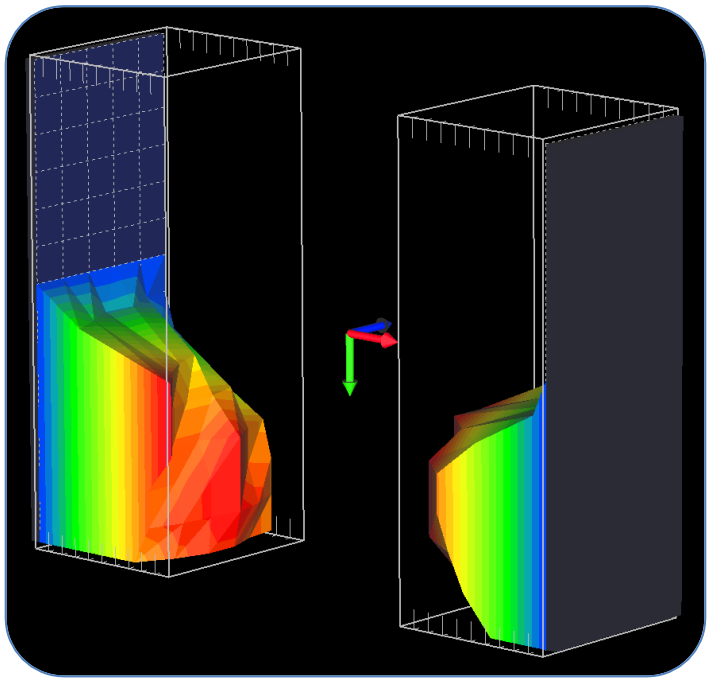
WTS family of Tactile Sensor Modules

- Command Set Reference Manual -

Firmware Version 1.2.0

March 2013



Contents

Weiss Robotics GmbH & Co. KG
In der Gerste 2
D-71636 Ludwigsburg, Germany

Phone: +49 (0)7141 642062-6
Fax: +49 (0)7141 642062-9
email: office@weiss-robotics.com

1 Introduction

The WTS family of tactile transducers can be controlled over a serial UART connection or a USB port using a Virtual COM Port Driver. This manual describes how to set up the communication interface and gives a detailed explanation of the protocol as well as of the WTS Command Set.

The following assumptions are made throughout the manual unless otherwise noted:

- Hexadecimal values are noted with a trailing “h”, e.g. 12h while decimal values are not specially marked.
- The data transmission is based on a little endian representation of multi-byte words where the least significant byte is transferred first. This means that the integer number 1234h is represented by two consecutive bytes 34h and 12h.
- Reserved values must always be set to 0 and read as undefined. This ensures compatibility with future versions.

The following data types are used by the command set:

- **integer:** Integer number of either 8, 16 or 32 Bit length
- **string:** An ASCII text that must not contain any control characters
- **bit vector:** usually flags where each bit has its special meaning
- **enum:** Enumeration. Similar to integer but every value has a special meaning.

1.1 Data Acquisition

The basic data acquisition path of the WTS family of tactile transducers is shown in Figure 1. It consists of a Frame Digitizer that samples the tactile sensing matrix and creates a digital representation of the pressure profile. This Frame Digitizer meets the strict timing requirements to achieve an optimum sensor signal quality.

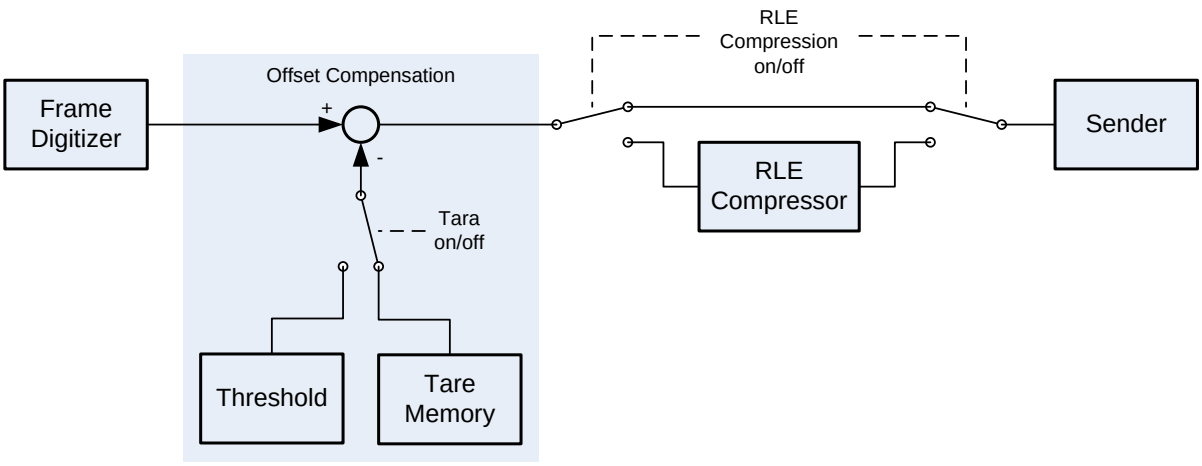


Figure 1: WTS Data Acquisition Path

The acquired frame data is passed to an Offset Compensation that applies either a constant threshold value (see Set Threshold (34h) Command on page 27) or a cell-by-cell tare value that was determined before by issuing the Tare Sensor Matrix (23h) Command (see page 18). For both modes, the output signal will always remain positive, i.e. negative values will be clipped to 0. The offset compensation mode is set by the last of these two commands that was issued.
After Offset Compensation stage, the frame can be passed to a RLE Compressor, depending on the flag settings of the respective acquisition command. This stage will compress zero values thus effectively reducing the amount of data to be sent in the last stage, the Frame Sender.

☞ **A detailed description of the compression algorithm together with a code example can be found in Appendix B.**

Data can either be acquired in a continuous loop or as a single frame, so the WTS transducer can be flexibly adapted to the application requirements. In loop mode, the transducer uses double-buffering for maximum throughput.

1.2 Sensor Cell Numbers

On rectangular sensors, the sensor cells of the tactile sensing matrix are numbered starting at the top left corner and proceeding line-wise to the bottom right, as shown in Figure 2 (left) for a converter with a horizontal resolution of eight sensor cells.

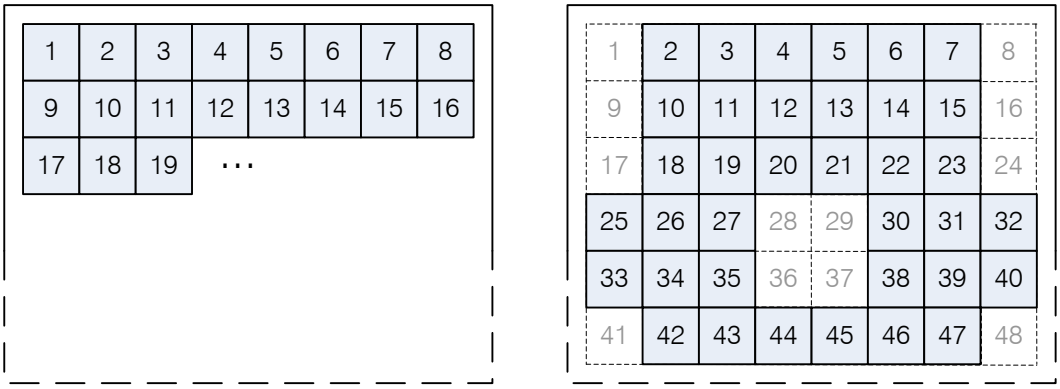


Figure 2: Sensor cell enumeration

For non-rectangular sensors, the sensor cell numbers are determined by its bounding rectangle, again starting with number 1 in the top left edge. Depending on the sensor shape, there may be sensors without a “Number 1” sensor cell, see Figure 2 (right).

1.3 Connecting to the WTS Tactile Sensor Module

The WTS family of tactile sensor modules offers two digital interfaces for communication: serial UART and USB 2.0. They can be selected via the WTS Configuration Shell that is accessible over the UART interface:

1. Connect a VT-100 compatible terminal emulator (e.g. minicom on Linux or HyperTerminal on Microsoft Windows) to the WTS’ UART, using 115200 Baud, no Parity and one Stop Bit. Be sure, that the terminal emulation mode is set to VT-100. A sample configuration file for HyperTerminal can be found on the Companion CD.
2. Switch on the power supply of the WTS. The WTS will offer a text-based configuration shell.

```
++=================================================================++
||          Welcome to the system configuration shell              ||
++=================================================================++
|                                                                   |
|  Enter 'help' to get a list of possible commands. To obtain       |
|  detailed information about a command, enter <command> help.      |
|                                                                   |
|  Serial Number: 00000000                                          |
|  Firmware version: 1.0.0                                          |
|                                                                   |
+-------------------------------------------------------------------+
|  Copyright 2011 Weiss Robotics, Germany. All rights reserved.     |
+-------------------------------------------------------------------+

USER> _
```

3. Type in “interface” at the command prompt. The WTS will answer:

```
USER> interface
Current interface is: serial, bitrate: 115200 Baud
```

4. You can change the interface settings by entering
   a. “interface USB” -> changes the communication interface to USB
   b. “interface serial” -> changes the communication interface to UART. The WTS will ask you for the desired bit rate
   c. “interface auto” -> enables automatic interface selection
5. Type in “exit” to leave the configuration shell. The settings are stored nonvolatile inside the internal configuration memory.

☞ **An In-depth description of the WTS Configuration Shell can be found in the document “WTS Configuration Shell Manual”**

1.3.1 Serial UART Interface

The bit rate of the serial UART interface can be set via the WTS Configuration Shell using the *interface* command. The default bit rate is 115.200 Baud.

⚠ **To connect the WTS to a host that provides a generic RS 232 interface (“COM-Port”), a level translation circuitry is required, as the WTS uses LVTTL levels on its UART pins.**

1.3.2 USB Interface

The USB Interface uses the standard CDC-Class for communication, so on most operating systems, the WTS can be accessed by a virtual COM-Port that will be generated if the device is connected.
On Microsoft Windows, the file wts.inf is required to install the COM port. It can be found on the WTS companion CD.
On Linux, the device is enumerated automatically and can be accessed without any additional driver files. The interface path is */dev/ttyACMx*, where x has to be replaced by the number of the actual interface (you might use the *dmesg* command inside a console window to determine it).

1.3.3 Automatic Interface Selection

Besides of an explicit setting of either USB or UART to be used for communication, an auto detection mode allows to select the right communication interface based on the state of the USB connection: if

USB is connected on startup and the device was properly enumerated by the Host (i.e. the device driver was installed), USB is selected as communication interface. Otherwise, the UART connection is used with the selected bit rate.

1.4 Communicating with the WTS

Regardless of the interface used, the WTS family of tactile sensor modules communicates with its host using binary data packets. The following chapters describe the general format of these commands.

1.4.1 Command Message Format

Command messages start with a preamble signaling the beginning of a new data packet. The table below illustrates the command format. An identification code describes the content of the packet. It is used as command identifier and distinguishes the several commands of the device. The two byte size value determines the size of the packet's payload in bytes. A two byte CRC checksum is added to each packet to verify data integrity.

☞ **A source code example for calculating the checksum over a message is given in Appendix C.**

To check a received message, you have to calculate the CRC again over the received data (with the preamble) including the received checksum. If the received data is correct, the calculated checksum is 0.

☞ **For your first steps, we recommend to use the Custom Command Editor function of the WTS Commander tool (see chapter 1.4.3) for interactively assembling valid data packets.**

Byte	Symbol	Description
0..2	PREAMBLE	Signals the begin of a new message and has to be AAAAAAh
3	COMMAND_ID	Command identifier
4..5	SIZE	Size of the packet's payload in bytes. Might be 0 for signaling packets.
6..n	PAYLOAD	Payload data

n+1..n+2	CHECKSUM	CRC checksum of the whole data packet, including the preamble. See Appendix C on how to calculate the checksum.

Example 1: Packet with ID = 1, no payload:

AAh AAh AAh 01h 00h 00h E8h 10h

Example 2: Packet with ID = 1, two bytes payload (12h, 34h), checksum is 666Dh:

AAh AAh AAh 01h 02h 00h 12h 34h 6Dh 66h

1.4.2 Command Acknowledge from the Device

Every command is acknowledged by the WTS using a standardized acknowledge packet according to the following format:

Byte	Symbol	Description
0..2	PREAMBLE	Signals the begin of a new message and has to be AAAAAAh
3	COMMAND_ID	Command Identifier
4..5	SIZE	Size of the packet's payload in bytes. This is n – 4, e. g. 2 for a packet with an error code other than E_SUCCESS or 6 for a packet returning E_SUCCESS and a 4-byte command-specific parameter.
6..7	STATUS_CODE	Status code, see Appendix A on page 37.
8..n	PARAMS	Command specific parameters. Only available, if the error code is E_SUCCESS.
n+1..n+2	CHECKSUM	CRC checksum of the whole data packet, including the preamble. See Appendix B on how to check this checksum. **When computing the CRC checksum over the whole data including this checksum field, the result has to be 0.**

Example 1: Acknowledging a successfully executed command without any return parameters (here: Loop Command):

Host sends:

AAh AAh AAh 06h 00h 00h 97h 26h

Device answers:

AAh AAh AAh 06h 02h 00h 00h 00h F9h F7h

Example 2: Acknowledging an erroneous command (here, Command ID 0x90 is unknown, so the device returns an E_CMD_UNKNOWN (000Eh) error with this ID).

Host sends:

AAh AAh AAh 90h 01h 00h 00h 00h B3h FDh

Device answers:

AAh AAh AAh 90h 02h 00h 0Eh 00h FDh 02h

Example 3: Acknowledging a successfully executed “Get Threshold”-Command, returning a 2-byte integer parameter (here 150):

Host sends:

AAh AAh AAh 35h 00h 00h F1h 2Ch

Device answers:

AAh AAh AAh 35h 04h 00h 00h 00h 96h 00h 97h 78h

1.4.3 Frame Format for Periodic Data Acquisition

When enabling periodic data acquisition via the Start Periodic Frame Acquisition (21h) Command (chapter 3.1.2), frames are acquired and sent using a special message format. The identifier of a data packed is always 00h. There is no other command using this identifier, so frame data can easily be recognized. The payload contains a four byte timestamp which reflects the sensor controller time in 1/10 milliseconds when the frame was acquired and can be used to reconstruct the chronological sequence of the data. The frame data itself is transferred using 16-Bit wide data words for each sensor cell. To achieve a higher throughput over the data interface, the frame data can be run-length encoded (RLE). Please see Appendix B for details.

Byte	Symbol	Description
0..2	PREAMBLE	Signals the begin of a new message and has to be AAAAAAh
3	COMMAND_ID	Command Identifier. 00h for data frames.
4..5	SIZE	Size of the packet’s payload in bytes. For uncompressed frames, this is 2n + 5, where n is the number of sensor cells of the matrix.

6...9	TIMESTAMP	Acquisition time stamp in 1/10 milliseconds.
10	FLAGS	**Frame-specific flags** D7..2 reserved D1 1: Frame data is RLE compressed 0: Frame data is uncompressed D0 reserved
11.. 2n+10	FRAMEDATA	Raw or compressed frame data. 2 Bytes per sensor cell value.
2n+11.. 2n+12	CHECKSUM	CRC checksum of the whole data packet, including the preamble. See Appendix B on how to check this checksum. **When computing the CRC checksum over the whole data including this checksum field, the result has to be 0.**

2 WTS Commander

The tool WTS Commander is provided free of charge and allows an easy familiarization with the WTS family of tactile sensor modules communication protocol and command set. It allows you to send basic commands to the connected WTS and contains a custom command editor to assemble own command packets. The data traffic to and from the device is displayed, so the communication can be understood easily. The software is running on Microsoft Windows XP and can be installed from either your Product CD that ships together with the WTS sensors.

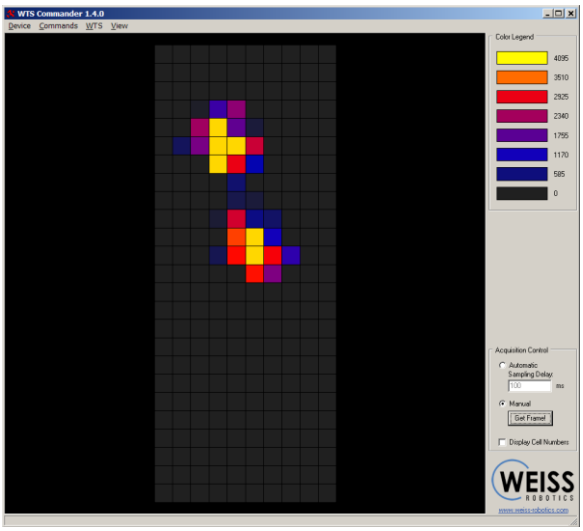


Figure 3: WSG Commander Main Window

Main Window

To connect to your WTS, select *Device*|*Connect* from the main menu and select the communication interface. Depending on the interface, additional settings may be necessary. Please note that the gripper has to be configured to use the selected interface via its Web Interface. Currently, the following interfaces are supported: RS232 (including USB virtual COM-Port), CAN-Bus via ESD-Cards as well as Ethernet TCP/IP and UDP/IP. Note, that CAN-Bus and Ethernet is not available on all WTS sensor modules.

Command Editor

Besides the predefined commands on the main window, you can compose your own commands using the Custom Command Entry dialog and send them to the device. To open this dialog, select *Commands*|*Command Editor…* from the main menu. You can either choose from predefined IDs or

enter your own values here. The payload can consist of bytes in either decimal or hexadecimal (starting with „0x“, e.g. 0x20) format, floating point values (followed by a „f“, e.g. 150.0f) or text strings (entered in quotation marks, e.g. ”text”). The entered data is converted online into a data packet. By clicking on the Send-Button, it is transferred to the device.

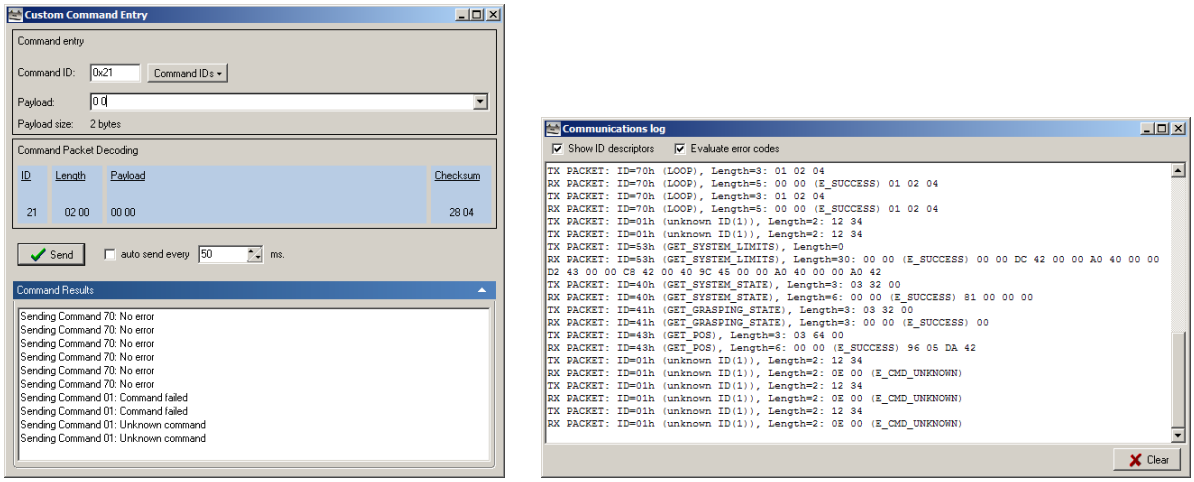


Figure 4: WTS Commander Custom Command Editor (left) and Communication Log (right)

Communication Log

To follow the communication between the WTS Commander and the target device, you may use the Communication Log Panel. It can be accessed using the main menu’s *View|Command log* entry. You can select the log panel to decode IDs and error codes automatically.

If you select one or more bytes inside the log, a popup-menu is displayed and you can convert the selected bytes into their integer or floating point representation as well as decode them as a text string.

The WTS Commander is intended as a tool to you to make the evaluation of the sensor’s command set as easy and comfortable as possibly. It comes with no warranty and is not intended to be used in any production environment!

3 Command Set Reference

The following chapter describes the command set of the WSG in detail.

3.1 Data Acquisition

3.1.1 Read Single Frame (20h)

Acquire and return a single frame from the sensor matrix. This command is intended to acquire discrete frames based on some external event. The frame data is returned as parameter of its acknowledgement message. Depending on the FLAGS field, the data will be RLE compressed. For details about the compression algorithm, please see Appendix C on page 39.

The WTS family of tactile sensors also allows periodic acquisition of frames, see chapter 3.1.2.

⚠ **This command is only available if periodic frame acquisition is disabled (chapter 3.1.2)**

Command ID: 20h

Command Parameters:

Byte	Symbol	Data Type	Description
0	FLAGS	bit vector	**Acquisition control flags** D7…D1 reserved D0 1: Enable RLE Compression of frame data 0: Disable RLE Compression

Returned Parameters:

Byte	Symbol	Data Type	Description
0..3	TIMESTAMP	integer	Acquisition time stamp in 1/10 milliseconds.

4	FLAGS	bit vector	**Frame-specific flags** D7..2 reserved D1 1: Frame data is RLE compressed 0: Frame data is uncompressed D0 reserved
5..6	CELL1	integer	Value of sensor cell 1
7..8	CELL2	integer	Value of sensor cell 2
…			
2n+5.. 2n+6	CELLn	integer	Value of sensor cell n

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

E_ACCESS_DENIED: Data acquisition is currently busy

E_INSUFFICIENT_RESOURCES: Out of memory

3.1.2 Start Periodic Frame Acquisition (21h)

Data can either be acquired in a continuous loop or as a single frame. This command configures the data acquisition loop of the tactile sensor. For single frame acquisition, see chapter 3.1.1. Each acquired frame is sent out via the communication protocol using 0 as identifier. Please see chapter 1.4.3 for a detailed description of the frame data format. In loop mode, the sensor uses double-buffering for maximum throughput.

As the bandwidth is often limited by the communication interface, RLE compression can be enabled to reduce the amount of data that is transferred. Please see Appendix C on page 39 for details about the compression algorithm.

If the host cannot process the incoming frames at the maximum sample rate, an additional delay can be added between the acquisitions of consecutive frames.

Please note that in case of RLE compressed frames, the data rate may vary depending on the frame content!

Command ID: 21h

Command Parameters:

Byte	Symbol	Data Type	Description
0	FLAGS	bit vector	**Acquisition control flags** D7…D1 reserved D0 1: Enable RLE Compression of frame data 0: Disable RLE Compression
1..2	DELAY	integer	Delay between the acquisitions of two consecutive frames in milliseconds. Set to 0 for maximum sample rate.

Returned Parameters:

No parameters are returned

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

E_ACCESS_DENIED: Data acquisition hardware is currently busy

E_INSUFFICIENT_RESOURCES: Out of memory

3.1.3 Stop Periodic Frame Acquisition (22h)

Stop any previously initiated periodic frame acquisition.

Command ID: 22h

Command Parameters:

No parameters required

Returned Parameters:

No parameters are returned

Possible Status Codes:

E_SUCCESS: Command succeeded.
E_NO_PARAM_EXPECTED: The command does not accept any parameter, but at least one was given.
E_INSUFFICIENT_RESOURCES: Out of memory

3.1.4 Tare Sensor Matrix (23h)

This command controls the tara of the sensor matrix. When taring the sensor matrix, the WTS will acquire a number of consecutive frames to determine the largest value for each sensor cell of the matrix. This value is then stored to the internal taring memory and used as new taring value. If the frame acquisition loop is currently active, it will be paused during taring.
The taring can be reverted by either resetting the taring memory to the currently set threshold value (see description of the parameters below) or by setting a new threshold value using the Set Threshold (34h) Command (see Chapter 3.2.5).

Command ID: 23h

Command Parameters:

Byte	Symbol	Data Type	Description
0	OPERATION	enum	**Determine the operation to be performed.** 0 un-tare the sensor matrix using the currently set threshold value 1 tare the sensor matrix 2..255: reserved

Returned Parameters:

No parameters are returned

Possible Status Codes:

E_SUCCESS: Command succeeded.
E_NO_PARAM_EXPECTED: The command does not accept any parameter, but at least one was given.
E_INSUFFICIENT_RESOURCES: Out of memory

3.2 Matrix Management

3.2.1 Get Matrix Information (30h)

Get specific properties of the sensor matrix realization, e.g. resolution, cell size etc. The returned data can be used to post-process the sensor's frame data and for visualization.

Command ID: 30h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0..1	RES_X	integer	Horizontal matrix resolution. For non-rectangular matrices, this is the horizontal resolution of the surrounding rectangle measured in sensor cells.
2..3	RES_Y	integer	Vertical matrix resolution. For non-rectangular matrices, this is the vertical resolution of the surrounding rectangle measured in sensor cells.
4..5	CELL_WIDTH	integer	Width of one sensor cell in 1/100 millimeters.
6..7	CELL_HEIGHT	integer	Height of one sensor cell in 1/100 millimeters.
8..9	FULLSCALE	integer	Full scale value of the output signal.

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_NO_PARAM_EXPECTED: The command does not accept any parameter but at least one was given.

E_INSUFSFICIENT_RESOURCES: Out of memory

3.2.2 Set Acquisition Mask Window (31h)

A mask can be applied to the matrix in order to increase the speed of the frame data acquisition or simply to disable uninteresting regions. Those areas will then be skipped when acquiring data, thus resulting in zero-values in the output frame data. When enabling RLE compression, this can produce a significant overall speed increase of the frame acquisition. The command accepts two X/Y coordinates as parameters to span the mask window. Sensor cell 1 has the coordinates 1/1, see Figure 5 for an example that defines an acquisition mask of 3 x 4 sensor cells on an 8 x 6 sensor matrix. When acquisition is initiated, only the sensor cells inside the mask window are sampled while the values of the sensor cells outside remain zero regardless of their real load.

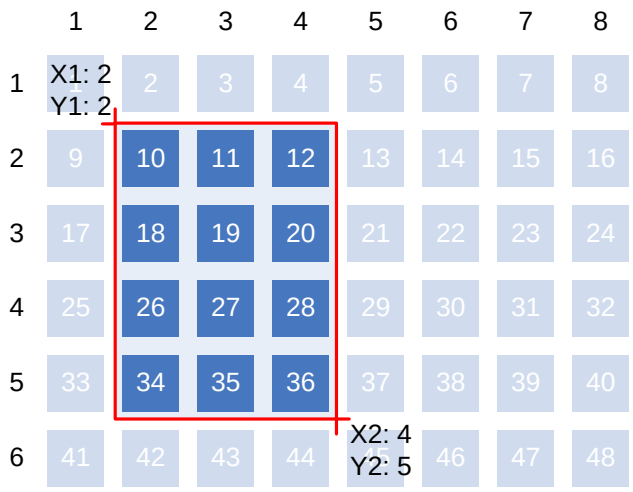


Figure 5: Windowed mask definition

On non-rectangular matrices, the coordinates are aligned with the bounding rectangle.

If your application requires a more detailed control over the acquisition mask, you might use the Set Advanced Acquisition Mask (32h) command in chapter 3.2.3.

The acquisition mask can only be changed while the data acquisition is idle.

Weiss Robotics GmbH & Co. KG
In der Gerste 2
D-71636 Ludwigsburg, Germany

Phone: +49 (0)7141 642062-6
Fax: +49 (0)7141 642062-9
email: office@weiss-robotics.com

Command ID: 31h

Command Parameters:

Byte	Symbol	Data Type	Description
0	X1	integer	X-Position of the upper-left corner (1..RES_X)
1	Y1	integer	Y-Position of the upper-left corner (1..RES_Y)
2	X2	integer	X-Position of the bottom-right corner (1..RES_X)
3	Y2	integer	Y-Position of the bottom-right corner (1..RES_Y)

Returned Parameters:

No parameters are returned

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

E_RANGE_ERROR: One or more values are out of range.

E_INVALID_PARAMETER: One or more dependent parameters mismatch (e.g. X1 > X2).

E_ACCESS_DENIED: Data acquisition is currently busy

E_INSUFFICIENT_RESOURCES: Out of memory

Weiss Robotics GmbH & Co. KG
In der Gerste 2
D-71636 Ludwigsburg, Germany

Phone: +49 (0)7141 642062-6
Fax: +49 (0)7141 642062-9
email: office@weiss-robotics.com

3.2.3 Set Advanced Acquisition Mask (32h)

As described in chapter 3.1.3, a mask can be applied to the matrix to control which sensor cells should be sampled and which not. Setting the mask bit of a sensor cell to 0 disables sampling of this cell. This command gives access to the discrete mask bits of the matrix.
Figure 6 shows a cross-style mask applied to a 7 x 6 sensor matrix (dark blue colored cells) that might be used to monitor the movement of a quickly oscillating object on the matrix surface.

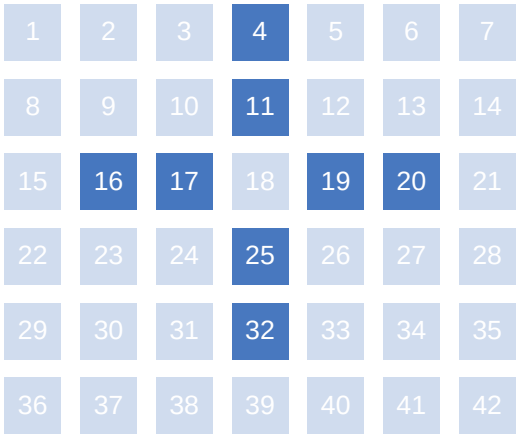


Figure 6: Advanced acquisition mask control

The mask bits are encoded in a series of consecutive bytes (mask sequence). Each byte holds 8 mask bits while the least significant bit (LSB) contains the mask bit of the cell with the lowest number. The bits are continuously assigned to the sensor cells from left to right. Spare bits will be set to 0. The length of the mask can be determined by the following formula:

The mask given in Figure 6 is represented by the following mask sequence:

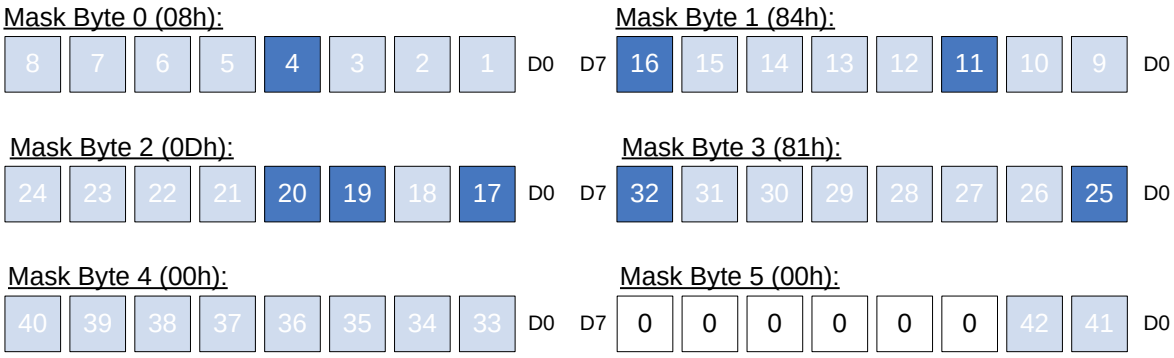


Figure 7: Mask sequence

Setting the mask of non-existent sensor cells (e.g. on non-rectangular sensor matrices) has no effect.

If your application requires a single acquisition window, you might use the Set Acquisition Mask Window (31h) Command (chapter 3.2.2) instead.

The acquisition mask can only be changed while the data acquisition is idle.

Command ID: 32h

Command Parameters:

Byte	Symbol	Data Type	Description
0	MASK	bit vector	**Mask Definition** D7 Mask of Cell 8 D6 Mask of Cell 7 D5 Mask of Cell 6 D4 Mask of Cell 5 D3 Mask of Cell 4 D2 Mask of Cell 3 D1 Mask of Cell 2 D0 Mask of Cell 1

1	MASK	bit vector	**Mask Definition** D7 Mask of Cell 16 D6 Mask of Cell 15 D5 Mask of Cell 14 D4 Mask of Cell 13 D3 Mask of Cell 12 D2 Mask of Cell 11 D1 Mask of Cell 10 D0 Mask of Cell 9
…			
n/8	MASK	bit vector	**Mask Definition** D7 Mask of Cell n D6 Mask of Cell n-1 D5 Mask of Cell n-2 D4 Mask of Cell n-3 D3 Mask of Cell n-4 D2 Mask of Cell n-5 D1 Mask of Cell n-6 D0 Mask of Cell n-7

Returned Parameters:

No parameters are returned

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

E_ACCESS_DENIED: Data acquisition is currently busy

E_INSUFFICIENT_RESOURCES: Out of memory

3.2.4 Get Acquisition Mask (33h)

Get the currently set acquisition mask. The acquisition mask is bit-wise encoded in the returned byte array as described in chapter 3.2.3. If the mask bit of a resp. sensor cell is set to '1', the sensor cell is sampled during data acquisition. If '0', the sensor cell is skipped on data acquisition and its value is set to 0 regardless of the real sensor signal. This is either the case on cells that have been masked out using one of the mask commands above or, on non-rectangular masks, if the cell is physically not present. If the sensor matrix' cell count is not a multiple of 8, the spare bits of the last mask sequence byte are set to 0.

Command ID: 33h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0	MASK	bit vector	**Mask Definition** D7 Mask of Cell 8 D6 Mask of Cell 7 D5 Mask of Cell 6 D4 Mask of Cell 5 D3 Mask of Cell 4 D2 Mask of Cell 3 D1 Mask of Cell 2 D0 Mask of Cell 1
1	MASK	bit vector	**Mask Definition** D7 Mask of Cell 16 D6 Mask of Cell 15 D5 Mask of Cell 14 D4 Mask of Cell 13 D3 Mask of Cell 12 D2 Mask of Cell 11 D1 Mask of Cell 10 D0 Mask of Cell 9
…			

n/8	MASK	bit vector	**Mask Definition** D7 Mask of Cell n D6 Mask of Cell n-1 D5 Mask of Cell n-2 D4 Mask of Cell n-3 D3 Mask of Cell n-4 D2 Mask of Cell n-5 D1 Mask of Cell n-6 D0 Mask of Cell n-7

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_NO_PARAM_EXPECTED: The command does not accept any parameter but at least one was given.

E_INSUFFICIENT_RESOURCES: Out of memory

3.2.5 Set Threshold (34h)

Set the threshold for the sensor matrix output. It defines at which output level a discrete sensing cell has a valid output signal. You can increase this value if you experience ghost contact e.g. due to mechanical stressing of the sensor itself or to manually tare the matrix. During frame acquisition, the calculation of the output signal is done using the following conditional equation:

output_signal := (matrix_output > threshold) ? matrix_output – threshold : 0

The threshold set with this command is only active until the next power cycle.

If the threshold value is set too low, the ground noise of the sensor matrix will come up. This effectively disables the run-length compression and therefore significantly reduces the throughput of your device.

The threshold value set with this command is only active until the next power cycle. To change the power-up default value, use the WTS Configuration Shell Interface.

Command ID: 34h

Command Parameters:

Byte	Symbol	Data Type	Description
0..1	THRESHOLD	integer	Threshold value to be set. Range: 0 to FULLSCALE. (See chapter 3.2 on how to determine the matrix full scale value).

Returned Parameters:

No parameters are returned

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

E_RANGE_ERROR: One or more values are out of range.

E_INSUFFICIENT_RESOURCES: Out of memory

3.2.6 Get Threshold (35h)

Get the currently set threshold value.

Command ID: 35h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0..1	THRESHOLD	integer	Currently set threshold value.

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_NO_PARAM_EXPECTED: The command does not accept any parameter, but at least one was given.

E_INSUFFICIENT_RESOURCES: Out of memory

3.2.7 Set Front-End Gain (36h)

Adjust the pressure sensitivity of a matrix by setting the gain of the Analog Front-End. The gain can be set using an integer value in the range of 0 to 255, where 0 is the most insensitive (lowest gain) and 255 is the most sensitive (highest gain) setting.

The gain value set with this command is only valid until the next power cycle. To change the power-up default value, use the WTS Configuration Shell Interface.

Command ID: 36h

Command Parameters:

Byte	Symbol	Data Type	Description
0	GAIN	integer	New gain value. Range: 0 to 255.

Returned Parameters:

No parameters are returned

Possible Status Codes:

E_SUCCESS: Command succeeded.
E_CMD_FORMAT_ERROR: Command length mismatch.
E_INSUFFICIENT_RESOURCES: Out of memory

3.2.8 Get Front-End Gain (37h)

Get the currently set analog front-end gain value. The gain is as an integer value ranging from 0 to 255, where 0 is the most insensitive (lowest gain) and 255 is the most sensitive (highest gain) setting.

Command ID: 37h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0	GAIN	integer	Currently set Analog Front-End gain value. Range: 0 to 255.

Possible Status Codes:

E_SUCCESS: Command succeeded.
E_NO_PARAM_EXPECTED: The command does not accept any parameter but at least one was given.
E_INSUFFICIENT_RESOURCES: Out of memory

3.2.9 Get Sensor Type (38h)

Return a string containing the sensor type.

Command ID: 38h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0..n	TYPE	string	Sensor type text string, e.g. “WTS 0406-38”

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_NO_PARAM_EXPECTED: A parameter was given, but not expected.

E_INSUFFICIENT_RESOURCES: Out of memory.

3.3 System Configuration and State

3.3.1 Read Device Temperature (46h)

Read the current temperature of the device. This value can be used to temperature-compensate the matrix output from within a user application or simply to monitor the operating temperature. The temperature accuracy of the built-in semiconductor sensor is typically ± 1.5°C.

Command ID: 46h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0..1	TEMP	integer	Temperature in 0.1°C-steps, e.g. FFF5h = -1.0 °C FFFFh = -0.1 °C 0000h = 0.0 °C 000Ah = 1.0 °C 00C8h = 20.0 °C 01F4h = 50.0 °C

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_NO_PARAM_EXPECTED: The command does not accept any parameter but at least one was given.

E_INSUFFICIENT_RESOURCES: Out of memory

3.3.2 Get System Information (50h)

Get information about the connected device. All products manufactured by Weiss Robotics supporting a binary interface provide this command.

Command ID: 50h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0	TYPE	enum	System Type: 0: unknown 4: WTS Tactile Sensor Module
1	HW_REV	integer	Hardware Revision
2..3	FW_VERSION	integer	Firmware Version: D15...12 major version D11...8 minor version 1 D7..4 minor version 2 D3..0 0 for release version, 1..15 for release candidate versions
4..7	SN	integer	Serial Number of the device

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_NO_PARAM_EXPECTED: The command does not accept any parameter but at least one was given.

E_INSUFFICIENT_RESOURCES: Out of memory

3.3.3 Set Device Tag (51h)

Set the device tag. This tag is a generic text string that can be set to any application-specific value, e.g. the location of the device or any additional process information that is used in conjunction with it. The maximum length of the device tag is 64 characters. The text string must not contain any control characters (i.e. ASCII codes < 32). Any terminating NUL characters are automatically stripped from the string.

Command ID: 51h

Command Parameters:

Byte	Symbol	Data Type	Description
0..n	TAG	string	Device tag text string. Maximum length is 64 characters.

Returned Parameters:

No parameters are returned

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_OVERRUN: Tag value is too long.

E_INVALID_PARAMETER: Tag contains illegal characters.

E_INSUFFICIENT_RESOURCES: Out of memory.

3.3.4 Get Device Tag (52h)

Return the device tag. If no device tag is set, the function returns an E_NOT_AVAILABLE error.

Command ID: 52h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0..n	TAG	string	Device tag text string.

Possible Status Codes:

E_SUCCESS: Command succeeded.

E_NO_PARAM_EXPECTED: A parameter was given, but not expected.

E_NOT_AVAILABLE: No device tag present.

E_INSUFFICIENT_RESOURCES: Out of memory.

3.4 Connection Management

3.4.1 Loop (06h)

Loop-back command, which returns the received parameters. This command is intended to test the communication interface.

Command ID: 06h

Command Parameters:

Byte	Symbol	Data Type	Description
0	LOOPDATA	integer	**Payload data to be looped** *A maximum of 256 bytes of payload data can be looped.*

Returned Parameters:

The received LOOPDATA is identically returned within the command acknowledge (note that the two bytes error code is automatically added to the beginning of the message as described in chapter 1.4.2).

Possible Status Codes:

E_CMD_FORMAT_ERROR: Command length mismatch (more than 256 bytes of payload).
E_INVALID_PARAMETER: Parameter value undefined.
E_CMD_PENDING: No error, command is pending.

4 Appendix A: Status Codes

All commands are acknowledged with a status code. Table 1 lists the valid codes and describes their meaning and reason.

Error code	Symbol name	Description
0	E_SUCCESS	No error occurred, operation was successful
1	E_NOT_AVAILABLE	Function or data is not available
2	E_NO_SENSOR	No measurement converter is connected
3	E_NOT_INITIALIZED	Device was not initialized
4	E_ALREADY_RUNNING	The data acquisition is already running
5	E_FEATURE_NOT_SUPPORTED	The requested feature is currently not available
6	E_INCONSISTENT_DATA	One or more parameters are inconsistent
7	E_TIMEOUT	Timeout error
8	E_READ_ERROR	Error while reading data
9	E_WRITE_ERROR	Error while writing data
10	E_INSUFFICIENT_RESOURCES	No more memory available
11	E_CHECKSUM_ERROR	Checksum error
12	E_NO_PARAM_EXPECTED	A Parameter was given but none expected
13	E_NOT_ENOUGH_PARAMS	Not enough parameters to execute the command
14	E_CMD_UNKNOWN	Unknown command
15	E_CMD_FORMAT_ERROR	Command format error
16	E_ACCESS_DENIED	Access denied
17	E_ALREADY_OPEN	Interface is already open
18	E_CMD_FAILED	Error while executing a command
19	E_CMD_ABORTED	Command execution was aborted by the user

Weiss Robotics GmbH & Co. KG
In der Gerste 2
D-71636 Ludwigsburg, Germany

Phone: +49 (0)7141 642062-6
Fax: +49 (0)7141 642062-9
email: office@weiss-robotics.com

20	E_INVALID_HANDLE	Invalid handle
21	E_NOT_FOUND	Device or file not found
22	E_NOT_OPEN	Device or file not open
23	E_IO_ERROR	Input/Output error
24	E_INVALID_PARAMETER	Wrong parameter
25	E_INDEX_OUT_OF_BOUNDS	Index out of bounds
26	E_CMD_PENDING	No error but the command was not completed yet. Another return message will follow including an error code as soon as the command has been completed.
27	E_OVERRUN	Data overrun
28	E_RANGE_ERROR	Range error
29	E_AXIS_BLOCKED	Axis blocked
30	E_FILE_EXISTS	File already exists

Table 1: Possible Status Codes

5 Appendix B: Sample code to calculate the checksum

The following code demonstrates how to calculate the CRC checksum for communicating with the WTS family of tactile sensor modules (written in ANSI C).

```
#include <stdio.h>
#include <stdlib.h>

typedef struct
{
   unsigned short length;     //!< Length of the message's payload in bytes
                              //   (0, if the message has no payload)
   unsigned char id;          //!< ID of the message
   unsigned char *data;       //!< Pointer to the message's payload
} TMESSAGE;    //!< command message format

//! Status codes
typedef enum
{
    E_SUCCESS = 0,              //!< No error
    E_NOT_AVAILABLE,            //!< Device, service or data is not available
    E_NO_SENSOR,                //!< No sensor connected
    E_NOT_INITIALIZED,          //!< The device is not initialized
    E_ALREADY_RUNNING,          //!< Service is already running
    E_FEATURE_NOT_SUPPORTED,    //!< The asked feature is not supported
    E_INCONSISTENT_DATA,        //!< One or more dependent parameters mismatch
    E_TIMEOUT,                  //!< Timeout error
    E_READ_ERROR,               //!< Error while reading from a device
    E_WRITE_ERROR,              //!< Error while writing to a device
    E_INSUFFICIENT_RESOURCES,   //!< No memory available
    E_CHECKSUM_ERROR,           //!< Checksum error
    E_NO_PARAM_EXPECTED,        //!< No parameters expected
    E_NOT_ENOUGH_PARAMS,        //!< Not enough parameters
    E_CMD_UNKNOWN,              //!< Unknown command
    E_CMD_FORMAT_ERROR,         //!< Command format error
    E_ACCESS_DENIED,            //!< Access denied
    E_ALREADY_OPEN,             //!< The interface is already open
    E_CMD_FAILED,               //!< Command failed
    E_CMD_ABORTED,              //!< Command aborted
    E_INVALID_HANDLE,           //!< invalid handle
    E_NOT_FOUND,                //!< device not found
    E_NOT_OPEN,                 //!< device not open
    E_IO_ERROR,                 //!< I/O error
    E_INVALID_PARAMETER,        //!< invalid parameter
    E_INDEX_OUT_OF_BOUNDS,      //!< index out of bounds
    E_CMD_PENDING,              //!< Command execution needs more time
    E_OVERRUN,                  //!< Data overrun
    E_RANGE_ERROR,              //!< Range error
    E_AXIS_BLOCKED,             //!< Axis is blocked
    E_FILE_EXISTS               //!< File already exists
} TStat;

#define SER_MSG_NUM_HEADER_BYTES   3     //!< number of header bytes
#define SER_MSG_HEADER_BYTE     0xAA     //!< header byte value

const unsigned short CRC_TABLE[256] = {
   0000h, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
   0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
```

```
   0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
   0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
   0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
   0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
   0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
   0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
   0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
   0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
   0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
   0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
   0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
   0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
   0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
   0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
   0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
   0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
   0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
   0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
   0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
   0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
   0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
   0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
   0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
   0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
   0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
   0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
   0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
   0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
   0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
   0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};


/**********************************************************************/
/*!
 Calculates the CRC checksum of an array by using a table.
 The start value for calculating the CRC should be set to 0xFFFF.

 @param *data  points to the byte array from which checksum should
        be calculated
 @param size   size of the byte array
 @param crc    value calculated over another array and start value
        of the crc16 calculation

 @return CRC16 checksum
*/
/**********************************************************************/

static unsigned short checksum_update_crc16( unsigned char *data,
   unsigned int size, unsigned short crc )
{
   unsigned long c;
   /* process each byte prior to checksum field */
   for ( c=0; c < size; c++ )
   {
      crc = CRC_TABLE[ ( crc ^ *( data ++ )) & 0x00FF ] ^ ( crc >> 8 );
   }
   return( crc );
}


/**********************************************************************/
```

```
/*!
 Builds a data packet from the given message.
 You have to free the returned buffer, if you do not use it anymore.

 @param *msg   Pointer to the source message
 @param *size    Returns the size of the created buffer

 @return buffer containing the bytewise packet data or NULL in case
      of an error.
*/
/*********************************************************************/

static unsigned char *msg_build( TMESSAGE * msg, unsigned int *size )
{
   unsigned char *buf;
   unsigned short chksum;
   unsigned int c, len;

   len = MSG_NUM_HEADER_BYTES + 3 + 2 + msg->length;

   buf = malloc( len );
   if ( !buf )
   {
      *size = 0;
      return( NULL );
   }

   // Assemble the message header:
   for ( c=0; c<MSG_NUM_HEADER_BYTES; c++ ) buf[c] = MSG_HEADER_BYTE;
   buf[ MSG_NUM_HEADER_BYTES ] = msg->id;  // Message ID
   buf[ MSG_NUM_HEADER_BYTES + 1 ] = lo( msg->length );  // Msg. length low byte
   buf[ MSG_NUM_HEADER_BYTES + 2 ] = hi( msg->length );  // Msg. length high byte

   // Copy payload to buffer:
   if ( msg->length ) memcpy( &buf[ MSG_NUM_HEADER_BYTES + 3 ], msg->data, msg->length );

   // Calculate the checksum over the header, include the preamble:
   chksum = checksum_update_crc16( buf, MSG_NUM_HEADER_BYTES + 3 + msg->length, 0xFFFF );

   // Add checksum to message:
   buf[ MSG_NUM_HEADER_BYTES + 3 + msg->length ] = lo( chksum );
   buf[ MSG_NUM_HEADER_BYTES + 4 + msg->length ] = hi( chksum );

   *size = len;
   return( buf );
}


/*********************************************************************/
/*!
 Send a message to an open file handle

 @param *file  Handle of an open file to which the message should be sent
 @param *msg    Pointer to the message that should be sent

 @return E_SUCCESS, if successful, otherwise error code
*/
/*********************************************************************/

TStat msg_send( FILE * file, TMESSAGE * msg )
{
   unsigned int c, size;
```

```
   // Convert message into byte sequence:
   unsigned char *buf = msg_build( msg, &size );
   if ( !buf ) return( E_INSUFFICIENT_RESOURCES );

   // Transmit buffer:
   c = fwrite( buf, size, 1, file );

   // Free allocated memory:
   free( buf );
   if ( c != 1 ) return( E_WRITE_ERROR );

   return( E_SUCCESS );
}
```

6 Appendix C: Data Compression

To achieve a higher throughput, the acquired pressure profile frames can be run-length encoded. To enable this compression, the corresponding flags have to be set when configuring the data acquisition loop (see chapter 3.1.2) or when acquiring a single frame (see chapter 3.1.1).

If compression is enabled, all zero values are RLE compressed during frame transmission, as in general a tactile sensor signal contains plenty of. Non-zero values are not compressed and will be sent out normally. If zero values are found, they are counted but not sent, until the next non-zero value is found or the frame transmission is completed. In this case, the number of zeros is multiplied by -1 and sent out before the non-zero value, i.e. one zero value results in a transmitted value of -1, two zero-values to -2 and so on. The example in Figure 8 example clarifies this procedure:

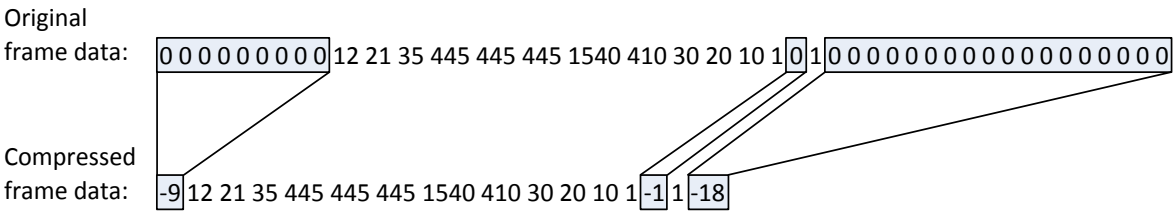


Figure 8: Frame data compression

Delphi Code example to decompress the frame data:

```
const NUM_SENSOR_CELLS    =  16*16;   // The number of texels in this example is 256

var input_data  : Record
                    length     :  Cardinal;
                    data       :  Array[0..NUM_SENSOR_CELLS+4] of Word;
                  end;
    actual_data : Record
                    timestamp  :  Cardinal;
                    flags      :  Byte;
```

```
                    data       :  Array[0..NUM_SENSOR_CELLS] of Word;
                  end;
    dataptr, temp, c, c1  :  Integer;

begin

  // extract the timestamp:
  actual_data.timestamp := cardinal( input_data.data[0] )
    or ( cardinal( input_data.data[1] ) shl 8 )
    or ( cardinal( input_data.data[2] ) shl 16 )
    or ( cardinal( input_data.data[3] ) shl 24 );
  flags := input_data.data[4];  // Tip: You can check the flags to see, if compression is
used
  c := 0;
  dataptr := 0;
  repeat

    // extract the data value:
    temp := integer( ( input_data.data[ 2 * c + 5 ] )
      or ( integer( input_data.data[ c * 2 + 6 ] ) shl 8 ));
    if temp > 32767 then dec( temp, 65536 );

    // Filling up the output structure with the actual data value:
    if temp >= 0 then
      begin
        actual_data.data[ dataptr ] := temp;
        inc( dataptr );
      end
    else begin
      // Expand to -temp zeroes:
      for c1:=1 to -temp do
        begin
          actual_data.data[ dataptr ] := 0;
          inc( dataptr );
        end;
    end;
    inc( c );
  until c >= ( input_data.length - 5 );
end;
```

Weiss Robotics GmbH & Co. KG
In der Gerste 2
D-71636 Ludwigsburg, Germany
e-mail: office@weiss-robotics.com

For further information and other products from Weiss Robotics, please visit our homepage at **http://www.weiss-robotics.com**.