SCHUNK

Software manual
2D Grasping-Kit
Smart Grasping Native Protocol V4

Original software manual

Hand in hand for tomorrow

Imprint

Technical changes:
We reserve the right to make alterations for the purpose of technical improvement.

Document number: 1614335

Version: 02.00 | 02/04/2025 | en

Dear Customer,

Thank you for trusting our products and our family-owned company, the leading technology supplier of robots and production machines.

Our team is always available to answer any questions on this product and other solutions. Ask us questions and challenge us. We will find a solution!

Best regards,

Your SCHUNK team

Customer Management
Tel. +49-7133-103-2503
Fax +49-7133-103-2189
cmg@de.schunk.com

Please read the operating manual in full and keep it close to the product.

Table of Contents

1 Simple Protocol General

Low-level pipeline interface for robot controllers.

The simple protocol is a communication protocol sitting on top of the TCP/IP stack. The protocol design is simple and straightforward, and thus feasible to implement even on the most archaic robots and industrial controllers.

1.1 Conventions

- **Port**: The server listens on tcp-port 42001
- **Endianness**: The order of multi-byte fields is big-endian
- **Floating-point numbers**: Single IEEE-Standard (IEC-60559)
- **Signed integers**: Signed integers are represented in two's complement notation
- **Units of measure**: Unless otherwise specified, lengths are measured in meters [m], and angles are measured in radians [rad]

1.2 Data Types

Following data types are define:

- **signed integers**: int8, int16, int32
- **unsigned integers**: uint8, uint16, uint32
- **floating point numbers**: float32

The number suffix of a data type denotes its size in bits (e. g. int16 is two bytes long).

1.3 Versioning

The server is backwards compatible.

Clients using earlier protocol versions will be served as expected and won't even notice the version mismatch. The server always replies with the same version number in the prefix as the one used in the requests.

However, if a server is outdated such that its protocol version is lower than the client's protocol version, the server will always reply with an empty message. The message prefix will have its length-field set to zero and its version-field set to the server's highest supported protocol version. In this case, clients are advised to alert the user that their server is outdated and incompatible with their current client version.

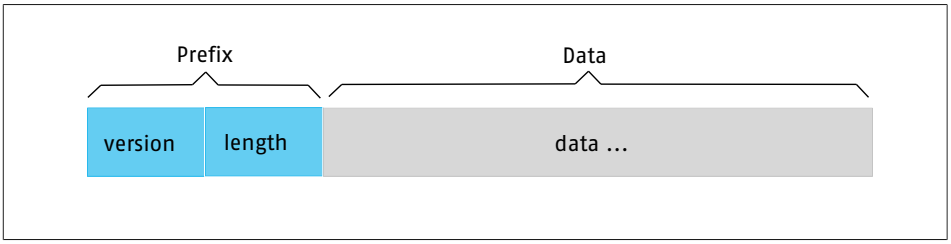
1.4 Layout

Each message frame comprises two parts: an invariant prefix that remains consistent across different protocol versions and a version-specific section. The details of these parts are elaborated in their respective chapters.

2 Simple Protocol Prefix

Invariant part of the low-level pipeline interface for robot controllers.

2.1 Layout



2.2 Prefix

All messages start with a *prefix* defining the protocol version and the size of the remaining message.

The prefix is invariant and does not change between different protocol versions.

	Byte index	Name	Type
Prefix	0	version	uint16
	2	lenght	uint32
Data	6	...	...

- **version**: The protocol version.
- **length**: The size of the remaining message in bytes (max. 4 GB), excluding the *prefix* itself (6B).

2.3 Data

The actual message content. The structure is version-specific and is described in the version chapters.

3 Simple Protocol V4

Version 4 of the low-level pipeline interface for robot controllers.

3.1 General

- Message size: Messages are always 80 bytes long.
- Data types: Only signed and unsigned integers are used; no floats.
- Layout: Fixed frame layout where each byte index is reserved for a specific field and is not reused.

3.2 Layout

Byte index	Data
0	Prefix
6	Header
10 – 79	Body

Requests and responses adhere to a fixed layout with a global mapping of fields to specific byte indices. Each byte index is reserved for a particular fieldand is not reused. Depending on the message type, only certain sections of the body will carry meaningful values, while others are to be disregarded. These disregarded sections are marked as *<offset>* or *<padding>* .

	Request			*Response*		
	Byte index	**Name**	**Type**	**Byte index**	**Name**	**Type**
Prefix	0	version	unit16	0	version	uint16
	2	length	unit32	2	length	uint32
Header	6	comm type	unit8	6	comm type	uint8
	7	reply code	unit8	7	reply code	uint8
	8	reply counter	unit8	8	reply counter	uint8
	9	msg type	unit8	9	msg type	uint8
Body	10	client	unit8	10	version	uint16
	11	grasp mode	unit8	12	state	uint8
	12	object id	uint16	13	grasp mode	uint8
	14	selection criterion	unit8	14	object id	uint16
	15	pose format	unit8	16	instance id	uint16
	16	grasp feedback	unit8	18	pose format	uint8
	17	*<offset>*	-	19	reference frame	uint8
	18	project id	uint16	20	gripper stroke	int32
	20	tool id	uint16	24	object offset	int32[3]

	Request				Response		
	Byte index	Name	Type		Byte index	Name	Type
	22	workspace id	uint16		36	center offset	int32[2]
	24	tool pose	int32[7]		44	grasp pose	int32[7]
	52 – 79	*<padding>*	–		72	candidate count	unit16
					74	object count	unit16
					76 – 79	*<padding>*	–

3.2.1 Prefix

Every message starts with a fixed prefix, containing the protocol version and the length of the remaining message.

	Byte index	Value	Name	Type
Prefix	0	4	version	uint16
	2	74	length	uint32
Data	6	…	…	…

Note: the length field indicates the remaining message size. In this fixed-frame protocol, this length is always 74 bytes.

The fields are detailed in the ▸ 2.1 [🗋 6].

3.2.2 Header

The *header* is the part that encodes the message type and its intent.

	Byte index	Name	Type
Prefix	0	…	…
Header	6	▸ comm type [🗋 8]	uint8
	7	▸ reply code [🗋 9]	uint8
	8	▸ reply counter [🗋 10]	uint8
	9	▸ msg type [🗋 10]	uint8
Body	10 – 79	…	…

3.2.2.1 comm type

Indicates whether the message is a request or a response.

value	description
01	request
02	response

3.2.2.2 reply code

The reply code indicates the result status of the processed request.

Clients should always check the reply code in every single response message and only progress if the reply code is set to SUCCESS. If the reply code is not set to SUCCESS, then the rest of the message shall be ignored. However, the reply counter will still be incremented.

value	name	description
General		
1	SUCCESS	The server has handled the request succesfully.
2	ERROR	The server has encountered an error while processing the request.
Message-specific		
3	NO_OBJECT	No object found.
4	NO_GRASP	No grasp found.
5	INVALID_OBJECT_CLASS	Object id is either invalid or does not exist.
6	CAMERA_NOT_CONNECTED	Camera is not connected.
7	CAMERA_NOT_CALIBRATED	Camera is not calibrated.
8	ROBOT_NOT_CALIBRATED	Robot is not calibrated.
9	WORKSPACE_NOT_CALIBRATED	Workspace is not calibrated.
10	NO_ACTIVE_PROJECT	No project exists or no project is activated.
11	NO_ACTIVE_TOOL	No tool exists or no tool is activated.

NOTE

The reply code is ignored in requests.

3.2.2.3 reply counter

A counter that increments with each reply, resetting to zero when it reaches 256.

NOTE

The reply counter is ignored in requests.

3.2.2.4 msg type

see Chapter ▸ 3.3 [🗋 10]

3.3 Messages

3.3.1 Message Types

Specifies the intent of the message and how the message body is to be interpreted.

value	name	description
General		
1	▸ GET_PROTOCOL_VERSION [🗋 11]	Get the server's protocol version.
2	▸ GET_STATE [🗋 12]	Get the server's current state.
3	▸ REGISTER_CLIENT [🗋 13]	Register the client system to the server.
4	▸ SET_PROJECT [🗋 14]	Set the active project on the server.
5	▸ SET_TOOL [🗋 15]	Set the active tool on the server.
6	▸ SET_WORKSPACE [🗋 16]	Set the active workspace on the server.
Grasping		
16	▸ GET_GRASP [🗋 17]	Get a grasp for the current scene.
17	▸ GRASP_FEEDBACK [🗋 21]	Provide feedback to the server about the last grasp result.
Detection		
32	▸ GET_OBJECT_COUNT [🗋 22]	Get the number of objects in the current scene.
Robot state		
48	▸ TOOL_POSE [🗋 23]	Notify the server about the current tool pose.

3.3.2 Messages

3.3.2.1 GET_PROTOCOL_VERSION

Returns the server's highest supported protocol version. This is useful to see if the client's version is outdated or not. If this is the case, the client is advised to inform the user accordingly.

Request:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
Header	2	length	74	uint32
	6	comm type	1	uint8
	7	reply code	–	uint8
	8	reply counter	–	uint8
	9	msg type	1	uint8
Body	10–79	*<padding>*	–	–

Response:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
Header	2	length	74	uint32
	6	comm type	2	uint8
	7	reply code	x	uint8
	8	reply counter	x	uint8
	9	msg type	1	uint8
Body	10	version	x	uint16
	10–79	*<padding>*	–	–

Version: The server's highest supported protocol version.

3.3.2.2 GET_STATE

Retrieves the server's state.

Request:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
Header	2	length	74	uint32
	6	comm type	1	uint8
	7	reply code	–	uint8
	8	reply counter	–	uint8
	9	msg type	2	uint8
Body	10–79	*<padding>*	–	–

Response:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
Header	2	length	74	uint32
	6	comm type	2	uint8
	7	reply code	x	uint8
	8	reply counter	x	uint8
	9	msg type	1	uint8
Body	10	*<offset>*	–	–
	12	state	x	uint8
	13–79	*<padding>*	–	–

Detail *state*:

value	name	description
1	INIT	The server is initializing.
2	OPERATIONAL	The server is operational and ready to serve requests.
3	STOPPED	The server has stopped operating.
4	ERROR	The server has encountered a critical error.

3.3.2.3 REGISTER_CLIENT

Registers the client system (for informational purposes).

This message should be sent right after the connection has been established. However, there are no restrictions on when to send and how often. If theclient's system is not officially listed (see client table in the request details), then simply ignore this message.

Request:

Contains information about the client's robot system.

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	1	uint8
	7	reply code	–	uint8
	8	reply counter	–	uint8
	9	msg type	3	uint8
Body	10	client	x	uint8
	11 – 79	*<padding>*	–	–

Client:

value	name	vendor
1	UR	Universal Robots
2	KUKA	KUKA
3	Yaskawa	Yaskawa
4	FANUC	FANUC
5	ABB	ABB
6	HORST	fruitcore robotics
128	Siemens PLC	Siemens
129	Allen-Bradley PLC	Allen-Bradley

Response:

The response body is empty.

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	2	uint8
	7	reply code	x	uint8
	8	reply counter	x	uint8
	9	msg type	3	uint8
Body	10–79	*<padding>*	–	–

3.3.2.4 SET_PROJECT

Sets the active project on the server.

Request:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	1	uint8
	7	reply code	–	uint8
	8	reply counter	–	uint8
	9	msg type	4	uint8
Body	10	*<offset>*	–	–
	18	project id	x	uint16
	20 – 79	*<padding>*	–	–

project id: The id of the project to activate.

Response:

The response body is empty. The reply code indicates whether the operation was successful or not.

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	2	uint8
	7	reply code	x	uint8
	8	reply counter	x	uint8
	9	msg type	4	uint8
Body	10–79	*<padding>*	–	–

3.3.2.5 SET_TOOL

Sets the active tool on the server.

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	1	uint8
	7	reply code	–	uint8
	8	reply counter	–	uint8
	9	msg type	5	uint8
Body	10	*<offset>*	–	–
	20	tool id	x	uint16
	22 – 79	*<padding>*	–	–

tool id: The id of the tool to activate.

Response:

DThe response body is empty. The reply code indicates whether the operation was successful or not.

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	2	uint8
	7	reply code	x	uint8
	8	reply counter	x	uint8
	9	msg type	5	uint8
Body	10–79	*<padding>*	–	–

3.3.2.6 SET_WORKSPACE

Sets the active workspace on the server.

Request:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	1	uint8
	7	reply code	–	uint8
	8	reply counter	–	uint8
	9	msg type	6	uint8
Body	10	*<offset>*	–	–
	22	workspace id	x	uint8
	24 – 79	*<padding>*	–	–

workspace id: ID des zu aktivierenden Arbeitsplatzes

Response:

The response body is empty. The reply code indicates whether the operation was successful or not.

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	2	uint8
	7	reply code	x	uint8
	8	reply counter	x	uint8
	9	msg type	6	uint8
Body	10–79	*<padding>*	–	–

3.3.2.7 GET_GRASP

Requests a grasp for the current scene.

Request:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	1	uint8
	7	reply code	-	uint8
	8	reply counter	-	uint8
	9	msg type	16	uint8
Body	10	*<offset>*	-	-
	11	grasp mode	x	uint8
	12	object id	x	uint16
	14	selection criterion	x	uint8
	15	pose format	x	uint8
	16 - 79	*<padding>*	-	-

grasp mode:

value	name	description
1	ACTIVE_GRASP	The returned grasp must match the target object's **active** user-defined grasp definition.
2	ANY_GRASP	The returned grasp must match **any** of the target object's user-defined grasp definitions
3	AUTO_GRASP	The server determines the optimal grasp, prioritizing user-defined grasp definitions when possible.

- *object id*:Specifies the object id of the target object. All objects with the specified object id are considered candidates. If the object id is zero, all non-obstacle objects are considered candidates. The target object is then selected from these candidates based on the specified selectioncriterion.
- *selection criterion*: Defines the method for selecting the target object from the available candidates.

value	name	description
1	RANDOM	Selects a random candidate.
2	CENTRAL	Selects the candidate that is most central in the image. If that candidate is not graspable, the grasp will fail.
3	ISOLATED	Selects the most isolated candidate among all other objects. If that candidate is not graspable, the next mostisolated candidate is selected, and so on.
4	ROWS	Selects candidates row-by-row from left to right. If a candidate is not graspable, the grasp will fail.

pose format: Specifies the pose format to use in the response message. Refer to the ▸ 5 [🗋 25] section to see all supported formats.

Response:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	2	uint8
	7	reply code	x	uint8
	8	reply counter	x	uint8
	9	msg type	16	uint8
Body	10	*<offset>*	-	-
	13	grasp mode	x	uint8
	14	object id	x	uint16
	16	instance id	x	uint16
	18	pose format	x	uint8
	19	reference frame	x	uint8
	20	gripper stroke	x	int32
	24	object offset	x	int32[3]
	36	center offset	x	int32[2]
	44	grasp pose	x	int32[7]
	72	candidate count	x	uint16
	74	object count	x	uint16
	76-79	*<padding>*	-	-

- *grasp mode*: Specifies the grasp mode, see request details for the enumeration values.

Note: This grasp mode is not necessarily the same as the requested grasp mode. For example, if an auto grasp is requested, the server willstill prefer user-defined grasps and only resort to auto grasps if former either does not exist or is not applicable.

- *object id*: The id of the target object.
- *instance id*: The instance id of the target object.
- *pose format*: The pose format used for the grasp pose-field. Refer to the ▸ 5 [🗋 25] section to see all supported formats.
- *reference frame*: Specifies the reference frame of the grasp pose. When the camera is mounted on the robot, this is often the tool or flange frame. Forcameras mounted on an external stand, it is often the base or world frame. The specific reference frame depends on how the robot wascalibrated with the camera.

value	name	description
1	BASE	Base/world frame (camera mounted on external stand)
2	TOOL	Tool/flange frame (camera mounted on robot)

- *gripper stroke*: The distance in micrometers between the gripper fingers to set before approaching the object. This field is relevant only for grippers and canbe ignored for other types of tools.

  Note: It is assumed that the gripper is at position zero when the fingers are touching each other.
- *object offset*: Specifies the (x, y, rz)-pose of the object frame, expressed in the grasp frame. The grasp frame corresponds to the tool frame at the moment the grasp is applied.
  - layout: [x, y, rz]
  - x, y: Position in micrometers.
  - rz: Orientation in microdegrees around the tool's z-axis.

  Use this offset to consistently place the object in the same pose after grasping. An example operation order is as follows:

  1. After grasping, move the tool to a fixed pose
  2. Rotate the tool by -rz around its z-axis
  3. Shift the tool by (-x,-y) relative to its own frame.

4. The object will now be positioned consistently, regardless of the grasp applied.

Important: Ensure that the robot's tool coordinate frame matches the tool coordinate frame defined by the server. The axes of bothcoordinate frames must be aligned.

- *center offset*: Specifies the (x, y)-offset relative to the grasp frame to align the target object's center point with the center of the camera image. The grasp frame corresponds to the tool frame at the moment the grasp is applied. This field is only relevant for setups where the camera is mountedon the robot.

  This adjustment helps minimize perspective distortion, especially with taller objects. You can move the tool (without grasping the object) tothe adjusted location and then request a new grasp with the selection criterion set to CENTRAL to grasp the object that is now positionedcentrally in the camera view.

  To move the tool to the adjusted location, shift the tool by (-x,-y) relative to the grasp frame.

  Important: Ensure that the robot's tool coordinate frame matches the tool coordinate frame defined by the server. The axes of bothcoordinate frames must be aligned.
- *grasp pose*: The grasp pose in the reference frame specified by the reference frame field. The format of the grasp pose is determined by the pose format field.
- *tool id*: The id of the tool being used.
- *candidate count*: The number of candidate objects, including the selected one. For instance, if the count is 1, it means that after applying the grasp, no objectswith the requested object id will remain.
- *object count*: The total number of all detected objects.

3.3.2.8 GRASP_FEEDBACK

Provides optional feedback to the server about the last grasp result.

Request:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	1	uint8
	7	reply code	–	uint8
	8	reply counter	–	uint8
	9	msg type	17	uint8
Body	10	*<offset>*	–	–
	16	grasp feedback	x	uint8
	11 – 79	*<padding>*	–	–

grasp feedback: Feedback about the last grasp result.

value	name	description
1	OK	The grasp was OK.
2	BAD	The grasp was not OK.

Response:

The response body is empty.

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	2	uint8
	7	reply code	x	uint8
	8	reply counter	x	uint8
	9	msg type	6	uint8
Body	10 – 79	*<padding>*	–	–

3.3.2.9 GET_OBJECT_COUNT

Requests the number of objects in the current scene.

Request:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	1	uint8
	7	reply code	–	uint8
	8	reply counter	–	uint8
	9	msg type	32	uint8
Body	10	*<offset>*	–	–
	12	object id	x	uint16
	14 – 79	*<padding>*	–	–

object id: The id of the objects to count. If the value is zero, all objects are counted.

Response:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	2	uint8
	7	reply code	x	uint8
	8	reply counter	x	uint8
	9	msg type	32	uint8
Body	10	*<offset>*	–	–
	72	candidate count	x	uint16
	74	object count	x	uint16
	76–79	*<padding>*	–	–

- *candidate count*: The number of detected objects with the requested id.
- *object count*: The total number of all detected objects.

3.3.2.10 TOOL_POSE

Notifies the server about the current tool pose in the robot's base/world frame.

Request:

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	1	uint8
	7	reply code	–	uint8
	8	reply counter	–	uint8
	9	msg type	48	uint8
Body	10	*<offset>*	–	–
	15	pose format	x	uint8
	16	*<offset>*	–	–
	24	tool pose	x	int32[7]
	52–79	*<padding>*	–	–

- *pose format*: Specifies the format of the tool pose. Refer to the ▶ 5 [🗋 25], section to see all supported formats
- *tool pose*: The tool pose in the robot's base/world frame.

Response:

The response body is empty.

	Byte index	Name	Value	Type
Prefix	0	version	4	uint16
	2	length	74	uint32
Header	6	comm type	2	uint8
	7	reply code	x	uint8
	8	reply counter	x	uint8
	9	msg type	48	uint8
Body	10–79	*<padding>*	–	–

4 Visualization

A visualization of the latest detection result is provided as an image resource that can be accessed via HTTP at:

`http://<Server-IP>/monitor/latest_result`

For older clients that cannot handle many pixels, a low-resolution version of the visualization is also available at:

`http://<server-ip>/monitor/latest_result_compressed`

5 Pose Formats

Following pose formats are supported:

value	name	type	layout	description
General formats				
1	Quaternion	int32[7]	position: x, y, z; orientation: qx, qy, qz, qw	Position and unnormalized quaternion. *Note: To normalize the quaternion, divide it by 10^6.* *Units*: Micrometer and radian.
2	Axis-Angle	int32[7]	position: x, y, z; orientation: x, y, z; –	Position and unnormalized rotation vector. *Units*: Micrometer and microradian. *Vendor*: Universal Robots
Vendor-specific formats				
16	WPR	int32[7]	position: x, y, z; orientation: W, P, R; –	Position and orientation WPR with W rotating around the fixed x-axis, then P rotating around the fixed y-axis, then R rotating around the fixed z-axis. *Rotation convention*: x-y-z (extrinsic). *Units*: Micrometer and microdegree. *Vendors*: Fanuc, Yaskawa
17	ABC	int32[7]	position: x, y, z; orientation: A, B, C; –	Position and orientation ABC with A rotating around the z-axis, then B rotating around the y'-axis, then C rotating around the x''-axis. *Rotation convention*: z-y'-x'' (intrinsic). *Units*: Micrometer and microdegree. *Vendor*: Kuka

6 Examples

Lets assume we've set up a TCP connection to the server successfully.

For simple grasping tasks, the most basic sequence of messages is:

1. ▸ GET_PROTOCOL_VERSION [🗋 26]
2. ▸ GET_STATE [🗋 27]
3. ▸ REGISTER_CLIEN [🗋 28]T
4. ▸ SET_PROJECT [🗋 29]
5. ▸ GET_GRASP [🗋 30]

6.1 GET_PROTOCOL_VERSION

The first thing we want to do after setting up a connection is to check the server's highest supported protocol version.

	Request				*Response*			
	Index	**Name**	**Value**	**Typ**	**Index**	**Name**	**Value**	**Typ**
Prefix	0	Version	4	unit16	0	Version	4	uint16
	2	Length	74	unit32	2	length	74	uint32
Header	6	comm type	1	unit8	6	comm type	2	uint8
	7	reply code	0	unit8	7	reply code	1	uint8
	8	reply counter	0	unit8	8	reply counter	0	uint8
Body	9	msg type	1	unit8	9	msg type	1	uint8
	10 – 79	*<padding>*	0	–	10	Version	4	uint16
					12 – 79	*<padding>*	0	–

Data

```
000 004 000 000 000 074 001 000
000 001 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
```

Data

```
000 004 000 000 000 074 002 001
000 001 000 004 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
```

In this example, the server answers with version = 4. This means that our client is up-to-date.

6.2 GET_STATE

Before sending further requests to the server, the server's state is to check.

	Request				*Response*			
	Index	**Name**	**Value**	**Type**	**Index**	**Name**	**Value**	**Type**
Prefix	0	version	4	unit16	0	version	4	uint16
	2	length	74	unit32	2	length	74	uint32
Header	6	comm type	1	unit8	6	comm type	2	uint8
	7	reply code	0	unit8	7	reply code	1	uint8
	8	reply counter	0	unit8	8	reply counter	1	uint8
Body	9	msg type	2	unit8	9	msg type	1	uint8
	10 – 79	*<padding>*	0	–	10	*<offset>*	0	–
					12	state	2	
					13 – 79	*<padding>*	0	–

Data

```
000 004 000 000 000 074 001 000
000 002 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
```

Daten

```
000 004 000 000 000 074 002 001
001 002 000 002 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
```

In this example, the server answers with state = 2. This means that the server is operational and proceeding is possible.

6.3 REGISTER_CLIENT

Register the client system, which is a Siemens PLC in this example.

	Request				*Response*			
	Index	**Name**	**Value**	**Type**	**Index**	**Name**	**Value**	**Type**
Prefix	0	version	4	unit16	0	version	3	uint16
	2	length	74	unit32	2	length	74	uint32
Header	6	comm type	1	unit8	6	comm type	2	uint8
	7	reply code	0	unit8	7	reply code	1	uint8
	8	reply counter	0	unit8	8	reply counter	2	uint8
Body	9	msg type	3	unit8	9	msg type	3	uint8
	10	client	128	unit8	10 – 79	*<padding>*	0	–
	11 – 79	*<padding>*	0	–				

Data

```
000 004 000 000 000 074 001 000
000 003 128 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
```

Daten

```
000 004 000 000 000 074 002 001
002 003 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
```

The response body is empty.

6.4 SET_PROJECT

We activate the project with index 5, which contains the objects we want to detect and grasp.

	Request				*Response*			
	Index	**Name**	**Value**	**Type**	**Index**	**Name**	**Value**	**Type**
Prefix	0	version	4	unit16	0	version	4	uint16
	2	length	74	unit32	2	length	74	uint32
Header	6	comm type	1	unit8	6	comm type	2	uint8
	7	reply code	0	unit8	7	reply code	1	uint8
	8	reply counter	0	unit8	8	reply counter	3	uint8
Body	9	msg type	4	unit8	9	msg type	4	uint8
	10	*<offset>*	0	-	10 – 79	*<padding>*	0	-
	18	project id	5	unit16				
	20 – 79	*<padding>*	0	-				

Data

```
000 004 000 000 000 074 001 000
000 004 000 000 000 000 000 000
000 000 000 005 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 005 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
```

Data

```
000 004 000 000 000 074 002 001
003 004 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
```

In this example activating the project with index 5 was successful.

The response body is empty.

Note: Check the reply code in the response header and proceed only if the reply code is 1 (indicating success).

6.5 GET_GRASP

We request an auto grasp for the object located most centrally. The grasp pose shall be returned in quaternion representation.

	Request				*Response*			
	Index	**Name**	**Value**	**Type**	**Index**	**Name**	**Value**	**Type**
Prefix	0	version	4	unit16	0	version	4	uint16
	2	length	74	unit32	2	length	74	uint32
Header	6	comm type	1	unit8	6	comm type	2	uint8
	7	reply code	0	unit8	7	reply code	1	uint8
	8	reply counter	0	unit8	8	reply counter	4	uint8
Body	9	msg type	16	unit8	9	msg type	16	uint8
	10	*<offset>*	0	-	10	*<offset>*	0	-
	11	grasp mode	3	unit8	13	grasp mode	2	uint8
	12	object id	0	unit16	14	object id	3	uint16
	14	selection criterion	2	unit8	16	instance id	0	uint8
	15	pose format	1	unit8	18	pose format	1	uint8
	16 - 79	*<padding>*	0	-	19	reference frame	2	uint8
					20	gripper stroke	86000	int32
					24	object offset	[16112, 8102, -25210142]	int32[3]
					36	center offset	[-123911, -225418, 68]	int32[2]
					44	grasp pose	[853798, 1111998, 502790, 775990, 630744, 0, 0]	int32[7]
					72	candidate count	2	uint16
					74	object count	4	uint16
					76 - 79	*<padding>*	0	-

Request				Response			
Index	Name	Value	Type	Index	Name	Value	Type
Data				Data			

```
000 004 000 000 000 074 001 000
000 016 000 003 000 000 002 001
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000
```

```
000 004 000 000 000 074 002 001
004 016 000 000 000 002 000 003
000 000 001 002 000 001 079 240
000 000 062 240 000 000 031 166
254 127 082 226 255 254 027 249
255 252 143 118 000 013 007 038
000 016 247 190 000 007 172 006
000 011 215 054 000 009 159 216
000 000 000 000 000 000 000 000
000 002 000 004 000 000 000 000
```

In this example, the returned grasp is a user-defined grasp (grasp mode = 2) and the object id of the selected central object is 3. Even though we'verequested an auto grasp, the system always prefers user-defined grasps and only resorts to auto grasps if former either does not exist or is notapplicable.

In the response, we can also see that the candidate count is 2, which means that after applying the grasp there will be one candidate object left in thescene.

Note: Requesting a grasp may fail for various reasons, such as an uncalibrated camera, inability to locate the object, or an impractical grasp. It is crucial to verify the reply code in the response header and proceed only if the reply code is 1 (indicating success).

Steps 1 to 4 are performed only once. Step 5 typically iterates in a loop.

SCHUNK SE & Co. KG
Spanntechnik | Greiftechnik | Automatisierungstechnik

Bahnhofstr. 106 – 134
D-74348 Lauffen/Neckar
Tel. +49-7133-103-0
info@de.schunk.com
schunk.com

Folgen Sie uns | *Follow us*

Wir drucken nachhaltig | *We print sustainable*

04-2025 02.00 | 2D Grasping-Kit | Software manual | en | 1614335