WSG Command Set

Reference Manual

Firmware Version 2.2.0

December 2011

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

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 WSG family of grippers can be controlled by different standard interfaces using a binary protocol. This manual gives a detailed explanation of the protocol used as well as over the WSG's command set. For getting started with the communication protocol, we recommend the free WSG Commander application running on Microsoft Windows.

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 numbers 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.

The following data types are used by the command set:

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

1.1 General Communication Protocol

Regardless of the interface used, the WSG communicates with its host using binary data packets. They consist of a preamble signaling the beginning of a new data packet. Table 1 illustrates the com-

mand format. An identification code describes the content of the packet. It is used as command ID 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 sample code for calculating the checksum over a message is given in Appendix C.**
If you decide not to use the CRC, e.g. on transmission-safe protocols as TCP/IP, you can disable the checksum evaluation using the WSG's Web Interface (Settings->Command Interface).

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 WSG Commander tool (see Chapter** Fehler! Verweisquelle konnte nicht gefunden werden.**) to the interactive assembly of valid data packets.**

Byte	Symbol	Description
0..2	PREAMBLE	Signals the begin of a new message and has to be AAAAAAh
3	COMMAND_ID	ID byte of the command.
4..5	SIZE	Size of the packet's payload in bytes. May 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. ⓘ **If checksum evaluation is disabled, these bytes are 0.**

Table 1: Communication packet structure

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

1.2 Command Acknowledge from the WSG

Every command is acknowledged by the WSG 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	ID of the command.
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	ERROR_CODE	Error code, see Chapter 0
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 C on how to check this checksum. Even if checksum evaluation is disabled, the WSG will always send a valid checksum with its response. **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: "Homing"-Command):

AAh AAh AAh 20h 02h 00h 00h 00h B3h FDh

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

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

Example 2: Acknowledging a successfully executed "Get Acceleration"-Command, returning a 4-byte floating point parameter (here: 150.0 mm/s², in hex: 00h 00h 16h 43h):

AAh AAh AAh 30h 06h 00h 00h 00h 00h 16h 43h DCh CBh

1.3 Asynchronous Commands

In case the command result is not immediately available, e.g. on movement or referencing commands, the WSG returns a notification that he did understand the received command and started its execution (command pending). However, the result will not be immediately available and therefore being sent in an additional packet when command execution is completed. The immediate response to such an asynchronous command will be a packet with E_CMD_PENDING as error code, followed by the additional packet returning the command's result:

Example: Acknowledging the reception of a GOTO-Command with E_CMD_PENDING (error code 001Ah):

AAh AAh AAh 21h 02h 00h 1Ah 00h 67h CBh

After the goal position was reached, the WSG sends the result with an additional packet (here E_SUCCESS, error code 0000h):

AAh AAh AAh 21h 02h 00h 00h 00h 28h 04h

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

2 WSG Commander

The tool WSG Commander is provided free of charge and allows an easy familiarization with the WSG's communication protocol and command set. It allows you to send basic commands to the WSG and contains a custom command editor to assemble own data packets. The data traffic to and from the WSG 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 WSG or can be downloaded at the WSG's Web Interface from the Support Area.

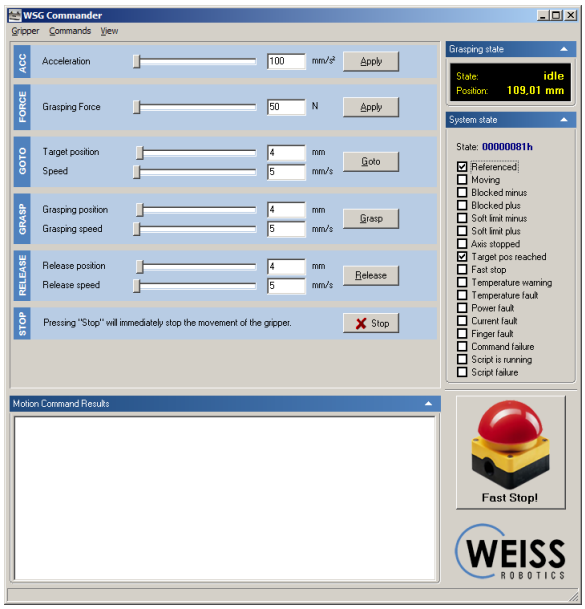


Figure 1: WSG Commander Main Window

Main Window

To connect to your WSG, select *Gripper|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, CAN-Bus via ESD-Cards as well as Ethernet TCP/IP and UDP/IP.

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 WSG. 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 gripper.

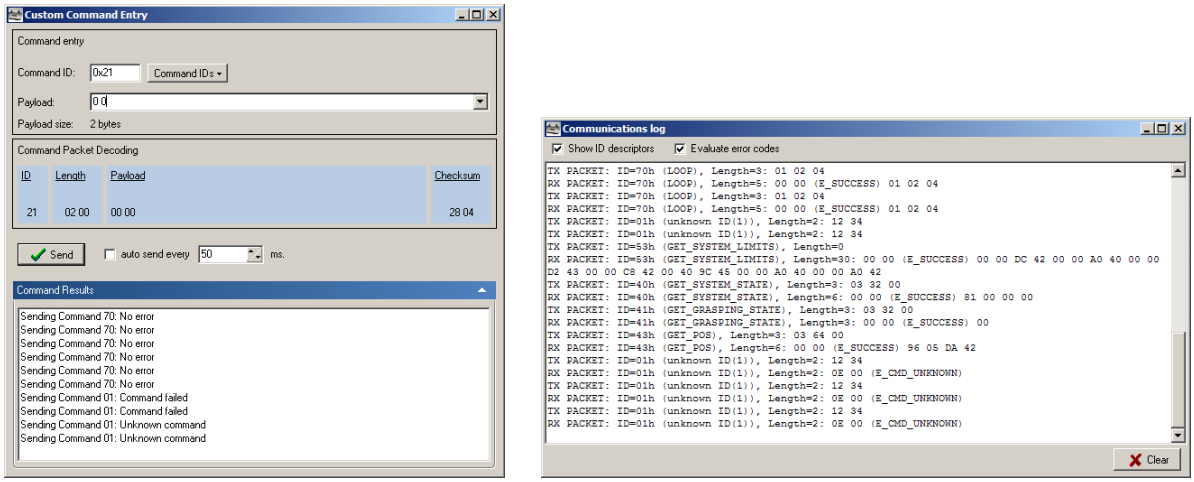


Figure 2: WSG Commander Custom Command Editor (left) and Communication Log (right)

Communication Log

To follow the communication between the WSG Commander and the gripper, 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 WSG Commander is intended as a tool to you to make the evaluation of the WSG’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 Connection Management

3.1.1 Loop (06h)

Loop-back command, which returns the received parameters. This command is intended to be used for testing 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 1.2).

Possible Error 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.

3.1.2 Disconnect Announcement (07h)

Announce the disconnection of the current interface. This command is only available with Ethernet TCP/IP connections and can be used to inform the device about a regular disconnection. Any movement that is executed when the disconnect announcement arrives is aborted immediately. If you send this command before closing the connection, the gripper does not enter FAST STOP on disconnect.

When issuing a Disconnect Announcement, the gripper will wait for disconnection. Commands arriving after a Disconnect Announcement will not be accepted anymore and return an E_ACCESS_DENIED error.

Command ID: 07h

Command Parameters:

No parameters.

Returned Parameters:

No parameters are returned.

Possible Error Codes:

E_SUCCESS: Command succeeded.
E_NO_PARAM_EXPECTED: The command does not accept any parameter, but at least one was given.
E_NOT_AVAILABLE: Command was used with a non-connection oriented interface

3.2 Motion Control

3.2.1 Homing (20h)

Execute a homing sequence to reference the gripper fingers. This command has to be executed prior to any other movement-related command. The direction of homing can be either explicitly specified or can be obtained from the gripper's configuration. During homing, the gripper moves its fingers into the specified direction until it reaches its mechanical end stop. The blocking position is used as new origin for all movement-related commands.

The best positioning performance will be achieved if homing is done into the direction you require the better positioning accuracy.

During homing soft limits are disabled!

Obstacles in the movement range of the fingers and collision with these during homing may result in a wrong reference point for the finger position!

Command ID: 20h

Command Parameters:

Byte	Symbol	Data Type	Description
0	DIRECTION	enum	**Homing direction** *0: use default value from system configuration (the default value can be changed via the Web Interface)* *1: Homing in positive movement direction* *2: Homing in negative movement direction*

Returned Parameters:

No parameters are returned

Possible Error Codes:

Immediate errors:

E_ACCESS_DENIED: Gripper is in FAST STOP state.

E_ALREADY_RUNNING: Gripper is currently moving. Issue a STOP command, first.

E_CMD_FORMAT_ERROR: Command length mismatch.

E_INVALID_PARAMETER: Parameter value undefined.

E_CMD_PENDING: No Error, command is pending.

Errors upon completion of the command:

E_SUCCESS: Command succeeded.

E_CMD_ABORTED: Homing sequence aborted.

E_AXIS_BLOCKED: Axis was blocked while moving away from the end stop.

E_TIMEOUT: Timeout while homing

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

3.2.2 Preposition Fingers (21h)

Move the gripper fingers to a defined opening width. This command is intended to preposition the gripper fingers prior to a grasp. For grasping a part, the Grasp Part (25h) command (see page 20) should be used. You can select between absolute movement, where the fingers are positioned to the given value and a relative movement, where the finger's opening width is changed relative to their current position.

This command is executed asynchronously. After reception of the command, the WSG returns a packet with an E_CMD_PENDING error, meaning it did understand and initiated execution of the command. After the goal position was reached, another message is returned, giving the result of the command. More details about asynchronous commands can be found in Chapter 1.3.

Speed and position values that are outside the gripper's physical limits are clamped to the highest/lowest available value. It is a good practice to get the gripper's limits (see Chapter 3.5.4) and check your movement parameters against it before issuing a movement-related command to ensure that the gripper behaves as intended.

For getting in-depth information about the current movement, you may use this command in conjunction with the Get Opening Width command, see Chapter 3.4.4.

Prepositioning always estimates the contact force by measuring the motor current (Force Approximation Mode), regardless of any Force Measurement Fingers that might have been installed.

To grasp or to release a part, please use the Grasp Part command (see page 20 for more details).

The gripper has to be homed and not in a FAST STOP state to start a movement!

Command ID: 21h

Command Parameters:

Byte	Symbol	Data Type	Description
0	FLAGS	bit vector	**D7...D2: unused, set to 0** **D1: Stop on Block** *1: Stop on block.* If a blocking condition in towards the movement direction of the fingers is detected, the motion command returns an E_AXIS_BLOCKED error and the motor is

			stopped, when a blocking condition is detected. *0: Clamp on block.* If a blocking condition in towards the movement direction of the fingers is detected, the motion command returns an E_AXIS_BLOCKED error. The motor is not turned off automatically, but clamps with the previously set force limit. **If the blocking condition is removed while clamping (e.g. the part between the fingers is being removed), the fingers will snap to the target position.** **D0: Movement Type** *1: relative movement* The passed width is treated as an offset to the current opening width. *0: absolute movement* The passed width is absolute to the closed fingers (0 mm).
1..4	WIDTH	float	Opening width in mm
5..8	SPEED	float	Traveling speed in mm/s

Returned Parameters:

No parameters are returned

Possible Error Codes:

Immediate errors:

E_ACCESS_DENIED: Gripper is in FAST STOP state.

E_NOT_INITIALIZED: Gripper is not referenced. Issue a Homing command, first.

E_ALREADY_RUNNING: Gripper is currently moving. Issue a STOP command, first.

E_RANGE_ERROR: Soft limits are enabled and the given position falls into these limits.

E_CMD_FORMAT_ERROR: Command length mismatch.

E_CMD_PENDING: No Error, command is pending.

Errors upon completion of the command:

E_SUCCESS: Command succeeded.

E_INSUFFICIENT_RESOURCES: Out of memory

E_AXIS_BLOCKED: Axis is blocked. This may indicate that a part was grasped.

E_RANGE_ERROR: A limit (either soft or hard limit) was reached during movement. Gripper is stopped.

E_TIMEOUT: Timeout while moving

E_CMD_ABORTED: The movement command was aborted, e.g. by a Stop command

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 Stop (22h)

Immediately stops any ongoing movement. The command sets the SF_AXIS_STOPPED flag. The AXIS STOPPED state does not need to be acknowledged; it is cleared automatically by the next movement command.

☞ **If you want to stop the gripper in case of an error, use the FAST STOP command instead.**

Command ID: 22h

Command Parameters:

No parameters expected

Returned Parameters:

No parameters are returned

Possible Error Codes:

E_SUCCESS: Command succeeded.
E_NO_PARAM_EXPECTED: A parameter was given, but not expected.
E_TIMEOUT: Timeout while stopping.

3.2.4 Issue FAST STOP (23h)

This function is similar to an “Emergency Stop”. It immediately stops any movement the fastest way and prevents further movement commands from being executed. The FAST STOP state can only be left by issuing a **FAST STOP Acknowledge message**. All movement-related commands are prohibited during FAST STOP and will produce an E_ACCESS_DENIED error.
The FAST STOP state is indicated in the **System Flags** and logged in the system’s log file, so this command should in general be used to react on certain error conditions.

To simply stop the current movement, you may want to use the STOP command instead (see Chapter 3.2.3).

Command ID: 23h

Command Parameters:

No parameters are required

Returned Parameters:

No parameters are returned

Possible Error Codes:

E_SUCCESS: Command succeeded.

3.2.5 Acknowledging a FAST STOP or Fault Condition (24h)

A previously issued **FAST STOP** or a severe error condition must be acknowledged using this command to bring the WSG back into normal operating mode.

Command ID: 24h

Command Parameters:

Byte	Symbol	Data Type	Description
0..2	ACK_KEY	string	Acknowledge key string, i.e. the letters “ack” (= 61h 63h 6Bh)

Returned Parameters:

No parameters are returned

Possible Error Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Acknowledge key is incorrect.

3.2.6 Grasp Part (25h)

Grasp a part by passing its nominal width and the speed at which the part should be grasped. When the command is issued, the gripper moves its fingers to the nominal part width and tries to clamp the expected part with the previously set grasping force. If the gripper can establish the desired grasping force within the defined clamping travel, a part is grasped. If the fingers fall through the clamping travel without establishing the grasping force, no part was found. The clamping travel can be set using the WSG's Web interface. The grasping state is updated with the result of this operation (either PART HOLDING or NO PART) as well as the grasping statistics (see Chapter 3.4.2). If no part was found, the command returns E_CMD_FAILED.

You may reduce the grasping speed with sensitive parts to limit the impact due to the mass of the gripper fingers and the internal mechanics.

The Grasping State reflects the current state of the process. You can read it using the Get Grasping State command (see Chapter 3.4.2).

Command ID: 25h

Command Parameters:

Byte	Symbol	Data Type	Description
0..3	WIDTH	float	Nominal width of the part to be grasped in mm.
4..7	SPEED	float	Grasping speed in mm/s

Returned Parameters:

No parameters are returned

Possible Error Codes:

Immediate errors:

E_ACCESS_DENIED: Gripper is in FAST STOP state.

E_ALREADY_RUNNING: Gripper is currently moving. Issue a STOP command, first.

E_CMD_FORMAT_ERROR: Command length mismatch.

E_RANGE_ERROR: WIDTH parameter violates the soft limits.

E_CMD_PENDING: No Error, command is pending.

Errors upon completion of the command:

E_SUCCESS: Command succeeded.

E_CMD_ABORTED: Grasping aborted.

E_CMD_FAILED: No part found.

E_TIMEOUT: Timeout while grasping.

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

3.2.7 Release Part (26h)

Release a previously grasped part.

Command ID: 26h

Command Parameters:

Byte	Symbol	Data Type	Description
0..3	OPENWIDTH	float	Opening width in mm to release the part safely.
4..7	SPEED	float	Opening speed in mm/s

Returned Parameters:

No parameters are returned

Possible Error Codes:

Immediate errors:

E_ACCESS_DENIED: Gripper is in FAST STOP state.

E_ALREADY_RUNNING: Gripper is currently moving. Issue a STOP command, first.

E_CMD_FORMAT_ERROR: Command length mismatch.

E_RANGE_ERROR: OPENWIDTH parameter violates the soft limits.

E_CMD_PENDING: No Error, command is pending.

Errors upon completion of the command:

E_SUCCESS: Command succeeded.

E_CMD_ABORTED: Releasing aborted.

E_TIMEOUT: Timeout while releasing.

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

3.3 Motion Configuration

3.3.1 Set Acceleration (30h)

Set the axis acceleration for consecutive movements, started with e.g. Grasp or Preposition Fingers commands.

On startup, a default value is used for acceleration. You can use the Web Interface to change this default value. The acceleration value that is set using the "Set Acceleration" command is only valid for the current session, i.e. if the WSG is restarted, this setting is lost.

Command ID: 30h

Command Parameters:

Byte	Symbol	Data Type	Description
0..3	ACC	float	Acceleration in mm/s². The value is clamped, if it is outside the device's capabilities.

Returned Parameters:

No parameters are returned

Possible Error Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Parameter length is incorrect.

3.3.2 Get Acceleration (31h)

Return the currently set axis acceleration.

Command ID: 31h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0..3	ACC	float	Acceleration in mm/s²

Possible Error Codes:

E_SUCCESS: Command succeeded.

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

3.3.3 Set Force Limit (32h)

Set the force limit for consecutive prepositioning and grasp commands. The force limit is the maximum grasping force that is applied on a mechanical contact.

On startup, a default value is used for the force limit. You can use the Web Interface to change this default value. The force value set by this command is only valid for the current session, i.e. if the WSG is restarted, this setting is lost.

Note: The force limit is defined as the sum of the nominal force times the number of fingers.

The force in prepositioning mode is always estimated using the motor current. Please keep in mind, that this might not as accurate as a true force measurement!

Command ID: 32h

Command Parameters:

Byte	Symbol	Data Type	Description
0..3	FORCE	float	**Force Limit in Newtons.** The value is clamped, if it is outside the device's capabilities. The given value is clamped if it is lower than the minimum grasping force and if exceeding the nominal force or if exceeding the overdrive force, depending on the Overdrive Mode flag set (see Overdrive Mode, Chapter 3.3.8).

Returned Parameters:

No parameters are returned

Possible Error Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Parameter length is incorrect.

3.3.4 Get Force Limit (33h)

Return the force limit that was previously set by the **Set Force Limit** command.

Note: The force limit is defined as the sum of the nominal force times the number of fingers.

Command ID: 33h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0..3	ACC	float	Force Limit in Newtons

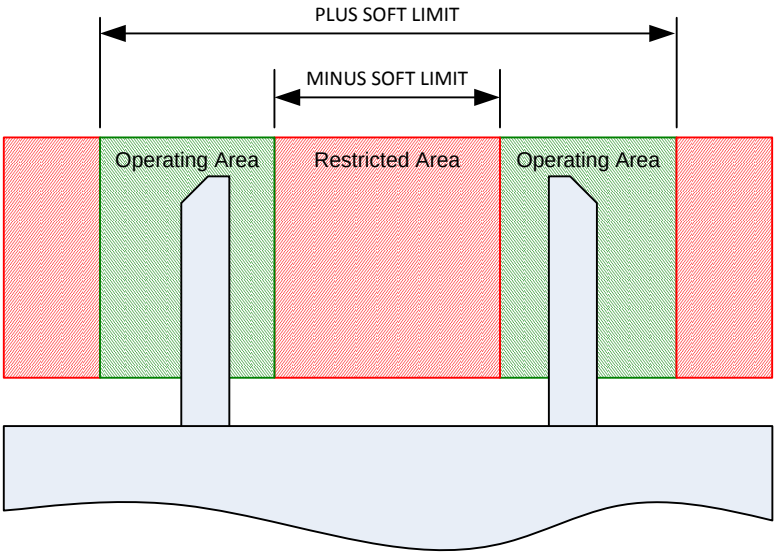
Possible Error Codes:

E_SUCCESS: Command succeeded.

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

3.3.5 Set Soft Limits (34h)

Set Soft Limits for both minus and plus direction. With soft limits, you can effectively prevent the fingers to move into a certain area. If soft limits are set, the gripper returns a range error for movement commands if the given finger position is outside these limits and ensures, that the fingers do not enter the restricted area.
If the fingers are moving into the restricted area, a FAST STOP is issued that has to be acknowledged prior to any further movement-related command being accepted (see Chapter 3.2.5).



The width of the fingers is not considered by the gripper. The opening width is always to the inner side of the base jaws.

If the gripper fingers are outside the allowed range after setting the soft limits, the resp. system flag is set and movement is only allowed in the direction out of the restricted area.

By using this command, you can only set soft limits for the current session (i.e. up to the next power cycle). If you want to set soft limits that are loaded per default on power-up, you can use the WSG's Web Interface.

Command ID: 34h

Command Parameters:

Byte	Symbol	Data Type	Description
0..3	LIMIT_MINUS	float	Soft limit opening width in negative movement direction (mm).
4..7	LIMIT_PLUS	float	Soft limit opening width in positive movement direction (mm).

Returned Parameters:

No parameters are returned

Possible Error Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

3.3.6 Get Soft Limits (35h)

Return the soft limits, if set. If no soft limits are currently set, the command will return an E_NOT_AVAILABLE error.

Command ID: 35h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0..3	LIMIT_MINUS	float	Soft limit position in negative movement direction (mm).
4..7	LIMIT_PLUS	float	Soft limit position in positive movement direction (mm).

Possible Error Codes:

E_SUCCESS: Command succeeded.

E_NOT_AVAILABLE: No soft limits have been set.

E_INSUFFICIENT_RESOURCES: Out of memory

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

3.3.7 Clear Soft Limits (36h)

Clear any previously set soft limits.

Command ID: 36h

Command Parameters:

No parameters required

Returned Parameters:

No parameters are returned

Possible Error Codes:

E_SUCCESS: Command succeeded.

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

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

3.3.8 Overdrive Mode (37h)

Enable or disable Force Overdrive Mode. Per default, the gripper only allows to set a grasping force that is not higher than the nominal value, which can be applied with a duty cycle of 100%. If you set overdrive mode, the grasping force can be increased up to the overdrive limit (see the Get System Limits command in Chapter 3.5.4).

Use the overdrive feature with care! If overdrive mode is enabled and a force is set higher than the nominal force value, the gripper's power dissipation will be increased. Depending on the duty cycle used, this may result in an excessive overheat and forces the gripper to turn off its power electronics. In some cases, excessive overload may also damage the device.

If overdrive mode is disabled and the current grasping force limit is beyond the gripper's nominal force limit, it is automatically reduced to the nominal force.

When Entering and Exiting Overdrive Mode, an resp. entry is created in the system log.

Command ID: 37h

Command Parameters:

Byte	Symbol	Data Type	Description
0	FLAGS	bit vector	**D7...D1: unused, set to 0** **D0: Enable Overdrive Mode** *1: Overdrive Mode enabled.* When setting the grasping force limit, the maximum allowed value is the overdrive force. *0: Overdrive Mode disabled.* When setting the grasping force limit, the maximum allowed value is the nominal force.

Returned Parameters:

No parameters are returned

Possible Error Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

3.3.9 Tare Force Sensor (38h)

Zeroes the connected Force Sensor used for the Force Control Loop.

This command is only allowed, if not in Force Control Mode (i.e. the grasping state must not be "holding" when issuing this command).

Command ID: 38h

Availability

This command is available from Firmware Version 1.1.0 onwards

Command Parameters:

No parameters required

Returned Parameters:

No parameters are returned

Possible Error Codes:

E_SUCCESS: Command succeeded.
E_NOT_AVAILABLE: No force sensor installed.
E_ACCESS_DENIED: Command is not allowed in Force Control Mode!
E_NO_PARAM_EXPECTED: The command does not accept any parameter, but at least one was given.

3.4 System State Commands

3.4.1 Get System State (40h)

Get the current system state. This command supports the automatic transmission of update packets in either fixed time intervals or if the system state changes. This gives you a precise control over the bus load of your system.
When sending this command with automatic updates disabled (FLAGS'0=0), one return packet containing the current system state is immediately returned.

If you select to send automatic update messages only in case the system state changed, the time interval between two packets still is maintained, even if the changing rate of the system state is higher than PERIOD_MS.

Command ID: 40h

Command Parameters:

Byte	Symbol	Data Type	Description
0	FLAGS	bit vector	D7...D2: unused, set to 0 D1: Change-sensitive Update: 1: Update on change only 0: Update always D0: Automatic Update: 1: auto update is enabled 0: auto update is disabled
1..2	PERIOD_MS	integer	Minimum period between two automatically sent packets in milliseconds.

Returned Parameters:

Byte	Symbol	Data Type	Description
0..3	SSTATE	bit vector	System state. See **Appendix B** for an explanation of the system state.

Possible Error Codes:

E_SUCCESS: Command succeeded.
E_CMD_FORMAT_ERROR: Command length mismatch.

3.4.2 Get Grasping State (41h)

Get the current Grasping State. The Grasping State reflects the current state of the grasping process and can be used to monitor it. The following states are possible and will be encoded into a single number:

- ***Idle (0)***
  The grasping process is in idle state and is waiting for a command.
- ***Grasping (1)***
  The fingers are currently closing to grasp a part. The part has not been grasped, yet
- ***No part found (2)***
  The fingers have been closed, but no part was found at the specified nominal width. This state will be active until the next grasp or release command is issued.
- ***Part lost (3)***
  A part was grasped but then lost before the fingers have been opened again. This state will be active until the next grasp or release command is issued.
- ***Holding (4)***
  A part was grasped successfully and is now being hold with the grasping force.
- ***Releasing (5)***
  The fingers are currently opening towards the opening width to release a part.
- ***Positioning (6)***
  The fingers are currently pre-positioned using a “move” command.

The Get Grasping State command supports the automatic transmission of update packets in either fixed time intervals or if the grasping state changes. This gives you a precise control over the bus load of your system.
When sending this command with automatic updates disabled (FLAGS’0=0), one return packet containing the current grasping state is immediately returned.

If you select to send automatic update messages only in case the grasping state changed, the time interval between two packets still is maintained, even if the changing rate of the grasping state is higher than PERIOD_MS.

Command ID: 41h

Command Parameters:

Byte	Symbol	Data Type	Description

0	FLAGS	bit vector	D7...D2: unused, set to 0 D1: Change-sensitive Update: 1: Update on change only 0: Update always D0: Automatic Update: 1: auto update is enabled 0: auto update is disabled
1..2	PERIOD_MS	integer	Minimum period between two automatically sent packets in milliseconds.

Returned Parameters:

Byte	Symbol	Data Type	Description
0	GSTATE	enum	Grasping state: 0: Idle 1: Grasping 2: No part found 3: Part lost 4: Holding 5: Releasing 6: Positioning 7 to 255: Reserved

Possible Error Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

3.4.3 Get Grasping Statistics (42h)

Get the current statistics for the number of executed grasps, lost or not found parts.

Command ID: 40h

Command Parameters:

Byte	Symbol	Data Type	Description
0	FLAGS	bit vector	D7...D1: unused, set to 0 D0: Reset Statistics: 1: reset grasping statistics after reading 0: do not reset

Returned Parameters:

Byte	Symbol	Data Type	Description
0..3	TOTAL	integer	Number of total grasps
4..5	NO_PART	integer	Number of grasps, where no part was found at the given position
6..7	LOST_PART	integer	Number of grasps, where the part was lost before the gripper was opened again.

Possible Error Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

3.4.4 Get Opening Width (43h)

Get the current finger position. This command supports the automatic transmission of update packets in either fixed time intervals or if the finger opening width changes. This gives you a precise control over the bus load of your system.
When sending this command with automatic updates disabled (FLAGS'0=0), one return packet containing the current finger position is immediately returned. A change is detected, if the width changes for an absolute amount of at least 0.01 mm.

If you select to send automatic update messages only in case the finger opening width did change, the time interval between two packets still is maintained, even if the changing rate of the system state is higher than PERIOD_MS.

The command returns the distance between the fingers, not their absolute position!

Command ID: 43h

Command Parameters:

Byte	Symbol	Data Type	Description
0	FLAGS	bit vector	D7...D2: unused, set to 0 D1: Change-sensitive Update: 1: Update on change only 0: Update always D0: Automatic Update: 1: auto update is enabled 0: auto update is disabled
1..2	PERIOD_MS	integer	Minimum period between two automatically sent packets in milliseconds.

Returned Parameters:

Byte	Symbol	Data Type	Description
0	WIDTH	float	Finger opening width in millimeters.

Possible Error Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

3.4.5 Get Speed (44h)

Get the current finger speed. This command supports the automatic transmission of update packets in either fixed time intervals or if the finger speed changes. This gives you a precise control over the bus load of your system.
When sending this command with automatic updates disabled (FLAGS'0=0), one return packet containing the current finger speed is immediately returned. A change is detected, if the finger speed changes for an absolute amount of at least 0.05 mm/s.

If you select to send automatic update messages only in case the finger speed changed, the time interval between two packets still is maintained, even if the changing rate of the system state is higher than PERIOD_MS.

The command returns the relative speed of the fingers to each other.

Command ID: 44h

Command Parameters:

Byte	Symbol	Data Type	Description
0	FLAGS	bit vector	D7...D2: unused, set to 0 D1: Change-sensitive Update: 1: Update on change only 0: Update always D0: Automatic Update: 1: auto update is enabled 0: auto update is disabled
1..2	PERIOD_MS	integer	Minimum period between two automatically sent packets in milliseconds.

Returned Parameters:

Byte	Symbol	Data Type	Description
0	SPEED	float	Finger speed in mm/s.

Possible Error Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

3.4.6 Get Force (45h)

Get the current grasping force. This command supports the automatic transmission of update packets in either fixed time intervals or if the grasping force changes. This gives you a precise control over the bus load of your system.
When sending this command with automatic updates disabled (FLAGS'0=0), one return packet containing the current grasping force is immediately returned. A change is detected, if the grasping force changes for an absolute amount of at least 0.05 N.

If you select to send automatic update messages only in case the grasping force changed, the time interval between two packets still is maintained, even if the changing rate of the system state is higher than PERIOD_MS.

The command returns the grasping force, i.e. the sum of the nominal force times the fingers.

Command ID: 45h

Command Parameters:

Byte	Symbol	Data Type	Description
0	FLAGS	bit vector	D7...D2: unused, set to 0 D1: Change-sensitive Update: 1: Update on change only 0: Update always D0: Automatic Update: 1: auto update is enabled 0: auto update is disabled
1..2	PERIOD_MS	integer	Minimum period between two automatically sent packets in milliseconds.

Returned Parameters:

Byte	Symbol	Data Type	Description
0	FORCE	float	Grasping force in Newtons.

Possible Error Codes:

E_SUCCESS: Command succeeded.

E_CMD_FORMAT_ERROR: Command length mismatch.

3.5 System Configuration

3.5.1 Get System Information (50h)

Get some information about the connected gripper that can be used e.g. to evaluate the functional range of the gripper.

Command ID: 50h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0	TYPE	enum	Gripper Type: 0: unknown 1: WSG 50
1	HWREV	integer	Hardware Revision
2..3	SWREV	integer	Firmware Revision
4..7	SN	integer	Serial Number

Possible Error 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

3.5.2 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 gripper or any additional process information that is used in conjunction with the gripper. The maximum length of the Device Tag is 64 characters. The text string must not contain any control characters. 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 Error 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.

3.5.3 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 Error 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.

3.5.4 Get System Limits (53h)

Get the gripper's physical limits for stroke, speed, acceleration and force. You can use these values when sending movement-related commands to the gripper to ensure that all parameters are within the system's limits.

Command ID: 53h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0..3	STROKE	float	Gripper stroke in mm
4..7	MIN_SPEED	float	Minimum speed in mm/s
8..11	MAX_SPEED	float	Maximum speed in mm/s
12..15	MIN_ACC	float	Minimum acceleration in mm/s²
16..19	MAX_ACC	float	Maximum acceleration in mm/s²
20..23	MIN_FORCE	float	Minimum grasping force in N
24..27	NOM_FORCE	float	Nominal grasping force in N (duty cycle of 100%)
28..31	OVR_FORCE	float	Maximum overdrive grasping force in N (can only be set in Overdrive Mode, see Chapter 3.3.8)

Possible Error 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

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

4 Appendix A: Error Codes

All commands are acknowledged with an error code. Table 2 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_NO_PARAM_EXPECTED	A Parameter was given, but none expected
13	E_NOT_ENOUGH_PARAMS	Not enough parameters for executing 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

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, if the function was 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 2: Possible error codes

5 Appendix B: System State Flags

The system state flags are arranged as a 32-bit wide integer value that can be read using the Get System State command (see Chapter 3.4.1). Each bit has a special meaning listed below.

Bit No.	Flag Name	Description
D31..21	reserved	These bits are currently unused but may be used in a future release of the WSG firmware.
D20	SF_SCRIPT_FAILURE	**Script Error.** An error occurred while executing the script and the script was aborted. This flag is reset when starting a script.
D19	SF_SCRIPT_RUNNING	**A script is currently running.** The flag is reset if the script either terminated normally, a script error occurred or the script was terminated manually by the user.
D18	SF_CMD_FAILURE	**Command Error.** The last command returned an error.
D17	SF_FINGER_FAULT	**Finger Fault.** The status of at least one finger is different from “operating” and “not connected”. Please check the finger flags for a more detailed error description.
D16	SF_CURR_FAULT	**Engine Current Error.**
D15	SF_POWER_FAULT	**Power Error.** The power supply is outside the valid range.
D14	SF_TEMP_FAULT	**Temperature Error.** The gripper hardware has reached a critical temperature level. All movement-related commands are disabled, until the temperature falls below the critical level.
D13	SF_TEMP_WARNING	**Temperature Warning.** The gripper hardware will soon reach a critical temperature level.

D12	SF_FAST_STOP	**Fast Stop.** The gripper was stopped due to an error condition. You have to acknowledge the error in order to reset this flag and to re-enable movement-related commands.
D11..10	reserved	These bits are currently unused but may be used in a future release of the WSG firmware.
D9	SF_FORCECNTL_MODE	**Force Control Mode.** True Force Control is currently enabled by using the installed Force Measurement Finger (WSG-FMF). If this flag is not set, the grasping force is controlled by approximation based on the motor current.
D8	SF_OVERDRIVE_MODE	**Overdrive Mode.** Gripper is in overdrive mode and the grasping force can be set to a value up to the overdrive force limit. If this bit is reset, the grasping force cannot be higher than the gripper's nominal grasping force value.
D7	SF_TARGET_POS_REACHED	**Target position reached.** Set after a Goto or Grasp command, if the target position was successfully reached. This flag is reset on the next movement command.
D6	SF_AXIS_STOPPED	**Axis stopped.** A previous movement command was aborted using the stop command. This flag is reset on the next movement command.
D5	SF_SOFT_LIMIT_PLUS	**Positive direction soft limit reached.** The fingers reached the defined soft limit in positive moving direction. A further movement into this direction is not allowed anymore. This flag is cleared, if the fingers have been moved away from the soft limit position.
D4	SF_SOFT_LIMIT_MINUS	**Negative direction soft limit reached.** The fingers reached the defined soft limit in negative moving direction. A further movement into this direction is not allowed anymore. This flag is cleared, if the fingers have been moved away from the soft limit position.
D3	SF_BLOCKED_PLUS	**Axis is blocked in positive moving direction.** You may use this flag to detect that a part was grasped.

D2	SF_BLOCKED_MINUS	**Axis is blocked in negative moving direction.** You may use this flag to detect that a part was grasped.
D1	SF_MOVING	**The Fingers are currently moving.** This flag is reset automatically if the movement stops.
D0	SF_REFERENCED	**Fingers Referenced.** If set, the gripper is referenced and accepts movement commands.

6 Appendix C: Sample code for calculating the checksum

The following code demonstrates how to calculate the CRC checksum for communicating with the WSG (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 );
}
```

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.