PENKO Engineering B.V.

Your Partner for Fully Engineered Factory Solutions

Protocol description:
PENKO Two Phase (TP)

PENKO
an ETC Company

PENKO TP Protocol

1 Table of Contents

2 Introduction

The PENKO TP protocol is a Binary burst two phase protocol. The protocol can be used over serial (RS232, RS422, RS485, USB) and Ethernet (UDP) connections.

The TP protocol is used between PENKO devices, PC's, PLC's and other PENKO equipment and is used for programming PENKO equipment, asking the status of I/O and reading out indicator values.

The TP protocol is based on two parts; a request and a reply. Both phases use the same shape for sending data. The difference is made by who takes the initiative for the phase. In the first phase of the protocol, the request, the initiative is taken by the master. Most of the times this will be a PC application, but it can also be an application on a PLC or other embedded device. The second phase is initiated by the slave. This is the reply message. This role will be mostly fulfilled by an embedded device such as an indicator or a remote I/O unit. After sending the reply, the communication cycle is closed and a new request can be sent.

The latest addition to the TP protocol is PDI, PENKO Device Interface. PDI is available in the current line of PENKO devices and shows the complete device configuration in a tree structure. Every property in the tree has a unique path number, and through these path numbers every property can be accessed using TP_PDI.

3 TP serial

The TP serial frames are constructed as follows.

Request frame:

Byte		Byte	Byte[]	Byte	Byte	
DLE	**STX**	**Address**	**Data**	**Checksum**	**DLE**	**ETX**

Reply frame:

Byte		Byte	Byte[]	Byte	Byte	
DLE	**STX**	**Address**	**Data**	**Checksum**	**DLE**	**ETX**

Frame description:

Frame part	Description
DLE + STX	Preamble
Address	Port address set in device. For USB connection the address is always 0.
Data	Command code, Operation code and parameters, described in chapter Data description
Checksum	Checksum, described in Checksum calculation
DLE + ETX	Postamble

Used characters:

Character	Decimal	Hexadecimal	Description
STX	2	0x02	Start of TeXt
ETX	3	0x03	End of TeXt
DLE	16	0x10	Data Link Escape

The postamble must be the only DLE+ETX sequence in the protocol in order to indicate the end of a frame. For every character in the address, data or checksum that equals the DLE character, an extra DLE character is added to ensure the uniqueness of the postamble. The extra DLE characters are not included in the checksum calculation.

Checksum calculation:

The checksum is the inverted first complement of the sum of the address byte and all data bytes. The possible extra DLE characters, as described above, are not included in this calculation.

Example:

The sum of the address byte and data bytes (excluding possible extra DLE characters) is 0x1234

Get the first complement: 0x1234 ***AND*** 0xFF = 0x34

Inverse the result: 0x34 ***XOR*** 0xFF = 0xCB

4 TP UDP

The TP UDP frames are constructed as follows.

Request frame:

Byte	Byte	Byte	Byte	Byte[]
0x00	**0x00**	**0x00**	**0x00**	**Data**

Reply frame:

Byte	Byte	Byte	Byte	Byte[]
0x00	**0x00**	**0x00**	**0x00**	**Data**

Frame description:

Frame part	Description
0x00	Preamble, default 4 x 0x00, reserved for future extensions
Data	Command code, Operation code and parameters, described in chapter Data description

Compared to the serial TP frame, the TP protocol over Ethernet has no address, preamble, postamble, checksum and extra DLE characters in the communication frame.

5 TP data description

The data frames in the serial and UDP communication are constructed as follows.

Request data frame:

Byte	Byte	Byte[]
Command code	**Operation code**	**Parameters**

Reply data frame:

Byte	Byte	Byte[]
Command code	**Operation code**	**Parameters**

Frame description:

Frame part	Description
Command code	Feature of the device to perform the request on
Operation code	Operation to perform on the feature, not present in all features
Parameters	Parameters belonging to the operation, not present in all operation

A complete TP serial frame:

Byte		Byte	Byte[]	Byte	Byte	
DLE	**STX**	**Address**	**Data**	**Checksum**	**DLE**	**ETX**

Byte	Byte	Byte[]
Command code	**Operation code**	**Parameters**

A complete TP UDP frame:

Byte	Byte	Byte	Byte	Byte[]
0x00	**0x00**	**0x00**	**0x00**	**Data**

Byte	Byte	Byte[]
Command code	**Operation code**	**Parameters**

The following command codes are available. Usually the command code in the request frame is replied in the reply frame.

Command codes (request and reply):

Command code	Name	Description
0x01	RTC	Real-Time Clock
0x46	INDICATOR	Indicator functions
0x5A	VERSION	S/W version
0x5D	ID	H/W & application id
0x5E	FLASH	FLASH functions
0x64	ECHO	Echo functions
0x78	SBC	Batch Controller
0xB4	PDI	PDI functions

In some cases only an acknowledge command code is returned, or in case a reply cannot be sent, one of the error command codes is returned.

Reply codes:

Command code	Name	Description
0x53	BUSY	Device busy internal
0x54	ERROR	Function parameter error
0x55	ACK	Function accepted and done
0x57	DISABLED	Host functions disabled
0x58	NAK	Internal status conflict
0x59	ILLEGAL	Unknown command

The available features depend on the hardware platform. Most features have an operation code to check the availability.

5.1 Reply code 0x53: Device busy internal

When a function is applied while the slave device has been long engaged in another activity, e.g. user input, this result code can be returned. When receiving this result the master can decide if he continues to poll the relevant slave until it can answer or that another device is accessed. This result code is only applicable to single-threaded applications, which often have to do with user input.

5.2 Reply code 0x54: Function parameter error

Upon receiving a function call, a check is done on the number of bytes received. When the number of bytes does not match the number specified for the requested feature, this feature is not implemented and this code is returned. This error can be caused by a transmission error in which a character is lost but the checksum is still correct. More likely it is that a mistake was made with building the request packet that a wrong packet size is specified.

5.3 Reply code 0x55: Function accepted and done

A function to activate an action on the slave device without returning data will give this result code to the master after the execution of the action.

5.4 Reply code 0x57: Host functions disabled

Slave devices with a configurable user interface can disable the protocol driver so remote configuration and / or control is no longer possible. In this case the protocol driver delivers this result code back. This makes it possible for the master to give a detailed error notification to the user and possibly remove the device from the communication.

5.5 Reply code 0x58: Internal status conflict

Performing a function can be connected to the internal status of the slave device. If this is the case, this function result code comes back. Such a situation may occur when, e.g., the parameters of a slave process are changed while the process is active. It is therefore prudent to keep track of the status of the slave device on the master so this conflict can be prevented.

5.6 Reply code 0x59: Unknown command

When a function is applied which is not known to the slave device, this result code is returned. By asking for the device ID there can be determined what kind of device is hidden behind the device address. On this basis the commands that are valid for this application can be determined. Possibly you may need to request additional information such as the version number from the slave device.

5.7 Command code 0x01: Real-Time Clock

The real-time clock interface gives the possibility to set or read any existing real-time clock from the host application. This simple interface consists of three functions that make use of a fixed request-reply-structure. The table below shows the available functions.

OpCode	Description
0x00	Feature detection
0x01	Read current RTC setting
0x02	Set new RTC setting

Check if the real-time clock feature is available

The availability of the real-time clock interface can be determined by means of this function. When the interface is available, ACK will be replied.

Request:

Command	OpCode
0x01	0x00

Reply (ACK):

Command
0x55

Read the real-time clock

After sending a request with this opcode a structure with fixed format is supplied back in which the internal current date and time are indicated. For the coding of the date and time so called BCD notation is used. For example, the value 12 (decimal) listed as 0x12 (hexadecimal).

Request:

Command	OpCode
0x01	0x01

Reply:

Command	OpCode	Year	Month	Day	Hour	Minute	Second
0x01	0x01	0x14	0x05	0x12	0x09	0x42	0x28

12 May 2014 09:42:28

Set the real-time clock

Setting the correct date and time can be done through this opcode. A structure with fixed format has to be sent as displayed below. Here too, the coding of the date and time is by means of BCD notation, just as in the answer to the QUERY function.

Request:

Command	OpCode	Year	Month	Day	Hour	Minute	Second
0x01	0x02	0x14	0x05	0x12	0x09	0x42	0x28

Reply (ACK):

Command
0x55

5.8 Command code 0x46: Indicator functions

The indicator interface is intended to control basic indicator functions like zero and tare. A second feature is reading the internal registers like ADC sample, Net, Gross, etc.

OpCode	Description
0x00	Indicator function is available
0x01	Read indicator registers
0x02	Basic indicator commands

Check if the indicator feature is available

The availability of the interface can be detected with this function. When present an ACK reply follows. If the interface is not present, there is automatically answered with the ERROR command.

Request:

Command	OpCode
0x46	0x00

Reply (ACK):

Command
0x55

Read the indicator

This command reads the registers of the indicator. The structure of the command is shown below.

Request:

Command	OpCode	Indicator query
0x46	0x01	***query bits***

Reply:

Command	OpCode	Indicator query	Indicator data
0x46	0x01	***query bits***	***data bits***

The following query bit combinations are available:

Query name	Query bits	Description
SAMPLE	0x00000001	A/D sample
FREE	0x00000002	Return always 0
FREE	0x00000004	Return always 0
STATUS*	0x00000008	Status bits weigher
GROSS X 10	0x00000010	Gross weight x 10
NET10	0x00000020	Net weight x10
FGROSS X 10	0x00000040	Filtered gross weight x10
FNET X 10	0x00000080	Filtered net weight x10
TARE X 10	0x00000100	Tare weight x10
PTARE X 10	0x00000200	Preset tare weight
GROSS	0x00000400	Gross weight
NET	0x00000800	Net weight
FGROSS	0x00001000	Filtered gross weight
FNET	0x00002000	Filtered net weight
TARE	0x00004000	Tare weight
PTARE	0x00008000	Preset tare weight
DISPLAY	0x00010000	Display weight

* Status bits weigher:

Status name	Status bits	Description
HWOVERLOAD	0x00000001	Hardware overload detected
MAXLOAD	0x00000002	Overload detected
STABLE	0x00000004	Stable signal
STABLERNG	0x00000008	In stable range
ZEROSET	0x00000010	Zero corrected
ZEROCENTER	0x00000020	Center of zero
ZERORANGE	0x00000040	In zero range
ZEROTRACK	0x00000080	Zero tracking possible
TARE	0x00000100	Tare active
PTARE	0x00000200	Preset tare active
NEWSAMPLE	0x00000400	New sample available
BADCAL	0x00000800	Calibration invalid
CALENABLED	0x00001000	Calibration enabled
INDUSTRIAL	0x00002000	Industrial/certified operation
NOTLEVEL	0x00004000	System level blocking
RESERVED15	0x00008000	Reserved bit fields
*	0xFFFF0000	Weigher format bits see below

* Weigher format bits:

Bit number	Description
#15	Signed/unsigned
	0 = Unsigned
	1 = Signed
#14	Zero suppressing
	0 = Nonzero suppressing
	1 = Zero suppressing
#11 - #8	Display step size
	0000 = Step 1
	0001 = Step 2
	0010 = Step 5
	0011 = Step 10
	0100 = Step 20
	0101 = Step 50
	0110 = Step 100
	0111 = Step 200
	1000 = Step 500
	1001 = Step 1000
	1010 = Step 2000
	1011 = Step 5000
#2 - #0	Decimal point position
	000 = 000000
	001 = 00000.0
	010 = 0000.00
	011 = 000.000
	100 = 00.0000
	101 = 0.00000

Control the indicator

With this command, control commands are sent to the indicator like set zero, set tare, reset tare. The structure of the command is shown below.

Request:

Command	OpCode	Indicator control
0x46	0x02	***control bits***

Reply:

Command	OpCode	Indicator control
0x46	0x02	***control bits***

The following control bit combinations are available:

Control name	Control bits	Description
ZEROSET	0x00000001	Zero set
ZERORESET	0x00000002	Zero reset
TARESET	0x00000010	Tare set, 4 byte tare value is required
TAREON	0x00000020	Auto tare
TARERESET	0x00000040	Tare reset
PTARESET	0x00000080	Preset tare set, 4 byte preset tare value is required

Examples

Get weigher status:

Request:

Command	OpCode	Query bits
0x46	0x01	0x00000008

Reply:

Command	OpCode	Query bits	Data bits
0x46	0x01	0x00000008	0xC00324CC

Weigher status is **0x24 CC** = Stable & Stable range & In zero range & Zero tracking possible & New sample available & Industrial operation (see status bits table)

Weigher format is **0xC0 03** = 3 decimals & Step size 1 & Zero suppressing & Signed (see format bits table)

Get gross x 10 weigher value:

Request:

Command	OpCode	Query bits
0x46	0x01	0x00000010

Reply:

Command	OpCode	Query bits	Data bits
0x46	0x01	0x00000010	0x0000162B

Gross x 10 weigher value is 0x00 00 16 2B = 5675

Set zero:

Request:

Command	OpCode	Control bits
0x46	0x02	0x00000001

Reply:

Command	OpCode	Control bits
0x46	0x02	0x00000001

If indicator is within zero tracking range, zero is set.

Set preset tare to 200 (internal weigher works with x10 values, so set 2000 -> hex 0x07 D0):

Request:

Command	OpCode	Control bits	Preset tare value
0x46	0x02	0x00000080	0x000007D0

Reply:

Command	OpCode	Control bits
0x46	0x02	0x00000080

Preset tare is set with the specified value.

5.9 Command code 0x5A: S/W version

The S/W version number of the device can be requested using this command.

Request:

Command
0x5A

Reply:

Command	Major	Minor	Build
0x5A	0x01	0x03	0x06

This would be replied when the software version for example is 1.3.0.9.0.6

All current PENKO devices use a 6 digit version number:

Major release	Minor release	Patch release	Audience release	Critical situation release	Build number
1	**3**	**0**	**9**	**0**	**6**

5.10 Command code 0x5D: H/W & application id

This command provides the ability to identify the type of device in the system. The universal design of the protocol makes it possible to apply several totally different devices in the same communication system.

Request:

Command
0x5D

Reply:

Command	Type (MSB)	Type (LSB)
0x5D	0x06	0x18

This would be replied when the hardware ID is 0618.

5.11 Command code 0x5E: FLASH functions

The device actually has two applications, a Flash-BOOT application (FBOOT) and a target application. The FBOOT application is fixed inserted into the Flash ROM and cannot be overwritten or deleted. For this both software and hardware blocks are present.

The task of the FBOOT application is to ensure a stable start situation and from here make it possible to program a new (improved) target application program in the Flash ROM. To achieve all this, the Flash programming interface is available.

When both FBOOT and target application have a two-phase protocol, there can be switched between both programs under control of the programming program. This way, it's possible to build a very user friendly programming application in which the user must perform a minimum number of operations. The functions for erasing and writing the flash ROM are not activated in the target application so that it's only possible to program the ROM from the FBOOT application.

During the download of a new application program, it's important that the programming program has knowledge of the structure of the Flash-ROM. To transfer this knowledge of the device to the programming program, the interface has the special function FLASH_INFO. A classification of the Flash memory can be accessed without the need to build an extensive database of Flash memory brands and types.

In addition to programming the application using this interface it's possible to program and read the device serial number. Programming of the serial number is only allowed once and is performed during the production test. After writing it's not possible to change the serial number.

OpCode	Description	FBOOT	Application
0x00	Feature detection	yes	yes
0x01	Boot FBOOT-application	no	yes
0x02	Erase sector	yes	no
0x03	Write block	yes	no
0x04	Read block	yes	yes
0x05	Boot application	yes	no
0x06	Query application	yes	yes
0x07	Query flash type information	yes	yes
0x08	Query/program serial number	yes	yes

Check if the feature is available
The availability of the Flash interface can be detected with this function. If present, a reply with the system command ACK follows. If the interface is not present there is automatically answered with the ERROR command.

Request:

Command	OpCode
0x5E	0x00

Reply (ACK):

Command
0x55

Boot FBOOT-application
Before the Flash ROM can be provided of a new application, the Flash BOOT program must be launched. This happens with this command. Executing this command is confirmed with ACK. When the device is already in the FBOOT mode there is answered with a NAK command.

Request:

Command	OpCode
0x5E	0x01

Reply (ACK):

Command
0x55

Erase flah
The Flash ROM is divided into a number of sectors, each with its own size. The first sector, starting at address 0, is reserved for the FBOOT-application. The target application starts, depending on the used Flash ROM, at 0x4000 or 0x8000. The type and structure of the Flash ROM can be requested with the FLASH_INFO function.

Before being able to program a sector, it must first be erased. This happens with the FLASH_ERASE command. For this the start address of the sector is set in the address field and the function is called. An address in the first sector, the FBOOT-Application sector, is not allowed and results in an ERROR reply.

Executing this command may take some time to complete, about one second. Attention must be paid to the settings of the protocol timeout parameters. The delete action is checked and ACK is reported on success. If erasing is not successful there is answered with a NAK command.

Request:

Command	OpCode	Address
0x5E	0x02	0x00000000

Reply (ACK):

Command
0x55

Write flash

After erasing the sector, it can be written. Writing is done by write the data in small chunks into the Flash. Every cycle the write address is increases with the size of the previously written block.

The number of bytes per function call can be written is determined by the maximum number of bytes that fits in a request. Assuming a frame size of 256 bytes, this means that a maximum of 246 bytes can be sent. If, in the future, use is made of a gateway function, this will need to be further bounded with 2 bytes per hop.

As a check, the so determined size of the data length is completed to the len field and passed to the function. Then the data is added to the function in the open array data.

After programming, the data is compared with the programmed data and if they match, the command is answered with ACK. In case of an incorrect programming action a NAK is forwarded.

Request:

Command	OpCode	Address	# of bytes to write	Data
0x5E	0x03	0x00000000	0x00	Byte[]

Reply (ACK):

Command
0x55

Read flash

For verification, it's possible to read back the written data. Care must be taken to ensure that the size of the requested data is not more than what can be answered in a frame. Although the data length is limited by the device within the maximum frame, in case of use of a gateway it's possible that this will still overflow. In the response, the address and the data length are normally not changed, unless there is an excessive length requested. Then it will be limited. The requested data is then added to the request structure by means of the open data array.

With an incorrect length of the request frame an ERROR is reported back.

Request:

Command	OpCode	Address	# of bytes to read
0x5E	0x04	0x00000000	0x00

Reply:

Command	OpCode	Address	# of bytes to read	Data
0x5E	0x04	0x00000000	0x00	Byte[]

Boot application

After complete programming of the target application in the Flash ROM, this can be booted using this command. Executing this command is confirmed by an ACK command. Starting the application is only possible from the FBOOT-application. If this is not the case then a NAK command is given as an answer.

After the restart there can be requested if the target application is indeed active, using the query application command.

Request:

Command	OpCode
0x5E	0x05

Reply (ACK):

Command
0x55

Read application

This function can be used to ask information about which program is running. It can also be called to gain the checksum of the target application, which will be calculated then.

When FBOOT-application is active, a 1 comes back into the application code field. If the target application is active, a 0 comes back here. The checksum field contains the calculated checksum of the target application.

Request:

Command	OpCode
0x5E	0x06

Reply:

Command	OpCode	Checksum	Application code
0x5E	0x06	0x0000	0x00

Read flash type information

During programming an application, the programming program must take into account the sector classification of the Flash ROM. This is of course possible on the basis of the code device which can be requested with the ID command, but it's more practical to use a universal method. This function is developed for that purpose.

The principle of this method is that the programmer program first asks the device the size of the Flash-ROM to be programmed. As there may be several different Flash-ROMs in a system, it's important to know from which address the information is desired. For this purpose, the following request is sent out.

The response contains information about the type and brand of the Flash ROM. This information can be used for visual feedback to the user interface and for determining the application start address. The brand and type codes correspond to the codes contained in the manufacturer's documentation. Some examples are given in the table below.

8 bit	16 bit	Type
0x0020	-	Am29F010
0x00A4	-	Am29F040
0x0051	0x2251	Am29F200T
0x0052	0x2252	Am29F200B
0x0023	0x2223	Am29F400T
0x00AB	0x22AB	Am29F400B

On the basis of the identification code of the Flash ROM some additional details are sent in the reply; the total size of the Flash ROM, the number of sectors and a table with the size of each sector.

With the aid of the table sector, it's now possible to implement a correct programmer algorithm. Attention must be paid if a block to write is going over the border of a sector. When this occurs, first a flash erase of the sector should take place.

By using a sector table stored in the device instead of using a table in the programmer program it's possible to provide a device with a different type/brand flash ROM without the need to adapt the programmer program.

Request:

Command	OpCode	Address
0x5E	0x07	0x00000000

Reply:

Command	OpCode	Address	Manufacturer id	Device id	Flash size	# of sectors	Sector table
0x55	0x07	0x00000000	0x0000	0x0000	0x00000000	0x0000	0x00000000

Read or program a serial number

This function is used to retrieve or set the serial number of the device. If the serial number is 0xFFFFFFFF, it's not yet programmed and it can be programmed in the device. A programmed number cannot be replaced by another number. The build-up of the serial number consists of 8 BCD digits. From left to right the production year and week number and the serial number can be read back. An example: serial number is 0x14190001. This is the first device which is manufactured in week 19 of 2014.

Request:

Command	OpCode
0x5E	0x08

Reply:

Command	OpCode	Serial
0x5E	0x08	0x00000000

Request:

Command	OpCode	Serial
0x5E	0x08	0x00000000

Reply (ACK):

Command
0x55

5.12 Command code 0x64: Echo functions

This interface is designed to carry out connection and performance testing. The functionality of this command is present in each device so that it can be used for checking of the connection, a type of PING function.

The data sent to this interface is unmodified sent back from the slave to the master again. It's therefore possible with this interface to perform tests to determine the performance that the connection can take out. The processing of this command requires a minimal effort on the side of the slave, so there can be carried out a pure measurement on the link performance.

Request:

Command	Data
0x64	Byte []

Reply:

Command	Data
0x64	Byte []

5.13 Command code 0x78: Controller interface

The controller interface provides all controller data. The interface is divided into the following blocks:

- System functions
- Input/output
- Extended registers
- Indicator
- Labels

5.13.1 System functions

System interface is for diagnostic purposes such as the recognition of the base interface and in-operation status.

OpCode	Description
0x00	Feature available
0x01	System diagnostics

Check if the feature is available

The availability of the interface can be detected with this function. If present, a reply with the system command ACK follows. If the interface is not present there is automatically answered with the ERROR command.

Request:

Command	OpCode
0x78	0x00

Reply (ACK):

Command
0x55

Get the system diagnostics

System diagnostics such as number of resets and operation time is retrieved with this command. The structure of the command is shown below.

Request:

Command	OpCode
0x78	0x01

Reply:

Command	OpCode	# of resets	Alive time in seconds	Up time in seconds
0x78	0x01	0x00000000	0x00000000	0x00000000

5.13.2 Input/output

With the input / output interface, inputs, outputs and markers can be read. Markers can also be written. Markers are outputs but without physical outputs.

OpCode	Description
0x14	Get the I/O structure info
0x15	Read the I/O status
0x16	Set markers
0x17	Reset markers

Get the I/O structure info

The I/O parameters are retrieved with this command.

Request:

Command	OpCode
0x78	0x14

Reply:

Command	OpCode	# of inputs	# of outputs	# of markers	# of internal markers
0x78	0x14	0x0000	0x0000	0x0000	0x0000

Input offset	Output offset	Marker offset	Internal marker offset	Device offset
0x0000	0x0000	0x0000	0x0000	0x0000

Example

Get I/O structure info:

Request:

Command	OpCode
0x78	0x14

Reply:

Command	OpCode	# of inputs	# of outputs	# of markers	# of internal markers
0x78	0x14	0x0028	0x0028	0x0258	0x03E8

Input offset	Output offset	Marker offset	Internal marker offset	Device offset
0x0000	0x00C8	0x0190	0x2328	0x03E8

Result:

Number of inputs	40
Number of outputs	40
Number of markers	600
Number of internal markers	1000
Input offset	0
Output offset	200
Marker offset	400
Internal marker offset	9000
Device offset	1000

Read the I/O status
The I/O status is retrieved with this command. Multiple groups of I/O can be requested.

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes
0x78	0x15	0x00	0x00	0x0000	0x00	0x00

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes	I/O data
0x78	0x15	0x00	0x00	0x0000	0x00	0x00	Byte []

For every task (# of tasks) a parameter field (start address + Reserved + # of bytes) has to be added to the frame. The reply frame will show the full request frame extended with the I/O data of all tasks.

Examples

Read the status of inputs 1 - 8:

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes
0x78	0x15	0x00	0x01	0x0000	0x00	0x01

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes	I/O data
0x78	0x15	0x00	0x01	0x0000	0x00	0x01	0x01

The I/O data shows that input 1 is high (I/O data = 0x01).

Read the status of outputs 1 - 8:

The outputs have an offset of 200 according to the I/O structure as shown above. The start address has to be entered as number of bytes. 200 = 25 bytes. 25 = hex 0x19.

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes
0x78	0x15	0x00	0x01	0x0019	0x00	0x01

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes	I/O data
0x78	0x15	0x00	0x01	0x0019	0x00	0x01	0x03

The I/O data shows that output 1 and 2 are high (I/O data = 0x03).

Read the status of markers 401 - 408:

The markers have an offset of 400 according to the I/O structure as shown above. The start address has to be entered as number of bytes. 400 = 50 bytes. 50 = hex 0x32.

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes
0x78	0x15	0x00	0x01	0x0032	0x00	0x01

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes	I/O data
0x78	0x15	0x00	0x01	0x0032	0x00	0x01	0x07

The I/O data shows that markers 401, 402 and 403 are high (I/O data = 0x07).

Read the status of markers 401 - 416:

The same request is used as for marker 401 - 408, only in this case the returned I/O data is 2 bytes.

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes
0x78	0x15	0x00	0x01	0x0032	0x00	0x02

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes	I/O data	I/O data
0x78	0x15	0x00	0x01	0x0032	0x00	0x01	0x07	0x01

The first I/O data byte (0x07) shows that markers 401, 402 and 403 are high. The second I/O data byte (0x01) shows that marker 409 is high.

Read the status of outputs 1 - 8 and markers 401 - 408:

Two requests are made in this case. Therefor set # of tasks to 2. The first task (Start address + Reserved + # of bytes) is setup to read the status of outputs 1 - 8. The second task is setup to read the status of markers 401 - 408.

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes	Start address
0x78	0x15	0x00	0x02	0x0019	0x00	0x01	0x0032

Reserved	# of bytes
0x00	0x01

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of bytes	Start address
0x78	0x15	0x00	0x02	0x0019	0x00	0x01	0x0032

Reserved	# of bytes	I/O data	I/O data
0x00	0x01	0x03	0x07

The first I/O data byte (0x03) shows that output 1 and 2 are high. The second I/O data byte (0x07) shows that markers 401, 402 and 403 are high.

Set markers

The markers are set with this command. For successful processing, the command is answered with the system command ACK.

Request:

Command	OpCode	Reserved	# of markers	Marker
0x78	0x16	0x00	0x00	0x0000

Reply (ACK):

Command
0x55

For every marker (# of markers) a marker address (0x0000) has to be added to the frame.

Examples

Set marker 401 (hex 0x0191):

Request:

Command	OpCode	Reserved	# of markers	Marker
0x78	0x16	0x00	0x01	0x0191

Reply (ACK):

Command
0x55

Marker 401 is set.

Set markers 401 and 402 (hex 0x0191 and 0x0192):

Request:

Command	OpCode	Reserved	# of markers	Marker	Marker
0x78	0x16	0x00	0x02	0x0191	0x0192

Reply (ACK):

Command
0x55

Markers 401 and 402 are set.

Reset markers

The markers are reset with this command. After successful processing, the command is answered with the system command ACK.

Request:

Command	OpCode	Reserved	# of markers	Marker
0x78	0x17	0x00	0x00	0x0000

Reply (ACK):

Command
0x55

For every marker (# of markers) a marker address (0x0000) has to be added to the frame.

Examples

Reset marker 401 (hex 0x0191):

Request:

Command	OpCode	Reserved	# of markers	Marker
0x78	0x17	0x00	0x01	0x0191

Reply (ACK):

Command
0x55

Marker 401 is reset.

Reset markers 401 and 402 (hex 0x0191 and 0x0192):

Request:

Command	OpCode	Reserved	# of markers	Marker	Marker
0x78	0x17	0x00	0x02	0x0191	0x0192

Reply (ACK):

Command
0x55

Markers 401 and 402 are reset.

5.13.3 Extended registers

With this interface, access is obtained to extended registers that can be read and written. The extended registers are 32 bit signed numbers. The registers 1 – 100 are battery backed up so they will always store their content.

OpCode	Description
0x1E	Get register info
0x1F	Read registers
0x20	Write registers

Get register info

The Extended registers parameters are retrieved with this command.

Request:

Command	OpCode
0x78	0x1E

Reply:

Command	OpCode	# of registers
0x78	0x1E	0x0000

Read registers

The Extended registers are accessed with this command. Multiple groups extended registers can be accessed through this structure.

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of registers
0x78	0x1F	0x00	0x00	0x0000	0x00	0x00

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of registers	Register data
0x78	0x1F	0x00	0x00	0x0000	0x00	0x00	Byte []

For every task (# of tasks) a parameter field (start address + Reserved + # of registers) has to be added to the frame. The reply frame will show the full request frame extended with the register data of all tasks.

Examples

Read extended register 1:

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of registers
0x78	0x1F	0x00	0x01	0x0000	0x00	0x01

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of registers	Register data
0x78	0x1F	0x00	0x01	0x0000	0x00	0x01	0x00000001

Extended register 1 has a value of 1.

Read extended registers 1 and 2:

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of registers
0x78	0x1F	0x00	0x01	0x0000	0x00	0x02

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of registers
0x78	0x1F	0x00	0x01	0x0000	0x00	0x02

Register data	Register data
0x00000001	0x00000002

Extended register 1 has a value of 1 and extended register 1 has a value of 2.

Read extended registers 1 and 11:

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of registers
0x78	0x1F	0x00	0x02	0x0000	0x00	0x01

Start address	Reserved	# of registers
0x000A	0x00	0x01

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of registers
0x78	0x1F	0x00	0x01	0x0000	0x00	0x01

Start address	Reserved	# of registers	Register data	Register data
0x000A	0x00	0x01	0x00000001	0x00000011

Extended register 1 has a value of 1 (0x00 00 00 01) and extended register 11 has a value of 17 (0x00 00 00 11).

Write registers

The Extended registers are written with this command. After successful processing, the command is answered with the system command ACK.

Request:

Command	OpCode	Register	Data
0x78	0x20	0x0000	0x00000000

Reply (ACK):

Command
0x55

Examples

Write to extended register 1 (first register is address 0x0000) value 123 (hex 0x0000007B):

Request:

Command	OpCode	Register	New value
0x78	0x20	0x0000	0x0000007B

Reply (ACK):

Command
0x55

Extended register 1 now has the value 123.

Write to extended register 2 (address 0x0001) value 2400 (hex 0x00000960):

Request:

Command	OpCode	Register	New value
0x78	0x20	0x0001	0x00000960

Reply (ACK):

Command
0x55

Extended register2 now has value 2400.

5.13.4 Indicator

This interface provides access to the indicator registers and is read only. Indicator registers are both internal and external scales connected to the device. Each register represents the indicator display value including formatting and status information.

OpCode	Description
0x28	Get indicator info
0x29	Read indicator

Get indicator info

The indicator registers parameters are retrieved with this command. Device offset is the logical numbering offset within a device. For example, indicators can be numbered starting at 100 instead of 1. Communication numbering always starts from 0.

Request:

Command	OpCode
0x78	0x28

Reply:

Command	OpCode	# of indicators	Device offset
0x78	0x28	0x0000	0x0000

Read indicator

The indicator registers are accessed with this command. Multiple groups of indicator registers can be requested through this structure. The field parameter indicates the number of structures.

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of indicators
0x78	0x29	0x00	0x00	0x0000	0x00	0x00

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of indicators	Indicators
0x78	0x29	0x00	0x00	0x0000	0x00	0x00	Byte []

For every task (# of tasks) a parameter field (start address + Reserved + # of indicators) has to be added to the frame. The reply frame will show the full request frame extended with the indicator data of all tasks.

The 32 bit Indicator field is constructed as follows:

8 bit status	24 bit signed indicator value

Status bits:

Status bits	Description
0x00	No decimal point position
0x01	Decimal point “00000.0”
0x02	Decimal point “0000.00”
0x03	Decimal point “000.000”
0x04	Decimal point “00.0000”
0x05	Decimal point “0.00000”
0x06	Decimal point “.000000”
0x07	Format mask
0x08	Indicator tare active
0x10	Indicator stable active
0x20	Indicator zero range
0x40	Indicator error
0x80	indicator avail, value is valid

Examples

Read indicator 1:

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of indicators
0x78	0x29	0x00	0x01	0x00 00	0x00	0x01

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of indicators	Indicator
0x78	0x29	0x00	0x01	0x00 00	0x00	0x01	0xBA002710

The first byte (0xBA) indicates the status.

“B” means that the indicator is in zero range, indicator stable is active and the indicator is available, the value is valid.

“A” means that the decimal point is “0000.00” and the indicator tare is active.

The next three bytes indicate the weight value. 0x00 27 10 so the value is 10000.

Read indicator 1 and 2:

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of indicators
0x78	0x29	0x00	0x01	0x0000	0x00	0x02

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of indicators
0x78	0x29	0x00	0x01	0x0000	0x00	0x02

Indicator	Indicator
0xBA002710	0xBA00137E

First indicator:

The first byte (in this case 0xBA) indicates the status.

“B” means that the indicator is in zero range, indicator stable is active and the indicator is available, the value is valid.

“A” means that the decimal point is “0000.00” and the indicator tare is active.

The next three bytes indicate the weight value. 0x00 27 10 so the value is 10000.

Second indicator:

The first byte (in this case 0xBA) indicates the status.

“B” means that the indicator is in zero range, indicator stable is active and the indicator is available, the value is valid.

“A” means that the decimal point is “0000.00” and the indicator tare is active.

The next three bytes indicates the weight value. 0x00 13 7E so the value is 4990.

Read indicator 1 and 3:

Request:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of indicators
0x78	0x29	0x00	0x02	0x0000	0x00	0x01

Start address	Reserved	# of indicators
0x0002	0x00	0x01

Reply:

Command	OpCode	Reserved	# of tasks	Start address	Reserved	# of indicators
0x78	0x29	0x00	0x02	0x0000	0x00	0x01

Start address	Reserved	# of indicators	Indicator	Indicator
0x0002	0x00	0x01	0xBA002710	0xBA00137E

First indicator:

The first byte (in this case 0xBA) indicates the status.

“B” means that the indicator is in zero range, indicator stable is active and the indicator is available, the value is valid.

“A” means that the decimal point is “0000.00” and the indicator tare is active.

The next three bytes indicate the weight value. 0x00 27 10 so the value is 10000.

Second indicator:

The first byte (in this case 0xBA) indicates the status.

“B” means that the indicator is in zero range, indicator stable is active and the indicator is available, the value is valid.

“A” means that the decimal point is “0000.00” and the indicator tare is active.

The next three bytes indicates the weight value. 0x00 13 7E so the value is 4990.

5.13.5 Labels

This interface gives access to the labels. Labels are text fields with a fixed length and are used for printer layouts and external screens.

OpCode	Description
0x32	Get label info
0x33	Load labels from flash
0x34	Save labels to flash
0x35	Read labels from device
0x36	Write labels to device

Get label info

The structure of the labels is retrieved with this command.

Request:

Command	OpCode
0x78	0x32

Reply:

Command	OpCode	Max label width	Max # of labels
0x78	0x32	0x0000	0x0000

Load labels from flash

With this command the labels are loaded from the device flash memory and are placed in the device working memory. A successful operation is replied with an ACK command.

Request:

Command	OpCode
0x78	0x33

Reply (ACK):

Command
0x55

Save labels to flash

With this command the labels are read from the device working memory and saved to the device flash memory. A successful operation is replied with an ACK command.

Request:

Command	OpCode
0x78	0x34

Reply (ACK):

Command
0x55

Read labels from device

With this command one or more labels are read from the device. Every label is null terminated.

Request:

Command	OpCode	Start label	# of labels
0x78	0x35	0x0000	0x0000

Reply (ACK):

Command	OpCode	Start label	# of labels	Labels
0x78	0x35	0x0000	0x0000	Byte []

Examples

Read the first two labels (first label is address 0x0000):

Request:

Command	OpCode	Start label	# of labels
0x78	0x35	0x0000	0x0002

Reply (ACK):

Command	OpCode	Start label	# of labels	Label	Label
0x78	0x35	0x0000	0x0002	0x544558542020203100	0x544558542020203200

Label 1 = 0x544558542020203100 = TEXT 1 *(the "00" is the null termination)*

Label 2 = 0x544558542020203200 = TEXT 2 *(the "00" is the null termination)*

Write labels to device

With this command one or more labels are written to the device working memory. A successful operation is replied with an ACK command. Every label shorter than the maximum number of character must be null terminated.

Request:

Command	OpCode	Start label	# of labels	Labels
0x78	0x36	0x0000	0x0000	Byte []

Reply (ACK):

Command
0x55

Examples

Write PENKO to label 5 (address 0x0004):

PENKO in hex = 0x50 45 4E 4B 4F + null termination = 0x50454E4B4F00

Request:

Command	OpCode	Start label	# of labels	Labels
0x78	0x36	0x0004	0x0001	0x50454E4B4F00

Reply (ACK):

Command
0x55

5.14 Command code 0xB4: PDI functions

See the **PENKO PDI Protocol** document, available on www.penko.com

About PENKO

At PENKO Engineering we specialize in weighing. Weighing is inherently chemically correct, independent of consistency, type or temperature of the raw material. This means that weighing any kind of material guaranties consistency and thus, it is essential to sustainable revenue generation in any industry. As a well-established and proven solution provider, we strive for the ultimate satisfaction of custom design and/or standard applications, increasing your efficiencies and saving you time, saving you money.

Whether we are weighing raw materials, components in batching, ingredients for mixing or dosing processes, - or weighing of static containers and silos, or - in-motion weighing of railway wagons or trucks, by whatever means required during a process, we are essentially forming vital linkages between processes and businesses, anywhere at any time. We design, develop and manufacture state of the art technologically advanced systems in accordance with your strategy and vision. From the initial design brief, we take a fresh approach and a holistic view of every project, managing, supporting and/or implementing your system every step of the way. Curious to know how we do it? www.penko.com

Certifications

PENKO sets high standards for its products and product performance which are tested, certified and approved by independent expert and government organizations to ensure they meet – and even – exceed metrology industry guidelines. A library of testing certificates is available for reference on:
http://penko.com/nl/publications_certificates.html

PENKO Professional Services

PENKO is committed to ensuring every system is installed, tested, programmed, commissioned and operational to client specifications. Our engineers, at our weighing center in Ede, Netherlands, as well as our distributors around the world, strive to solve most weighing-system issues within the same day. On a monthly basis PENKO offers free training classes to anyone interested in exploring modern, high-speed weighing instruments and solutions. Training sessions on request: **www.penko.com/training**

PENKO Alliances

PENKO's worldwide network: Australia, Brazil, China, Denmark, Germany, Egypt, Finland, France, India, Italy, Netherlands, Norway, Poland, Portugal, Slovakia, Spain, Syria, Turkey, United Kingdom, South Africa, Slovakia Sweden and Switzerland, Singapore.
A complete overview you will find on: **www.penko.com/dealers**

PENKO Engineering B.V. ▪ Schutterweg 35, NL 6718XC Ede ▪ Tel +31 (0) 318525630 ▪ Fax +31 (0) 31852971 ▪ info@penko.com
Web ▪ www.penko.com ▪ Copyright © 2014 ETC All rights reserved. 7600L1106 EN PENKO TP Protocol R6