WSG Series of Intelligent Servo-Electric Grippers

Command Set Reference Manual

Firmware Version 4.0.0

July 2015

www.weiss-robotics.com

Contents

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 as well as of the WSG's command set. To get started with the communication protocol, we recommend the free WSG Commander application running on Microsoft Windows.

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

- Hexadecimal values are noted with a trailing "h", e.g. 12h, while decimal values are not specially marked.
- The data transmission is based on a little endian representation of multi-byte words, where the least significant byte is transferred first. This means that the integer number 1234h is represented by two consecutive bytes 34h and 12h.
- Floating point values are represented by 4 byte 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: 32 Bit floating point number according to IEEE 754
- 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 Connecting to the WSG Grippers

The WSG family of grippers offers a number of interfaces, each of which uses the same binary message protocol as described in this manual. Available interfaces are RS-232 serial connection, Ethernet TCP/IP, Ethernet UDP/IP and CAN-Bus. The Profibus interface is fundamentally different from these interfaces. It is covered separately in the Fieldbus Interface Manual.
The interface configuration can be changed using the WSG's web interface. Connect the WSG to the local network or directly to your computer's network interface and point your favorite web browser to the gripper's IP address, e.g. by typing the default **http://192.168.1.20** into the address bar and pressing "Enter". After the page has loaded, choose "*Settings -> Command Interface*" from the menu to configure the standard interface.

1.2 Communicating with the WSG Grippers

Regardless of the interface being used, the WSG communicates with its client using binary data packets. The following chapters describe the general format of these commands.

1.2.1 Command Message Format

The table below illustrates the generic command format. All packets start with a preamble signaling the beginning of a new data packet. 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-16 checksum is added to each packet to verify data integrity.

☞ **A source code example for calculating the CRC-16 checksum over a message is given in Appendix D.**
If you decide not to use the CRC, e.g. on transmission-safe protocols like TCP/IP, you can disable the WSG's checksum evaluation using the WSG's web interface (see chapter 1.4.1).

To check a received message, you have to calculate the CRC checksum again over the received data frame including preamble and 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 2) to interactively assemble 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 D 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

1.2.2 Command Acknowledge from the Device

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	STATUS_CODE	Status code, see chapter Appendix A
8..n	PARAMS	Command specific parameters. Only available, if the status code is E_SUCCESS.
n+1..n+2	CHECKSUM	CRC checksum of the whole data packet, including the preamble. See Appendix D 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 3: 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 available immediately, e.g. on movement or referencing commands, the WSG returns a notification that it did understand the received command and started its execution (command pending). However, the result will be sent in an additional packet after the command execution has completed. The immediate response to such an asynchronous command will be a packet with E_CMD_PENDING as status 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 target 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

1.4 Interface Options

Dependent on the interface type, there are some additional settings beyond the basic interface configuration that influence the WSG’s behavior. The following section gives a short overview.

1.4.1 Disable CRC Checksum

As described in the previous chapters, a two-byte CRC-16 checksum is appended to each message to ensure data integrity when exchanging messages using the WSG’s communication protocol. In some cases, however, it may become necessary to turn off the WSG’s checksum evaluation, e.g. for testing purposes when implementing your own version of the WSG binary protocol.

This can be done using the web interface, by choosing “*Settings -> Command Interface*” from the menu and disabling the appropriate checkbox.

If CRC checksum evaluation is disabled, the WSG will ignore the value of the checksum field in all messages it receives, though the two-byte checksum field must still be present in all messages.

Please note that the checksum evaluation will only be disabled for incoming messages. The WSG’s outgoing messages will always be equipped with a valid checksum, regardless of this option being enabled or not.

This feature is intended for experts only. It is strongly recommended to leave the CRC checksum enabled to ensure maximum data integrity!

1.4.2 Disable Error State on TCP Disconnect

On connection oriented protocols like TCP, accidentally lost connections must be considered as severe errors. This is why the WSG automatically issues a FAST STOP whenever a TCP connection to the command interface is closed by the client without a previous announcement (see chapter 3.1.2).
Sometimes it may become necessary to turn off this feature. This can be done using the web interface, by choosing "*Settings -> Command Interface"* from the menu. If TCP is selected as default interface, the appropriate checkbox can be disabled.

This feature is intended for experts only. Lost connections can be a serious problem, so the WSG should always perform a FAST STOP in this case. Try to adjust your application before disabling this error condition.

2 WSG Commander

The tool WSG Commander is provided free of charge and allows you to get familiar with the WSG's communication protocol and command set. It allows to send basic commands to the WSG and contains a custom command editor to assemble your 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® and can be installed from either the Product CD that ships together with the WSG or can be downloaded at the WSG's web interface by choosing "*Help -> Documentation*" from the menu.

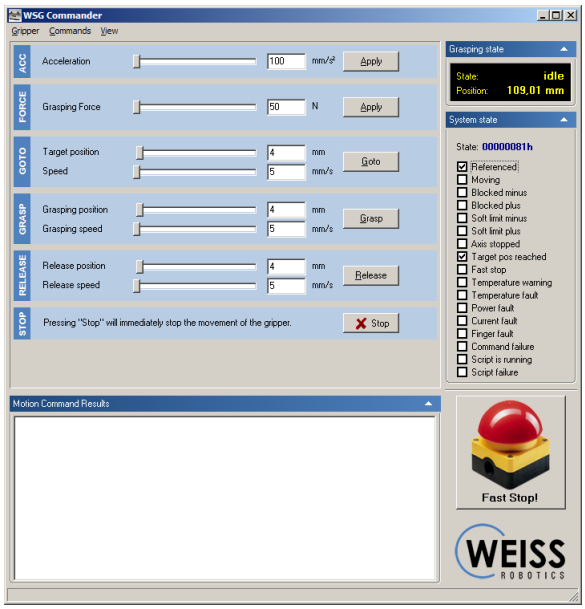


Figure 1: WSG Commander Main Window

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.

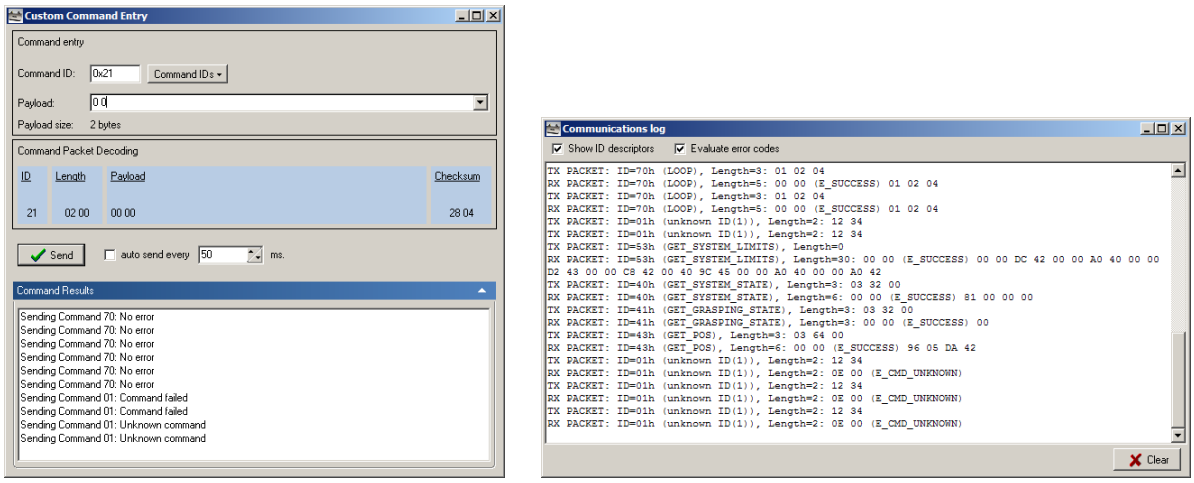


Figure 2: WSG Commander Custom Command Editor (left) and Communication Log (right)

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 provided as a tool to make the evaluation of the WSG's command set as easy and comfortable as possible. 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 to test the communication interface.

Command ID: 06h

Command Parameters:

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

Returned Parameters:

The received LOOPDATA is identically returned within the command acknowledge (note, that the two bytes error code is automatically added to the beginning of the message as described in chapter 1.2.2).

Possible Status Codes:

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

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 notify the device about a regular disconnection. Any finger movement that is executed when the disconnect announcement arrives is aborted immediately. By sending this command before closing the connection, the gripper will 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 Status 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 motion-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 motion-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 Status 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

3.2.2 Pre-position Fingers (21h)

Move the gripper fingers to a defined opening width. This command is intended to pre-position the gripper fingers prior to a gripping a part. To grip or release a part, the *Grip Part* (25h, cf. chapter 3.2.6) and *Release Part* (26h, cf. chapter 3.2.7) commands must 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 the command has been received, the WSG returns a packet with an E_CMD_PENDING error, meaning it did understand and initiated execution of the command. After the target 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 motion parameters against it before issuing a motion-related command to ensure that the gripper behaves as intended.

To get in-depth information about the current finger movement, you may use this command in conjunction with the Get Opening Width command, see chapter 3.4.4.

Pre-positioning 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 grip or release a part, please use the Grip Part (25h, cf. chapter 3.2.6) and Release Part (26h, cf. chapter 3.2.7) commands.

The gripper has to be referenced and may not be in 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 towards the movement direction of the fingers is detected, the motion command returns an E_AXIS_BLOCKED error and the motor is stopped. *0: Clamp on block.* If a blocking condition 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 a fixed limit of 50% of the gripper's nominal force. ⚠ **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 motion* The passed width is treated as an offset to the current opening width. *0: absolute motion* 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 Status 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.
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

3.2.3 Stop (22h)

Immediately stops any ongoing finger 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 motion-related command.

☞ **If you would like 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 Status 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 finger movement the fastest way and prevents further motion-related commands from being executed. The FAST STOP state can only be left by issuing a **FAST STOP Acknowledge message**. All motion-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 finger movement without raising an error condition, 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 Status 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 Status Codes:

E_SUCCESS: Command succeeded.
E_CMD_FORMAT_ERROR: Acknowledge key is incorrect.

3.2.6 Grip Part (25h)

Grip a part by passing its nominal width and the speed at which the part should be gripped. When the command is issued, the gripper moves its fingers to the given part width and tries to clamp the expected part with the previously set gripping force. If the gripper can establish the desired gripping force within the defined clamping travel, a part is gripped. If the fingers fall through the clamping travel without establishing the gripping force, no part was found. The clamping travel can be set using the WSG's web interface. The gripping state is updated with the result of this operation (either HOLDING or NO PART) as well as the gripping statistics (see chapter 3.4.2). If no part was found, the command returns E_CMD_FAILED.
You may reduce the gripping speed with sensitive parts to limit the impact due to the mass of the gripper fingers and the internal mechanics.
The gripping state reflects the current state of the process. You can read it using the *Get Gripper State* command (see chapter 3.4.2).
Please note that it is not possible to send a subsequent grip command that will move the fingers in the opposite direction. In general, a grip command should always be followed by a release command (see chapter 3.2.7) before the next grip command is issued. However, it is possible to re-grip in the same direction, e.g. if the previously submitted griping width was too large.

Command ID: 25h

Command Parameters:

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

Returned Parameters:

No parameters are returned

Possible Status 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: Gripping aborted.
E_CMD_FAILED: No part found.
E_TIMEOUT: Timeout while gripping.

3.2.7 Release Part (26h)

Release a previously gripped 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 Status 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.

3.3 Motion Configuration

3.3.1 Set Acceleration (30h)

Set the axis acceleration for consecutive motion-related commands, e.g. *Grip Part* or *Pre-position Fingers*.

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 Status 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 Status 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 pre-positioning and gripping commands. The force limit is the maximum gripping 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 product of the effective force times the number of fingers.

The force in pre-positioning mode is always estimated using the motor current. Please keep in mind, that this might not be 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 gripping force and if exceeding the nominal force.

Returned Parameters:

No parameters are returned

Possible Status 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 product of the effective force times the number of fingers.

Command ID: 33h

Command Parameters:

No parameters required

Returned Parameters:

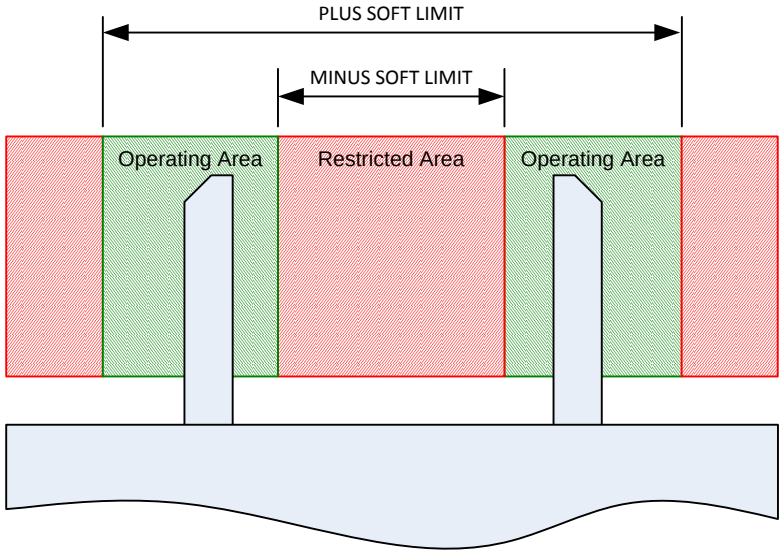
Byte	Symbol	Data Type	Description
0..3	ACC	float	Force Limit in Newtons

Possible Status 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 the 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 motion-related commands if the 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 motion-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 finger 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 by 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 motion direction (mm).
4..7	LIMIT_PLUS	float	Soft limit opening width in positive motion direction (mm).

Returned Parameters:

No parameters are returned

Possible Status 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 Status 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 Status Codes:

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

3.3.8 Overdrive Mode (37h)

Enable or disable force overdrive mode. By default, the gripper only allows to set a gripping force that is not higher than the nominal value, which can be applied with a duty cycle of 100%. By enabling overdrive mode, the gripping 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 higher than the nominal force value is set, 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.

Overdrive mode is not supported by all WSG grippers. Please refer to the User's Manual for further information.

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

When entering or leaving overdrive mode, a 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 gripping force limit, the maximum allowed value is the overdrive force. *0: Overdrive mode disabled.* When setting the gripping 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.

Force sensors are not available on all WSG grippers. Please refer to the User's Manual for further information.

This command is only allowed if not in force control mode (i.e. the gripping 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 Status 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.**

⚠ **The system state flags are not intended to control the gripping process. Please use the gripping state instead (see chapter 3.4.2).**

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. Minimum value is 10 ms.

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 Status Codes:

E_SUCCESS: Command succeeded.
E_CMD_FORMAT_ERROR: Command length mismatch.

3.4.2 Get Gripper State (41h)

Get the current gripper state. The gripper state can be used to control and monitor the gripping process. The following states are possible and will be encoded into a single number:

- ***Idle (0)***
  The gripping process is in idle state and is waiting for a command.
- ***Gripping (1)***
  The fingers are currently closing to grip a part. The part has not been gripped, yet
- ***No part found (2)***
  The fingers have been closed, but no part was found at the specified nominal width within the given clamping range or the desired gripping force could not be reached. This state will persist until the next Grip Part, Release Part or Pre-position Fingers command is issued.
- ***Part lost (3)***
  A part was gripped but then lost before the fingers have been opened again. This state will persist until the next Grip Part, Release Part or Pre-position Fingers command is issued.
- ***Holding (4)***
  A part was gripped successfully and is now being hold with the gripping 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 Pre-position Fingers command or the gripper is homing.
- ***Error (7)***
  An error occurred during the gripping process. This state will persist until the next Grip Part, Release Part or Pre-position Fingers command is issued.

The *Get Gripper State* command supports the automatic transmission of update packets in either fixed time intervals or if the gripper 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 gripper state is immediately returned.

If you select to send automatic update messages only in case the gripper state changed, the time interval between two packets still is maintained, even if the changing rate of the gripper 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. Minimum value is 10 ms.

Returned Parameters:

Byte	Symbol	Data Type	Description
0	GSTATE	enum	**Gripper state:** 0: Idle 1: Gripping 2: No part found 3: Part lost 4: Holding 5: Releasing 6: Positioning 7: Error 8 to 255: Reserved

Possible Status Codes:

E_SUCCESS: Command succeeded.
E_CMD_FORMAT_ERROR: Command length mismatch.

3.4.3 Get Gripping Statistics (42h)

Get the current statistics for the number of executed grips, 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 gripping statistics after reading* *0: do not reset*

Returned Parameters:

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

Possible Status 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 opening width is immediately returned. A change is detected, if the width changes for an absolute amount of at least 0.01 mm.

If the gripper is not referenced, the command will return an opening width of 0 and not send any messages on changing opening width.

If you select to send automatic update messages only in case the finger opening width 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 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. Minimum value is 10 ms.

Returned Parameters:

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

Possible Status 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. Minimum value is 10 ms.

Returned Parameters:

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

Possible Status Codes:

E_SUCCESS: Command succeeded.
E_CMD_FORMAT_ERROR: Command length mismatch.

3.4.6 Get Force (45h)

Get the current gripping force. This command supports the automatic transmission of update packets in either fixed time intervals or if the gripping 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 gripping force is immediately returned. A change is detected, if the gripping force changes for an absolute amount of at least 0.05 N.

⚠ **If you select to send automatic update messages only in case the gripping 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 gripping force, i.e. the sum of the effective 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. Minimum value is 10 ms.

Returned Parameters:

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

Possible Status Codes:

E_SUCCESS: Command succeeded.
E_CMD_FORMAT_ERROR: Command length mismatch.

3.4.7 Get Temperature (46h)

Get the current device temperature. This can be used to check the operating conditions of the device.

Command ID: 46h

Command Parameters:

No parameters required

Returned Parameters:

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

Possible Status Codes:

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

3.5 System Configuration

3.5.1 Get System Information (50h)

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

Command ID: 50h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0	TYPE	enum	Device Type: 0: unknown 1: WSG 50 2: WSG 32 (3: Force-Torque Sensor KMS 40) (4: Tactile Sensing Module WTS) 5: WSG 25 6: WSG 70
1	HWREV	integer	Hardware Revision
2..3	FW_VERSION	integer	Firmware Version: D15..12 major version D11..8 minor version 1 D7..4 minor version 2 D3..0 0 for release version, 1..15 for release candidate versions
4..7	SN	integer	Serial Number

Possible Status Codes:

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

3.5.2 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 Status Codes:

E_SUCCESS: Command succeeded.
E_OVERRUN: Tag value is too long.
E_INVALID_PARAMETER: Tag contains illegal characters.
E_INSUFFICIENT_RESOURCES: Out of memory.

3.5.3 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 Status Codes:

E_SUCCESS: Command succeeded.
E_NO_PARAM_EXPECTED: A parameter was given, but not expected.
E_NOT_AVAILABLE: No device tag present.
E_INSUFFICIENT_RESOURCES: Out of memory.

3.5.4 Get System Limits (53h)

Get the gripper's physical limits for stroke, speed, acceleration and force. You can use these values when sending motion-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 gripping force in N
24..27	NOM_FORCE	float	Nominal gripping force in N (duty cycle of 100%)
28..31	OVR_FORCE	float	Maximum overdrive gripping force in N[1]

Possible Status Codes:

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

[1] Overdrive mode is not supported by all WSG grippers. Please refer to the User's Manual for further information.

3.6 Finger Interface

Some modules of the WSG series of grippers provide a sensor port in each base jaw to which sensor fingers can be connected to. The following commands are used to access and control these fingers.

The finger interface is not available on all WSG grippers. Please refer to the User's Manual for further information.

3.6.1 Get Finger 1 Info (60h)

Return information about the connected finger. Use this command to determine the type of the connected finger and to get the size of one data frame returned by this finger.

Command ID: 60h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0	TYPE	enum	Finger type: 0: generic or no finger installed 1: WSG-FMF 2: WSG-DSA 3..255: reserved
1..2	SIZE	integer	Size of one data frame in bytes that is returned by the *Finger 1 Get Data* (ID 62h) command. If the finger doesn't support the *Get Data* command, SIZE is 0000h.

Possible Status Codes:

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

3.6.2 Get Finger 1 Flags (61h)

Return the state flags for finger 1.

Command ID: 61h

Command Parameters:

No parameters required

Returned Parameters:

<table>
<tr><th>Byte</th><th>Symbol</th><th>Data Type</th><th>Description</th></tr>
<tr><td>0..1</td><td>FLAGS</td><td>bit vector</td><td>Finger Flags
These flags represent the state of the selected finger.
<table><tr><th>Bit Index:</th><th>Name</th><th>Description</th></tr><tr><td>Bit 0:</td><td>POWER_ON</td><td>If set, finger is powered up</td></tr><tr><td>Bit 1:</td><td>CONFIG_AVAIL</td><td>The connected finger provides configuration data (i.e. an intelligent finger was detected on this sensor port)</td></tr><tr><td>Bit 2:</td><td>COMM_OPEN</td><td>A communication interface is open</td></tr><tr><td>Bit 3..7:</td><td>(reserved)</td><td></td></tr><tr><td>Bit 8:</td><td>POWER_FAULT</td><td>An Over-Current fault was detected</td></tr><tr><td>Bit 9:</td><td>COMM_FAULT</td><td>A communication fault occurred during runtime</td></tr><tr><td>Bit 10..15</td><td>(reserved)</td><td></td></tr></table></td></tr>
</table>

Possible Status Codes:

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

3.6.3 Finger 1 Power Control (62h)

Enables or disables the power supply for finger 1. This may be used in conjunction with custom hardware to control the behavior of the finger.
Enabling the power supply is executed as an asynchronous command because the system will wait some time until the finger is powering up. In this case, the first command result is E_CMD_PENDING followed by an E_SUCCESS after approx. 500 ms.
Disabling power is always executed immediately, i.e. without the E_CMD_PENDING mechanism.

The power supply can only be controlled, if the finger is of generic type.

Command ID: 62h

Command Parameters:

Byte	Symbol	Data Type	Description
0	ON/OFF	enum	Set this byte to 0 to disable the power supply or to 1 to enable it. The power supply can only be controlled for generic fingers. All fingers are powered up on system startup by default.

Returned Parameters:

No parameters are returned.

Possible Status Codes:

Immediate errors:
E_SUCCESS: Command succeeded (when disabling power).
E_CMD_FORMAT_ERROR: Command length mismatch.
E_CMD_FAILED: Over-current detected while enabling the finger's power supply.
E_CMD_PENDING: Power was enabled, waiting for the finger to startup.

Errors upon completion of the command (only when enabling power):
E_SUCCESS: Enabling the finger's power supply succeeded.

3.6.4 Get Finger 1 Data (63h)

Return the current finger data for predefined finger types. The length of the finger-specific data can be obtained using the *Get Finger 1 Info* command (see chapter 3.6.1).

☞ **The content and length of the returned data depends on the installed finger type. Please see the documentation of the resp. finger.**

Command ID: 63h

Command Parameters:

No parameters required

Returned Parameters:

Finger-specific data.

Possible Status Codes:

E_SUCCESS: Command succeeded.
E_CMD_FORMAT_ERROR: Command length mismatch.
E_IO_ERROR: A communication error occurred while accessing the finger.
E_NOT_AVAILABLE: The selected finger does not support finger-specific data.

3.6.5 Get Finger 2 Info (70h)

Return information about the connected finger. Use this command to determine the type of the connected finger and to get the size of one data frame returned by this finger.

Command ID: 70h

Command Parameters:

No parameters required

Returned Parameters:

Byte	Symbol	Data Type	Description
0	TYPE	enum	Finger type: 0: generic or no finger installed 1: WSG-FMF 2: WSG-DSA 3..255: reserved
1..2	SIZE	integer	Size of one data frame in bytes that is returned by the *Finger 2 Get Data* (ID 72h) command. If the finger doesn't support the *Get Data* command, SIZE is 0000h.

Possible Status Codes:

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

3.6.6 Get Finger 2 Flags (71h)

Return the state flags for finger 2.

Command ID: 71h

Command Parameters:

No parameters required

Returned Parameters:

<table>
<tr><th>Byte</th><th>Symbol</th><th>Data Type</th><th>Description</th></tr>
<tr><td>0..1</td><td>FLAGS</td><td>bit vector</td><td>Finger Flags
These flags represent the state of the selected finger.
<table><tr><th>Bit Index:</th><th>Name</th><th>Description</th></tr><tr><td>Bit 0:</td><td>POWER_ON</td><td>If set, finger is powered up</td></tr><tr><td>Bit 1:</td><td>CONFIG_AVAIL</td><td>The connected finger provides configuration data (i.e. an intelligent finger was detected on this sensor port)</td></tr><tr><td>Bit 2:</td><td>COMM_OPEN</td><td>A communication interface is open</td></tr><tr><td>Bit 3..7:</td><td>(reserved)</td><td></td></tr><tr><td>Bit 8:</td><td>POWER_FAULT</td><td>An Over-Current fault was detected</td></tr><tr><td>Bit 9:</td><td>COMM_FAULT</td><td>A communication fault occurred during runtime</td></tr><tr><td>Bit 10..15</td><td>(reserved)</td><td></td></tr></table></td></tr>
</table>

Possible Status Codes:

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

3.6.7 Finger 2 Power Control (72h)

Enables or disables the power supply for finger 2. This may be used in conjunction with custom hardware to control the behavior of the finger.
Enabling the power supply is executed as an asynchronous command because the system will wait some time until the finger is powering up. In this case, the first command result is E_CMD_PENDING followed by an E_SUCCESS after approx. 500 ms.
Disabling power is always executed immediately, i.e. without the E_CMD_PENDING mechanism.

The power supply can only be controlled, if the finger is of generic type.

Command ID: 72h

Command Parameters:

Byte	Symbol	Data Type	Description
0	ON/OFF	enum	Set this byte to 0 to disable the power supply or to 1 to enable it. The power supply can only be controlled for generic fingers. All fingers are powered up on system startup by default.

Returned Parameters:

No parameters are returned.

Possible Status Codes:

Immediate errors:
E_SUCCESS: Command succeeded (when disabling power).
E_CMD_FORMAT_ERROR: Command length mismatch.
E_CMD_FAILED: Over-current detected while enabling the finger's power supply.
E_CMD_PENDING: Power was enabled, waiting for the finger to startup.

Errors upon completion of the command (only when enabling power):
E_SUCCESS: Enabling the finger's power supply succeeded.

3.6.8 Get Finger 2 Data (73h)

Return the current finger data for predefined finger types. The length of the finger-specific data can be obtained using the *Get Finger 2 Info* command (see chapter 3.6.5).

☞ **The content and length of the returned data depends on the installed finger type. Please see the documentation of the resp. finger.**

Command ID: 73h

Command Parameters:

No parameters required

Returned Parameters:

Finger-specific data.

Possible Status Codes:

E_SUCCESS: Command succeeded.
E_CMD_FORMAT_ERROR: Command length mismatch.
E_IO_ERROR: A communication error occurred while accessing the finger.
E_NOT_AVAILABLE: The selected finger does not support finger-specific data.

Appendix A. Status Codes

All commands are acknowledged with a status code. Table 2 lists the valid status codes and describes their reason.

Status 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	The command was not completed, yet. Another return message will follow including a status 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 status codes

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 a script and the script has been aborted. The flag is reset whenever a script is started.
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 has been 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.** The engine has reached its maximum thermal power consumption. The flag will be reset automatically as soon as the engine has recovered. Then the corresponding Fast Stop can be committed.
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 motion-related commands are disabled until the temperature falls below the critical level.

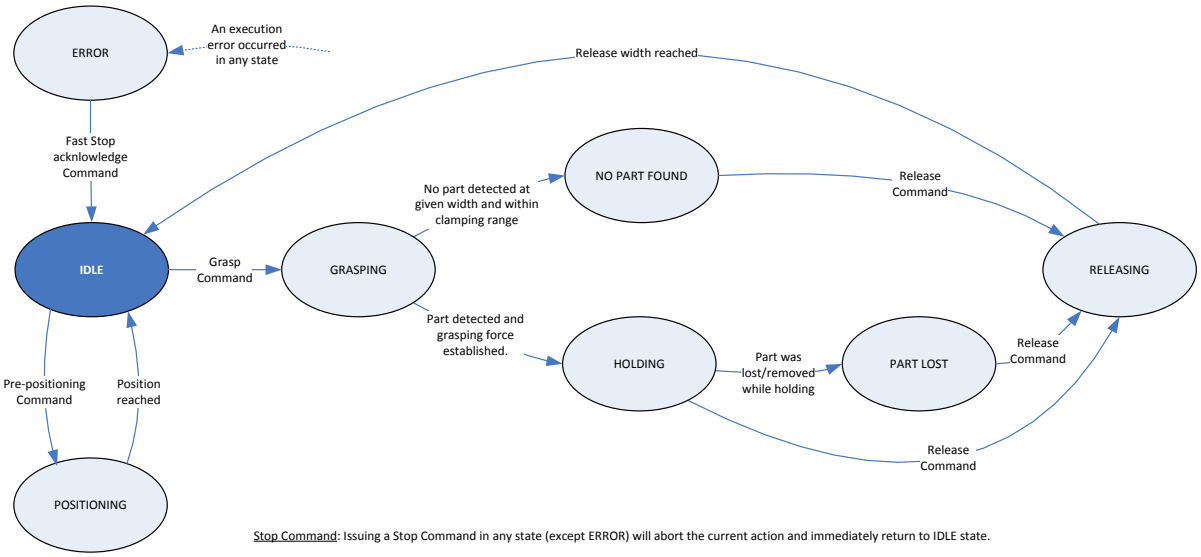
D13	SF_TEMP_WARNING	**Temperature Warning.** The gripper hardware will soon reach a critical temperature level.
D12	SF_FAST_STOP	**Fast Stop.** The gripper has been stopped due to an error condition. You have to acknowledge the error in order to reset this flag and to re-enable motion-related commands.
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 gripping force is controlled by approximation based on the motor current.
D8	SF_OVERDRIVE_MODE	**Overdrive Mode[2].** Gripper is in overdrive mode and the gripping force can be set to a value up to the overdrive force limit. If this bit is not set, the gripping force cannot be higher than the gripper's nominal gripping force value.
D7	SF_TARGET_POS_REACHED	**Target position reached.** Set if the target position was reached. This flag is not synchronized with SF_MOVING, so it is possible that there is a delay between SF_MOVING being reset and SF_TARGET_POS becoming active.
D6	SF_AXIS_STOPPED	**Axis stopped.** A previous motion command has been aborted using the stop command. This flag is reset on the next motion command.

[2] Overdrive mode is not supported by all WSG grippers. Please refer to the User's Manual for further information.

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 any more. This flag is cleared if the fingers are 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 any more. This flag is cleared if the fingers are moved away from the soft limit position.
D3	SF_BLOCKED_PLUS	**Axis is blocked in positive moving direction.** Set if the axis is blocked in positive moving direction. The flag is reset if either the blocking condition is resolved or a stop command is issued.
D2	SF_BLOCKED_MINUS	**Axis is blocked in negative moving direction.** Set if the axis is blocked in negative moving direction. The flag will be reset if either the blocking condition is resolved or a stop command is issued.
D1	SF_MOVING	**The Fingers are currently moving.** This flag is set whenever a movement is started (e.g. MOVE command) and reset automatically as soon as the movement stops.
D0	SF_REFERENCED	**Fingers Referenced.** If set, the gripper is referenced and accepts motion-related commands.

Appendix C. Gripper States

The following diagram illustrates the gripper states and transitions as intended to be used in normal operation.



Appendix D. Sample code to calculate 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 );
}
```

www.weiss-robotics.com