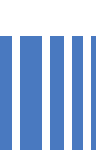


# **DSACON32**

## **Command Set Reference Manual**

Firmware Revision 272

October 2009



## Contents

1	Introduction .....	3
1.1	Sensor Cell Numbers .....	3
1.2	General Communication Protocol .....	4
1.3	Data Format of the Acquired Frames .....	5
2	Command Set Reference .....	7
2.1	Query the Controller Configuration .....	7
2.2	Query the Sensor Configuration .....	8
2.3	Query Matrix Configuration .....	9
2.4	Configuring the Data Acquisition .....	10
2.5	Query Controller Features .....	11
2.6	Read the Matrix Mask .....	12
2.7	Set the Dynamic Mask .....	14
2.8	Read the descriptor string .....	15
2.9	Loop .....	16
2.10	Query the Controller State .....	17
2.11	Set Properties Sample Rate .....	17
2.12	Set Properties Control Vector for a Matrix .....	19
2.13	Get Properties Control Vector of a Matrix .....	20
2.14	Adjust Matrix Sensitivity .....	21
2.15	Get Sensitivity Adjustment Info .....	22
2.16	Set Matrix Threshold .....	23
2.17	Get Matrix Threshold .....	24
3	Appendix A: Error Codes .....	25
4	Appendix B: Data Compression .....	26
4.1	The Legacy RLE Compression .....	27
4.2	Enhanced RLE Compression .....	29
5	Appendix C: Sample code for calculating the checksum .....	30

## 1 Introduction

The sensor controllers of the DSACON32 family are powerful data acquisition systems to measure contact pressure distributions in conjunction with tactile measurement converts from Weiss Robotics. For the visualization of the acquired data, the software "DSA Explorer Basic Edition" is available which runs under MS Windows 2000 and XP. It is included in the delivery of the sensor system. The Professional Edition of this software with additional features can be purchased directly from Weiss Robotics.

To use the sensor system in your own application, the sensor controller can also be controlled directly from a user application over standard interfaces. Please note, that some of the described interfaces may be optional depending on the sensor controller's model or the options you purchased.

 **The following assumptions are used 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.
- Floating point values are represented by 4 byte long single precision floating point values according to IEEE 754 using the following standard encoding:  
D31: sign  
D30...23: exponent  
D22...0: mantissa
- Any set of values is indexed starting with 0, i.e. an array with n elements has an index range of 0...n-1.

### 1.1 Sensor Cell Numbers

The sensor cells of the measurement converter are numbered starting at the left top corner and proceeding line-wise to the bottom right, as shown in Figure 1 (left) for an converter with a horizontal resolution of eight sensor cells.

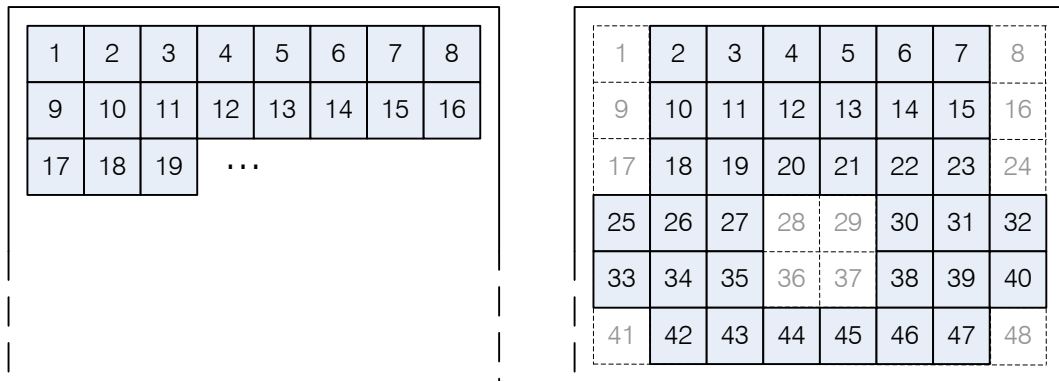


Figure 1: Sensor cell enumeration

For non-rectangular, 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 1 (right).

## 1.2 General Communication Protocol

Regardless of the interface used, the sensor controller communicates with its host via binary data packets. They consist of a preamble signaling the beginning of a new data packet. An identification byte describes the content of the packet. It is used to distinguish the several commands of the controller. The two byte size value determines the size of the packet’s payload in bytes. For simple signaling packets without any payload, this value is 0, and the following payload and the checksum are omitted. If the packet contains data, a two byte CRC16 checksum is added. The polynomial is  $x^{16}+x^{12}+x^5+1$  (CCITT-16). The start value for calculating the CRC is FFFFh. A sample code for calculating the checksum over a message is given in Appendix C: Sample code for calculating the checksum. To check a received message, you have to calculate the CRC again over the received data (without the preamble) including the received checksum. If the received data is correct, the calculated checksum is 0.

To send a command to the sensor controller, the packet ID has to be set according to the command ID. An ID of 00h identifies current measurement values. Such messages are exclusively sent by the sensor controller.

Byte	Symbol	Description
0...2	PREAMBLE	Signals the begin of a new message and has to be AAAAAAh
3	PACKET_ID	ID byte of the packet.
4...5	SIZE	Size of the packet's payload in bytes. May be 0 for signaling packets.
6...n	PAYLOAD	Payload data
n+1	CHECKSUM	CRC16 checksum of the whole data packet, excluding the preamble. The polynom is $x^{16}+x^{12}+x^5+1$ (CCITT-16).  <b>Please note:</b> The checksum is only transferred with packets containing a payload. On signaling packets, it is omitted.

Table 1: Communication packet structure

Example 1:      Signaling packet:

AAh AAh AAh 01h 00h 00h

Example 2:      Packet with two bytes payload (CDh, AAh), checksum is 83D9h:

AAh AAh AAh 01h 02h 00h CDh ABh D9h 83h

### 1.3 Data Format of the Acquired Frames

When the data acquisition is running, the sensor controller continuously sends data packets containing the acquired data. The format of the data packet was already described below. The ID of a data packet is always 00h. The payload contains a four byte timestamp which reflects the sensor controller time in milliseconds when the frame was acquired and can be used to reconstruct the chronological sequence of the data.

Byte	Symbol	Description
0...2	PREAMBLE	Signals the begin of a new message and has to be AAAAAAh
3	PACKET_ID	ID byte of the packet = 00h
4...5	SIZE	Size of the packet's payload in bytes.

6...9	PAYLOAD	TIMESTAMP	Sensor controller time in milliseconds when the passed frame was acquired
10		FLAGS	D7...1 reserved D1..0 00: No compression 01: Frame is compressed using legacy RLE compression 02: Frame is compressed using the enhanced RLE algorithm
11...n		FRAME_DATA	Raw or compressed frame data
n+1	CHECKSUM		CRC16 checksum of the whole data packet. The polynom is $x^{16}+x^{12}+x^5+1$ .

Table 2: Communication packet structure

Example: The table below shows an uncompressed data frame of a sensor matrix with 16 sensor cells, sampled at processor time 8197 ms:

Data	Description
AAh AAh AAh	Preamble
00h	Packet ID
25h 00h	Size of the payload is 37 bytes (4 bytes timestamp, 1 byte flags and 2x16 bytes for the frame data)
05h 20h 00h 00h	Time in milliseconds (here: 8197 ms) since starting the sensor controller when the frame was acquired
00h	Flags (no compression enabled)
00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 04h FFh 00h 00h 00h 00h 00h 00h 12h 1Ah 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h	Raw frame data, starting with the upper left sensor cell of the measurement converter and going on line by line.
CCh 48h	Checksum (calculated byte-wise over the packet ID, the payload size, the timestamp, flags and the frame data) is 48CCh

Table 3: Data frame example

## 2 Command Set Reference

In the following the available instructions of the DSACON32 sensor controllers are described and their syntax is clarified by simple examples.

### 2.1 Query the Controller Configuration

Description: Returns the current controller configuration.

Command ID: 01h

Parameters: none

Answer: If an error occurs, a data packet describing the error is sent. Otherwise, the following data packet is returned:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code
2...5	SERIAL_NO	Serial number of the sensor controller
6	HW_REVISION	Hardware revision of the controller (BCD coded) D7...4 upper revision number D3...0 lower revision number
7...8	SW_BUILD	Build number of the current DAQ firmware
9	STATE_FLAGS	Controller state flags D7 1: Sensor controller operable 0: No sensor connected (error condition) D6 1: Data acquisition is running D5...0 reserved
10	FEATURE_FLAGS	Controller feature flags D7 reserved D6 1: USB interface available D5 1: CAN-Bus interface available D4 1: Serial interface (RS232 ) available D3...0 reserved
11	SENSCON_TYPE	Sensor controller type. Currently, the following types are available: 0: DSACON16

		1: DSACON32-S 2: DSA100-256 3: DSACON32-M (OEM-Version) 4: DSACON32-H 5: DSACON32-C 6: DSA9205i 7: DSAMOD-5i
12...15	CAN_BAUDRATE	Actual bit rate of the CAN-Bus interface in kbps (kbits per second)
16...17	CAN_ID	Base ID of the CAN-Bus node

## 2.2 Query the Sensor Configuration

Description: Returns the configuration of the connected tactile transducer.

Command ID: 02h

Parameters: none

Answer: If an error occurs, a data packet describing the error is sent. Otherwise, the following data packet is returned:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code
2...3	NUMBER_MATRICES	Number of matrices connected to the distributed measurement transducer. The configuration of each matrix can be retrieved using the Query matrix configuration command, see chapter 2.3.
4...5	GENERATED_BY	Firmware revision used to set up the sensor configuration
6	HW_REVISION	Revision number of the measurement transducer
7...10	SERIAL_NO	Serial number of the distributed measurement transducer
11	FEATURE_FLAGS	Advanced features of the measurement transducer D7...1 reserved D0 true, if a descriptor string is available



## 2.3 Query Matrix Configuration

Description: Returns the configuration a sensor matrix connected to the distributed tactile transducer

Command ID: 0Bh

Parameters: The following payload format is used:

Byte	Symbol	Description
0	MATRIX_NUMBER	Number of the matrix to be queried

Answer: If an error occurs, a data packet describing the error is sent. Otherwise, the following data packet is returned:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code
2...5	TEXEL_WIDTH	Width of one texel in millimeters. Floating point value.
6...9	TEXEL_HEIGHT	Height of one texel in millimeters. Floating point value.
10...11	CELLS_X	Number of horizontal texels
12...13	CELLS_Y	Number of vertical texels
14...19	TRANSDUCER_ID	Unique 48-bit value that identifies the matrix
20...22	RESERVED	Reserved.
22	HW_REVISION	Revision number (xx.xx) of the measurement transducer D7...4 upper revision number D3...0 lower revision number
23...26	MATRIX_CENTER_X	X-coordinate of the matrix center (in mm). Floating point value.
27...30	MATRIX_CENTER_Y	Y-coordinate of the matrix center (in mm). Floating point value.
31...34	MATRIX_CENTER_Z	Z-coordinate of the matrix center (in mm). Floating point value.
35...38	THETA_X	Rotation of matrix about x-axis (in degrees). Floating point value.

39...42	THETA_Y	Rotation of matrix about y-axis (in degrees). Floating point value.
43...46	THETA_Z	Rotation of matrix about z-axis (in degrees). Floating point value.
47...50	FULLSCALE	Fullscale value of the output signal. For sensor controller platforms with 12-Bit converters, this is 4,095, for those with 10-Bit converters 1,024.
51	FEATURE_FLAGS	Advanced features of the tactile transducer D7...3 reserved D2 true, if the matrix supports masking of single sensor cells D1 true, if the sensitivity is adjustable by the user D0 true, if a descriptor string was previously set

## 2.4 Configuring the Data Acquisition

**Description:** Configures the data acquisition process of the firmware. The data acquisition can only be started, if the sensor controller is in configuration mode, i.e. a properly configured measurement converter is connected and was recognized during boot up.

If you want to increase the data throughput, you can enable a RLE compression for the frame data. Two algorithms are supported, please see chapter 4. To acquire a single frame, pass 0 as frame rate.

**Command ID:** 03h

**Parameters:** For configuration, the following payload format is used

Byte	Symbol	Description
0	FLAGS	Flags that do control the data acquisition D7 Acquisition control: 1: enabled 0: disabled D6...1 reserved D1..0 Data compression: 2: Data will be RLE compressed using the enhanced

		RLE compression (see chapter 4.2) 1: Data will be RLE compressed using legacy RLE compression (see chapter 4.1) 0: No data compression
1...2	FRAMERATE	Desired frame rate in frames per second (fps). If the value cannot be realized with the current configuration, it will be automatically overwritten with the nearest possible frame rate. If this parameter is set to 0, only a single frame is acquired.

Answer: If an error occurs, a data packet describing the error is sent. Otherwise, the following data packet is returned:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code

Example 1: Start data acquisition with eight frames per second:

Command: ID=03h, length=3, data: A0h 08h 00h

Answer: ID=03h, length=2, data: 00h 00h

Example 2: Stop the currently running data acquisition:

Command: ID=03h, length=3, data: 00h 00h 00h

Answer: ID=03h, length=2, data: 00h 00h

## 2.5 Query Controller Features

Description: Returns the available and currently enabled features. To enable or disable installed features, you can use the command shell interface of the DSACON32 controller.

Command ID: 10h

Parameters: none

Answer: If an error occurs, a data packet describing the error is sent. Otherwise, the following data packet is returned:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code
2...3	INST_FEAT	Installed features (for codes, see Table 4)
4...5	EN_FEAT	currently enabled features (for codes, see Table 4)

The list describes the available feature codes. If more than one feature is enabled, the values are logically or-ed:

Feature code	Symbol name	Description
0001h	FEAT_FILTER	Digital frame filter module
0002h	FEAT_PROPERTIES	Frame properties module
0004h	FEAT_GRASPING	Reactive grasping module

Table 4: DSACON32 controller feature codes

Example 1: Query features (Filter and properties module installed, but only filter module enabled):

Command: ID=10h, length=0

Answer: ID=10h, length=6, data: 00h 00h 03h 00h 01h 00h

## 2.6 Read the Matrix Mask

Description: Returns the sensor cell mask of the matrix with the passed index. The matrix mask information is stored as single bits, thus the message can be calculated by the following formula:

Payload length = 2 + RoundUp( <number of sensor cells> / 8 )

Each matrix has to mask types. One factory set and not editable "static mask", defining the physical shape of the sensor matrix (e.g. on non-rectangular matrices) and one "dynamic mask" which can be set by the user. This mask can be used to speed up sampling the matrix by un-masking areas of the sensor surface which are not interesting at the moment. The mask feature allows the

user to analyze the whole sensor surface and after identifying the region of interest to sample this with a much higher speed than it is possible for the whole matrix.

Command ID: 04h

Parameters:

Byte	Symbol	Description
0	MASK_TYPE	00h: Static mask 01h: Dynamic mask
1	INDEX	Determines the matrix index whose mask shall be returned.

Answer: If an error occurs, a data packet describing the error is returned. Otherwise, the following data packet is returned:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code
2...m	MASK_DATA	Mask data (for cell numbers, see chapter 1.1):  Byte 2: D7 Mask of sensor cell 8 D6 Mask of sensor cell 7 D5 Mask of sensor cell 6 D4 Mask of sensor cell 5 D3 Mask of sensor cell 4 D2 Mask of sensor cell 3 D1 Mask of sensor cell 2 D0 Mask of sensor cell 1  Byte 3: D7 Mask of sensor cell 10 D6 Mask of sensor cell 9 ...  Byte m: D7..3 '0' D2 Mask of sensor cell n D1 Mask of sensor cell n-1 D0 Mask of sensor cell n-2

Example 1: Reading the dynamic mask of matrix 2, which has 5 x 4 sensor cells which are all enabled:

Command: ID=04h, length=2, 01h 02h

Answer: ID=04h, length=5, data: 00h 00h FFh FFh 1Fh

Example 2: Reading the static mask of matrix 1, which has 5 x 4 sensor cells which are all enabled, except the first line:

Command: ID=04h, length=2, 00h 01h

Answer: ID=04h, length=5, data: 00h 00h C0h FFh 1Fh

## 2.7 Set the Dynamic Mask

Description: Sets the dynamic sensor cell mask of the matrix with the passed index. The mask information is stored as single bits, thus the message can be calculated by the following formula:

Payload length = 2 + RoundUp( <number of sensor chells> / 8 )

For details, see Chapter 2.5 – Query Controller Features.

Command ID: ABh

Parameters:

Byte	Symbol	Description
0	MASK_TYPE	Mask type has to be 01h (dynamic mask)
1	INDEX	Determines the matrix index whose mask should be set.
2...m	MASK	Mask data (for cell numbers, see chapter 1.1): Byte 2: D7 Mask of sensor cell 8 D6 Mask of sensor cell 7 D5 Mask of sensor cell 6 D4 Mask of sensor cell 5 D3 Mask of sensor cell 4 D2 Mask of sensor cell 3 D1 Mask of sensor cell 2 D0 Mask of sensor cell 1 Byte 3:

		D7      Mask of sensor cell 10 D6      Mask of sensor cell 9 ...  Byte m: D7..3   '0' D2      Mask of sensor cell n D1      Mask of sensor cell n-1 D0      Mask of sensor cell n-2
--	--	---

Answer:            The following data packet is returned:

Byte	Symbol	Description
0..1	ERROR_CODE	0000h, if successful, otherwise error code

Example 1:        Set the dynamic mask of matrix 2, which has 5 x 4 sensor cells to enable all cells:

Command: ID=ABh, length=5, 01h 02h FFh FFh 1Fh

Answer: ID=ABh, length=2, data: 00h 00h (no error)

## 2.8 Read the descriptor string

Description:      If a descriptor string was set via the configuration shell, it can be queried by sending an empty message with the ID 05h to the sensor controller. If the string is available, the controller returns a message containing the string. The length of the message is equal to the string length plus the two byte error code.

Command ID:    05h

Parameters:

Byte	Symbol	Description
0	DESCR_TYPE	00h: Sensor descriptor 01h: Matrix descriptor
1	INDEX	Only necessary, if DESCR_TYPE is 01h (matrix). Determines

		the matrix index whose descriptor shall be returned.
--	--	--

Answer: If an error occurs, a data packet describing the error is returned. Otherwise, the following data packet is returned:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code
2...n	STRING	Descriptor string

Example 1: Reading the descriptor string:  
Command: ID=05h, length=0  
Answer: ID=05h, length=20, data: 00h 00h "Descriptor string"

Example 2: Reading the descriptor string, but not available:  
Command: ID=05h, length=0  
Answer: ID=05h, length=2, data: 01h 00h (not available)

## 2.9 Loop

Description: For test purposes, a loop command can be send to the controller which will be returned immediately. The controller returns a message with the same ID.

Command ID: 06h

Parameters: none

Answer: Signaling packet with ID = 06h

Example: Loop test:  
Command: ID=06h, length=0  
Answer: ID=06h, length=0



## 2.10 Query the Controller State

Description: Returns the current controller state.

Command ID: 0Ah

Parameters: none

Answer: The following data packet is returned:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if controller state could be retrieved
2...3	STATE	Bit field describing the controller state. D15...7 reserved D6 Sensor emulation running D5 Calibration bus matching error D4 No Calibration bus available D3 Sensor not configured D2 Sensor configuration memory has wrong format (e.g. created by an older firmware revision) D1 No sensor connected D0 Controller not configured
4...7	TEMP	Current temperature of the controller (measured in °C). Floating point value.

Example: Query the controller state:

Command: ID=0Ah, length=0

Answer: ID=0Ah, length=6, data: 00h 00h 00h 00h <(float)32.1>

## 2.11 Set Properties Sample Rate

Description: Set sample rate at which the properties of the data frames should be send, turn off or request properties for a single data frame.

Command ID: 0Ch

Parameters:

Byte	Symbol	Description
0..1	SAMPLE_RATE	Properties sample rate (in data frames). 0000h: Turn off (no properties will be sent) FFFFh: Send properties for exactly one data frame All values n from 1 to 65534: Send properties for every n-th data frame

Answer: The following data packet is returned immediately:

Byte	Symbol	Description
0..1	ERROR_CODE	0000h, if successful, otherwise error code

Packets containing properties will be sent with the same ID=0Ch in the following format. Note that only those properties indicated by the BIT\_VECTOR will be included, so the length of those packets is variable. However, the order of the property values is obtained:

Byte	Symbol	Description
0..1	ERROR_CODE	0000h, if successful, otherwise error code. If this error code is different to 0000h (E_SUCCESS), only the matrix index is included (i.e. the message payload length is 3).
2	MATRIX_INDEX	Index of the matrix for which the properties were determined
3..6	TIMESTAMP	4 bytes timestamp of the matching data frame
7	BIT_VECTOR	Bit vector indicating which properties are sent with this packet: D7..6 reserved D5 Maximum force D4 Average force D3 Average force over contact area D2 Contact area size D1 Resulting force D0 Center of Gravity
8..11	CENTROID_X	Floating point value, indicating the x coordinate of the Center of Gravity ( <i>optional</i> )
12..15	CENTROID_Y	Floating point value, indicating the y coordinate of the Center of Gravity ( <i>optional</i> )

16..19	RESULTING_FORCE	Integer value indicating the resulting force ( <i>optional</i> )
20..23	CA_SIZE	Floating point value, indicating the size of the contact area ( <i>optional</i> )
24..27	AVG_FORCE_CA	Floating point value, indicating the average force over contact area ( <i>optional</i> )
28..31	AVG_FORCE	Floating point value, indicating the average force over the whole sensor area ( <i>optional</i> )
32..35	MAX_FORCE	Maximum force value of the matrix ( <i>optional</i> )

Example: Send properties for every 5<sup>th</sup> data frame:

Command: ID=0Ch, length=2, data: 05h 00h

Answer: ID=0Ch, length=2, data: 00h 00h

The properties will be sent in a new packet with the same ID.

## 2.12 Set Properties Control Vector for a Matrix

Description: Set the properties control vector (indicating which properties should be sent) for a certain matrix.

Command ID: 0Dh

Parameters:

Byte	Symbol	Description
0	INDEX	Determines the matrix index for which the properties should be set
1	BIT_VECTOR	Bit vector indicating which properties should be sent: D7...6 reserved D5 Maximum force D4 Average force D3 Average force over contact area D2 Contact area size D1 Resulting force D0 Center of Gravity



Answer:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code

Example: Send all properties for matrix 1

Command: ID=0Dh, length=2, data: 01h 1Fh

Answer: ID=0Dh, length=2, data: 00h 00h

## 2.13 Get Properties Control Vector of a Matrix

Description: Get the properties control vector (indicating which properties should be sent) of a certain matrix

Command ID: 0Eh

Parameters:

Byte	Symbol	Description
0	INDEX	Determines the matrix index for which the properties should be set

Answer:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code
2	BIT_VECTOR	Bit vector indicating which properties should be sent.  D7...6 reserved D5 Maximum force D4 Average force D3 Average force over contact area D2 Contact area size D1 Resulting force D0 Center of Gravity


Example: Get properties vector of matrix 3, which has all properties turned on

Command: ID=0Eh, length=1, data: 03h

Answer: ID=0Eh, length=3, data: 00 00 1Fh

## 2.14 Adjust Matrix Sensitivity

**Description:** Adjusts the pressure sensitivity of a matrix. This command tunes amplification factor of the matrix' integrated analog signal conditioning circuit. Not every matrix supports this feature, so check first using the "Get Sensitivity Adjustment Info" Command (see chapter 2.15). If the matrix doesn't support adjusting its sensitivity, the command returns E\_ACCESS\_DENIED. To make your settings non-volatile (i.e. they won't be lost after a power cycle), set bit FLAGS.7.

 **PLEASE NOTE: as the maximum write endurance of the configuration memory used is about 100.000 times, you should not make an excessive use of this option!**

Command ID: 0Fh

Parameters:

Byte	Symbol	Description
0	FLAGS	Bit vector to set the sensitivity adjustment options. D7      1: sensitivity is set non-volatile 0: sensitivity setting is volatile, i.e. changes are lost after a power cycle D6...2   reserved D1      1: adjust all matrices to the passed value, ignore the matrix index passed in byte 1 0: use byte 1 as matrix index D0      1: reset sensitivity to factory default 0: do not reset
1	INDEX	Matrix index. Ignored, if FLAGS.1 is set
2...5	SENSITIVITY	Matrix sensitivity as a floating point value. Range: 0 to 1.0, where 0 is the lowest sensitivity and 1.0 is the highest sensitivity to set. Ignored, if FLAGS.0 is set.



Answer:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code

## 2.15 Get Sensitivity Adjustment Info

Description: Returns information about the sensitivity adjustment options of the selected matrix.

Command ID: 12h

Parameters:

Byte	Symbol	Description
0	INDEX	Matrix index

Answer:


Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code
2	ADJ_FLAGS	Bit vector indicating the sensitivity adjustment options. D7...2 reserved D1 1: user can change the sensitivity 0: sensitivity cannot be changed by the user D0 1: the matrix supports sensitivity adjustment 0: sensitivity adjustment not available
3...6	CUR_SENS	Currently set sensitivity value. Floating point value. 0 is minimum sensitivity, 1.0 is maximum sensitivity.
7...10	FACT_SENS	Sensitivity value that is used, if a factory restore command is issued. Floating point value. 0 is minimum sensitivity, 1.0 is maximum sensitivity.


## 2.16 Set Matrix Threshold

**Description:** Sets the threshold for a single matrix or for the whole sensor. This threshold 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. The calculation of the output signal is done using the following conditional equation:

$$\text{output\_signal} := (\text{raw\_signal} > \text{threshold}) ? \text{raw\_signal} - \text{threshold} : 0$$

To reset the threshold to its initial value, use the factory settings flag (FLAGS.0). You can process either the threshold of a single matrix or for the whole sensor, depending on setting FLAGS.1. Per default, the set threshold is volatile and therefore lost on a power cycle. To store your settings to the non-volatile configuration memory so they are used on the next time, you power on the sensor system as well, set bit FLAGS.7.

 **PLEASE NOTE: as the maximum write endurance of the configuration memory is about 100.000 times!**

 **If you set the threshold value too low, the ground noise of the transducer will come up. This effectively disables the run-length compression and therefore significantly reduces the throughput of your interface connection.**

Command ID: 13h

Parameters:

Byte	Symbol	Description
0	FLAGS	<p>Bit vector to set the sensitivity adjustment options.</p> <p>D7      1: Threshold is set non-volatile                   0: Threshold setting is volatile, i.e. changes are lost after a power cycle</p> <p>D6...2   reserved</p> <p>D1      1: Set the threshold for the whole sensor. The matrix index will be ignored.                   0: use byte 1 as matrix index</p> <p>D0      1: reset threshold to factory default                   0: do not reset</p>



1	INDEX	Matrix index. Ignored, if FLAGS.1 is set
2...3	THRESHOLD	Threshold value as unsigned integer value. Range: 0 to 4095. Ignored, if FLAGS.0 is set.

Answer:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code

Example 1: Set threshold of matrix 2 to 150 for the current session (i.e. volatile):

Command: ID=13h, length=4, data: 00h 02h 96h 00h

Answer: ID=13h, length=2, data: 00h 00h

Example 2: Set threshold of all matrices to 150 for the current session (i.e. volatile):

Command: ID=13h, length=4, data: 01h 00h 96h 00h

Answer: ID=13h, length=2, data: 00h 00h

## 2.17 Get Matrix Threshold

Description: Returns the currently used threshold of the selected matrix.

Command ID: 14h

Parameters:

Byte	Symbol	Description
0	INDEX	Matrix index

Answer:

Byte	Symbol	Description
0...1	ERROR_CODE	0000h, if successful, otherwise error code
2...3	THRESHOLD	Current threshold value of the selected matrix



Example: Get threshold matrix 2

Command: ID=14h, length=1, data: 02h

Answer: ID=14h, length=4, data: 00h 00h 96h 00h

-> Matrix 2 uses a threshold of 96h = 150d.

### 3 Appendix A: Error Codes

All commands are acknowledged with an error code. Table 5 lists the valid error codes and describes their 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_CMD_NOT_ENOUGH_PARAMS	Not enough parameters for executing the command

13	E_CMD_UNKNOWN	Unknown command
14	E_CMD_FORMAT_ERROR	Command format error
15	E_ACCESS_DENIED	Access denied
16	E_ALREADY_OPEN	Interface is already open
17	E_CMD_FAILED	Error while executing a command
18	E_CMD_ABORTED	Command execution was aborted by the user
19	E_INVALID_HANDLE	Invalid handle
20	E_DEVICE_NOT_FOUND	Device not found
21	E_DEVICE_NOT_OPENED	Device not opened
22	E_IO_ERROR	Input/Output Error
23	E_INVALID_PARAMETER	Wrong parameter
24	E_INDEX_OUT_OF_BOUNDS	Index out of bounds
25	E_CMD_PENDING	No error, but the command was not completed, yet. Another return message will follow including an error code, if the function was completed.
26	E_OVERRUN	Data overrun
27	RANGE_ERROR	Range error

Table 5: Valid error codes

## 4 Appendix B: Data Compression

For achieving a higher throughput, the acquired data can be run-length encoded. To enable this compression, the corresponding flags have to be set when configuring the data acquisition. Two compression algorithms are currently supported: the Legacy RLE compression and the Enhanced RLE Compression. While the Legacy RLE Compression is compatible with DSACON32 sensor controllers running older versions of the firmware, the Enhanced RLE Compression achieves a slightly higher throughput, especially with large sensor matrices.



## 4.1 Legacy RLE Compression

Usually, the measurement values are 12 bits wide but are transferred in 16 bit words (see Figure 2).

High byte		Low byte	
0000	MMMM	MMMM	MMMM
Always "0"	Measurement value		

Figure 2: Data word without compression

The upper four bits are used by the compression algorithm. It counts the number of equal consecutive measurement values and submits it in the upper four bits (see Figure 3).

High byte		Low byte	
0000	MMMM	MMMM	MMMM
Number of equal consecutive values	Measurement value		

Figure 3: Data word with compression

A series of 16 measurement values "0 0 0 0 0 125 560 1201 1201 550 110 0 0 0 0 0" is therefore converted to 5x "0", 1x "125", 1x "560", 2x "1201", 1x "550", 1x "110", 5x "0" and transferred as "20480 4221 4656 9393 4646 4206 20480". The original 16 words are compressed to 7 words. This equals a compression factor of 56 %. Please not, that only the measurement values but neither the header nor the timestamp of the data packet are compressed. The compression is only used on data packets with an ID = 00h.

The following code is an example for a decompression of the received data by the host software. The received raw data and its size are passed in the structure "input data". It is the payl-

oad of the received data packet. The function extracts the timestamp and decompresses the measurement data and saves it into the structure “actual data”.

#### Code example using the Classic RLE compression:

```
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, cl : 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 for the compression used
    c := 0;
    dataptr := 0;
    repeat

        // extract the meareument value:
        temp := integer( ( input_data.data[ 2 * c + 5 ] )
            or ( integer( input_data.data[ c * 2 + 6 ] ) shl 8 )) and $0FFF;

        // Extract the number of consecutive words with the same value:
        count := input_data.data[ c * 2 + 6 ] shr 4;

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



## 4.2 Enhanced RLE Compression

In this mode, only zeroes are RLE compressed, since usually a tactile sensor signal contains a lot of them. They are merged as with the Legacy RLE compression, but were coded as a negative number. This overcomes the limitation of the Legacy RLE compression for a maximum of 16 merged elements a time and therefore further reduces the data. To clarify this approach, please see the following example:

Given the matrix data: 0 0 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

results in the following code: -9 1 2 3 4 5 3 2 1 1 1 -18

 **Enhanced RLE Compression is only available from Firmware Rev. 272 onwards.**

Code example using the Enhanced RLE compression:

```
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, cl : 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 for the compression used
    c := 0;
    dataptr := 0;
    repeat

        // extract the meareument 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 measurement value:
```



```

if temp >= 0 then
  begin
    actual_data.data[ dataptr ] := temp;
    inc( dataptr );
  end
else begin
  // Expand to -temp zeroes:
  for cl:=1 to -temp do
    begin
      actual_data.data[ dataptr ] := 0;
      inc( dataptr );
    end;
  end;
  inc( c );
until c >= ( input_data.length - 5 );
end;

```

## 5 Appendix C: Sample code for calculating the checksum

The following code demonstrates how to calculate the CRC checksum for communicating with the DSACON32.

```

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

#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_CCITT16[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, 0xcdbdc, 0xfdbbf, 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
};

extern DSA_STAT comm_write( unsigned char byte ); // Prototype for sending a byte

/*****
/*!
Calculates the CCITT compatible CRC16 checksum of an array
by using a table.
The crc16 polynom is x^16 + x^12 + x^5 + 1. The tart value for calculating the CRC is
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
*/
*****/

unsigned short checksum_update_crc16( unsigned char *data, unsigned short size, unsigned
short crc )
{
    unsigned long c;
    /* process each byte prior to checksum field */
    for ( c=0; c < size; c++ )
    {
        crc = CRC_TABLE_CCITT16[ ( crc ^ *( data ++ ) ) & 0x00FF ] ^ ( crc >> 8 );
    }
    return crc;
}

DSA_STAT comm_msg_send( TMESSAGE * msg )
{
    DSA_STAT result;
    unsigned int c;
    unsigned short chksum = 0;
    unsigned char chksum_byte;

```



```
chksum = 0xFFFF; // initialize checksum to 0xFFFF

// Message preamble:
for ( c=0; c<SER_MSG_NUM_HEADER_BYTES; c++ )
{
    result = comm_write( SER_MSG_HEADER_BYTE );
    if ( result != E_SUCCESS ) return( result );
}

// Message ID:
chksum = checksum_update_crc16( &(msg->id), 1, chksum );
result = comm_write( msg->id );
if ( result != E_SUCCESS ) return( result );

// Message Length:
chksum_byte = lo( msg->length );
chksum = checksum_update_crc16( &chksum_byte, 1, chksum );
result = comm_write( lo( msg->length ) );
if ( result != E_SUCCESS ) return( result );
chksum_byte = hi( msg->length );
chksum = checksum_update_crc16( &chksum_byte, 1, chksum );
result = comm_write( hi( msg->length ) );
if ( result != E_SUCCESS ) return( result );

// Message data (if available):
if ( msg->length > 0 )
{
    //calculate checksum from message data
    chksum = checksum_update_crc16( (unsigned char*) msg->data, msg->length, chksum );
    result = comm_write_bytes( msg->data, msg->length );
    if ( result != E_SUCCESS ) return( result );
    result = comm_write( lo( chksum ) );
    if ( result != E_SUCCESS ) return( result );
    result = comm_write( hi( chksum ) );
    if ( result != E_SUCCESS ) return( result );
}
comm_flush(); // Flush io buffers
return( E_SUCCESS );
}
```





## Weiss Robotics

Bluecherstrasse 17  
D-71636 Ludwigsburg, Germany  
e-mail: kontakt@weiss-robotics.de

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

---

© 2009 Weiss Robotics, all rights reserved.

All technical data mentioned in this data sheet can be changed to improve our products without prior notice. Used trademarks are the property of their respective trademark owners. Our products are not intended for use in life support systems or systems whose failure can lead to personal injury.