAVAILTEMP      = 0x8036,
  IOLINK_SMI_ERRORTYPE_PAR_SETINVALID        = 0x8040,
  IOLINK_SMI_ERRORTYPE_PAR_SETINCONSIST      = 0x8041,
  IOLINK_SMI_ERRORTYPE_APP_DEVNOTRDY         = 0x8082,
  IOLINK_SMI_ERRORTYPE_UNSPECIFIC            = 0x8100,

  /* Table C.2 Derivced ErrorTypes */
  IOLINK_SMI_ERRORTYPE_COM_ERR           = 0x1000,
  IOLINK_SMI_ERRORTYPE_I_SERVICE_TIMEOUT = 0x1100,
  IOLINK_SMI_ERRORTYPE_M_ISDU_CHECKSUM   = 0x5600,
  IOLINK_SMI_ERRORTYPE_M_ISDU_ILLEGAL    = 0x5700,

  /* Table C.3 SMI related ErrorTypes */
  IOLINK_SMI_ERRORTYPE_ARGBLOCK_NOT_SUPPORTED   = 0x4001,
  IOLINK_SMI_ERRORTYPE_ARGBLOCK_INCONSISTENT    = 0x4002,
  IOLINK_SMI_ERRORTYPE_DEV_NOT_ACCESSIBLE       = 0x4003,
  IOLINK_SMI_ERRORTYPE_SERVICE_NOT_SUPPORTED    = 0x4004,
  IOLINK_SMI_ERRORTYPE_DEV_NOT_IN_OPERATE       = 0x4005,
  IOLINK_SMI_ERRORTYPE_MEMORY_OVERRUN           = 0x4006,
  IOLINK_SMI_ERRORTYPE_PORT_NUM_INVALID         = 0x4011,
  IOLINK_SMI_ERRORTYPE_ARGBLOCK_LENGTH_INVALID  = 0x4034,
  IOLINK_SMI_ERRORTYPE_SERVICE_TEMP_UNAVAILABLE = 0x4036,
```