ВАЛДАЙ РОБОТЫ

ООО «ВАЛДАЙ РОБОТЫ»
ИНН: 9717138305 КПП 771701001
Тел: +7 909 155 58 89
Email: general.secretary@vald.ai
Сайт: www.vald.ai

СКАЛА-П

Руководство пользователя

Оглавление

1. Аннотация

Документ содержит указания по работе с программным обеспечением «СКАЛА-П» (далее – программное обеспечение, ПО).

Важно: Для взаимодействия с ПО настоятельно рекомендуется использовать специализированное графическое ПО "Скала-Ю".

2. Подготовка к запуску ПО

Примечание: при работе с реальным промышленным роботом описываемое ПО запускается автоматически. Ручной запуск может быть необходим при работе с виртуальными моделями. Перед запуском программы необходимо запустить NATS-сервер на той же машине, на которой будет происходить запуск программы.

Для ручного взаимодействия с API на рабочей машине должен быть установлен NATS. Для удобства можно использовать Docker:

```
docker run --network=host -it synadia/nats-box:latest
```

Дальнейшее взаимодействие с ПО производится вызовом команд:

```
./nats pub robot.command # для публикации команд
./nats sub robot.command # для наблюдения за присылаемыми от ПО данными
```

3. Запуск ПО

Примечание: при работе с реальным промышленным роботом описываемое ПО запускается автоматически. Ручной запуск может быть необходим при работе с виртуальными моделями.
Запуск программы происходит вызовом исполняемого файла **robot-controller**. Программа запускается обязательно с правами суперпользователя.

Пример:

```
sudo ./robot-controller
```

4. Предоставляемое NATS API

Управляющие команды роботу могут отправляться посредством NATS-сервера, также с помощью него можно отслеживать текущее состояние робота. Все сообщения передаются в формате JSON.

Управляющие программы публикуются на robot.command.

Данные о результате выполнения команды публикуются на robot.data или robot.err. Также на robot.io публикуются изменения созданных портов ввода-вывода.
Любая команда обязательно состоит из трех атрибутов:

- type
- function
- arguments
- type - обозначает тип команды, к какой области работы робота она относится.

Предоставляются следующие типы команд:

- mov - любые интерполяции робота.
- iParam - настройка параметров интерполяции.
- actualData - чтение актуальных данных робота.
- storage - хранилище систем координат и инструментов.
- ecat - EtherCAT сеть (двигатели робота).
- mtcp - ModbusTCP сеть (Входы/Выходы).
- interpreter - интерпретатор программ робота.
- state - сброс ошибок и переключение состояний (тормоза).
- cyclePub - управление циклической публикацией актуальных данных.
- settings - конфигурация робота

function - непосредственно команда - то, что должен выполнить робот.

arguments - аргументы команды, необходимые или опциональные для работы команды.

5. Подготовка к запуску робота

Примечание: Актуально только при работе с реальным роботом. При работе с виртуальной моделью применение указанных команд не требуется.

После запуска контроллера мы готовы начать работу с роботом. Подготовка к запуску состоит из нескольких этапов:

- Подключение EtherCAT сети
- Подключение периферийных устройств
- Снятие робота с тормоза

5.1. Подключение EtherCAT сети

Подключение EtherCAT сети производится командой типа "ecat" с функцией "connect". После успешного подключения на robot.data будет опубликовано сообщение "ECAT connected."

5.2. Подключение периферийных устройств

При наличии зарегистрированных ModbusTCP устройств необходимо для каждого из них вызвать команду типа "mtcp" с функцией "connect". После успешного подключения на robot.data будет опубликовано сообщение "Connected."

5.3. Снятие робота с тормоза

Движение робота без снятия с тормоза невозможно ни в одном типе перемещения. Чтобы снять робота с торомоза необходимо вызвать команду типа "state" с функцией "set" и аргументом "state": "brakes off". После успешного снятия робота с торомоза на robot.data будет опубликовано сообщение "Brakes OFF.". После снятия с тормоза робот готов к перемещениям.

6. Юстировка робота

Для корректной работы робота необходимо провести юстировку осей робота, чтобы понять смещение нуля моторов относительно нуля оси.

Важно: Без корректной юстировки любые перемещения, кроме осевых будут происходить некорреткно.

Для проведения юстировки необходимо последовательно привести каждую ось, с помощью ручного режима перемещения в осевом режиме, в нулевое пололжение и вызвать команду "zero axis" для нужного номера оси. Нумерация осей идет с 0, поэтому первая ось - 0, вторая - 1 и т.д.

После того, как все оси отъюстированы разрешено использование других режимов перемещения.

7. Запоминание точек

При написании программ, создании систем координат или калибровке инструмента необходимо передавать роботу наборы точек. Одним из удобных способов получения этих точек является запоминание точек с робота. Чтобы запомнить точку необходимо вызвать

команду типа "actualData" с функцией "read" и аргументом "parameter": "point". В результате выполнения команды на robot.data будет опубликовано сообщение с атрибутом "value", который будет равен актуальной точке ЦТИ в актуальной системе координат.

Пример публикации сообщения на robot.data:

```
{
"type": "point",
"value": {
"a": 0,
"b": 3.14,
"c": 0,
"x": 800,
"y": 0,
"z": 500,
"k": 0
}
}
```

Значение "value" является точкой в правильном формате для сохранения. Рекомендуется сохранять точки программы как отдельные json объекты со своим собственным названием, а точки для создания систем координат или калибровки инструмента в массив объектов json без названий.

Пример текста файла для сохранения точек для программы:

```
{
    "Заготовка1-низ": {
        "a": 0,
        "b": 3.14,
        "c": 0,
        "x": 800,
        "y": 0,
        "z": 500,
        "k": 0
    },
    "Заготовка1-верх": {
        "a": 0,
```

```
        "b": 3.14,
        "c": 0,
        "x": 800,
        "y": 0,
        "z": 700,
        "k": 0
    },
    "конвейер1": {
        "a": 0,
        "b": 3.14,
        "c": 0,
        "x": 800,
        "y": 800,
        "z": 500,
        "k": 0
    },
    "круконвейер2": {
        "a": 0,
        "b": 3.14,
        "c": 0,
        "x": 800,
        "y": -800,
        "z": 500,
        "k": 0
    }
}
```

Пример текста файла для сохранения точек для создания системы координат:


```
[
    {
        "a": 0,
        "b": 1.57,
        "c": 0,
        "x": 800,
        "y": 0,
        "z": 800,
        "k": 0
    },
    {
        "a": 0,
        "b": 3.14,
        "c": 0,
        "x": 1000,
        "y": 0,
        "z": 800,
        "k": 0
    },
    {
        "a": -1.57,
        "b": 1.57,
        "c": 0,
        "x": 1000,
        "y": 150,
        "z": 800,
        "k": 0
    }
]
```

8. Создание системы координат

Для создания системы координат необходимо сохранить массив из трех точек в пространстве. Первая точка является точкой начала новой системы координат. Вторая точка лежит строго на оси X новой системы координат. Рекомендуется снимать вторую точку как можно дальше от первой, таким образом достигается большая точность. Третья точка лежит строго на плоскости, образованной осями X и Y новой системы координат.

Полученный массив из трех точек необходимо передать аргументом "value" в команде типа "storage" с функцией "create". Также обязательно в аргументах необходимо указать тип создаваемой сущности "parameter": "coordSystem" и имя новой системы координат "name": "имяВашейСистемы". Пример команды есть в описании команды в разделе NATS API.

9. Создание инструмента

Для создания инструмента необходимо сохранить массив из четырех или шести точек в пространстве. Первые 4 точки снимаются таким образом, что ЦТИ, калибруемого инструмента должна оставаться неподвижной в пространстве, но с изменяющейся ориентацией. При необходимости калибровки только смещения точки ЦТИ с сохранением ориентации фланца - достаточно четырех точек. Если необходимо также откалибровать и ориентацию инструмента необходимо запомнить ещё 2 точки. Пятая точка является точкой на оси X инструмента относительно ориентации четвертой точки. Шестая точка является точкой на плоскости XY относительно ориентации четвертой точки. Вследствие требований к пятой и шестой точек рекомендуется снимать четвертую точку таким образом, чтобы ось X инструмента была сонаправлена с -Z актуальной системы координат. Таким образом достигается максимальное удобство при снятии пятой и шестой точек.

Полученный массив из четырех или шести точек необходимо передать аргументом "value" в команде типа "storage" с функцией "create". Также обязательно в аргументах необходимо указать тип создаваемой сущности "parameter": "tool" и имя нового инструмента "name": "имяВашегоИнструмента". Пример команды есть в описании команды в разделе NATS API.

10. Перечень предоставляемых команд через NATS API

Type <mov>

Команды типа mov отвечают за перемещения робота. Предоставляются следующие виды перемещения: линейное, круговое, из точки в точку и ручное.

lin

Команда линейного перемещения осуществляет интерполяцию ЦТИ (центральной точки инструмента) по линейной траектории, точка начала которой находится в актуальной точке

положения ЦТИ на момент начала перемещения, а конечная точка задается в аргументах команды.

Пример команды:

```
{
    "arguments": {
        "endPoint": {
            "a": 0,
            "b": 3.14,
            "c": 0,
            "x": 1600,
            "y": 0,
            "z": 500
        }
    },
    "function": "lin",
    "type": "mov"
}
```

circ

Команда кругового перемещения осуществляет интерполяцию ЦТИ по круговой траектории, точка начала которой находится в актуальной точке положения ЦТИ на момент начала перемещения, а конечная точка задается в аргументах команды. Радиус кругового перемещения рассчитывается автоматически на основании переданной в аргументах промежуточной точки на траектории. Также, при необходимости, есть возможность выбрать альтернативную траекторию на дуге окружности, построенной по трем точкам (начальная, промежуточная и конечная), проходя не через промежуточную точку.

Пример команды:

```
{
    "arguments": {
        "endPoint": {
            "a": 0,
            "b": 3.14,
            "c": 0,
            "x": 1600,
            "y": 0,
            "z": 500
```

```
        },
        "midPoint": {
            "a": 0,
            "b": 3.14,
            "c": 0,
            "x": 1200,
            "y": 400,
            "z": 500
        },
        "altArc": false
    },
    "function": "circ",
    "type": "mov"
}
```


ptp

Команда перемешения из точки в точку осуществляет интерполяцию осей робота таким образом, что каждая ось робота совершает вращение только в одну сторону в рамках одного перемещения. При этом траектория ЦТИ может иметь произвольных характер и гарантирована только её конечная позиция, которая передается в аргументах команды.

Пример команды:


```
{
    "arguments": {
        "endPoint": {
            "a": 0,
            "b": 3.14,
            "c": 0,
            "x": 1600,
            "y": 0,
            "z": 500
        }
    },
    "function": "ptp",
    "type": "mov"
}
```

hand

Команда ручного перемещения осуществляет ручное управление перемещением робота в реальном времени. Есть 3 режима ручного перемещения: координатный (coord), инструментальный (tool), осевой (ang).

Координатный режим позволяет выполнять перемещения относительно актуальной системы координат.

Инструментальный режим позволяет выполнять перемещения относительно ориентации актульного инструмента.

Осевеой режим позволяет выполнять вращение каждой оси робота по отдельности. В аргументах команды передается выбранный режим, а также 16 битный регистр с данными о

желаемом направлении движения. Чтобы осуществлять непрерывное движение команду ручного перемещения нужно отправлять не реже, чем 15мс.

Таблица с определением значения каждого бита регистра:

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
				-C	+C	-B	+B	-A	+A	-Z	+Z	-Y	+Y	-X	+X
				-6	+6	-5	+5	-4	+4	-3	+3	-2	+2	-1	+1

Пример команды:

```
{
    "arguments": {
        "type": "coord",
        "buttons16bit": 5
    },
    "function": "hand",
    "type": "mov"
}
```

Type <iParam>

Команды типа iParam отвечают за управление параметрами перемещения робота.

Чтобы получить доступ к параметрам предоставляются две основные функции: read и write – с помощью них можно читать или записывать указанный параметр.

Предоставляются следующие параметры: userSpeed, tool, coordSystem.

userSpeed

Скорость работы робота в долях (0...1). Является регулировкой всех типов скоростей и ускорений робота.

Пример команды:

```
{
    "arguments": {
        "parameter": "userSpeed",
        "value": 0.1
    },
    "function": "write",
    "type": "iParam"
}
```

tool

Актуальный инструмент робота. Чтобы выбрать инструмент - он должен быть предварительно создан.

Пример команды:

```
{
    "arguments": {
        "parameter": "tool",
        "value": "захват"
    },
    "function": "write",
    "type": "iParam"
}
```

coordSystem

Актуальная система координат робота. Чтобы выбрать систему координат - она должна быть предварительно создана.

Пример команды:

```
{
    "arguments": {
        "parameter": "coordSystem",
        "value": "стол"
    },
    "function": "write",
    "type": "iParam"
```

```
}
```

Type <actualData>

Команды типа actualData отвечают за получение данных робота.

Чтобы получить доступ к данным робота предоставляется функция read, с помощью которой можно читать указанные данные.

Предоставляются следующие данные: status, angles, point, tool, coordSystem.

status

Статус робота из следующего списка: waiting, interpolating, hand interpolation, program.

waiting - робот ожидает следующей команды

interpolating - робот выполняет команду интерполяции

hand interpolation - робот выполняет команду ручной интерполяции

program - робот исполняет программу

Пример команды:

```
{
    "arguments": {
        "parameter": "status"
    },
    "function": "read",
    "type": "actualData"
}
```

angles

Актуальные углы поворота осей робота в радианах.

Пример команды:

```
{
    "arguments": {
        "parameter": "angles"
        },
    "function": "read",
    "type": "actualData"
}
```

point

Актуальные координаты ЦТИ относительно актуальной системы координат.

Пример команды:

```
{
    "arguments": {
        "parameter": "point"
    },
    "function": "read",
    "type": "actualData"
}
```

tool

Название актуального инструмента.

Пример команды:

```
{
    "arguments": {
        "parameter": "tool"
    },
    "function": "read",
    "type": "actualData"
}
```

coordSystem

Название актуальной системы координат.

Пример команды:

```
{
    "arguments": {
        "parameter": "coordSystem"
    },
    "function": "read",
    "type": "actualData"
}
```

Type <storage>

Команды типа storage отвечают за управление жизненным циклом сущностей – систем координат и инструментов.

Для обеспечения жизненного цикла предоставляются следующие функции:

- write
- read
- erase
- create
- rename
- list

write

Создает новую сущность с заданным названием и указанного типа по матрице трансформации. Для систем координат матрица трансформации должна описывать перенос и ориентацию системы координат относительно системы координат робота. Для инструментов матрица трансформации описывает перенос и ориентацию ЦТИ относительно фланца робота.

Пример команды:

```
{
    "arguments": {
        "parameter": "tool",
        "name": "захват",
        "value": [
            [0.0, 0.0, -1.0, 0.0],
            [0.0, 1.0, 0.0, 0.0],
            [1.0, 0.0, 0.0, 100.0],
            [0.0, 0.0, 0.0, 1.0]
        ]
    },
    "function": "write",
    "type": "storage"
}
```

read

Матрица трансформации сущности заданного типа и названия.

Пример команды:

```
{
    "arguments": {
        "parameter": "tool",
        "name": "захват"
    },
```

```
    "function": "read",
    "type": "storage"
}
```

erase

Удалить сущность по типу и названию.

Пример команды:

```
{
    "arguments": {
        "parameter": "tool",
        "name": "myTool1"
    },
    "function": "erase",
    "type": "storage"
}
```

create

Создает новую сущность с заданным названием и указанного типа по точкам в пространстве. Для системы координат достаточно трех точек. Для инструмента есть два варианта создания: по четырем и по шести точкам. При создании по четырем - будет учтено только смещение ЦТИ относительно фланца, при создании по шести - будет учтена также и ориентация.

Пример команды:

```
{
    "arguments": {
        "parameter": "coordSystem",
        "name": "стол",
        "value": [
            {
                "x": 800,
                "y": 0,
                "z": 800,
                "a": 0,
                "b": 1.57,
                "c": 0
            },
            {
```

```
                "x": 1000,
                "y": 0,
                "z": 800,
                "a": 0,
                "b": 3.14,
                "c": 0
            },
            {
                "x": 1000,
                "y": 150,
                "z": 800,
                "a": -1.57,
                "b": 1.57,
                "c": 0
            }
        ]
    },
    "function": "create",
    "type": "storage"
}
```


rename

Переименоние сущности заданного типа и заданного названия.

Пример команды:


```
{
    "arguments": {
        "parameter": "tool",
        "name": "захват",
        "value": "горелка"
    },
    "function": "rename",
    "type": "storage"
}
```


list

Список всех сущностей заданного типа

Пример команды:

```
{
    "arguments": {
        "parameter": "tool"
    },
    "function": "list",
    "type": "storage"
}
```

Type <mtcp>

Команды типа mtcp отвечают за управление ModbusTCP устройствами, подключенными к контроллеру. Контроллер выступает master устройством в таком режиме.

Каждое новое подключенное устройство является объектом сущности device. Каждый новых или дублирующий регистр этого устройства является объектом сущности port.

Каждое название устройства и порта должно быть уникальным. Для обеспечения управления ModbusTCP устройствами предоставляются следующие функции:

- createDevice
- deleteDevice
- setDeviceIP
- connect
- disconnet
- createPort
- deletePort
- read
- write

createDevice

Создает новое устройство с указанным именем, ip-адресом, портом и протоколом.

Стандартными значениями являются: порт - 502, протокол - modbusTCP.

Пример команды:

```
{
    "arguments": {
        "deviceName": "ПР103",
        "ipAddress": "192.168.1.99",
        "netPort": 502,
```

```
        "protocol": "modbusTCP"
    },
    "function": "createDevice",
    "type": "mtcp"
}
```

deleteDevice

Удаляет существующее устройство с указанным именем.

Пример команды:

```
{
    "arguments": {
        "deviceName": "ПР103"
    },
    "function": "deleteDevice",
    "type": "mtcp"
}
```

setDeviceIP

Изменяет ip-адрес и порт уже существующего устройства.

Пример команды:

```
{
    "arguments": {
        "deviceName": "ПР103",
        "ipAddress": "192.168.2.88",
        "netPort": 502
    },
    "function": "setDeviceIP",
    "type": "mtcp"
}
```

connect

Устанавливает подключение к устройству с указанным названием.

Пример команды:

```
{
    "arguments": {
        "deviceName": "ПР103"
```

```
    },
    "function": "connect",
    "type": "mtcp"
}
```


disconnect

Прерывает соединение с устройством с указанным названием.

Пример команды:


```
{
    "arguments": {
        "deviceName": "ПР103"
    },
    "function": "disconnect",
    "type": "mtcp"
}
```


createPort

Создает новый порт для существующего устройства. Порт может быть 6 разных типов:

- DI (0) - дискретный вход
- DO (1) - дискертный выход
- AI (2) - аналоговый вход
- AO (3) - аналоговый выход
- GI (4)- групповой вход
- GO (5) - групповой выход

Разные типы портов хранят разные типы данных:

- дискретные - bool
- аналоговые - float32 (занимает 2 регистра)
- групповые - uint16

Пример команды:


```
{
    "arguments": {
        "deviceName": "ПР103",
        "portName": "Двых4",
        "portType": 1,
        "regAddress": 470,
        "bit": 3
```

```
    },
    "function": "createPort",
    "type": "mtcp"
}
```

deletePort

Удаляет существующий порт с указанным названием.

Пример команды:

```
{
    "arguments": {
        "portName": "Двых4"
    },
    "function": "deletePort",
    "type": "mtcp"
}
```

read

Осуществляет чтение порта с указанным названием.

Пример команды:

```
{
    "arguments": {
        "portName": "Двых4"
    },
    "function": "read",
    "type": "mtcp"
}
```

write

Осуществляет запись порта с указанным названием.

Пример команды:

```
{
    "arguments": {
        "portName": "Двых4",
        "value": true
    },
    "function": "write",
```

```
    "type": "mtcp"
}
```

Type <interpreter>

Команды типа interpreter отвечают за управление программами на роботе. Каждая программа состоит из двух файлов: текст программы и набор точек программы.

Для обеспечения работы программ на роботе предоставлены следующие функции:

- setActualProgramName
- getActualProgramName
- upload
- download
- execute
- stop
- list

setActualProgramName

Выбирает для исполнения программу с указанным названием. Программа уже должна быть загружена на контроллер.

Пример команды:

```
{
    "arguments": {
        "programName": "Программа1"
    },
    "function": "setActualProgramName",
    "type": "interpreter"
}
```

getActualProgramName

Имя выбранной программы для исполнения.

Пример команды:

```
{
    "arguments": {},
    "function": "getActualProgramName",
    "type": "interpreter"
}
```

upload

Загружает программу с указанным названием, текстом и точками программы в контроллер робота.

Пример команды:


```
{
    "arguments": {
        "points": {
            "вск1": {
                "a": 0,
                "b": 3.14,
                "c": 0,
                "x": 0,
                "y": 0,
                "z": 0
            },
            "вск2": {
                "a": 0,
                "b": 3.14,
                "c": 0,
                "x": 0,
                "y": 0,
                "z": 0
            },
            "вторая": {
                "y": -1600
            },
            "круг1": {
                "x": 300,
                "y": 0,
                "z": 0
            },
            "круг2": {
                "x": 150,
                "y": 300,
                "z": 0
```

```
            },
            "первая": {
                "b": -3.14,
                "x": 800,
                "y": 0
            }
        },
        "programName": "Программа1",
        "programText": "начало\n твт(т[первая], 100%, ск[default], инстр[def
    },
    "function": "upload",
    "type": "interpreter"
}
```


download

Выгружает программу с указанным названием с контроллера робота.

Пример команды:


```
{
    "arguments": {
        "programName": "Программа1"
    },
    "function": "download",
    "type": "interpreter"
}
```


execute

Запускает исполнение актуальной программы.

Пример команды:


```
{
    "arguments": {},
    "function": "execute",
    "type": "interpreter"
}
```


stop

Прерывает исполнение программы.

Пример команды:

```
{
    "arguments": {},
    "function": "stop",
    "type": "interpreter"
}
```

list

Список имен всех программ, загруженных на контроллер робота.

Пример команды:

```
{
    "arguments": {},
    "function": "list",
    "type": "interpreter"
}
```

delete

Удаление программы с контроллера робота

Пример команды:

```
{
    "arguments": {
        "programName": "Программа1"
    },
    "function": "delete",
    "type": "interpreter"
}
```

Type <state>

Команды типа state осуществляют управление состоянием, режимами работы робота и сбросом ошибок.

Режимы работы: P1 (ручной безопасный), P2 (ручной тестировочный) и A (автоматический).

Состояния: на тормозах, снят с тормозов.

Предоставляются следующие функции:

- resetError
- set

- get
- set mode
- get mode

resetError

Сбрасывает ошибку робота и его осей, если она присутствует.

Пример команды:

```
{
    "arguments": {},
    "function": "resetError",
    "type": "state"
}
```

set

Устанавливает выбранное состояние робота. (brakes off/brakes on)

Пример команды:

```
{
    "arguments": {
        "state": "brakes off"
    },
    "function": "set",
    "type": "state"
}
```

get

Актуальное состояние робота.

Пример команды:

```
{
    "arguments": {},
    "function": "get",
    "type": "state"
}
```

set mode

Устанавливает выбранный режим работы. (hand1/hand2/auto)

Пример команды:

```
{
```

```
    "arguments": {
        "mode": "auto"
    },
    "function": "set mode",
    "type": "state"
}
```

get mode

Актуальный режим работы.

Пример команды:

```
{
    "arguments": {
    },
    "function": "get mode",
    "type": "state"
}
```

Type <cyclePub>

Команды типа cyclePub осуществляют управление публикацией телеметрии робота на robot.cyclicData.

Предоставляются следующие функции:

- set
- get
- enable
- disable
- set

Устанавливает, какие данные нужно добавлять в телеметрию, а какие нет. В телеметрию могут быть добавлены любые данные из раздела actualData.

Пример команды:

```
{
    "arguments": {
        "angles": true,
        "point": true,
        "status": true,
        "tool": false,
```

```
        "coordSystem": false
    },
    "function": "set",
    "type": "cyclePub"
}
```

get

Показывает, какие данные добавлены в телеметрию, а какие нет.

Пример команды:

```
{
    "arguments": {},
    "function": "get",
    "type": "cyclePub"
}
```

enable

Включает публикацию телеметрии с заданной в мс частотой публикации.

Пример команды:

```
{
    "arguments": {
        "cycleTime": 100
    },
    "function": "enable",
    "type": "cyclePub"
}
```

disable

Выключает публикацию телеметрии.

Пример команды:

```
{
    "arguments": {},
    "function": "disable",
    "type": "cyclePub"
}
```

Type <settings>

Команды типа settings осуществляют конфигурацию параметров робота, а также обнуление положения осей.

Параметры робота:

- d1 - высота от точки отсчета системы координат робота до второй оси
- a2 - расстояние между второй и третьей осями
- d4 - расстояние между точкой крепления четвертой оси и пятой осью
- d6 - расстояние между пятой осью и точкой фланца
- offsetX - смещение точки крепления четвертой оси относительной третьей оси в системе
- координат четвертой оси по направлению X
- offsetZ - смещение точки крепления четвертой оси относительной третьей оси в системе
- координат четвертой оси по направлению Z
- ratio - массив с передаточным отношением каждой оси
- tOfDiscret - время цикла отправки EtherCAT PDO в секундах
- encoderTicks - массив с количеством тиков энкодера на оборот каждой оси
- axisDirection - массив значений 1/-1 для каждой оси, где 1 - прямое направление вращения
- оси, а -1 - инвертированное
- lowerAxisLimit - массив нижней границы углов поворота каждой оси в радианах
- upperAxisLimit - массив верхней границы углов поворота каждой оси в радианах
- maxAxisAcc - массив максимальных ускорений осей в рад/с^2
- maxAxisVel - массив максимальных ускорений осей в рад/с
- maxCoordAcc - максимальное ускорение ЦТИ
- maxCoordVel - максимальная скорость ЦТИ

Предоставляются следующие функции:

- set
- get
- zero axis

set

Устанавливает значение указанного параметра в указанное значение.

Пример команды:

```
{
    "arguments": {
        "maxCoordAcc": 2.0
    },
    "function": "set",
```

```
    "type": "settings"
}
```

get

Показывает значение указанного параметра.

Пример команды:

```
{
    "arguments": {},
    "function": "get",
    "type": "settings"
}
```

zero axis

Обнуление положения выбранной оси. Нумерация осей с 0. Используется при юстировке робота.

Пример команды:

```
{
    "arguments": {
        "axisNum": 0
    },
    "function": "zero axis",
    "type": "settings"
}
```