



GRIPLINK PLUGIN FOR STÄUBLI

Version 4.0.0

April 2026



Contents

1	Introduction	3
1.1	Notation and Symbols.....	3
1.2	Intended Use.....	3
1.3	System Requirements.....	4
1.4	License Terms	4
2	Changes from Previous Versions	5
2.1	New error handling — x_nErrorCode	5
2.2	Removal of the implicit device-state call in GRIP, RELEASE and HOME	5
2.3	Command rename: devstate is now deviceState	5
2.4	Switch to floating-point parameters.....	6
2.5	New commands — FLEXGRIP and FLEXRELEASE	6
2.6	GRIP and RELEASE — Parameter x_bWstrEnabled replaced by x_nDoBlock	6
2.7	WAITVAL — Extended Parameters.....	7
2.8	CLAMP — Parameter type changed	7
2.9	Commands that are still available (deprecated).....	7
2.10	Note on Migration	8
3	Installation	9
3.1	Preparing the robot	9
3.2	Installation of the software via STAUBLI Robotics Suite.....	10
3.3	Setting up the socket connection for the GRIPLINK Controller	11
3.3.1	Teach Pendant	11
3.3.2	STAUBLI Robotics Suite	12
3.4	Transferring the VAL3 application via Transfer Manager	14
3.4.1	Configuring the connection to the CS9 controller	15
3.4.2	Establishing the connection in Transfer Manager	15
3.4.3	Verifying the target controller	16
3.4.4	Selecting the required components	16
3.4.5	Reloading the application on the SP2 pendant	17
3.5	Behavior in the Event of an Error	18
3.6	Emergency Stop	19
4	Command Reference	20
4.1	Example programs — GRIPLINK_Examples	21
4.2	Establish Connection — CONNECT	23
4.3	Close Connection — DISCONNECT	24
4.4	Check Connected Device — DEVASSERT	25
4.5	Enable Device — ENABLE	26

4.6	Disable Device – DISABLE.....	27
4.7	Home Reference – HOME.....	29
4.8	Grip Workpiece via Preset – GRIP.....	30
4.9	Release Workpiece via Preset – RELEASE.....	32
4.10	Gripping Workpiece via Parameters – FLEXGRIP.....	33
4.11	Release Workpiece via Parameters – FLEXRELEASE.....	34
4.12	Configuring a Grip Preset – SETGRIPCFG.....	35
4.13	Query current Device Status – DEVICESTATE.....	37
4.14	Waiting for a Device State Change – WSTR.....	38
4.15	Querying Device Value – VALUE.....	40
4.16	Set Device Value – SETVALUE.....	41
4.17	Waiting for a Device Value – WAITVAL.....	42
4.18	Control Force Retention – CLAMP.....	43
4.19	Controlling the Status LED – LED.....	44
4.20	Deprecated Commands.....	45
4.20.1	Simultaneous Gripping of Workpieces – MGRIP.....	45
4.20.2	Simultaneous Release of Workpieces – MRELEASE.....	46
4.20.3	Waiting for Multiple Devices to Change Their State – MWAITFOR.....	48
5	Troubleshooting.....	49
5.1	Error Messages.....	49
5.1.1	GRIPLINK error response with status code.....	49
5.1.2	Socket variable not linked to hardware I/O.....	49
5.1.3	Instruction timeout due to missing socket communication.....	50
5.1.4	Socket communication interrupted during operation.....	50
Appendix A	Device status.....	52
Appendix B	Status codes.....	53

1 Introduction

GRIPLINK technology enables IO-Link-compatible automation components to be connected to robot systems from leading manufacturers via a network connection. The GRIPLINK plugin for Stäubli serves as the control-side interface and facilitates the seamless integration of WEISS ROBOTICS' GRIPLINK technology into robot systems from Stäubli.

The term “GRIPLINK Controller” is used in this manual to refer to all products that integrated GRIPLINK technology.



These include both standalone GRIPLINK Controllers (e.g., GRIPLINK-ET4) as well as devices with embedded GRIPLINK functionality, such as WPG or INTRAPAL servo grippers from Weiss Robotics. All of these products implement the same GRIPLINK protocol and can therefore be controlled in the same way via the plugin.



This manual describes the functions of the GRIPLINK plugin. For information on installation, commissioning, and operation of the GRIPLINK Controller, please refer to the Operating Manual for the respective module. These can be found online at www.griplink.de

1.1 Notation and Symbols

For clarity, the following symbols are used in this manual:



Function- or safety-related note. Failure to observe these instructions may endanger the safety of personnel and the system, damage the device, or impair the device's function.



Additional information to aid in understanding the described topic.



Reference to further information.

1.2 Intended Use

The “GRIPLINK-Plugin” software is intended for communication between the GRIPLINK Controller from Weiss Robotics and a robot controller. It enables the integration and control of GRIPLINK-compatible devices via the standardized GRIPLINK communication interface.

The requirements of the applicable guidelines as well as the installation and operating instructions in this manual must be observed and complied with. Any other or additional use is considered improper. The manufacturer is not liable for any resulting damage.

1.3 System Requirements

This plugin is compatible with **GRIPLINK protocol version 3** and higher.

One of the following Stäubli robot controllers is required for operation:

- CS9 with software version s8.10.6 or higher

The following robot options are required to run the software:

- Stäubli SRS



Contact your Stäubli dealer to purchase the robot options.



The IP address of the GRIPLINK Controller must be on the same subnet as that of the robot controller. The GRIPLINK Controller manual describes the exact procedure for changing the IP address.

1.4 License Terms

The GRIPLINK plugin is protected by copyright. The applicable license terms are included with the software package. By installing the software, you accept these license terms.

2 Changes from Previous Versions

Version 4.0.0 changes the parameterization, the command signatures and the behavior in the event of an error. This version is not fully backward compatible; existing applications must be reviewed and adapted.

2.1 New error handling — `x_nErrorCode`

From version 4.0.0 onwards, every public command returns its status through the by-reference parameter `x_nErrorCode`.

- -1: no response, or an unparseable response, from the GRIPLINK controller.
- 0 (`E_SUCCESS`): the command was executed successfully.
- 1 – 27: status codes of the GRIPLINK controller (see Appendix B).



A connection loss or socket timeout is reported as a VAL3 runtime error (e.g. `ERR 60`, `ERR 123`, `ERR 125`).

The error response must be implemented in the user program. A complete description is given in Section 3.5.

2.2 Removal of the implicit device-state call in `GRIP`, `RELEASE` and `HOME`

In version 3.x.x, `grip()`, `release()` and `home()` returned the gripper's device state through the by-reference parameter `x_nDevstate`. From version 4.0.0 this output is removed; the by-reference parameter is now `x_nErrorCode` in all three commands. If the device state is required after a blocking `grip()`, `release()` or `home()`, it must be queried explicitly via `deviceState()`.

Up to Version 3.x.x	From Version 4.0.0
<code>grip(port, preset, devstate&, bWstrEnabled:bool)</code>	<code>grip(port, preset, nDoBlock, errorCode&)</code>
<code>release(port, preset, devstate&, bWstrEnabled:bool)</code>	<code>release(port, preset, nDoBlock, errorCode&)</code>
<code>home(port, devstate&)</code>	<code>home(port, errorCode&)</code>

2.3 Command rename: `devstate` is now `deviceState`

The command `devstate()` has been renamed to `deviceState()` in version 4.0.0. Calls of the form `call griplink:devstate(...)` must be replaced by `call griplink:deviceState(...)`.

2.4 Switch to floating-point parameters

The scaled integer parameters used previously have been replaced by floating-point values. In earlier versions, physical values (e.g., positions or velocities) were transmitted with a scaling factor (typically 1000).



Starting with version 4.0.0, values are transferred directly in physical units.

This change affects the following commands:

- SETGRIPCFG
- VALUE
- SETVAL
- WAITVAL

Effects

- Scaling factors (e.g., factor of 1000) may no longer be used
- Parameter values must be converted to actual physical values (e.g., mm instead of mm × 1000)

Example

Up to version 3.x.x	Starting with version 4.0.0
POSITION = 19500	POSITION = 19.5

2.5 New commands – FLEXGRIP and FLEXRELEASE

The following new commands were introduced in version 4.0.0:

- **FLEXGRIP**: Gripping operation with freely selectable motion parameters (position, force, speed, acceleration).
- **FLEXRELEASE**: Release of a gripped workpiece with freely selectable motion parameters (position, speed, acceleration).

2.6 GRIP and RELEASE — Parameter x_bWstrEnabled replaced by x_nDoBlock

For grip() and release(), the previous parameter x_bWstrEnabled (bool) has been replaced by x_nDoBlock (num). The blocking behavior is now controlled directly within the command: x_nDoBlock = 1 waits for the target state to be reached (corresponds to the old x_bWstrEnabled = true); x_nDoBlock = 0 returns immediately (corresponds to the old x_bWstrEnabled = false). The separate command WSTR remains unaffected and can still be used to monitor state changes, in combination with x_nDoBlock = 0 for non-blocking calls.

Up to version 3.x.x	From version 4.0.0
<code>grip(port, preset, devstate&, bWstrEnabled:bool)</code>	<code>grip(port, preset, nDoBlock, errorCode&)</code>
<code>release(port, preset, devstate&, bWstrEnabled:bool)</code>	<code>release(port, preset, nDoBlock, errorCode&)</code>

2.7 WAITVAL – Extended Parameters

The WAITVAL command has been significantly expanded. In addition to the switch to floating-point values, new parameters have been introduced that enable precise monitoring of a threshold value with a tolerance window and timeout.

Up to version 3.x.x	From version 4.0.0
<code>waitval(port, index, value)</code>	<code>waitval(port, index, threshold, windowsize, timeout, errorCode&)</code>

2.8 CLAMP – Parameter type changed

The parameter `x_bEnable` (bool) has been replaced by `x_nEnable` (num). Existing calls with true/false must be changed to 1/0.

Up to version 3.x.x	From version 4.0.0
<code>clamp(port, bEnable:bool)</code>	<code>clamp(port, nEnable, errorCode&)</code>

2.9 Commands that are still available (deprecated)

The following commands are still available but are considered **deprecated**:

- MGRIP
- MRELEASE
- MWAITFOR

These commands are retained for backward compatibility but should no longer be used for new projects. For new applications, it is recommended to use the following commands:

- GRIP/RELEASE in combination with DOBLOCK = 0 instead of MGRIP/MRELEASE
- WSTR for status monitoring instead of MWAITFOR

The new approach offers greater flexibility in program design, as processes and logic can be individually adapted to the specific application. At the same time, it enables better control over program flow and the synchronization of multiple devices. Furthermore, the use of a uniform interface ensures consistent control of different device types.

2.10 Note on Migration

Due to the changes in version 4.0.0, existing programs are **not fully backward compatible**. Existing applications must be reviewed and adapted, particularly with regard to the following points:

- Removal of scaling factors
- Switching from `x_bWstrEnabled` to `x_nDoBlock` for GRIP and RELEASE
- Adaptation of WAITVAL calls to the new signature
- Conversion from true/false to 1/0 for the CLAMP parameter



It is recommended to perform a complete functional test of the application after migration.

3 Installation



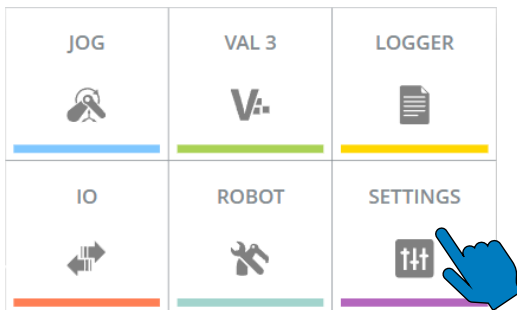
Ensure that you are using the latest version of the GRIPLINK plugin. The latest version can be downloaded at www.griplink.de/software.

3.1 Preparing the robot

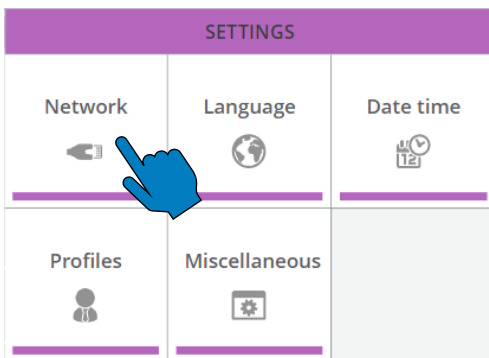
Turn on the robot and configure the IP address (e.g., 192.168.1.14). Make sure that the robot and the GRIPLINK Controller are on the same network.

To do this, follow these steps:

1. Click on 
2. Navigate to “SETTINGS”



3. Navigate to “Network”



4. Set the IP address for the interface in use to a valid address

NETWORK SETTINGS

Hostname: Haldir

FTP Server Protocol: FTPS + FTP

SRS Port: 851

Remote Access Port: 5900

Interfaces

Interface	Status	IP Address
J204	connected	192.168.1.14
J205	connected	192.168.1.15

Gateways

No gateway

J204

Hostname: Haldir

FTP Server Protocol: FTPS + FTP

SRS Port: 851

Remote Access Port: 5900

MAC Address: 0a-00-27-00-00-08

DHCP: disabled

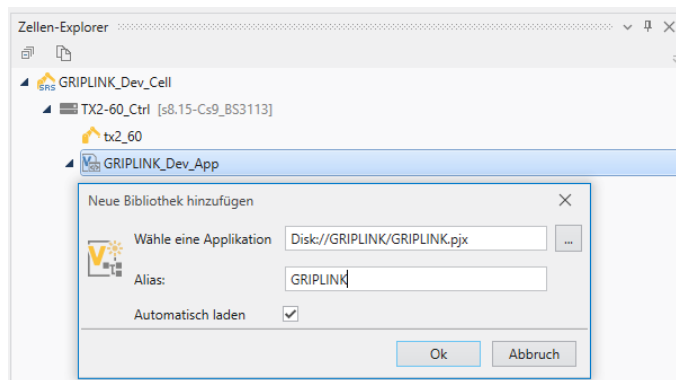
IP: 192.168.1.14

Mask: 255.255.255.0

OK Cancel

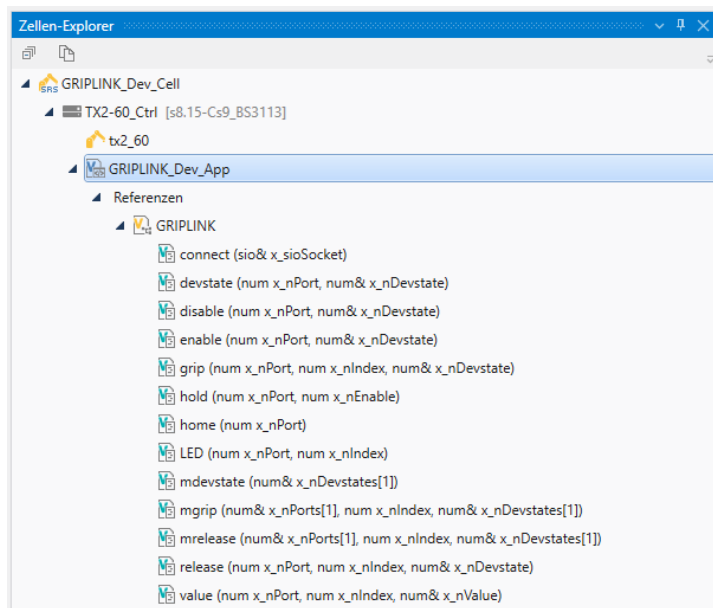
3.2 Installation of the software via STAUBLI Robotics Suite

1. Download the plugin file "griplink_staubli_<Version>.zip".
2. Unzip the ZIP archive into the "usrapp" directory:
".../<cell name>/<controller name>/usr/usrapp"
3. Open the cell in SRS and navigate to the Cell Explorer.
4. Right-click the VAL3 program in where you want to use the plugin and add the plugin program "GRIPLINK.pjx" as a library.



In the "Alias" field, enter a unique name that you will use to access the plugin functions in your application's subprograms.

5. A program named "GRIPLINK" now appears in the Cell Explorer



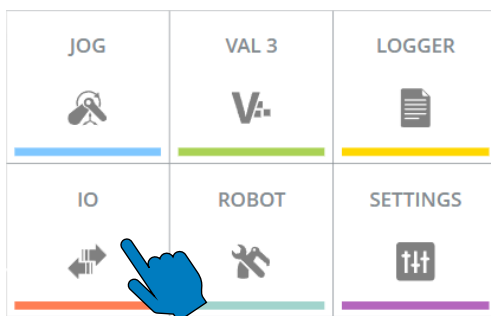
3.3 Setting up the socket connection for the GRIPLINK Controller

The network interface is accessed via socket variables, which must be configured beforehand.

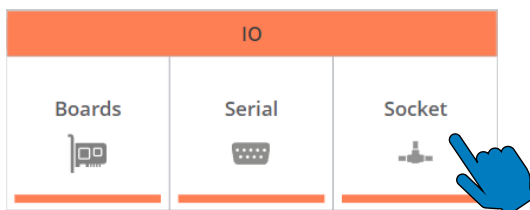
3.3.1 Teach Pendant

To do this, follow these steps:

1. Click 
2. Navigate to "IO"



3. Navigate to "Socket"



4. Select “TCP Clients” and open the input screen by clicking on “+”. Enter the name of the socket connection and the remaining interface parameters. Confirm by clicking “OK”.



Set the following values:

Field	Value	Unit
Server IP	IP address of the GRIPLINK Controller	-
Port	10001	-
Timeout	60	seconds
End of string	10	-



Make sure you enter the correct name that you also use in the robot program.

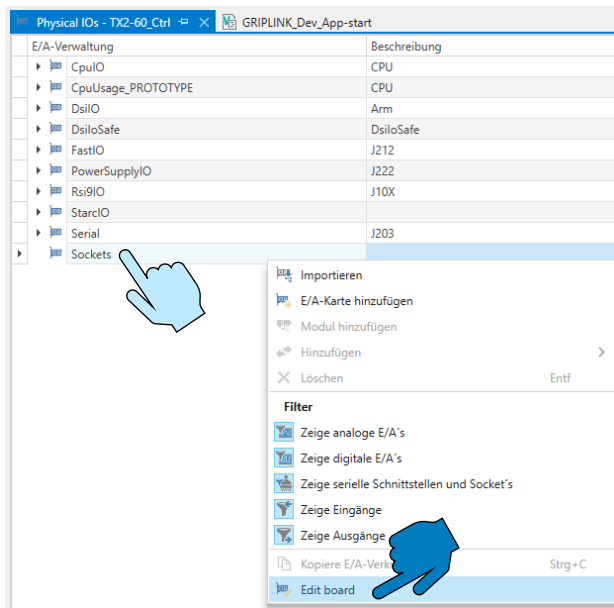


Other values may cause malfunctions!

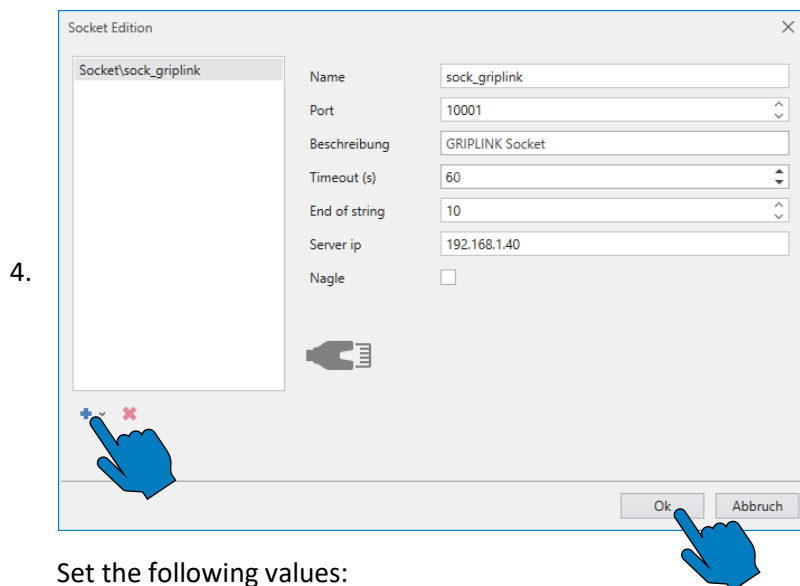
3.3.2 STAUBLI Robotics Suite

To do this, follow these steps:

1. Open the I/O management of the robot controller in SRS
2. Open the input screen for a new socket connection by right-clicking on “Sockets”



3. Add a new socket by clicking on "+". Enter the name of the socket connection and the remaining interface parameters. Confirm by clicking "OK."



Set the following values:

Field	Value	Unit
Server IP	IP address of the GRIPLINK Controller	-
Port	10001	-
Timeout	60	seconds
End of string	10	-



Make sure you enter the correct name that you also use in the robot program.



Using different values may cause the system to malfunction!

5. The socket connection now appears in the table.

Sockets		
sock_griplink	GRIPLINK Socket	Socket\sock_griplink

6. The socket connection can now be assigned to a socket variable in the robot program

7.

The screenshot shows the 'Physical I/Os - TX2-60_Ctrl' table with the following data:

Index	lo
0	Socket\sock_griplink

The 'Sockets' list shows the following entries:

E/A-Verwaltung	Beschreibung	Format	Logical Name	E/A-Verknüpfung...
Serial	J203			
Sockets				
sock_griplink	GRIPLINK Socket		GRIPLINK_Dev ...	Socket\sock_gr...

The 'Globale Variablen' panel shows the following structure:

- GRIPLINK_Dev_Cell
 - TX2-60_Ctrl [s8.15-Cs9_BS3113]
 - tx2_60
 - GRIPLINK
 - GRIPLINK_Dev
 - mdesc
 - 3/4 num
 - sio
 - sioGriplink
 - sioGriplink[0]

When loading VAL3 applications from the STAUBLI Robotics Suite onto the controller, it may be necessary to relink the SIO variable via the Application Manager. To do this, create an SIO socket connection (see section 3.3.1).



Click and open the Application Manager.

Click on "VAL3 Application" → "Hard Disk" → "Program" → "Global Variables" → "sio" → <Variable Name>.

Click on "Link (F2)" → "Edit (F6)" → Socket → "TCP Clients" → "Socket Name" → "OK (F8)" → "OK (F8)" → "Save (F8)"

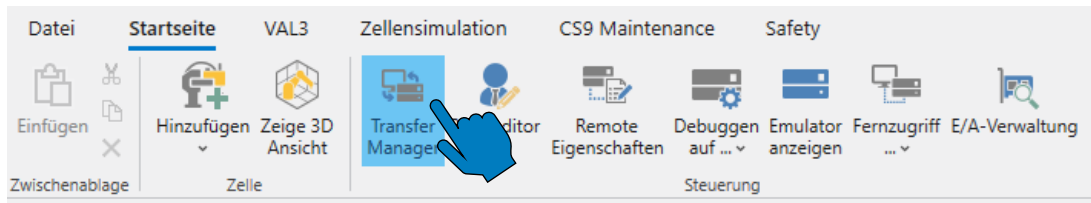
3.4 Transferring the VAL3 application via Transfer Manager

The VAL3 applications and the required configurations are transferred to the CS9 controller via the Transfer Manager of Stäubli SRS.

The Ethernet interfaces must be configured separately. Both interfaces must be on different networks:

- J204 (Transfer Manager): 172.31.0.1
- J205 (GRIPLINK TCP/IP communication): 192.168.1.1

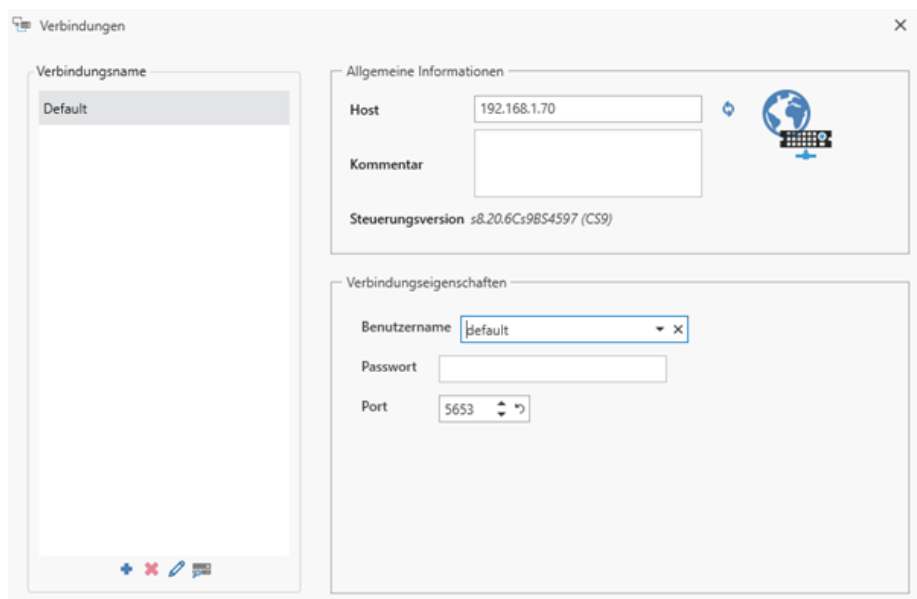
In Stäubli SRS, the Transfer Manager is opened via the top menu bar:



3.4.1 Configuring the connection to the CS9 controller

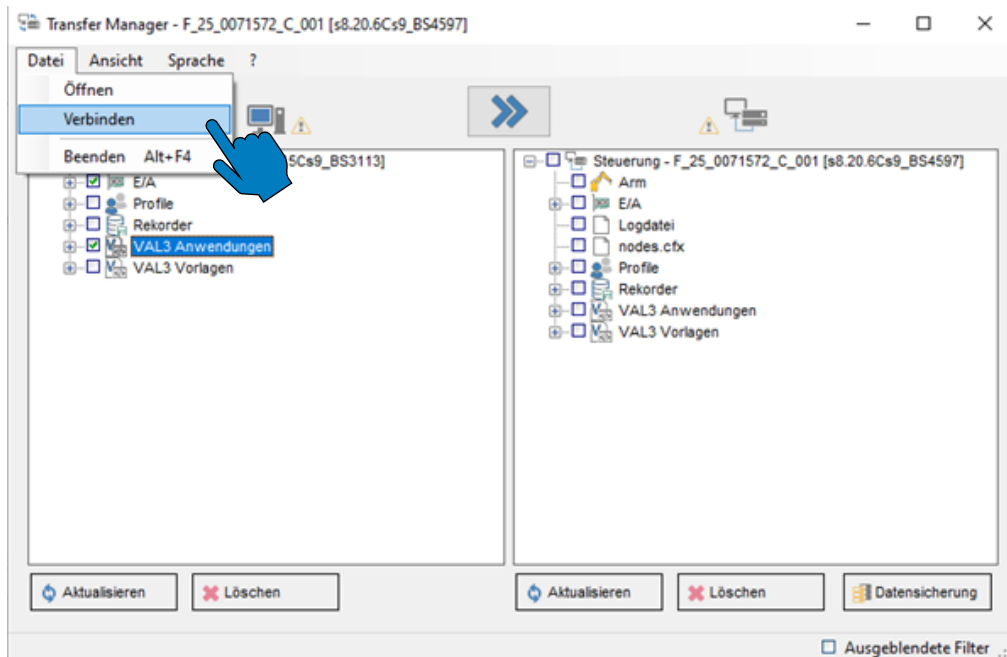
In the connection dialog the connection to the CS9 controller is configured. The controller's IP address is entered as the host.

After a successful connection the controller's software version is displayed:



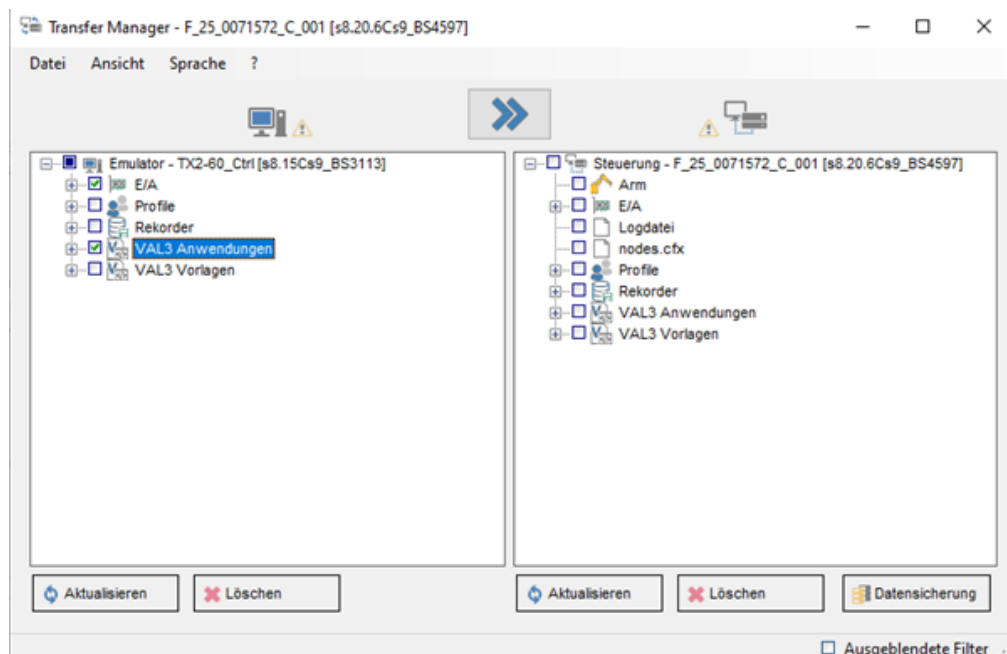
3.4.2 Establishing the connection in Transfer Manager

After opening the Transfer Manager, the connection to the CS9 controller is established via File → Connect. On the left side the local project or the emulator is shown; on the right side the connected CS9 controller is shown.



3.4.3 Verifying the target controller

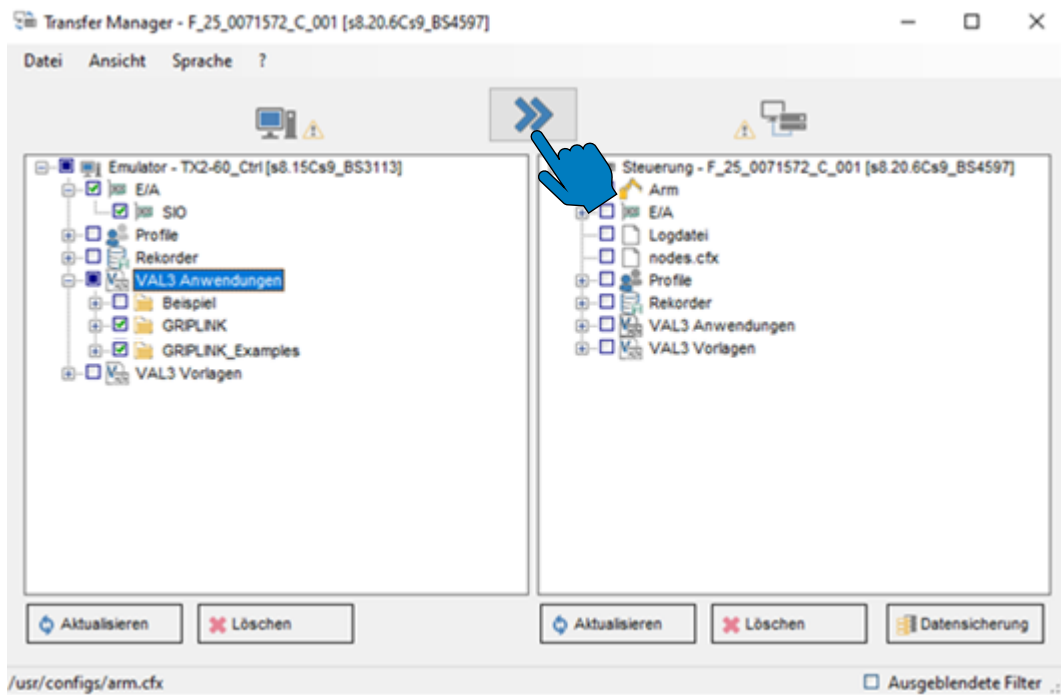
After a successful connection the CS9 controller is visible in the right-hand pane of the Transfer Manager. The controller is displayed with its software version. This confirms that the Transfer Manager is connected to the intended target controller.



3.4.4 Selecting the required components

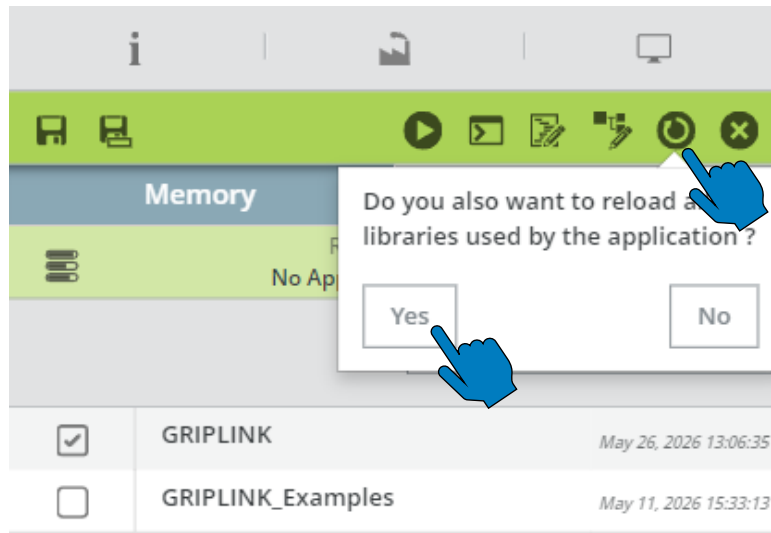
In the Transfer Manager the components required for the transfer are selected. These components are transferred to the disk of the CS9 controller:

- I/O (SIO)
- VAL3 application GRIPLINK
- VAL3 application GRIPLINK_Examples



3.4.5 Reloading the application on the SP2 pendant

After the transfer, the VAL3 applications are initially only on the disk of the CS9 controller. For the application to be executed, it is reloaded on the SP2 pendant. This causes the application and the libraries it uses to be loaded from the disk into the working memory of the controller.



3.5 Behavior in the Event of an Error

Every public command of the GRIPLINK plugin returns its execution status through the by-reference parameter `x_nErrorCode`. The value must be evaluated after every call and handled in the user program in the event of an error.

Possible values of the status code:

- -1: the controller response was empty or could not be parsed by VAL3.
- 0 (E_SUCCESS): command executed successfully.
- 1 – 27: status codes from the GRIPLINK controller (see Appendix B).



A true connection loss or socket timeout is reported through VAL3 runtime errors (e.g. ERR 60, ERR 123, ERR 125).

Recommended call pattern:

After every GRIPLINK call, check `x_nErrorCode` and, if it is non-zero, call the routine `errorHandling()`.

The `errorHandling.pgx` template in the `GRIPLINK_Examples` project decodes the status code into a log entry and contains a clearly marked section where the application-specific error response (`stopMove`, `movej`, `taskSuspend`, `taskKill`) can be added.

```
call griplink:grip(x_nPortGripper, 0, 1, l_nErrorCode)
if l_nErrorCode != 0
    call errorHandling(l_nErrorCode)
    return
endif
```

For an overview of typical error messages, their causes, and possible solutions, see the section 5.1.

3.6 Emergency Stop

The current emergency-stop state can be queried using the VAL3 function `esStatus()` (see the VAL3 reference manual). Depending on the application, the appropriate response may be to disable the gripper, keep it in the HOLDING state, or display a message to the operator.



An emergency stop interrupts the robot motion. The TCP/IP connection to the GRIPLINK controller is not torn down by VAL3 during the emergency stop.



The appropriate response to an emergency stop depends on the safety concept of the application. The decision on how the TCP/IP connection and the gripper should react during an emergency stop is to be implemented in the user program.

4 Command Reference

The GRIPLINK plugin provides a range of functions for controlling and monitoring GRIPLINK-compatible devices. These commands can be used to activate and control gripper modules as well as to query statuses and process values.

Both single and multiple commands are available. The commands are called using the user-defined alias of the library reference.

In this command reference, parameters, as in VAL3, are distinguished in two forms:

Passed by value

A Parameter passed by value sends a local copy of the data to the command. Since the command operates on this copy, any modifications remain local and do not affect the original variable in the calling program.

Examples:

- num x_nPort
- num x_nValueIndex

Passed by reference

A parameter passed by reference is indicated by the "&" symbol following the type. In this case, not only a value is passed, but also the reference to the original variable itself. This allows the method to directly modify the variable's value. This mechanism is used in the GRIPLINK plugin to return result values.

Examples:

- num& x_nValue
- num& x_nDeviceState
- sio& x_sioSocket

Multiple commands

Multiple commands (prefix M) allow multiple devices to be addressed simultaneously. These commands are particularly suitable for handling large or flexible workpieces with multiple gripping modules.

Basic program flow

1. Establish a connection to the GRIPLINK Controller using **CONNECT**.
This command must be executed once at the beginning of the program.
2. Optional: Initialize or reference devices.
For certain device types (e.g., servo grippers without absolute encoders), a homing run with **HOME** may be required. Other device types, such as sensors, do not require referencing.
3. Activate devices using **ENABLE** and configure them, provided the device in question supports this.

4. Execute device functions, e.g., gripping or releasing for gripper modules (GRIP, RELEASE), or query device states and values.
5. Close the connection to the GRIPLINK Controller using **DISCONNECT** if the interface must be physically disconnected (e.g., during a tool change) or when the program ends.

To ensure reliable communication and optimal performance, the application should be implemented so that the connection to the GRIPLINK Controller remains active throughout the entire program runtime. In this case, **CONNECT** is executed once at the start of the program, and **DISCONNECT** is called only at the end of the program or in the event of an error.

Example: Minimal program

```
// Connect to GRIPLINK controller
call griplink:connect(sioGriplink, l_nErrorCode)

// Enable gripper
call griplink:enable(0, l_nGripperState, true, l_nErrorCode)

// Grip part
call griplink:grip(0, 0, 1, l_nErrorCode)

// Release part
call griplink:release(0, 0, 1, l_nErrorCode)

// Disconnect
call griplink:disconnect()
```

The available commands of the GRIPLINK plugin are described below. The library prefix “griplink” is used for the code examples.

4.1 Example programs — GRIPLINK_Examples

In addition to the call patterns listed in this command reference, the plugin package includes the complete VAL3 project GRIPLINK_Examples. The project contains runnable, tested example programs that show how the commands are combined in real workflows.

- **SimpleGripRelease:** a minimal grip cycle composed of connect, enable, home, grip, release.
- **AsyncPreposition:** non-blocking pre-positioning of the gripper in parallel with robot motion (release with x_nDoBlock = 0 followed by wstr).
- **CheckWorkpiece:** a grip cycle with workpiece check via value() to evaluate OK/NOK.
- **FlexgripCycle:** a gripping operation with freely selectable motion parameters (position, force, speed, acceleration)

- **MultiGripRelease:** synchronized gripping and releasing of multiple grippers using `grip()/release()` instead of the deprecated M-commands.
- **FaultRecoveryViaDisable:** recovery from a FAULT state via disable followed by enable.
- **GRIPLINKDemo:** a complete demo sequence that combines several of the patterns above.
- **errorHandling:** template for error handling (see Section 3.5). This routine is called from the other example programs whenever `x_nErrorCode` returns a non-zero value.

It is recommended to base new applications on these example programs rather than rebuilding the individual call patterns from this reference from scratch. The example programs are designed to be taken over as a template into your own VAL3 project and adapted to your application.

4.2 Establish Connection – CONNECT

This command establishes the connection between the GRIPLINK Controller and the robot controller. To do this, a Socket.IO variable is passed by reference, through which the subsequent commands are executed. This variable must first have been linked to a TCP/IP socket (see section 3.3).

The command also checks whether the connected GRIPLINK Controller meets the minimum requirements of the GRIPLINK protocol (see section 1.3).

Syntax

```
connect(sio& x_sioSocket, num& x_nErrorCode)
```

Parameters passed by reference

| | |
|-------------------|--|
| sio& x_sioSocket | Reference to the socket variable of the TCP/IP socket being used |
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |

Common causes of errors

Socket variable not initialized correctly

→ Check settings

GRIPLINK Controller not connected to the robot

→ Check network connection and power supply

GRIPLINK Controller configured differently

→ Check the GRIPLINK Controller's network settings via the web interface

The GRIPLINK Controller's protocol version does not match the GRIPLINK plugin

→ Update the GRIPLINK Controller's firmware

Example

Establishes a connection between the robot controller and the GRIPLINK Controller via the sioGriplink socket variable:

```
call griplink:connect(sioGriplink, l_nErrorCode)
```



The CONNECT command must be called before any other GRIPLINK command. If GRIPLINK commands are executed before a CONNECT, this triggers the error “ERR 123 IO variable not linked to an input or output.”

4.3 Close Connection – DISCONNECT

This command closes the connection between the robot controller and the GRIPLINK Controller. This may be necessary if the network connection to the GRIPLINK Controller is interrupted by a tool changer.



Disconnecting an existing connection is not required during normal program execution and should only be used with caution!

Syntax

```
disconnect()
```

Parameters

none

Example

Disconnect the connection between the robot and the GRIPLINK Controller when a tool change is required:

```
call griplink:disconnect()
```

```
// Change tool here!
```

```
call griplink:connect(sioGriplink, l_nErrorCode)
```

4.4 Check Connected Device – DEVASSERT

This command checks whether the intended device is connected to the selected port. If the device connected to the port does not have the specified vendor and product IDs, a status code is written to the parameter `x_nErrorCode`.

Syntax

```
devassert(num x_nPort, num x_nVid, num x_nPid, num& x_nErrorCode)
```

Parameters passed by value

| | |
|--------------------------|---|
| <code>num x_nPort</code> | Port index (0 to 31) |
| <code>num x_nVid</code> | Vendor ID specified in the IODD of the expected device |
| <code>num x_nPid</code> | Product ID specified in the IODD of the expected device |

Parameters passed by reference

| | |
|------------------------------------|--|
| <code>num& x_nErrorCode</code> | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |
|------------------------------------|--|

Example

Make sure that the STERIGRIP 200-030 device from Weiss Robotics (VID 815, PID 52) is connected to Port 0:

```
call GRIPLINK:devassert(0, 815, 52, l_nErrorCode)
```



You can find the Vendor and Product IDs in the device description provided by the manufacturer or in the IODD.

4.5 Enable Device – ENABLE

This command activates the device connected to the selected port. IO-Link gripper modules then switch to the RELEASED state and execute the release action of the currently selected grip preset.

The parameter `x_bWstrEnabled` can be used to enable waiting for a state transition (WSTR). If WSTR is enabled, command execution waits until the device state on the selected port changes to ENABLED or FAULT. If no state change occurs, command execution ends with a TIMEOUT.

If WSTR is enabled, the device state is written to the reference parameter `x_nDeviceState` after the command has been executed.

Syntax

```
enable(num x_nPort, num& x_nDeviceState, bool x_bWstrEnabled, num& x_nErrorCode)
```

Parameters passed by value

| | |
|---------------------|--|
| num x_nPort | Port index (0 to 31) |
| bool x_bWstrEnabled | Set to "true" to enable WSTR
Set to "false" to disable WSTR |

Parameters passed by reference

| | |
|---------------------|--|
| num& x_nDeviceState | Contains the device state after successful command execution, provided that WSTR is enabled |
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |

Example

Enables the device on port 0 and writes the new device state to `l_nDeviceState`:

```
call GRIPLINK:enable(0, l_nDeviceState, true, l_nErrorCode)
```



WSTR should only be used if the device was previously in the DISABLED state, as otherwise a timeout error will occur during command execution. The new device state is only written to the return variable if WSTR has been enabled!



Behavior may vary depending on the connected device. Please refer to the instructions in the respective device driver manual!

4.6 Disable Device – DISABLE

This command disables the device connected to the selected port. The command can be used, for example, to release the fingers of a gripper module or to temporarily disable the laser beam of a distance sensor.

The `x_bWstrEnabled` parameter can be used to enable waiting for a state transition (WSTR). If WSTR is enabled, command execution waits until the device state on the selected port changes to DISABLED or FAULT; if this does not happen, execution terminates with a TIMEOUT error.

Error Acknowledgment Using DISABLE

DISEABLE can also be used to acknowledge device errors, for example, if the connected device is in the FAULT state.

Syntax

```
disable(num x_nPort, num& x_nDeviceState, bool x_bWstrEnabled, num& x_nErrorCode)
```

Parameters passed by value

| | |
|---------------------|--|
| num x_nPort | Port index (0 to 31) |
| bool x_bWstrEnabled | Set to "true" to enable WSTR
Set to "false" to disable WSTR |

Parameters passed by reference

| | |
|---------------------|--|
| num& x_nDeviceState | Contains the device state after successful command execution, provided that WSTR is enabled |
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |

Example

A servo gripper is connected to Port 0. At system startup, the system should check for any error conditions before referencing and acknowledge them:

```
// Read current device state
call griplink:deviceState(0, l_nGripperState, l_nErrorCode)

// Check for FAULT state (7)
if l_nGripperState == 7

    // Gripper is in FAULT state
    call griplink:disable(0, l_nGripperState, true, l_nErrorCode)
    delay(1)
    call griplink:enable(0, l_nGripperState, true, l_nErrorCode)
endIf
call griplink:home(0, l_nErrorCode)
```



WSTR should only be used if the device was not previously in the DISABLED state, as otherwise a timeout will occur during command execution.



The new device state is written to the reference parameter only if WSTR is enabled.

4.7 Home Reference – HOME

Performs a homing cycle/reference drive on the selected device. The command executes the device's built-in home-referencing function and waits until it is complete.

Syntax

```
home(num x_nPort, num& x_nErrorCode)
```

Parameters passed by value

| | |
|-------------|----------------------|
| num x_nPort | Port index (0 to 31) |
|-------------|----------------------|

Parameters passed by reference

| | |
|-------------------|--|
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |
|-------------------|--|

Example

References the gripper at port 0:

```
call GRIPLINK:home(0, l_nErrorCode)
```



Behavior may vary depending on the connected device. Please refer to the instructions in the respective device driver manual!

4.8 Grip Workpiece via Preset – GRIP

The GRIP command performs a grip using the specified grip preset. The optional `x_nDoBlock` parameter can be used to specify whether the command returns immediately after the movement starts or only after it is complete:

`x_nDoBlock = 0` → The command returns immediately after the movement starts (non-blocking).

`x_nDoBlock = 1` → The command waits until the gripping process is complete (blocking).

The current state of the gripping module can be queried at any time using the `DEVICESTATE` command. In non-blocking mode (`x_nDoBlock = 0`), the `WSTR` command can also be used to wait for a state change to detect the end of the release.



The gripping parameters can be configured via the GRIPLINK Controller's web interface or using the "SET GRIPCFG" command (see section 4.12).

Syntax

```
grip(num x_nPort, num x_nPreset, num x_nDoBlock, num& x_nErrorCode)
```

Parameters passed by value

| | |
|----------------|--|
| num x_nPort | Port index (0 to 31) |
| num x_nPreset | Grip preset index (valid range depends on the device) |
| num x_nDoBlock | If 0, the command returns immediately after the motion starts. If 1, GRIPLINK waits until gripping is complete |

Parameters passed by reference

| | |
|-------------------|--|
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |
|-------------------|--|

Example

The gripper module at Port 0 should grip using grip preset 2. If no workpiece is found, the gripper should wait, reopen, and try again:

```
l_bLoop = true
while l_bLoop
    // Grip
    call griplink:grip(0, 2, 1, l_nErrorCode)
    call griplink:deviceState(0, l_nDeviceStatePort0, l_nErrorCode)

    if l_nDeviceStatePort0 == 5
        // Holding a part
```

```

        logMsg("Gripper in HOLDING state", 1)
        delay(1)
        l_bLoop = false

    elseif l_nDeviceStatePort0 == 4
        logMsg("Gripper in NO PART state", 2)
        delay(1)

        // Release
        call griplink:release(0, 2, 1, l_nErrorCode)

    elseif l_nDeviceStatePort0 == 7
        logMsg("Gripper in FAULT state", 3)
        return
    else
        logMsg("Gripper in invalid state", 3)
        return
    endif
endWhile

```

4.9 Release Workpiece via Preset – RELEASE

The RELEASE command releases a previously gripped workpiece using the parameters of the specified grip preset. The optional parameter x_nDoBlock can be used to control whether the command returns immediately after the start of the movement or only after its completion:

x_nDoBlock = 0 → The command returns immediately after the movement starts (non-blocking).

x_nDoBlock = 1 → The command waits until the release is complete (blocking).

The current device state of the gripping module can be queried at any time using the DEVICESTATE command.

In non-blocking mode (x_nDoBlock = 0), the WSTR command can also be used to wait for a state change to detect the end of the release.



The gripping parameters can be configured via the GRIPLINK Controller's web interface or using the "SET GRIPCFG" command (see section 0).

Syntax

```
release(num x_nPort, num x_nPreset, num x_nDoBlock, num& x_nErrorCode)
```

Parameters passed by value

| | |
|----------------|--|
| num x_nPort | Port index (0 to 31) |
| num x_nPreset | Grip preset index (valid range depends on the device) |
| num x_nDoBlock | If 0, the command returns immediately after the movement starts. If 1, GRIPLINK waits until the release movement is complete |

Parameters passed by reference

| | |
|-------------------|--|
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |
|-------------------|--|

Example

See example in 4.8

4.10 Gripping Workpiece via Parameters – FLEXGRIP

The FLEXGRIP command performs a gripping operation with freely selectable motion parameters.

Unlike the GRIP command, no predefined grip configuration (preset) is used here; instead, the motion sequence is determined entirely by the command parameters.

This allows the user to specify individual target positions, forces, speeds, and accelerations without having to rely on a saved grip preset.

The command can be executed either in blocking or non-blocking mode.

In blocking mode, the command returns only after the gripping process is complete.

After the command is called, the gripping process starts with the specified motion parameters. If the parameter `x_nDoBlock = 0` is set, the command returns immediately while the motion is executed in the background. If `x_nDoBlock = 1`, the command waits until the process is complete (workpiece gripped or no workpiece detected).

The current state can be queried at any time using the DEVICESTATE command. Alternatively, the completion of the command can be detected with WSTR if the command was executed non-blocking.

Syntax

```
flexgrip(num x_nPort, num x_nPosition, num x_nForce,  
num x_nSpeed, num x_nAcceleration, num x_nDoBlock, num& x_nErrorCode)
```

Parameters passed by value

| | |
|---------------------|--|
| num x_nPort | Port index (0 to 31) |
| num x_nPosition | Target position for "No Part Limit" in mm |
| num x_nForce | Gripping force in N |
| num x_nSpeed | Gripping speed in mm/s (0 = use automatic calculation) |
| num x_nAcceleration | Finger acceleration in mm/s ² (0 = use default value) |
| num x_nDoBlock | 0 = non-blocking, 1 = blocking |

Parameters passed by reference

| | |
|-------------------|--|
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |
|-------------------|--|

Example

To grip a workpiece with the gripper on port 0 and 60 N force, No-Part-Limit = 15 mm, speed = 45.5 mm/s, standard acceleration, non-blocking:

```
call griplink:flexgrip(0, 15.0, 60, 45.5, 0, 0, l_nErrorCode)
```

Blocking variant:

```
call griplink:flexgrip(0, 15.0, 60, 45.5, 0, 1, l_nErrorCode)
```

4.11 Release Workpiece via Parameters – FLEXRELEASE

The FLEXRELEASE command releases a previously gripped workpiece using freely selectable motion parameters. It can be used both for simple release and for targeted pre-positioning of a servo gripper. The motion parameters are defined entirely via the command arguments. After the command is called, the gripper module opens the fingers to the specified target position.

In non-blocking execution, the command returns immediately while the movement runs in the background. In blocking execution, the command waits until the movement is complete. Progress can be monitored using the DEVICESTATE command or the WSTR query command. The state typically changes to DS_RELEASED upon completion.

Syntax

```
flexrelease(num x_nPort, num x_nPosition, num x_nSpeed, num x_nAcceleration, num  
            x_nDoBlock, num& x_nErrorCode)
```

Parameters passed by value

| | |
|---------------------|--|
| num x_nPort | Port index (0 to 31) |
| num x_nPosition | Target position (release limit) in mm |
| num x_nSpeed | Opening speed in mm/s (0 = use default value) |
| num x_nAcceleration | Finger acceleration in mm/s ² (0 = use default value) |
| num x_nDoBlock | 0 = non-blocking, 1 = blocking |

Parameters passed by reference

| | |
|-------------------|--|
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |
|-------------------|--|

Example

Pre-positioning the gripper at port 0 to 19.5 mm with default parameters, non-blocking:

```
call griplink:flexrelease(0, 19.5, 0, 0, 0, l_nErrorCode)
```

Blocking variant:

```
call griplink:flexrelease(0, 19.5, 0, 0, 1, l_nErrorCode)
```

4.12 Configuring a Grip Preset – SETGRIPCFG

The SETGRIPCFG command is used to adjust the parameters of a grip preset. A grip preset defines the motion and force parameters that a gripper module uses when executing a GRIP or RELEASE command. Up to eight parameters can be configured for each grip preset.



The meaning and permissible value ranges of these parameters are device-specific and must be obtained from the respective device or driver documentation.

Syntax

```
setgripcfg(num x_nPort, num x_nPreset, string x_sTag, num& x_nParams[8],  
num& x_nErrorCode)
```

Parameters passed by value

| | |
|---------------|--|
| num x_nPort | Port index (0 to 31) |
| num x_nPreset | Index of the grip preset (valid range depends on the device) |
| string x_sTag | Preset tag |

Parameters passed by reference

| | |
|-------------------|--|
| num& x_nParams[8] | Array with 8 elements for the device-specific preset parameters 0 through 7. |
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |

Example: Parameters for gripping modules from Weiss Robotics via GRIPLINK-ET4

| | |
|-----------------|--|
| num x_nPort | Port index (0 to 31) |
| num x_nPreset | Index of the grip preset (permissible value range depends on the device) |
| string x_sTag | Preset tag |
| x_nParams[0] | No part limit (range depends on gripper model, in mm) |
| x_nParams[1] | Release Limit (value range depends on gripper model, in mm) |
| x_nParams[2] | Force Factor (value range depends on gripper model, in % [0..100]) |
| x_nParams[4..7] | Not used |

Example

For the IEG PLUS gripper module on port 3 of the GRIPLINK Controller, set the parameters of grip preset 1:

Preset Tag = "bearing"
No Part Limit = 31 mm
Release Limit = 41 mm
Force Factor = 59%

```
// l_nGripPresetParameters defined as an array of 8 integers  
l_nGripPresetParameters[0] = 31  
l_nGripPresetParameters[1] = 41  
l_nGripPresetParameters[2] = 59  
call griplink:setgripcfg(3, 1, "bearing", l_nGripPresetParameters,  
l_nErrorCode)
```

4.13 Query current Device Status – DEVICESTATE

This command returns the device status and writes it to the reference parameter x_nDeviceState.

Syntax

```
deviceState(num x_nPort, num& x_nDeviceState, num& x_nErrorCode)
```

Parameters passed by value

| | |
|-------------|----------------------|
| num x_nPort | Port index (0 to 31) |
|-------------|----------------------|

Parameters passed by reference

| | |
|---------------------|--|
| num& x_nDeviceState | Reference to return variable of the device status. |
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |

Example

Wait until the device state of the gripper module on port 2 is "HOLDING" (5):

```
// Grip with gripper
call griplink:grip(2, 0, 0, l_nErrorCode)
while l_nGripperState != 5
  call griplink:deviceState(2, l_nGripperState, l_nErrorCode)
  delay(0.1)
endWhile
```

4.14 Waiting for a Device State Change – WSTR

The WSTR command waits for a change in the device state at the specified port. It can be used, for example, to check whether an asynchronously executed release motion (RELEASE with x_nDoBlock=0) has been completed. This command blocks until a state change occurs on the specified device. After the state change, the new device state is written to the reference parameter x_nDeviceState.



If the state change does not occur within a specified time interval, a timeout error is triggered and program execution is interrupted.

Syntax

```
wstr(num x_nPort, num& x_nDeviceState, num& x_nErrorCode)
```

Parameters passed by value

| | |
|-------------|----------------------|
| num x_nPort | Port index (0 to 31) |
|-------------|----------------------|

Parameters passed by reference

| | |
|---------------------|--|
| num& x_nDeviceState | Reference to return variable of the device state |
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |

Example

Pre-position the gripper fingers using Preset 2 while the robot moves to the pickup position. Upon arrival, wait until the fingers are open, then move to the pickup position to grip the workpiece.

```
begin
  num l_nGripperState

  // Start pre-positioning the fingers
  call griplink:release(0, 2, 0, l_nErrorCode)

  // Move to approach point
  movej ()
  waitEndMove ()

  // Wait until the gripper changed its state
  call griplink:wstr(0, l_nGripperState, l_nErrorCode)
  if l_nGripperState != 3
    logMsg("Gripper not in RELEASED state", 3)
    return
  endif

  // Move to pick point
  movej ()
  waitEndMove ()
```

```
    // Grip workpiece  
    call griplink:grip(0, 2, 1, l_nErrorCode)  
end
```

4.15 Querying Device Value – VALUE

This command reads an indexed device value and writes the result to the reference parameter.



The valid value range for the value index can be found in the driver documentation for the respective device.

Syntax

```
value(num x_nPort, num x_nValueIndex, num& x_nValue, num& x_nErrorCode)
```

Parameters passed by value

| | |
|-------------------|---|
| num x_nPort | Port index (0 to 31) |
| num x_nValueIndex | Index of the device value (permissible value range depends on the device) |

Parameters passed by reference

| | |
|-------------------|--|
| num& x_nValue | Contains the read device value after successful execution |
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |

Example

Read the primary value (Value index 0, here: finger position) from the device at Port 3, a servo gripper, and store the value in the local variable l_nPartWidth. Use this for an IO/NIO check:

```
l_nGripperState = 0
l_nPartWidth    = 0

// Grip workpiece
call griplink:grip(3, 0, 1, l_nErrorCode)

// Get part width
call griplink:value(3, 0, l_nPartWidth, l_nErrorCode)

// Is part width 30 +/- 0.1?
if l_nPartWidth < 29.9 or l_nPartWidth > 30.1
  // Part is NOK -> Move to Drop position
  movej()
  call griplink:release(3, 0, 1, l_nErrorCode)
else
  // Part is OK -> Move to Use position
  movej()
  call griplink:release(3, 0, 1, l_nErrorCode)
endif
```

4.16 Set Device Value – SETVALUE

This command can be used to set a device-specific value. The value can be selected via an index, which has a valid value range that depends on the device. The value to be written can have different meanings depending on the connected device.



Refer to the driver manual for the respective device for the value ranges of the value index and the value to be written.

Syntax

```
setvalue(num x_nPort, num x_nValueIndex, num x_nValue, num& x_nErrorCode)
```

Parameters passed by value

| | |
|-------------------|---|
| num x_nPort | Port index (0 to 31) |
| num x_nValueIndex | Index of the device value (permissible value range depends on the device) |
| num x_nValue | Value (permissible value range depends on the device) |

Parameters passed by reference

| | |
|-------------------|--|
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |
|-------------------|--|

Example

Set the value with index 2 to 10.5 on port 0 of the device:

```
call griplink:setvalue(0, 2, 10.5, l_nErrorCode)
```

4.17 Waiting for a Device Value – WAITVAL

The WAITVAL command waits until a specific device value reaches the previously set target value or falls within a defined tolerance range. It can be used, for example, to check whether a sensor or position value has reached a target range before the program continues.

The command blocks until one of the following conditions is met:

- The monitored value is within the specified tolerance window, or
- the timeout has been exceeded.



The WAITVAL command is compatible with GRIPLINK protocol version 3. The monitored value corresponds to the index value that can be read via VALUE.



If the value being monitored does not reach the detection window within the specified TIMEOUT interval, a timeout error is triggered and program execution is interrupted.

Syntax

```
waitval(num x_nPort, num x_nIndex, num x_nThreshold, num x_nWindowsize, num  
x_nTimeout, num& x_nErrorCode)
```

Parameters passed by value

| | |
|-------------------|---|
| num x_nPort | Port index (0 to 31) |
| num x_nIndex | Index of the device value (permissible value range depends on the device) |
| num x_nThreshold | Target value in the unit of the measured value |
| num x_nWindowsize | Size of the tolerance window around the target value |
| num x_nTimeout | Timeout in ms |

Parameters passed by reference

| | |
|-------------------|--|
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |
|-------------------|--|

Example

Wait until the distance sensor connected to Port 2 reaches the target value 100.0 at value index 0 within a tolerance window of ± 0.5 . After 5000 ms, the process is terminated due to a timeout.

```
call griplink:waitval(2, 0, 100.0, 0.5, 5000, l_nErrorCode)
```

4.18 Control Force Retention – CLAMP

With the CLAMP command, gripper modules with integrated gripper force safety can de-energize the drive in the HOLDING state to reduce the gripper module's power consumption. The workpiece continues to be held securely.

When CLAMP is activated, the drive switches off in the HOLDING state. When CLAMP is deactivated, gripper force control remains active in the HOLDING state, and the gripper module's controller continuously adjusts the gripper force.



The CLAMP command is not available on all gripping modules.

Syntax

```
clamp(num x_nPort, num x_nEnable, num& x_nErrorCode)
```

Parameters passed by value

| | |
|---------------|----------------------|
| num x_nPort | Port index (0 to 31) |
| num x_nEnable | CLAMP state |

Parameters passed by reference

| | |
|-------------------|--|
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |
|-------------------|--|

Example

Enable clamping on the Weiss Robotics CRG 200-085 at port 1:

```
call griplink:clamp(1, 1, l_nErrorCode)
```

4.19 Controlling the Status LED – LED

This command activates a predefined LED preset for the status LED of a connected device. This allows the operating status or progress of a process to be visually indicated directly on the end-effector.



LED presets can be configured via the GRIPLINK Controller's web interface.



The LED command is not supported by all devices.

Syntax

```
led(num x_nPort, num x_nPreset, num& x_nErrorCode)
```

Parameters passed by value

| | |
|---------------|---|
| num x_nPort | Port index (0 to 31) |
| num x_nPreset | Index of the LED preset (valid range depends on the device) |

Parameters passed by reference

| | |
|-------------------|--|
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B). |
|-------------------|--|

Example

Execute a gripping command with the gripping module at port 2 and read the gripping position. Change the color of the light ring to light pattern 0 if the finger position is greater than or equal to 8.1 mm, and to light pattern 1 if it is less:

```
// Grip with gripper at port 3 and preset 0, wait for state transition
call griplink:grip(2, 0, 1, l_nErrorCode)
```

```
// Read finger position and set respective LED preset
call griplink:value(2, 0, l_nGripperPosition, l_nErrorCode)
```

```
if (l_nGripperPosition >= 8.1)
    call GRIPLINK:led(2, 1, l_nErrorCode)
else
    call GRIPLINK:led(2, 2, l_nErrorCode)
endif
```

4.20 Deprecated Commands

This section lists commands that are still included in the GRIPLINK plugin for backward compatibility reasons, but whose use is no longer recommended.

These commands have been replaced by functionally equivalent but more flexible mechanisms. In particular, the newer commands allow for finer control of program flow and better adaptation to different applications and device types.

Therefore, these deprecated commands should not be used in new programs. Instead, it is recommended to use the corresponding current commands and programming patterns.

4.20.1 Simultaneous Gripping of Workpieces – MGRIP

This command executes a simultaneous gripping command with the selected gripping modules using the selected grip preset index.

The function argument `x_bMwaitforEnabled` can be used to enable waiting for a state transition (MWAITFOR) of the respective devices. As a result, command execution waits until the state of all selected devices has changed. If no state change occurs, the command returns with the error code TIMEOUT.



This command is deprecated and is provided only for compatibility reasons. For new applications, it is recommended to use GRIP with `x_nDoBlock = 0` in combination with WSTR instead.

Syntax

```
mgrip(num x_nPreset, num x_nPorts, bool x_bMwaitforEnabled, num&
x_nErrorCode)
```

Parameters passed by value

| | |
|---------------|---|
| num x_nPreset | Index of the grip preset (valid range of values is device-dependent) |
| num x_nPorts | Selected grip modules as a 32-bit vector:
Bit x = 1: Device at port x is selected
Bit x = 0: Device at port x is not selected |

Example: Select grippers at ports 0 and 3

→ Set bits 0 and 3

→ $\text{Ports} = (1 \ll 0) + (1 \ll 3) = 1 + 8 = 9$

| | |
|-------------------------|--|
| bool x_bMwaitforEnabled | Set to "true" to enable MWAITFOR
Set to "false" to disable MWAITFOR |
|-------------------------|--|

Parameters passed by reference

| | |
|-------------------|---|
| num& x_nErrorCode | Returns the status code:
-1 if the controller response is empty or cannot be parsed; |
|-------------------|---|

0 on success;

1..27 = GRIPLINK controller status codes (see Appendix B).

Example

Gripper modules at ports 0 and 3 grip the workpiece using grip preset 2 and then wait for state changes:

```
// Grip with grippers at ports 0 and 3
call griplink:mrip(2, 9, true, l_nErrorCode)

// Get states of the grippers
call griplink:deviceState(0, l_nDeviceStateGripper0, l_nErrorCode)
call griplink:deviceState(3, l_nDeviceStateGripper3, l_nErrorCode)
...
// Release with grippers at ports 0 and 3
call griplink:mrelease(2, 9, true, l_nErrorCode)

// Get actual states of the grippers
call griplink:deviceState(0, l_nDeviceStateGripper0, l_nErrorCode)
call griplink:deviceState(3, l_nDeviceStateGripper3, l_nErrorCode)
```

4.20.2 Simultaneous Release of Workpieces – MRELEASE

This command executes a simultaneous release command with the selected gripper modules using the selected grip preset index.

The function argument `x_bMwaitforEnabled` can be used to enable waiting for a state transition (MWAITFOR) of the respective devices. As a result, command execution waits until the state of all selected devices has changed. If no state change occurs, the command returns with the error code TIMEOUT.



This command is deprecated and is provided only for compatibility reasons. For new applications, it is recommended to use RELEASE with `x_nDoBlock = 0` in combination with WSTR instead.

Syntax

```
mrelease(num x_nPreset, num x_nPorts, bool x_bMwaitforEnabled, num&
x_nErrorCode)
```

Parameters passed by value

| | |
|---------------|---|
| num x_nPreset | Index of the grip preset (valid range of values is device-dependent) |
| num x_nPorts | Selected grip modules as a 32-bit vector:
Bit x = 1: Device at port x is selected
Bit x = 0: Device at port x is not selected |

Example: Select grippers at ports 0 and 3

→Set bits 0 and 3

→Ports = $(1 \ll 0) + (1 \ll 3) = 1 + 8 = 9$

bool x_bMwaitforEnabled Set to "true" to enable WSTR
Set to "false" to disable WSTR

Parameters passed by reference

num& x_nErrorCode Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B).

Example

See example in 4.20.1

4.20.3 Waiting for Multiple Devices to Change Their State – MWAITFOR

The MWAITFOR command waits for multiple devices to change their state.



If the state change does not occur on at least one selected port within a specified time interval, a timeout error is triggered and program execution is interrupted.



This command is deprecated and is provided only for compatibility reasons. For new applications, it is recommended to use WSTR instead.

Syntax

```
mwaitfor(num x_nPorts, num& x_nErrorCode)
```

Parameters passed by value

num x_nPorts Selected grip modules as a 32-bit vector:
Bit x = 1: Device at port x is selected
Bit x = 0: Device at port x is not selected

Example: Select grippers at ports 0 and 3

→ Set bits 0 and 3

→ Ports = $(1 \ll 0) + (1 \ll 3) = 1 + 8 = 9$

Parameters passed by reference

num& x_nErrorCode Returns the status code:
-1 if the controller response is empty or cannot be parsed;
0 on success;
1..27 = GRIPLINK controller status codes (see Appendix B).

Example

```
// Wait for devices at port 0 and 3 to change their state  
call griplink:mwaitfor(9, l_nErrorCode)
```

5 Troubleshooting

The GRIPLINK plugin outputs error messages during operation. Important messages, possible causes, and solutions are explained below.

5.1 Error Messages

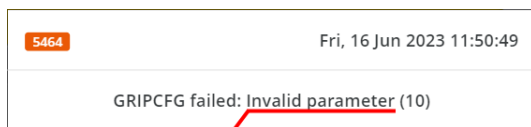
5.1.1 GRIPLINK error response with status code

The GRIPLINK Controller reports failed command executions with an ERR response. The meaning of the individual status codes is described in **Appendix B**.

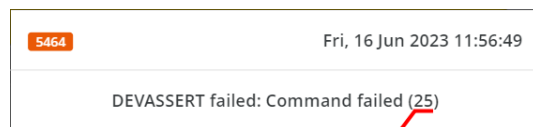
The GRIPLINK plugin evaluates this response and displays it in the logger in the following format:

<Command name> failed: <Error description> (<Status code>)

Messages in the logger:



Error Description



Error Code

Possible causes

- The GRIPLINK Controller rejected the command with an ERR response.
- The command call in the robot program is incorrect.
- A function argument is invalid or lies outside the permissible value range.
- The command is not permitted in the current device state.

Resolution

- Check the status code and error description; see **Appendix B**.
- Check the command call in the VAL3 program.
- Use correct function arguments and observe the permissible value ranges.
- Check the device state, e.g., whether the device is referenced, connected, and ready for operation.

5.1.2 Socket variable not linked to hardware I/O

This message appears if the socket variable used in VAL3 has not been correctly linked to a hardware I/O.

| | |
|--|---------------------------|
| 5404 | Fri, 16 Jun 2023 15:33:53 |
| VAL3 runtime error. IO not linked to a hardware input-output.. VialFilling~. | |

| | |
|--|---------------------------|
| 5404 | Fri, 16 Jun 2023 15:33:53 |
| ERR 123 IO not linked to a hardware input-output.. VialFilling~. | |

Possible cause

- The socket variable is not linked.

Solution

- Link the socket variable correctly (see Section 3.3.2).

5.1.3 Instruction timeout due to missing socket communication

This message appears if successful socket communication with the GRIPLINK Controller cannot be established during a command call or if the response is not received within the expected time.

| | |
|--|---------------------------|
| 5404 | Tue, 05 May 2026 10:37:22 |
| ERR 60 Instruction timeout. GRIPLINK_Dev~. | |

| | |
|---|---------------------------|
| 5404 | Tue, 05 May 2026 10:37:22 |
| VAL3 runtime error. Instruction timeout. GRIPLINK_Dev~. | |

Possible causes

- The GRIPLINK Controller is not reachable via the socket connection.
- The socket connection was not opened successfully.
- The IP address or port is configured incorrectly.
- The GRIPLINK Controller does not respond within the expected time.
- Communication was interrupted during command execution.

Troubleshooting

- Check the Ethernet connection, IP address, and port.
- Check the socket configuration and connection setup in the VAL3 program.
- Check the power supply and operating status of the GRIPLINK Controller.
- Re-establish the connection and restart the program.

5.1.4 Socket communication interrupted during operation

This message appears when a socket connection has already been established between the robot controller and the GRIPLINK Controller, but communication is interrupted during operation.

| | |
|---|---------------------------|
| 5404 | Tue, 05 May 2026 11:18:18 |
| VAL3 runtime error. No answer from input-output..
GRIPLINK_Dev~. | |

| | |
|--|---------------------------|
| 5404 | Tue, 05 May 2026 11:18:18 |
| ERR 125 No answer from input-output.. GRIPLINK_Dev~. | |

Possible cause

- The network connection was interrupted during operation.
- The network cable has been disconnected or is damaged.
- A connector is loose or not properly seated.
- The power supply to the GRIPLINK Controller has been interrupted or is unstable.

Solution

- Check the Ethernet cable and connectors.
- Check the cables for a secure fit, breaks, or damage.
- Check the power supply and ensure that the GRIPLINK Controller is continuously powered.
- After an interruption, re-establish the connection.
- Restart the program.

Appendix A Device status

| Device status | Code | Description |
|-----------------|------|---------------------------|
| NOT CONNECTED | 0 | No device connected |
| NOT INITIALIZED | 1 | Device is not initialized |
| DISABLED | 2 | Idle state |
| RELEASED | 3 | Part released |
| NO PART | 4 | No part found |
| HOLDING | 5 | Holding a part |
| ENABLED | 6 | Enabled state |
| FAULT | 7 | Fault state |

Appendix B Status codes

If an error occurs while executing a GRIPLINK command, a corresponding status code is output. This describes the cause of the error and can be used for diagnostics in the robot program or during commissioning. The following table provides an overview of the possible status codes and their meanings.

| Status code | Identifier | Description |
|-------------|--------------------------|---|
| -1 | - | No response from the controller, or the response could not be processed. Set by the plugin. |
| 0 | E_SUCCESS | No error. Command successfully executed. |
| 1 | E_OVERRUN | Data overrun |
| 2 | E_RANGE_ERROR | Value out of range |
| 3 | E_NOT_AVAILABLE | Function or data not available |
| 4 | E_NOT_INITIALIZED | Device not initialized |
| 5 | E_TIMEOUT | Timeout |
| 6 | E_INSUFFICIENT_RESOURCES | Not enough memory available |
| 7 | E_CHECKSUM_ERROR | Checksum error |
| 8 | E_ACCESS_DENIED | Access denied |
| 9 | E_INVALID_HANDLE | Invalid handle |
| 10 | E_INVALID_PARAMETER | Invalid parameter |
| 11 | E_INDEX_OUT_OF_BOUNDS | Index out of bounds |
| 12 | E_IO_ERROR | Generic I/O error |
| 13 | E_READ_ERROR | Read error |
| 14 | E_WRITE_ERROR | Write error |
| 15 | E_NOT_FOUND | Resource not found |
| 16 | E_NOT_OPEN | File or device not open |
| 17 | E_EXISTS | Resource already exists |
| 18 | E_NO_COMM | Connection error |
| 19 | E_STATE_CONFLICT | Invalid state |

| | | |
|-----------|---------------------|-----------------------------------|
| 20 | E_NOT_SUPPORTED | Command or function not supported |
| 21 | E_INCONSISTENT_DATA | Data inconsistent |
| 22 | E_CMD_SYNTAX | Syntax error |
| 23 | E_CMD_UNKNOWN | Unknown command |
| 24 | E_CMD_ABORTED | Command aborted |
| 25 | E_CMD_FAILED | Command failed |
| 26 | E_AXIS_BLOCKED | Axis is blocked |
| 27 | E_PENDING | Action pending / Not yet finished |

© 2026 WEISS ROBOTICS GmbH & Co. KG. All rights reserved.

GRIPLINK and PERMAGRIP are registered trademarks of WEISS ROBOTICS GmbH & Co. KG. All other trademarks are the property of their respective owners.

The technical data provided in this document is subject to change without notice for the purpose of product improvement. Trademarks are the property of their respective owners. Our products are not intended for use in life-support systems or in systems where a malfunction could result in personal injury.

