ENERGY & CLIMATE CONTROL

Гибкая логика в PDU R-2MCx, R-3MCx, R-4MCx

Описание Python API

Версии ПО: 1.16.1 / 3.6.1 / 4.6.1

Москва, 2026

Оглавление

1. Введение

Этот документ содержит руководство по использованию Python API для поддержки гибкой логики в интеллектуальных блоках распределения питания REM™ второго и более старших серий (далее – PDU в общем, либо PDU2, либо PDU3 и т.д.). В контексте описания функций основного Контроллера PDU также будет использоваться термин «Контроллер».

Данная функция позволяет создавать, загружать и выполнять пользовательские скрипты, написанные на языке Python, для автоматизации задач, расширения функциональности контроллера и интеграции с внешними системами.

Гибкая логика предоставляет возможность:

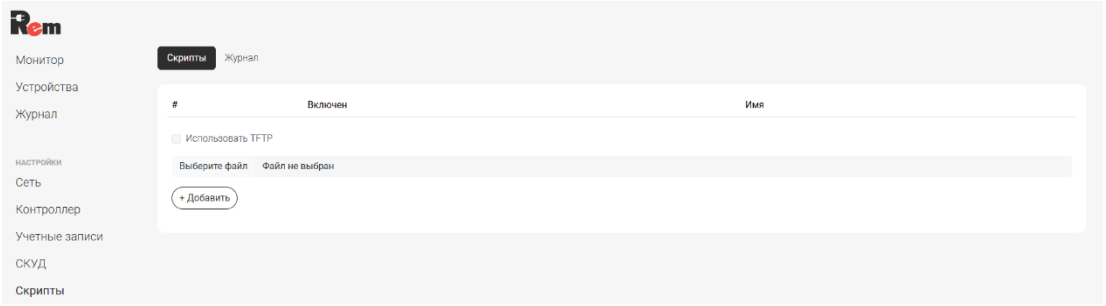
- Автоматизировать работу с розетками, датчиками, внешними устройствами. Реализовывать разные сценарии поведения контроллера при различных внешних условиях;
- Сохранять фотографии с подключенных камер на накопители, посылать их на удаленный сервер;
- Интегрировать контроллер со внешними системами верхнего уровня (IoT);
- Работать с устройствами, подключаемыми по шине RS485, у которых нет поддержки в основном ПО контроллера, как по Modbus RTU, так и по проприетарным протоколам.

Для успешной работы с гибкой логикой рекомендуется предварительное знакомство с основами языка Python, а также с основными функциями контроллера, описанными в руководстве по эксплуатации (РЭ).

2. Работа с готовыми скриптами

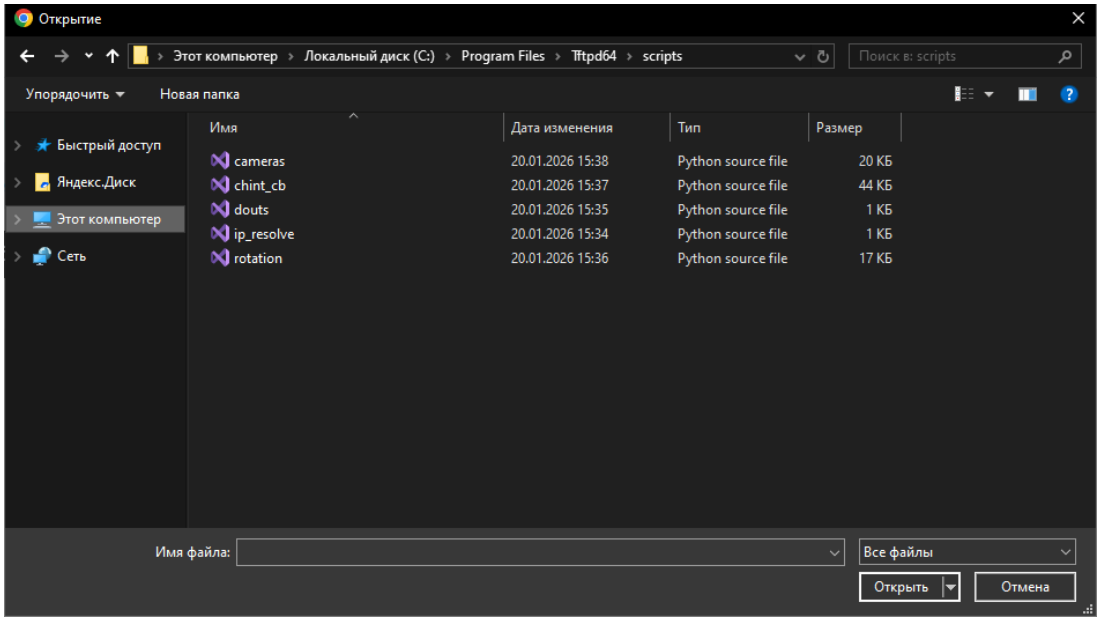
2.1. Загрузка скриптов через WEB-интерфейс

Для загрузки скриптов через WEB-интерфейс, перейдите на вкладку «Скрипты».

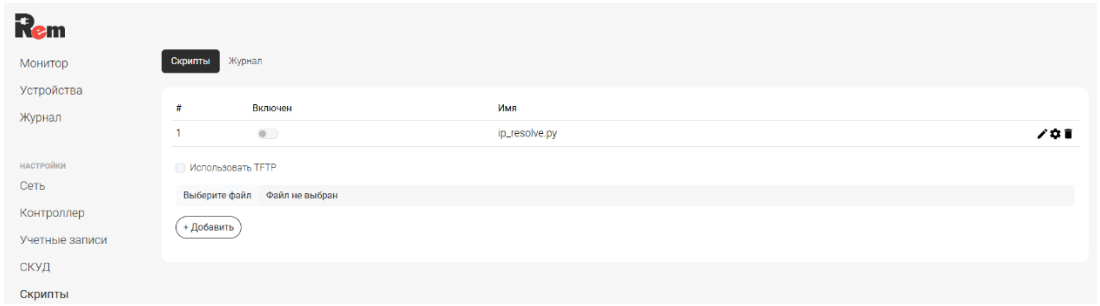


Затем выберите вариант загрузки – прямой (с файловой системы пользователя), либо через TFTP.

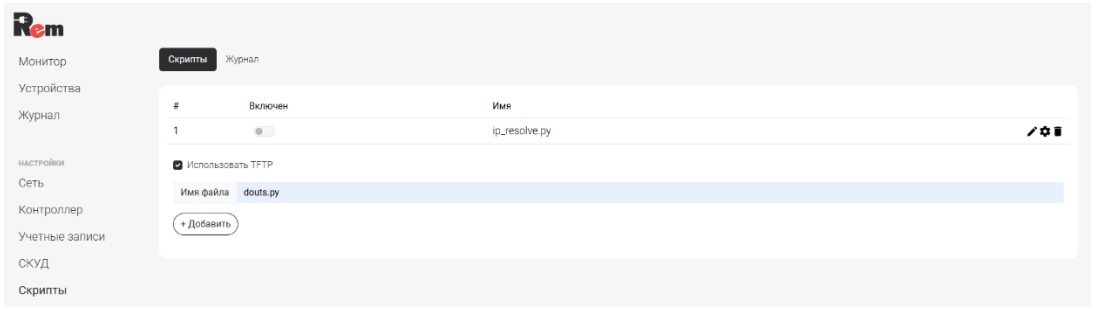
При прямой загрузке нажмите на поле «Выберите файл». Откроется окно «Открыть» - на своей файловой системе найдите нужный скрипт, нажмите «Открыть».



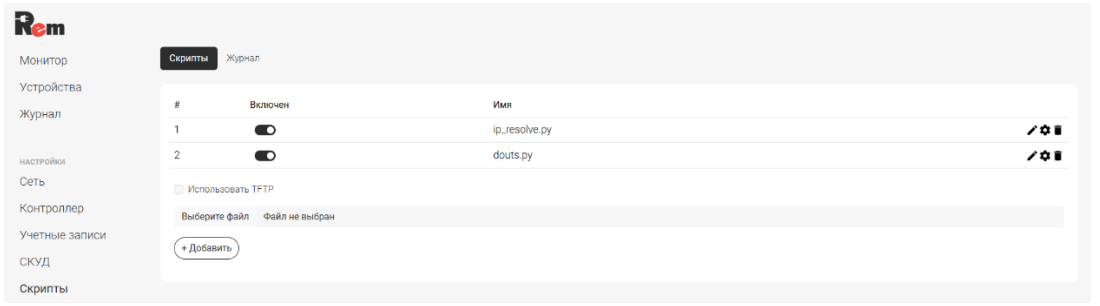
Затем на странице «Скрипты» нажмите кнопку «+Добавить».



При загрузке скрипта с TFTP-сервера, отметьте соответствующий пункт, а затем впишите имя файла скрипта на TFTP-сервере в поле для ввода. Имя скрипта указывается с расширением «.py». Затем нажмите кнопку «+Добавить».



Скрипт будет загружен, вы увидите его на странице «Скрипты». По умолчанию скрипт не запущен. Для запуска – переведите ползунок «Включен». Появится всплывающее окно с просьбой подтвердить включение скрипта. При подтверждении скрипт будет запущен.



Для удаления скрипта нажмите на иконку корзины справа напротив скрипта на странице «Скрипты». Появится всплывающее окно с просьбой подтвердить удаление скрипта. При подтверждении скрипт будет удален.

2.2. Редактирование скриптов в WEB-интерфейсе

Выключенные скрипты можно редактировать в WEB-интерфейсе. Для этого нажмите на иконку редактирования справа напротив скрипта. Откроется окно с кодом скрипта.

После внесения изменений можно нажать кнопку «Применить». При успешном сохранении скрипта вы увидите соответствующее уведомление.

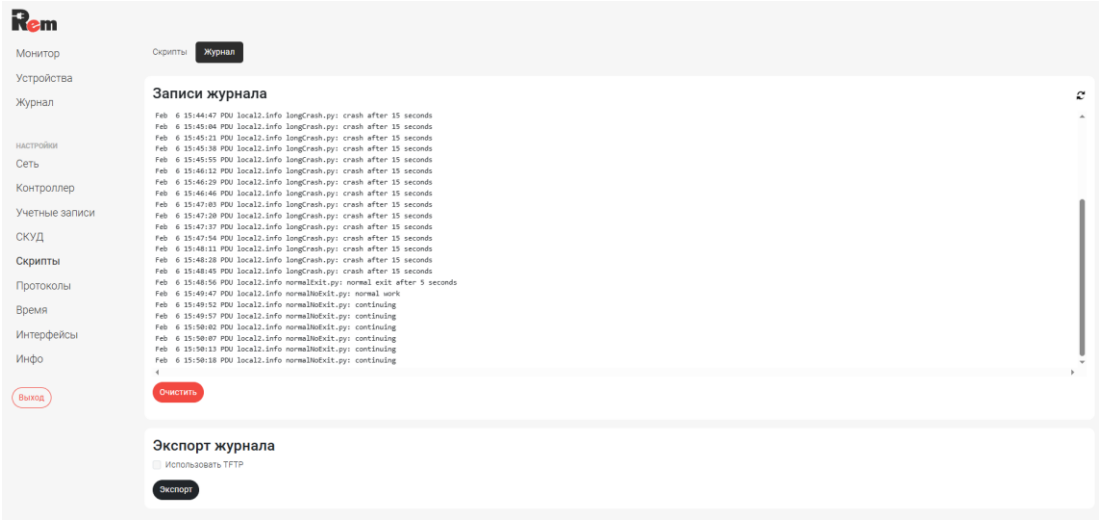
Если нажать кнопку «Закрыть» изменения не будут применены.

2.3. Настройки скриптов

У скриптов есть дополнительные настройки. Можно установить комментарий к скрипту, а также установить флаг перезапуска скрипта, если тот завершится ошибкой. В ПО стоит защита от частого перезапуска, поэтому если скрипт будет завершаться ошибкой слишком быстро, то Контроллер перестанет его перезапускать после 5 попыток. Для того чтобы открыть настройки скриптов надо нажать на иконку-шестерёнку настроек.

2.4. Журналирование, экспорт и импорт настроек скриптов

Ошибки, которые возникают при запуске или работы скриптов (неправильный синтаксис, ошибки выполнения) пишутся в обычный Журнал. Ошибки, сообщения от скриптов, которые посылаются с помощью API – пишутся журнал скриптов. В WEB-интерфейсе журнал доступен на вкладке «Скрипты» -> «Журнал».



Журнал можно очистить, экспортировать. Если журнал экспортируется через WEB-интерфейс, то он сохраняется на файловую систему пользователя. Для этого нажмите на кнопку «Экспорт журнала» на соответствующей странице.

Настройки скриптов можно экспортировать, вместе со всеми загруженными скриптами. При полном сбросе настроек контроллера удаляются скрипты как на файловой системе контроллера, так и из настроек, поэтому для быстрого переноса рекомендуется своевременно экспортировать настройки скриптов.

При запросе через REST API (см. описание REST API) на ФС пользователя загрузится архив с настройками скриптов и самими скриптами. Так же через REST API можно послать запрос на сохранение настроек скриптов на TFTP-сервер.

2.5. Примечания

- Имена скриптов должны быть не более 50 символов в длину;
- Имя файла должно содержать расширение «.py»;
- Размер файла скрипта должен быть не больше 1 МБ;
- Интерпретатор, используемый Контроллером, обладает рядом ограничений по сравнению со стандартным Python. Список поддерживаемых стандартных библиотек можно найти на сайте разработчика (https://docs.micropython.org/en/latest/library/index.html);
- Ряд операций может оказаться недоступным для выполнения в скриптах, это связано с ограничением прав у запускаемых скриптов в ПО Контроллера. Например, это операции с файлами или сетевые функции, которые требуют работы с DNS. Для обхода ограничений стоит использовать модуль, который помогает работать с файловой системой контроллера и API для разрешения IP-адреса по доменному имени;
- Команды для управления скриптами в CLI приведены в РЭ.

3. Общие сведения о REM Python API

Для того чтобы получить доступ к различным подсистемам контроллера и иметь возможность управлять им, подключёнными датчиками и внешними устройствами, необходимо использовать специальные Python модули (REM модуль) – они уже встроены в ПО контроллера и ниже описаны все их возможности, а также примеры применения.

4. Модуль rem_api

Данный модуль необходим в любом скрипте – он предоставляет функции инициализации и деинициализации API, без которых скрипт не начнет работу с контроллером. Помимо этого, в нем есть функция логирования, которая направляет сообщения в лог журнала скриптов. Описание функций представлено в таблице 4.1.

Таблица 4.1 – Описание функций модуля rem_api

Номер	Название	Аргументы	Описание
1.	init()	Нет	Начало работы с API Remer, ***без инициализации не будут работать функции других модулей REM API***. Возвращает True в случае успеха, вызывает ошибку, в случае неудачи.
2.	deinit()	Нет	Окончание работы с API Remer. После деинициализации не будут работать функции других модулей. Возвращает True.
3.	log(…)	string_obj – строка, которую необходимо вывести в лог	Вывод сообщения “string” в лог в журнал скриптов. Возвращает True.
4.	resolve(…)	string_obj – строка, доменное имя isIpv6 – флаг boolean, необязательный параметр	Определяет IP-адрес ресурса по его доменному имени и возвращает в виде строки. При передаче True вторым аргументом будет возвращен IPv6-адрес.

Пример использования:

```
"""
Пример использования rem_api: init, log, resolve.


Модуль rem_api экспортирует:
  - init()      — инициализация API скриптов и USB (обязательно в начале скрипта)
  - deinit()    — завершение (вызывать в finally)
  - log(msg)    — запись строки в лог скрипта (видно в веб-интерфейсе)
  - resolve(hostname [, prefer_ipv6=False]) — DNS-резолв, возвращает IP-строку


Скрипт: логирует старт, резолвит хост, пишет результат в лог и завершается.
"""


import rem_api


rem_api.init()


try:
    rem_api.log("Скрипт example_init_log_resolve запущен")

    host = "example.com"
    ip = rem_api.resolve(host)
    rem_api.log("{} -> {}".format(host, ip))

    # IPv6 (если нужен)
    try:
        ip6 = rem_api.resolve(host, True)
```

```
        rem_api.log("{} (IPv6) -> {}".format(host, ip6))
    except OSError as e:
        rem_api.log("IPv6 для {} недоступен: {}".format(host, e))

    rem_api.log("Скрипт завершён")
finally:
    rem_api.deinit()
```

5. Модуль rem_cameras

Модуль используется для работы с камерами, подключенными к Контроллеру. Позволяет сохранять фото с камеры на файловую систему контроллера, на TFTP-сервер, а также на накопитель. Описание функций модуля представлено в таблице 5.1.

Таблица 5.1 – Описание функций модуля rem_cameras

Номер	Название	Аргументы	Описание
1.	tftp_snapshot(…)	cam_idx_obj – целое число, индекс камеры (порта USB), подключенной к контроллеру file_name_obj – строка, имя файла, под которым будет сохранен снимок на TFTP-сервере (или ФС контроллера)	Сохраняет снимок с камеры на TFTP-сервер. Индекс начинается с 0. Порт USB (один на PDU2 / 4) – 0, Порт USB-2 (PDU3) – 1, Порт USB-3 (PDU3) – 2.
2.	snapshot(…)		Сохраняет снимок с камеры на ФС контроллера. Индекс начинается с 0.
3.	storage_snapshot(…)	cam_idx_obj – целое число, индекс камеры (порта USB), подключенной к контроллеру storage_idx_obj – целое число, индекс накопителя, на который будет сохранен снимок file_name_obj – строка, имя файла, под которым будет сохранен снимок на накопителе	Только для PDU3. Сохраняет снимок с камеры на внешний FLASH / uSD накопитель. Индекс накопителя можно выяснить командой storage в CLI, начинается с 1. Подробнее в РЭ.

Пример использования:

```
import rem_api
import rem_inputs
import rem_cameras
import time

# Инициализируем API
rem_api.init()
rem_api.log("Door monitor script started (НО Дверь)")

# Индексы входов
DIN1_INDEX = 0  # din1
DIN2_INDEX = 1  # din2

# Константы для мониторинга
MONITOR_INTERVAL = 1  # Проверка каждую секунду

# Индекс камеры (первая камера)
```

```
CAMERA_INDEX = 0

# Инициализация конфигурации дверей
def setup_doors():
    """Настройка входов din1 и din2 как дверных датчиков НО (Normally
Open)"""
    try:
        # Получаем din1 (индекс 0)
        din1 = rem_inputs.get(rem_inputs.DINS, DIN1_INDEX)
        # Получаем din2 (индекс 1)
        din2 = rem_inputs.get(rem_inputs.DINS, DIN2_INDEX)

        # Устанавливаем шаблон "Дверь НО" для обоих входов
        din1.type = rem_inputs.DIN_DOOR_NO
        din2.type = rem_inputs.DIN_DOOR_NO

        # Включаем входы
        din1.enabled = True
        din2.enabled = True

        # Включаем мониторинг
        din1.monitor = True
        din2.monitor = True

        # Переименовываем
        din1.name = "DoorFront"
        din2.name = "DoorBack"

        # Сохраняем изменения
        rem_inputs.save(din1)
        rem_inputs.save(din2)

        rem_api.log("Doors configured: din1 and din2 -> НО Дверь
(enabled, monitoring)")
        return True
    except Exception as e:
        rem_api.log(f"Error setting up doors: {str(e)}")
        return False

# Мониторинг состояния дверей
def monitor_doors():
    """Мониторинг состояния дверей и сохранение фото при открытии"""
    # Предыдущие состояния дверей
    prev_din1_state = None
    prev_din2_state = None

    while True:
        try:
```

```
            # Получаем текущее состояние дверей
            din1 = rem_inputs.get(rem_inputs.DINS, DIN1_INDEX)
            din2 = rem_inputs.get(rem_inputs.DINS, DIN2_INDEX)

            # Проверяем, загружено ли состояние
            if not din1.state_loaded:
                time.sleep(MONITOR_INTERVAL)
                continue
            if not din2.state_loaded:
                time.sleep(MONITOR_INTERVAL)
                continue

            # Инициализируем предыдущие состояния при первом запуске
            if prev_din1_state is None:
                prev_din1_state = din1.state
            if prev_din2_state is None:
                prev_din2_state = din2.state

            # Проверяем DIN1
            # Если дверь была закрыта, а теперь открыта
            if prev_din1_state != rem_inputs.DIN_DOOR_OPENED and
din1.state == rem_inputs.DIN_DOOR_OPENED:
                rem_api.log("DIN1: Дверь открыта - сохраняем фото на
TFTP")

                # Генерируем имя файла с временной меткой
                timestamp = int(time.time())
                filename = f"door1_din1_{timestamp}.jpg"

                try:
                    # Сохраняем фото с первой камеры на TFTP-сервер
                    rem_cameras.tftp_snapshot(CAMERA_INDEX, filename)
                    rem_api.log(f"DIN1: Photo saved to TFTP:
{filename}")
                except Exception as e:
                    rem_api.log(f"DIN1: Error saving photo to TFTP:
{str(e)}")

            # Проверяем DIN2
            # Если дверь была закрыта, а теперь открыта
            if prev_din2_state != rem_inputs.DIN_DOOR_OPENED and
din2.state == rem_inputs.DIN_DOOR_OPENED:
                rem_api.log("DIN2: Дверь открыта - сохраняем фото на
TFTP")

                # Генерируем имя файла с временной меткой
                timestamp = int(time.time())
                filename = f"door2_din2_{timestamp}.jpg"
```

```
                try:
                    # Сохраняем фото с первой камеры на TFTP-сервер
                    rem_cameras.tftp_snapshot(CAMERA_INDEX, filename)
                    rem_api.log(f"DIN2: Photo saved to TFTP:
{filename}")
                except Exception as e:
                    rem_api.log(f"DIN2: Error saving photo to TFTP:
{str(e)}")

            # Обновляем предыдущие состояния
            prev_din1_state = din1.state
            prev_din2_state = din2.state

            time.sleep(MONITOR_INTERVAL)

        except Exception as e:
            rem_api.log(f"Error in door monitoring loop: {str(e)}")
            time.sleep(MONITOR_INTERVAL)

# Основная функция
def main():
    """Главная функция скрипта"""
    # Настройка дверей при запуске
    if not setup_doors():
        rem_api.log("Failed to setup doors, exiting")
        rem_api.deinit()
        return

    rem_api.log("Door monitoring started")

    # Запускаем мониторинг
    try:
        monitor_doors()
    except KeyboardInterrupt:
        rem_api.log("Script interrupted")
    except Exception as e:
        rem_api.log(f"Fatal error: {str(e)}")
    finally:
        rem_api.deinit()

if __name__ == "__main__":
    main()
```

6. Модуль rem_devices

Модуль позволяет работать со внешними устройствами, подключенными к PDU. Подразумевается как считывание состояний, так и управление настройками устройств. Помимо этого, в экспериментальном формате доступна функция Python-устройств – возможность

работать с подключенными по к интерфейсам RS-485 Контроллера устройствами, у которых нет поддержки в основном ПО Контроллера.

Описание функций представлено в таблице 6.1.

Таблица 6.1 – Описание функций модуля rem_devices

Номер	Название	Аргументы	Описание
1.	get_all()	Нет	Получает список объектов device, описание которого приведено в таблице **Ошибка! Источник ссылки не найден.**. Настройки устройств и состояния не загружаются.
2.	get(…)	device_idx_or_name_obj – номер внешнего устройства в настройках (начинается с 0), либо имя устройства, заданное строкой	Получает объект device. Соответствует внешнему устройству с заданным индексом.
3.	save(…)	device_obj – объект типа device	Сохраняет настройки соответствующего внешнего устройства в соответствии со значениями, переданными с объектом device

Ниже в таблице 6.2 описаны атрибуты объектов device, которые можно получить функциями выше.

Таблица 6.2 – Описание объекта device

Номер	Название	Тип	Описание
1.	name	string	Название внутреннего устройства
2.	type	integer	Тип устройства: THERMOMETER – Термометр EMETER – Е-метр (не поддерживается) CONDITIONER – Кондиционер THERMOSTAT – Термостат LOCK – REM-замок DISPLAY – HMI-дисплей (доступны только настройки) FLOOD – Датчик протечки ATS – АВР DOORHUB – DoorHub (см. пункт 8) MPYDEV – Python-устройство (см. пункт 7)
3.	bus	integer	Серийный порт, к которому подключено устройство: BUS_1W – 1-Wire BUS_RS485_1 – RS485-1 BUS_RS485_2 – RS485-2 BUS_RS485_3 – RS485-3 BUS_RS232 – RS232
4.	group	integer	Группа устройства: PDU_GROUP_24_7 – 24/7 PDU_GROUP_GUARD – Охрана PDU_GROUP_INFO – Информационная PDU_GROUP_ENTRANCE – Входная
5.	enabled	boolean	Если True, то устройство включено
6.	monitoring	boolean	Если True, то устройство отображается в панели мониторинга
7.	send_emails	boolean	Если True, то устройство будет отсылать Email-уведомления при изменении состояния
8.	send_traps	boolean	Если True, то устройство будет отсылать trap-уведомления при изменении состояния
9.	uri	integer	Уникальный идентификатор устройства

Номер	Название	Тип	Описание
10.	status	integer	Статус устройства: STATUS_NORMAL – Норма STATUS_ALMIN – Незначительная авария STATUS_ALMAJ – Критическая авария
11.	state	integer	Состояние устройства: STATE_OK – Устройство работает нормально STATE_GONE – Связь с устройством потеряна STATE_NEVERSEEN – Устройство не найдено STATE_ERRORNEOUS – Устройство имеет внутренние ошибки STATE_BLOCKED – Устройство заблокировано (включен прозрачный порт) STATE_CONDITIONER_HEATING – Кондиционер в режиме нагревателя STATE_CONDITIONER_COOLING – Кондиционер в режиме охлаждения STATE_CONDITIONER_RUNNING – Кондиционер нормально работает STATE_CONDITIONER_STOPPED – Кондиционер остановлен STATE_FLOOD_DETECTED – Обнаружена протечка STATE_FLOOD_NOT_DETECTED – Протечка не обнаружена STATE_LOCK_BLOCKED – Замок закрыт STATE_LOCK_OPEN – Замок механически открыт STATE_LOCK_UNBLOCKED – Замок открыт ключом доступа STATE_LOCK_FULL_OPEN – Замок открыт ключом доступа и механически STATE_THERMOMETER_SEARCH – Поиск датчика температуры STATE_THERMOMETER_LOW_TEMP – Предупреждение по низкой температуре STATE_THERMOMETER_ALARM_LOW_TEMP – Тревога по низкой температуре STATE_THERMOMETER_HIGH_TEMP – Предупреждение по превышению температуры STATE_THERMOMETER_ALARM_HIGH_TEMP – Тревога по превышению температуры STATE_THERMOSTAT_HEATING – Термостат в режиме нагрева STATE_THERMOSTAT_COOLING – Термостат в режиме охлаждения
12.	state_loaded	boolean	Если True, то состояние устройства загружено в объект. Состояние есть только у устройств, которые включены в настройках
13.	params	dict	Словарь параметров из настроек Python-устройства. Доступ к значениям по ключу: value = params[key]
14.	almin_l	integer	Настройки термометра: нижняя граница температуры предупреждения, ℃
	almin_h		Настройки термометра: верхняя граница температуры предупреждения, ℃
	almaj_l		Настройки термометра: нижняя граница аварийной температуры, ℃
	almaj_h		Настройки термометра: верхняя граница аварийной температуры, ℃
15.	model	integer	Настройки кондиционера: модель 0 – Модель REM5 1 – Модель REM5U
	running		Настройки кондиционера: настройка работы 0 – Выключен 1 – Включен
	cooling_stop_pt		Настройки кондиционера: верхняя граница температуры охлаждения, ℃
	cooling_hyst		Настройки кондиционера: гистерезис температуры охлаждения, ℃

Номер	Название	Тип	Описание
	heating_stop_pt		Настройки кондиционера: нижняя граница температуры нагрева, ℃
	heating_hyst		Настройки кондиционера: гистерезис температуры нагрева, ℃
	int_fan_stop_pt		Настройки кондиционера: температура остановки внутреннего вентилятора, ℃
	temp_set_pt		Настройки кондиционера: установленная температура, $10 \cdot ℃$
	temp_sens_set_pt		Настройки кондиционера: гистерезис установленной температуры, $10 \cdot ℃$
	humid_set_pt		Настройки кондиционера: установленная влажность, $10 \cdot \%$
	humid_sens_set_pt		Настройки кондиционера: гистерезис установленной влажности, $10 \cdot \%$
	high_temp_set_pt		Настройки кондиционера: максимальная температура, $10 \cdot ℃$
	low_temp_set_pt		Настройки кондиционера: минимальная температура, $10 \cdot ℃$
	high_humid_set_pt		Настройки кондиционера: максимальная влажность, $10 \cdot \%$
	low_humid_set_pt		Настройки кондиционера: минимальная влажность, $10 \cdot \%$
16.	running	integer	Настройки термостата: настройка работы 0 – Выключен 1 – Включен
	temp_max_lvl		Настройки термостата: максимальная температура, $10 \cdot ℃$
	temp_min_lvl		Настройки термостата: минимальная температура, $10 \cdot ℃$
	humid_hyst		Настройки термостата: гистерезис влажности, $10 \cdot \%$
	humid_max_lvl		Настройки термостата: аварийная влажность, $10 \cdot \%$
	humid_alarm_lvl		Настройки термостата: влажность предупреждения, $10 \cdot \%$
	sensors_priority[4]		Настройки термостата: приоритет датчиков (1...4)
	sensor_ext		Настройки термостата: внешние датчики 0 – Включен T1 1 – Включен T2 255 – Никакой не включен
	sensors_enabled		Настройки термостата: включенные датчики $abcd_2$ a – 1, если T1 включен b – 1, если T2 включен c – 1, если TH включен d – 1, если INT включен
	relay_min_sw_time		Настройки термостата: минимальное время переключения реле, с
	otp_enabled		Настройки термостата: работа защиты от перегрева 0 – Выключена 1 – Включена
	otp_lvl		Настройки термостата: температура включения защиты от перегрева, $10 \cdot ℃$
	otp_hyst		Настройки термостата: гистерезис защиты от перегрева, $10 \cdot ℃$
	cold_start_enabled		Настройки термостата: работа холодного старта 0 – Выключена 1 – Включена
	cold_start_lvl		Настройки термостата: температура холодного старта, $10 \cdot ℃$
	fan_hyst		Настройки термостата: гистерезис вентилятора, $10 \cdot ℃$
	heat_hyst		Настройки термостата: гистерезис нагревателя, $10 \cdot ℃$
17.	id	integer	Настройки REM-замка: адрес устройства на шине Modbus
	open_delay		Настройки REM-замка: время открытия замка, с
18.	id	integer	Настройки датчика протечки: адрес устройства на шине Modbus
	baud		Настройки датчика протечки: скорость на шине Modbus
	alarm_delay		Настройки датчика протечки: задержка сигнализации, с
19.	id	integer	Настройки АВР: адрес устройства на шине Modbus
	priority_line		Настройки АВР: приоритетная линия 0 – Линия A 1 – Линия B

Номер	Название	Тип	Описание
			2 – Автоматический выбор
	u_min_a		Настройки АВР: нижняя граница напряжения на линии A, Вольт
	u_min_b		Настройки АВР: нижняя граница напряжения на линии B, Вольт
	u_max_a		Настройки АВР: верхняя граница напряжения на линии A, Вольт
	u_max_b		Настройки АВР: верхняя граница напряжения на линии B, Вольт
	nonsyn_enable	boolean	Настройки АВР: если True, разрешена работа с несинусоидальной сетью
	toggle_prior_on_disp		Настройки АВР: если True, включена возможность выбора приоритетной линии на станице мониторинга
	current_max	float	Настройки АВР: аварийное значение тока, А
	current_warn		Настройки АВР: значение тока по предупреждению, А
	hyst	integer	Настройки АВР: гистерезис тока, А
	prior_delay		Настройки АВР: задержка смены приоритетной линии
	quality_sens		Настройки АВР: качество чувствительности по току 0 – Низкое 1 – Среднее 2 – Высокое
20.	value	float	Состояние термометра: температура, ℃
21.	error	integer	Состояние кондиционера: флаг ошибок
	evaporator_temp		Состояние кондиционера: температура испарения, $10 \cdot$ ℃
	condenser_temp		Состояние кондиционера: температура охлаждения, $10 \cdot$ ℃
	indoor_temp		Состояние кондиционера: температура внутренняя, $10 \cdot$ ℃
	ret_air_temp		Состояние кондиционера: температура возврата воздуха, ℃
	supply_air_temp		Состояние кондиционера: температура входа воздуха, ℃
	ret_air_humid		Состояние кондиционера: влажность возврата воздуха, %
22.	t1_temp	float	Состояние термостата: температура T1, ℃
	t2_temp		Состояние термостата: температура T2, ℃
	th_temp		Состояние термостата: температура TH, ℃
	th_humid		Состояние термостата: влажность TH, %
	t_ext_value		Состояние термостата: температура TExt, ℃
	t_int_value		Состояние термостата: температура TINT, ℃
	active_state	boolean	Состояние термостата: если True, то термостат работает
	fan_state		Состояние термостата: если True, то вентилятор работает
	heat_state		Состояние термостата: если True, то нагреватель работает
23.	card	integer	Состояние REM-замка: приложенная карта
	error		Состояние REM-замка: флаг ошибки
	status		Состояние REM-замка: статус
	open		Состояние REM-замка: закрыт/открыт
24.	flood_detected	boolean	Состояние датчика протечки: если True, то замечена протечка
	alarm_detected		Состояние датчика протечки: если True, то замечена тревога
25.	active_line	integer	Состояние АВР: активная линия 0 – Нет 1 – A 2 – B
	a_voltage		Состояние АВР: напряжение на линии A
	b_voltage		Состояние АВР: напряжение на линии B
	output_current	float	Состояние АВР: выходной ток, А
	a_freq		Состояние АВР: частота на линии A, Гц
	b_freq		Состояние АВР: частота на линии B, Гц
26.	id	int	Настройки DoorHub: адрес Modbus
	baud		Настройки DoorHub: бодрейт
	lock_addr		Настройки DoorHub: адрес Modbus подключенного REM-замка
	display_addr		Настройки DoorHub: адрес Modbus подключенного HMI-дисплея
	sw_version	string	Версия ПО устройства
	acs_status	int	Состояние DoorHub: статус СКУД

Номер	Название	Тип	Описание
	climat_status		Состояние DoorHub: статус микроклимата
	power_status		Состояние DoorHub: статус питания
	door	dh_item	Состояние DoorHub: объект двери
	internal_ntc		Состояние DoorHub: объект встроенного датчика температуры
	sensors		Состояние DoorHub: список (list) датчиков, подключенных к DoorHub-у
	onewire		Состояние DoorHub: датчик 1-Wire, подключенный к DoorHub
Объекты типа dh_item будут описаны ниже в пункте 8.			

Пример использования:

Скрипт осуществляет ротацию кондиционеров по времени, а также отслеживает температуру внутри термошкафа включает форсированный режим работы кондиционеров при превышении максимальной температуры.

```
import rem_devices
import rem_api
import time

# Инициализируем API
rem_api.init()
rem_api.log("Conditioner rotation script started!")

# Конфигурация
TEMP_THRESHOLD = 45          # Максимальная температура, гр. Цельсия
HYSTERESIS = 5               # Гистерезис температуры
SWITCH_INTERVAL = 4 * 3600   # 4 часа - интервал ротации
RETRY_INTERVAL = 60          # 1 минута между попытками переинициализации

# Глобальные переменные
last_switch_time = 0          # Время последней ротации
active_conditioner_name = None  # Имя работающего кондиционера
both_running = False          # Флаг форсированного режима
emergency_mode = False        # Аварийный режим (один кондиционер)
critical_mode = False         # Критический режим (оба кондиционера)
last_retry_time = 0           # Время последней попытки переинициализации
conditioner_names = []        # Имена кондиционеров
current_conditioners = []     # Текущие объекты кондиционеров

# Функция проверки, является ли состояние устройства рабочим
def is_device_state_working(state):
    """Проверяет, является ли состояние устройства рабочим для ротации"""
    return state in [rem_devices.STATE_CONDITIONER_COOLING,
                     rem_devices.STATE_CONDITIONER_HEATING,
                     rem_devices.STATE_CONDITIONER_RUNNING,
                     rem_devices.STATE_CONDITIONER_STOPPED]

# Функция проверки, является ли состояние устройства проблемным
```

```
def is_device_state_problematic(state):
    """Проверяет, является ли состояние устройства проблемным"""
    return state in [rem_devices.STATE_GONE,
                     rem_devices.STATE_ERRORNEOUS,
                     rem_devices.STATE_NEVERSEEN,
                     rem_devices.STATE_BLOCKED]

# Функция находит кондиционеры по именам
def find_conditioners():
    global conditioner_names, current_conditioners

    conditioner_names = []
    current_conditioners = []

    try:
        # Пытаемся найти кондиционеры по именам
        for name in ["Conditioner1", "Conditioner2"]:
            try:
                device = rem_devices.get(name)
                if (device.type == rem_devices.CONDITIONER and
                    device.enabled and
                    is_device_state_working(device.state)):
                    conditioner_names.append(name)
                    current_conditioners.append(device)
                    rem_api.log(f"Found conditioner: {device.name}
(state: {device.state})")
            except Exception as e:
                rem_api.log(f"Conditioner {name} not found or not
available: {str(e)}")
    except Exception as e:
        rem_api.log(f"Error finding conditioners: {str(e)}")

    return len(conditioner_names) >= 2

# Функция получает актуальный объект кондиционера по имени
def get_conditioner_by_name(name):
    try:
        device = rem_devices.get(name)
        if (device and device.type == rem_devices.CONDITIONER and
            device.enabled and is_device_state_working(device.state)):
            return device
        return None
    except Exception as e:
        rem_api.log(f"Error getting conditioner {name}: {str(e)}")
        return None

# Функция обновляет текущие объекты кондиционеров
def refresh_conditioners():
```

```
    global current_conditioners

    refreshed_conditioners = []
    for name in conditioner_names:
        device = get_conditioner_by_name(name)
        if device:
            refreshed_conditioners.append(device)
        else:
            rem_api.log(f"Conditioner {name} not available")

    current_conditioners = refreshed_conditioners
    return len(current_conditioners)

# Функция устанавливает состояние кондиционера по имени
def set_conditioner_state(name, state):
    try:
        device = rem_devices.get(name)
        if device and device.type == rem_devices.CONDITIONER:
            device.running = 1 if state else 0
            rem_devices.save(device)
            rem_api.log(f"Setting conditioner {device.name} to {'ON'
if state else 'OFF'}")
            return True
        else:
            rem_api.log(f"Device {name} is not a conditioner")
            return False
    except Exception as e:
        rem_api.log(f"Error setting conditioner state for {name}:
{str(e)}")
        return False

# Функция проверки состояния кондиционеров
def check_conditioners_status():
    status_ok = True
    working_conditioners = []  # Список имен рабочих кондиционеров
    connection_lost = False

    if not refresh_conditioners():
        rem_api.log("No conditioners available during status check")
        return False, [], True

    for name in conditioner_names:
        try:
            device = get_conditioner_by_name(name)
            if not device:
                rem_api.log(f"Conditioner {name} not found")
                status_ok = False
                connection_lost = True
```

```
                continue

            # Проверка состояния устройства
            if is_device_state_problematic(device.state):
                if device.state == rem_devices.STATE_GONE:
                    rem_api.log(f"Connection lost with conditioner
{device.name}!")
                elif device.state == rem_devices.STATE_ERRORNEOUS:
                    rem_api.log(f"Internal errors on conditioner
{device.name}!")
                elif device.state == rem_devices.STATE_NEVERSEEN:
                    rem_api.log(f"Conditioner {device.name} never seen
since addition!")
                elif device.state == rem_devices.STATE_BLOCKED:
                    rem_api.log(f"Conditioner {device.name} is blocked
by controller settings!")

                status_ok = False
                if device.state in [rem_devices.STATE_GONE,
rem_devices.STATE_NEVERSEEN]:
                    connection_lost = True
            else:
                # Устройство в рабочем состоянии
                working_conditioners.append(name)

        except Exception as e:
            rem_api.log(f"Error checking conditioner {name}:
{str(e)}")
            status_ok = False
            connection_lost = True

    return status_ok, working_conditioners, connection_lost

# Функция получения температуры кондиционера
def get_conditioner_temperature(name):
    try:
        device = get_conditioner_by_name(name)
        if not device:
            rem_api.log(f"Conditioner {name} not found for temperature
reading")
            return None

        # Используем только indoor_temp (температура внутри шкафа)
        # Это поле заполняется для обеих моделей кондиционеров
        if hasattr(device, 'indoor_temp') and device.indoor_temp is
not None:
            temp = device.indoor_temp / 10.0
            rem_api.log(f"Temperature {device.name}: {temp}°C")
            return temp
        else:
```

```
            rem_api.log(f"No temperature data for {device.name}")
            return None

    except Exception as e:
        rem_api.log(f"Error reading temperature for {name}: {str(e)}")
        return None

# Функция запуска аварийного режима
def start_emergency_mode(working_names):
    global emergency_mode, critical_mode, both_running

    # Выключаем все кондиционеры сначала
    for name in conditioner_names:
        set_conditioner_state(name, 0)

    if len(working_names) == 0:
        # Критический режим - нет рабочих кондиционеров
        rem_api.log("CRITICAL MODE: No working conditioners, turning
both ON")
        for name in conditioner_names:
            set_conditioner_state(name, 1)
        critical_mode = True
        emergency_mode = False
        both_running = False
    else:
        # Обычный аварийный режим - есть хотя бы один рабочий
        device = get_conditioner_by_name(working_names[0])
        if device:
            rem_api.log(f"EMERGENCY MODE: Switching to conditioner
{device.name}")
            set_conditioner_state(working_names[0], 1)
        emergency_mode = True
        critical_mode = False
        both_running = False

# Функция проверки восстановления после аварии
def check_emergency_recovery():
    global emergency_mode, critical_mode

    status_ok, working_names, connection_lost =
check_conditioners_status()

    if critical_mode and len(working_names) >= 1:
        rem_api.log("Recovering from critical mode")
        start_emergency_mode(working_names)
        return True
    elif emergency_mode and len(working_names) >= 2:
        rem_api.log("Recovering from emergency mode, returning to
```

```
rotation")
        emergency_mode = False
        return initialize_rotation()

    return False

# Функция инициализации ротации
def initialize_rotation():
    global last_switch_time, active_conditioner_name, emergency_mode,
critical_mode, both_running

    # Проверяем статус устройств
    status_ok, working_names, connection_lost =
check_conditioners_status()

    if not status_ok or len(working_names) < 2:
        rem_api.log("Cannot start rotation - device issues detected")
        start_emergency_mode(working_names)
        return False

    # Выключаем все кондиционеры
    for name in conditioner_names:
        set_conditioner_state(name, 0)

    # Включаем первый кондиционер (по порядку в conditioner_names)
    active_conditioner_name = conditioner_names[0]
    set_conditioner_state(active_conditioner_name, 1)

    last_switch_time = time.time()
    emergency_mode = False
    critical_mode = False
    both_running = False

    device = get_conditioner_by_name(active_conditioner_name)
    if device:
        rem_api.log(f"Rotation started with conditioner
{device.name}")
    return True

# Функция переключения на следующий кондиционер
def switch_to_next_conditioner():
    global last_switch_time, active_conditioner_name

    # Выключаем текущий
    if active_conditioner_name:
        set_conditioner_state(active_conditioner_name, 0)

    # Переключаем на следующий
```

```
    if active_conditioner_name in conditioner_names:
        current_idx = conditioner_names.index(active_conditioner_name)
        next_idx = (current_idx + 1) % len(conditioner_names)
    else:
        next_idx = 0

    active_conditioner_name = conditioner_names[next_idx]
    set_conditioner_state(active_conditioner_name, 1)

    last_switch_time = time.time()

    device = get_conditioner_by_name(active_conditioner_name)
    if device:
        rem_api.log(f"Switching to conditioner {device.name}")

# Функция управления кондиционерами
def manage_conditioners():
    global last_switch_time, active_conditioner_name, both_running,
emergency_mode, critical_mode

    current_time = time.time()

    # Проверяем состояние устройств
    status_ok, working_names, connection_lost =
check_conditioners_status()

    # Обработка аварийных режимов
    if not status_ok:
        if not emergency_mode and not critical_mode:
            rem_api.log("Device issues detected, starting emergency
mode")
            start_emergency_mode(working_names)
        elif emergency_mode or critical_mode:
            # Проверяем восстановление в аварийном режиме
            check_emergency_recovery()
        return

    # Получаем температуру со всех кондиционеров
    temperatures = []
    for name in conditioner_names:
        temp = get_conditioner_temperature(name)
        if temp is not None:
            temperatures.append(temp)

    if not temperatures:
        rem_api.log("No temperature data available")
        return
```

```
    max_temp = max(temperatures)

    # Проверка температуры для форсированного режима
    if max_temp > TEMP_THRESHOLD:
        if not both_running:
            rem_api.log(f"High temperature detected ({max_temp}°C),
enabling FORCED mode")
            for name in conditioner_names:
                set_conditioner_state(name, 1)
            both_running = True
            emergency_mode = False
            critical_mode = False

    # Выход из форсированного режима
    elif both_running:
        # Получаем температуру со всех кондиционеров снова для выхода
        exit_temperatures = []
        for name in conditioner_names:
            temp = get_conditioner_temperature(name)
            if temp is not None:
                exit_temperatures.append(temp)

        if exit_temperatures:
            max_exit_temp = max(exit_temperatures)
            if max_exit_temp < TEMP_THRESHOLD - HYSTERESIS:
                rem_api.log(f"Temperature normalized
({max_exit_temp}°C), returning to rotation mode")
                both_running = False
                # Отключаем все
                for name in conditioner_names:
                    set_conditioner_state(name, 0)
                # Включаем активный
                if active_conditioner_name:
                    set_conditioner_state(active_conditioner_name, 1)
                last_switch_time = current_time

    # Обычная ротация
    if not both_running and not emergency_mode and not critical_mode:
        # Проверяем время ротации
        if current_time - last_switch_time >= SWITCH_INTERVAL:
            switch_to_next_conditioner()

# Основная функция
def main():
    global last_retry_time

    # Первоначальный поиск кондиционеров
    if not find_conditioners():
```

```
        rem_api.log("Error: Need at least 2 working conditioners for
rotation!")
        return

    # Инициализация
    if not initialize_rotation():
        last_retry_time = time.time()

    # Основной цикл
    while True:
        try:
            current_time = time.time()

            # Периодический перепоиск кондиционеров и попытка
переинициализации
            if current_time - last_retry_time >= RETRY_INTERVAL:
                # Перепоиск кондиционеров (индексы могут измениться)
                find_conditioners()

                # Попытка переинициализации в аварийных режимах
                if emergency_mode or critical_mode:
                    rem_api.log("Trying to reinitialize rotation...")
                    if initialize_rotation():
                        last_retry_time = current_time
                    else:
                        last_retry_time = current_time
                else:
                    last_retry_time = current_time

            # Основное управление
            manage_conditioners()

            time.sleep(60)

        except Exception as e:
            rem_api.log(f"Error in main loop: {str(e)}")
            time.sleep(60)

if __name__ == "__main__":
    main()
```

Примечание. Для корректной работы скрипта индекс кондиционеров в общем списке внешних устройств не должен меняться.

7. Модуль rem_devices – работа с Modbus-устройствами

Модуль rem_devices так же позволяет работать с устройствами, которые подключены к шинам RS-485 / RS-232 Контроллера. Работа может осуществляться как по Modbus RTU протоколу, так и по проприетарным протоколам. Для работы необходимо создать объект task, в него добавляются задачи специальными функциями. За один вызов можно добавить до 16

задач. После добавления задач вызывается функция perform, которая эти задачи выполняет. Функция возвращает объект result, который содержит результаты выполнения – статусы, ответы от устройства.

На стороне Контроллера необходимо добавить новое устройство, поменять его шаблон на «Python-устройство». В настройках доступен для изменения адрес, можно добавить до 10 параметров, которые потом станут доступны из скриптов. Параметры задаются в формате «ключ-значение».

Кроме того, сообщения, строки измерения для вывода в WEB-интерфейсе, а также статус устройства задается тоже с помощью задач.

Описание функций модуля, связанных с Python-устройствами представлено в таблице 7.1. Ниже в таблице 7.2 описание объекта task, его функций.

Таблица 7.1 – Описание функций Python-устройства объекта device

Номер	Название	Аргументы	Описание
1.	create_tasks()	Нет	Создает объект task, в который добавляются задачи
2.	perform(…)	task_obj – объект task с добавленными задачами timeout_obj – число, таймаут выполнения задачи в мс	Выполняет все задачи task. Вызов функции синхронный, завершается при выполнении всех задач, либо по таймауту. Возвращает объект result, в котором результаты выполнения задач в том же порядке, что и задачи в task. После вызова task очищается.

Таблица 7.2 – Описание функций объекта task

Номер	Название	Аргументы	Описание
1.	read_register(…)	register_type – enum, тип регистра addr_obj – число, адрес регистра count_obj – число, число регистров для чтения	Читает регистр(ы) устройства по заданному адресу. Типы регистров: REGISTER_HOLDING – аналоговые выходы REGISTER_COIL – дискретные выходы REGISTER_INPUT – аналоговые входы REGISTER_DISCRETE_INPUT – дискретные входы Адреса и количество регистров указываются в формате беззнаковых чисел UINT16.
2.	write_register(…)	register_type – enum, тип регистра addr_obj – число, адрес регистра data_obj – может быть одним значением при записи одного регистра, а может быть списком list значений для записи нескольких регистров	Записывает данные в регистр(ы) устройства по заданному адресу. Типы регистров: REGISTER_HOLDING – аналоговые выходы REGISTER_COIL – дискретные выходы Адреса и количество регистров указываются в формате беззнаковых чисел UINT16. Максимальное число регистров для записи – 123. Значение одного регистра в пределах UINT16.
3.	set_status(…)	status_obj – enum, статус устройства	Устанавливает статус устройства. Виды статуса: STATUS_NORMAL – Норма STATUS_ALMIN – Незначительная авария STATUS_ALMAJ – Критическая авария
4.	set_message(…)	msg_obj – string, сообщение устройства	Устанавливает строку сообщения устройства (отображение в WEB, CLI). Максимальная длина сообщения – 128 символов.
5.	set_measure(…)	msr_obj – string, измерение устройства для отображения в WEB, CLI	Устанавливает строку измерения устройства (отображение в WEB, CLI). Максимальная длина измерения – 128 символов. Для

Номер	Название	Аргументы	Описание
			корректного переноса по строкам в WEB измерения разделяются запятыми.
6.	set_snmp_data(…)	idx_obj – число, индекс поля в snmp-таблице устройства data_obj – число, данные для отображения в поле snmp-таблицы устройства	Устанавливает значение в нужное поле в snmp-таблице устройства. Значение должно быть INT32.
7.	send_bytes(…)	bytes_obj – набор байт для передачи устройству	Отправляет набор байт устройству. Используется для реализации коммуникации с устройством по проприетарному протоколу.

После выполнения perform вернется массив объектов result. Ниже в таблице 7.3 представлены поля объекта result.

Таблица 7.3 – Описание полей объекта result

Номер	Название	Тип	Описание
1.	error	enum	Enum ошибки Modbus, которая может возникнуть при коммуникации с устройством. Типы ошибок: MB_ERROR_OK – операция прошла успешно, ошибок нет MB_ERROR_NO_ANSWER – устройство не отвечает на запросы MB_ERROR_BAD_CRC – не сошлась контрольная сумма MB_ERROR_BAD_DATA – поврежденные данные
2.	value	int или list[int]	Данные переданные устройством в ответ за запрос на чтение регистров.
3.	raw	bytes	Полезные данные ответа (без адреса, функции, числа последующих байт и контрольной суммы). Либо полный ответ от устройства на запрос send_bytes.

Пример использования:

```
"""
Скрипт для работы с 3-фазным интеллектуальным УЗО Chint NB2-80ZT через
универсальные устройства (mpydev).


Скрипт демонстрирует:
- Работу с mpydev устройством на шине RS485
- Фильтрацию устройств по параметру "script_target" == "3phase"
- Опрос параметров измерения питания (токи, напряжения, мощности)
- Опрос состояний аварий
- Управление положением автомата на основе параметра "enabled" из
конфигурации
- Формирование строк измерения и сообщений
- Установку статуса устройства на основе считанных данных
- Логирование ошибок коммуникации и параметров питания


Параметры устройства настраиваются через web-интерфейс или CLI:
- script_target - должен быть равен "3phase" для работы этого скрипта
- enabled - управление автоматом: 1 = взвести, 0 = выключить
"""


import time
```

```
import rem_api
import rem_devices


# Параметры по умолчанию
DEFAULT_MODBUS_ADDRESS = 3  # Адрес по умолчанию для Chint NB2-80ZT

# Интервал логирования параметров питания (в секундах)
POWER_LOG_INTERVAL = 300  # 5 минут

# Команды дистанционного управления (функция 0x10, адрес 0x0000)
CMD_REMOTE_CLOSE_ALLOWED = 0x0002    # Разрешить дистанционное
замыкание
CMD_REMOTE_CLOSE_FORBIDDEN = 0x0003  # Запретить дистанционное
замыкание
CMD_REMOTE_CLOSE = 0x0006            # Дистанционное замыкание
CMD_REMOTE_OPEN = 0x0007             # Дистанционное размыкание

# Адреса регистров
REG_WORKING_STATUS = 0x0020      # Рабочее состояние BIT16
REG_ALARM_STATUS = 0x0021        # Состояния аварии BIT16
REG_CURRENT_L1_LOW = 0x0040      # Фазный ток L1 (младшее слово)
REG_CURRENT_L1_HIGH = 0x0041     # Фазный ток L1 (старшее слово)
REG_CURRENT_L2_LOW = 0x0042      # Фазный ток L2 (младшее слово)
REG_CURRENT_L2_HIGH = 0x0043     # Фазный ток L2 (старшее слово)
REG_CURRENT_L3_LOW = 0x0044      # Фазный ток L3 (младшее слово)
REG_CURRENT_L3_HIGH = 0x0045     # Фазный ток L3 (старшее слово)
REG_VOLTAGE_L1 = 0x0048          # Фазное напряжение L1
REG_VOLTAGE_L2 = 0x0049          # Фазное напряжение L2
REG_VOLTAGE_L3 = 0x004A          # Фазное напряжение L3
REG_POWER_L1_LOW = 0x0051        # Активная мощность L1 (младшее
слово)
REG_POWER_L1_HIGH = 0x0052       # Активная мощность L1 (старшее
слово)
REG_POWER_L2_LOW = 0x0053        # Активная мощность L2 (младшее
слово)
REG_POWER_L2_HIGH = 0x0054       # Активная мощность L2 (старшее
слово)
REG_POWER_L3_LOW = 0x0055        # Активная мощность L3 (младшее
слово)
REG_POWER_L3_HIGH = 0x0056       # Активная мощность L3 (старшее
слово)
REG_REMOTE_CONTROL = 0x0000      # Команды дистанционного управления
(W, функция 0x10)


def get_param(dev, param_name, default_value):
    """Получает значение параметра из конфига устройства"""
    params = dev.params
```

```
    if param_name in params:
        return params[param_name]
    return default_value


def get_error_name(error):
    """Возвращает понятное имя ошибки ModbusError_t"""
    if error == rem_devices.MB_ERROR_OK:
        return "OK"
    elif error == rem_devices.MB_ERROR_NO_ANSWER:
        return "Нет ответа от устройства"
    elif error == rem_devices.MB_ERROR_BAD_CRC:
        return "Не сошлась контрольная сумма (проверьте настройки
интерфейса)"
    elif error == rem_devices.MB_ERROR_BAD_DATA:
        return "Неполный ответ от устройства (проверьте настройки
интерфейса)"
    else:
        return "Неизвестная ошибка ({})".format(error)


def find_mpydev_devices():
    """Находит mpydev устройства на шине RS485 с параметром
script_target == "3phase" """
    devices = rem_devices.get_all()
    result = []
    for d in devices:
        if (d.type == rem_devices.MPYDEV or d.type ==
rem_devices.UNSPECIFIED) and \
           (d.bus == rem_devices.BUS_RS485_1 or d.bus ==
rem_devices.BUS_RS485_2):
            # Проверяем параметр script_target
            script_target = get_param(d, "script_target", "")
            if script_target == "3phase":
                result.append(d)
    if len(result) < 1:
        raise Exception("Не найдено устройств mpydev с параметром
script_target='3phase' на шине RS485")
    return result


def crc16_modbus(data):
    """Вычисляет CRC16 для Modbus RTU кадра"""
    crc = 0xFFFF
    for byte in data:
        crc ^= byte
        for _ in range(8):
            if crc & 0x0001:
                crc = (crc >> 1) ^ 0xA001
```

```
            else:
                crc >>= 1
    return crc


def bytes_to_hex(data):
    """Преобразует bytes или bytearray в hex строку (совместимо с MicroPython)"""
    if data is None:
        return "None"
    return ''.join(['%02x' % b for b in data])


def parse_uint32(low_word, high_word):
    """Парсит 32-битное беззнаковое число из двух 16-битных слов

    В Modbus RTU регистры передаются как: младший регистр (low_word) содержит младшие 16 бит,
    старший регистр (high_word) содержит старшие 16 бит.
    Значение = (low_word << 16) | high_word
    """
    return (low_word << 16) | high_word


def parse_int32(low_word, high_word):
    """Парсит 32-битное знаковое число из двух 16-битных слов

    В Modbus RTU регистры передаются как: младший регистр (low_word) содержит младшие 16 бит,
    старший регистр (high_word) содержит старшие 16 бит.
    Значение = (low_word << 16) | high_word
    Затем преобразуем в знаковое число.
    """
    value = (low_word << 16) | high_word
    # Преобразуем в знаковое число
    if value >= 0x80000000:
        value = value - 0x100000000
    return value


def read_register_result(res, index):
    """Читает значение регистра из результата запроса"""
    if res.error != rem_devices.MB_ERROR_OK or res.raw is None:
        return None

    raw = bytes(res.raw)
    if len(raw) < 3:  # byte_count + минимум 1 регистр (2 байта)
        return None
```

```
    # Формат: первый байт - количество байт данных, затем данные
    byte_count = raw[0]
    if byte_count < 2 or len(raw) < byte_count + 1:
        return None

    # Читаем значение регистра (big-endian)
    value = (raw[1] << 8) | raw[2]
    return value


def read_two_registers_result(res, index):
    """Читает два регистра из результата запроса и возвращает как 32-битное число"""
    if res.error != rem_devices.MB_ERROR_OK or res.raw is None:
        return None

    raw = bytes(res.raw)
    if len(raw) < 3:
        rem_api.log(
            "read_two_registers_result ({}): слишком короткий ответ, len={}, raw={}".format(
                index, len(raw), bytes_to_hex(raw)
            )
        )
        return None

    byte_count = raw[0]
    # Формат: [byte_count][data...]
    if byte_count == 2 and len(raw) >= 3:
        # Устройство вернуло только один регистр (low_word), high_word считаем 0
        low_word = (raw[1] << 8) | raw[2]
        high_word = 0
        rem_api.log(
            "read_two_registers_result ({}): получен только 1 регистр (byte_count=2), raw={}".format(
                index, bytes_to_hex(raw)
            )
        )
        return (low_word, high_word)
    elif byte_count >= 4 and len(raw) >= 5:
        # Нормальный случай: два регистра подряд (big-endian)
        low_word = (raw[1] << 8) | raw[2]
        high_word = (raw[3] << 8) | raw[4]
        return (low_word, high_word)
    else:
        rem_api.log(
```

```
            "read_two_registers_result ({}): неожиданный формат
ответа, byte_count={}, len={}, raw={}".format(
                index, byte_count, len(raw), bytes_to_hex(raw)
            )
        )
        return None


MEASURE_PHASE_ROTATION_INTERVAL = 60  # секунд между сменой
отображаемой фазы
_current_measure_phase_idx = 0
_last_phase_switch_time = 0.0


def format_measure(data):
    """Форматирует строку измерения для одной фазы.

    Формат: "Phase: Lx, I: <ток, 1 знак после запятой> A, U:
<напряжение, цел> V, P: <мощность, 2 знака после запятой> kW"
    """
    global _current_measure_phase_idx, _last_phase_switch_time

    phases = ['L1', 'L2', 'L3']
    available_phases = []
    for phase in phases:
        cur_key = 'current_{}'.format(phase.lower())
        volt_key = 'voltage_{}'.format(phase.lower())
        power_key = 'power_{}'.format(phase.lower())
        if (data.get(cur_key) is not None) or (data.get(volt_key) is
not None) or (data.get(power_key) is not None):
            available_phases.append(phase)

    if not available_phases:
        return "Phase: N/A, I: N/A, U: N/A, P: N/A"

    now = time.time()
    # Смена фазы раз в MEASURE_PHASE_ROTATION_INTERVAL секунд
    if now - _last_phase_switch_time >=
MEASURE_PHASE_ROTATION_INTERVAL:
        _last_phase_switch_time = now
        _current_measure_phase_idx = (_current_measure_phase_idx + 1)
% len(available_phases)

    phase = available_phases[_current_measure_phase_idx %
len(available_phases)]
    cur_key = 'current_{}'.format(phase.lower())
    volt_key = 'voltage_{}'.format(phase.lower())
    power_key = 'power_{}'.format(phase.lower())

    parts = ["Phase: {}".format(phase)]
```

```
    # Ток (мА -> А)
    current_ma = data.get(cur_key)
    if current_ma is not None:
        parts.append("I: {:.1f} A".format(current_ma / 1000.0))
    else:
        parts.append("I: N/A")

    # Напряжение (0.01В -> В)
    voltage_001v = data.get(volt_key)
    if voltage_001v is not None:
        parts.append("U: {:.0f} V".format(voltage_001v / 100.0))
    else:
        parts.append("U: N/A")

    # Мощность (0.1Вт -> кВт)
    power_001w = data.get(power_key)
    if power_001w is not None:
        # Значение в регистрах: 0.1 Вт (пример из даташита: 10992 ->
1099.2 Вт)
        # Перевод в кВт: /10 (до Вт) и /1000 (до кВт) => делим на
10000
        power_kw = abs(power_001w) / 10000.0
        parts.append("P: {:.2f} kW".format(power_kw))
    else:
        parts.append("P: N/A")

    return ", ".join(parts)


def format_message(data):
    """Форматирует сообщение: список аварий или состояние автомата
(Open/Close) """
    alarm_status = data.get('alarm_status', 0)
    working_status = data.get('working_status', 0)

    # Если есть аварии - возвращаем список аварий
    if alarm_status != 0:
        alarms = []

        # Бит 0: перенапряжение
        if alarm_status & (1 << 0):
            alarms.append("OV")  # Перенапряжение

        # Бит 1: пониженное напряжение
        if alarm_status & (1 << 1):
            alarms.append("UV")  # Пониженное напряжение
```

```
        # Бит 2: утечка
        if alarm_status & (1 << 2):
            alarms.append("LEAK")  # Утечка

        # Бит 3: перегрузка
        if alarm_status & (1 << 3):
            alarms.append("OL")  # Перегрузка

        # Бит 6: обрыв фазы
        if alarm_status & (1 << 6):
            alarms.append("PL")  # Обрыв фазы

        # Бит 7: повышенная частота
        if alarm_status & (1 << 7):
            alarms.append("OF")  # Повышенная частота

        # Бит 8: пониженная частота
        if alarm_status & (1 << 8):
            alarms.append("UF")  # Пониженная частота

        # Бит 9: потеря напряжения
        if alarm_status & (1 << 9):
            alarms.append("VL")  # Потеря напряжения

        if alarms:
            return ", ".join(alarms)

    # Если нет аварий - возвращаем состояние автомата
    # Проверяем положение переключателя (бит 7)
    # 1: замкнут (Close), 0: разомкнут (Open)
    switch_position = (working_status >> 7) & 1
    if switch_position == 1:
        return "Close"
    else:
        return "Open"


def compute_status(data):
    """Вычисляет статус устройства: ALMAJ если есть аварии или автомат
разомкнут"""
    working_status = data.get('working_status', 0)
    alarm_status = data.get('alarm_status', 0)

    # Проверяем наличие аварий
    if alarm_status != 0:
        return rem_devices.STATUS_ALMAJ
```

```
    # Проверяем положение переключателя (бит 7)
    # 1: замкнут, 0: разомкнут
    switch_position = (working_status >> 7) & 1
    if switch_position == 0:
        return rem_devices.STATUS_ALMAJ

    return rem_devices.STATUS_NORMAL

def read_register_function11(dev, modbus_address):
    """Читает регистр используя функцию 11 (0x11 - Report Slave ID)

    Согласно мануалу, функция 11 используется для чтения регистра
0x0020.
    Формат кадра: [адрес][0x11][CRC16]
    Ответ: [адрес][0x11][длина_данных][данные...][CRC16]
    """
    tasks = dev.create_tasks()

    # Формируем Modbus RTU кадр: [адрес][код_функции][CRC16]
    frame = bytearray([modbus_address, 0x11])
    crc = crc16_modbus(frame)
    frame.append(crc & 0xFF)  # Младший байт CRC
    frame.append((crc >> 8) & 0xFF)  # Старший байт CRC

    # Отправляем кадр используя send_bytes
    tasks.send_bytes(bytes(frame))

    try:
        res = dev.perform(tasks)
    except Exception as e:
        # Ошибка API - возвращаем None
        return None

    if len(res) == 0 or res[0].error != rem_devices.MB_ERROR_OK:
        return None

    # Парсим ответ: [адрес][0x11][длина_данных][данные...][CRC16]
    # Примечание: send_bytes возвращает полный Modbus RTU кадр включая
адрес и CRC
    raw = bytes(res[0].raw) if res[0].raw else None
    if raw is None or len(raw) < 5:  # Минимум: адрес + функция +
длина + 2 байта данных + 2 байта CRC
        return None

    # Проверяем формат ответа
    if len(raw) < 3:
        return None
```

```
    resp_addr = raw[0]
    resp_func = raw[1]

    # Проверяем ответ об ошибке (0x80 + 0x11 = 0x91)
    if resp_func == 0x91:  # 0x80 + 0x11
        return None

    if resp_addr != modbus_address or resp_func != 0x11:
        return None

    # Получаем длину данных
    data_length = raw[2]
    if len(raw) < 3 + data_length + 2:  # адрес + функция + длина + данные + CRC
        return None

    # Извлекаем данные (предполагаем, что первые 2 байта - значение регистра)
    if data_length >= 2:
        value = (raw[3] << 8) | raw[4]
        return value

    return None



def send_remote_command(dev, modbus_address, command):
    """Отправляет команду дистанционного управления (функция 0x10, адрес 0x0000)

    Согласно мануалу Chint NB2-80ZT: "(Функциональный код 0x10, адрес 0x0000, 2-байтовый код команды)"
    Используем функцию 0x10 (Write Multiple Registers).

    Формат запроса функции 0x10:

[адрес][0x10][адрес_регистра_H][адрес_регистра_L][количество_регистров_H][количество_регистров_L][количество_байт][данные_H][данные_L][CRC16]

    Для одного регистра (0x0000) с командой:

[адрес][0x10][0x00][0x00][0x00][0x01][0x02][команда_H][команда_L][CRC16]

    Возвращает True при успехе, False при ошибке.
    """
    tasks = dev.create_tasks()

    command_name = "взвод" if command == CMD_REMOTE_CLOSE else "отключение"
```

```
    # Формируем Modbus RTU кадр для функции 0x10 (Write Multiple
Registers)
    # Адрес регистра: 0x0000
    # Количество регистров: 0x0001 (один регистр)
    # Количество байт данных: 0x02 (2 байта)
    # Команда: 2 байта (старший и младший)
    frame = bytearray([
        modbus_address,      # Адрес устройства
        0x10,                # Функция Write Multiple Registers
        0x00, 0x00,          # Адрес регистра (0x0000)
        0x00, 0x01,          # Количество регистров (1)
        0x02,                # Количество байт данных (2)
        (command >> 8) & 0xFF,  # Старший байт команды
        command & 0xFF       # Младший байт команды
    ])

    # Вычисляем CRC16
    crc = crc16_modbus(frame)
    frame.append(crc & 0xFF)  # Младший байт CRC
    frame.append((crc >> 8) & 0xFF)  # Старший байт CRC

    # Отправляем кадр используя send_bytes
    tasks.send_bytes(bytes(frame))

    try:
        res = dev.perform(tasks)
    except Exception as e:
        error_msg = str(e)
        return None, "Ошибка API при чтении данных:
{}".format(error_msg)

    if len(res) == 0:
        rem_api.log("Не удалось отправить команду {}: нет ответа от
устройства".format(command_name))
        return False

    if res[0].error != rem_devices.MB_ERROR_OK:
        error_name = get_error_name(res[0].error)
        # Логируем сырые данные ответа для отладки
        raw = bytes(res[0].raw) if res[0].raw else None
        if raw:
            raw_hex = bytes_to_hex(raw)
            rem_api.log("Не удалось отправить команду {}: {} (ответ
устройства: {})".format(
                command_name, error_name, raw_hex))
        else:
            rem_api.log("Не удалось отправить команду {}:
{}".format(command_name, error_name))
        return False
```

```
    # Проверяем ответ функции 0x10
    # Формат ответа:
[адрес][0x10][адрес_регистра_H][адрес_регистра_L][количество_регистров
_H][количество_регистров_L][CRC16]
    raw = bytes(res[0].raw) if res[0].raw else None
    if raw is None or len(raw) < 8:  # Минимум: адрес + функция +
адрес_регистра(2) + количество(2) + CRC(2)
        raw_hex = bytes_to_hex(raw) if raw else "нет данных"
        rem_api.log("Неполный ответ на команду {}: длина ответа {},
данные: {}".format(
            command_name, len(raw) if raw else 0, raw_hex))
        return False

    resp_addr = raw[0]
    resp_func = raw[1]

    # Проверяем ответ об ошибке (0x80 + 0x10 = 0x90)
    if resp_func == 0x90:  # 0x80 + 0x10
        raw_hex = bytes_to_hex(raw)
        exception_code = raw[2] if len(raw) > 2 else "неизвестен"
        rem_api.log("Устройство вернуло ошибку на команду {}: код
исключения {}, полный ответ: {}".format(
            command_name, exception_code, raw_hex))
        return False

    if resp_addr != modbus_address or resp_func != 0x10:
        raw_hex = bytes_to_hex(raw)
        rem_api.log("Неверный формат ответа на команду {}: адрес={},
функция={:02X}, полный ответ: {}".format(
            command_name, resp_addr, resp_func, raw_hex))
        return False

    return True



def read_device_data(dev, modbus_address):
    """Читает данные из устройства Chint NB2-80ZT

    Использует функцию 11 (0x11) для чтения регистра 0x0020 (рабочее
состояние) согласно мануалу.
    Остальные регистры читаются стандартной функцией Modbus 0x03.

    Возвращает словарь с данными или None при ошибке коммуникации.
    В случае ошибки также возвращает информацию об ошибке в виде
кортежа (None, error_info).
    """
    tasks = dev.create_tasks()

    # Читаем рабочее состояние (0x0020) используя функцию 11
```

```
    # Читаем отдельно используя send_bytes
    working_status = read_register_function11(dev, modbus_address)


    # Читаем статус аварий (0x0021) - функция 0x03 (Read Holding
Registers)
    tasks.read_register(rem_devices.REGISTER_HOLDING,
REG_ALARM_STATUS, 1)


    # Читаем токи фаз (L1, L2, L3) - по 2 регистра на каждую фазу -
функция 0x03
    tasks.read_register(rem_devices.REGISTER_HOLDING,
REG_CURRENT_L1_LOW, 2)  # L1: 0x0040-0x0041
    tasks.read_register(rem_devices.REGISTER_HOLDING,
REG_CURRENT_L2_LOW, 2)  # L2: 0x0042-0x0043
    tasks.read_register(rem_devices.REGISTER_HOLDING,
REG_CURRENT_L3_LOW, 2)  # L3: 0x0044-0x0045


    # Читаем напряжения фаз (L1, L2, L3) - по 1 регистру на каждую
фазу - функция 0x03
    tasks.read_register(rem_devices.REGISTER_HOLDING, REG_VOLTAGE_L1,
1)  # L1: 0x0048
    tasks.read_register(rem_devices.REGISTER_HOLDING, REG_VOLTAGE_L2,
1)  # L2: 0x0049
    tasks.read_register(rem_devices.REGISTER_HOLDING, REG_VOLTAGE_L3,
1)  # L3: 0x004A


    # Читаем мощности фаз (L1, L2, L3) - по 2 регистра на каждую фазу
- функция 0x03
    tasks.read_register(rem_devices.REGISTER_HOLDING,
REG_POWER_L1_LOW, 2)  # L1: 0x0051-0x0052
    tasks.read_register(rem_devices.REGISTER_HOLDING,
REG_POWER_L2_LOW, 2)  # L2: 0x0053-0x0054
    tasks.read_register(rem_devices.REGISTER_HOLDING,
REG_POWER_L3_LOW, 2)  # L3: 0x0055-0x0056


    try:
        res = dev.perform(tasks)
    except Exception as e:
        error_msg = str(e)
        return None, "Ошибка API при чтении данных:
{}".format(error_msg)


    # Ожидаем 10 ответов: авария + 3 тока + 3 напряжения + 3 мощности
    # (рабочее состояние читается отдельно через функцию 11)
    expected_responses = 10


    if len(res) != expected_responses:
        # Логируем сырые данные ответов для отладки при ошибке
        raw_data_info = []
        for i, r in enumerate(res):
            if r.raw:
                raw_hex = bytes_to_hex(bytes(r.raw))
                error_status = get_error_name(r.error) if r.error !=
```

```
rem_devices.MB_ERROR_OK else "OK"
                raw_data_info.append("запрос {}: статус={},
данные={}".format(i, error_status, raw_hex))
        error_msg = "Неполный ответ от устройства (получено {} ответов
из {})".format(len(res), expected_responses)
        if raw_data_info:
            error_msg += "; сырые данные: " + "; ".join(raw_data_info)
        return None, error_msg

    # Проверяем, что хотя бы один запрос успешен
    success_count = 0
    error_info = []
    for i, r in enumerate(res):
        if r.error == rem_devices.MB_ERROR_OK:
            success_count += 1
        else:
            error_detail = get_error_name(r.error)
            # Добавляем сырые данные ответа для отладки
            if r.raw:
                raw_hex = bytes_to_hex(bytes(r.raw))
                error_detail += " (данные: {})".format(raw_hex)
            error_info.append("запрос {}: {}".format(i, error_detail))

    if success_count == 0 and working_status is None:
        error_msg = "Все запросы завершились с ошибкой"
        if error_info:
            error_msg += ": " + "; ".join(error_info)
        return None, error_msg

    data = {}

    # Парсим рабочее состояние (читается через функцию 11)
    if working_status is not None:
        data['working_status'] = working_status

    # Парсим статус аварий (индекс 0)
    alarm_status = read_register_result(res[0], 0)
    if alarm_status is not None:
        data['alarm_status'] = alarm_status
    else:
        raw = bytes(res[0].raw) if res[0].raw else None
        raw_hex = bytes_to_hex(raw) if raw else "нет данных"
        error_name = get_error_name(res[0].error)
        rem_api.log(
            "Предупреждение: не удалось прочитать статус аварий ({}),
raw={}".format(
                error_name, raw_hex
            )
```

```
        )


    # Парсим токи фаз (i32 - знаковое 32-битное число, единица
измерения: 0.001 А)
    current_l1_words = read_two_registers_result(res[1], 1)
    if current_l1_words is not None:
        data['current_l1'] = parse_int32(current_l1_words[0],
current_l1_words[1])  # в мА (0.001 А)


    current_l2_words = read_two_registers_result(res[2], 2)
    if current_l2_words is not None:
        data['current_l2'] = parse_int32(current_l2_words[0],
current_l2_words[1])  # в мА (0.001 А)


    current_l3_words = read_two_registers_result(res[3], 3)
    if current_l3_words is not None:
        data['current_l3'] = parse_int32(current_l3_words[0],
current_l3_words[1])  # в мА (0.001 А)


    # Парсим напряжения фаз
    voltage_l1 = read_register_result(res[4], 4)
    if voltage_l1 is not None:
        data['voltage_l1'] = voltage_l1  # в 0.01В


    voltage_l2 = read_register_result(res[5], 5)
    if voltage_l2 is not None:
        data['voltage_l2'] = voltage_l2  # в 0.01В


    voltage_l3 = read_register_result(res[6], 6)
    if voltage_l3 is not None:
        data['voltage_l3'] = voltage_l3  # в 0.01В


    # Парсим мощности фаз
    power_l1_words = read_two_registers_result(res[7], 7)
    if power_l1_words is not None:
        data['power_l1'] = parse_int32(power_l1_words[0],
power_l1_words[1])  # в 0.01Вт


    power_l2_words = read_two_registers_result(res[8], 8)
    if power_l2_words is not None:
        data['power_l2'] = parse_int32(power_l2_words[0],
power_l2_words[1])  # в 0.01Вт


    power_l3_words = read_two_registers_result(res[9], 9)
    if power_l3_words is not None:
        data['power_l3'] = parse_int32(power_l3_words[0],
power_l3_words[1])  # в 0.01Вт


    return data, None
```

```
def process_device(dev, modbus_address, last_log_time,
last_enabled_state, last_command_attempt_time, remote_close_allowed,
error_count):
    """Обрабатывает устройство: читает данные, управляет автоматом на
основе параметра enabled"""
    # Получаем свежее устройство из конфигурации для чтения параметров
    all_devices = rem_devices.get_all()
    current_dev = None
    for d in all_devices:
        if d.name == dev.name:
            current_dev = d
            break

    if current_dev is None:
        rem_api.log("Устройство {} не найдено в
конфигурации".format(dev.name))
        return last_log_time, last_enabled_state,
last_command_attempt_time, remote_close_allowed, error_count

    # Читаем параметр enabled из конфигурации
    enabled_param = get_param(current_dev, "enabled", "0")
    try:
        enabled = int(enabled_param)
    except (ValueError, TypeError):
        enabled = 0

    # Управляем автоматом на основе параметра enabled
    # Проверяем, изменилось ли значение enabled и прошло ли достаточно
времени с последней попытки
    current_time = time.time()
    last_attempt = last_command_attempt_time.get(dev.name, 0)
    time_since_last_attempt = current_time - last_attempt

    if enabled != last_enabled_state.get(dev.name, -1):
        # Если прошло меньше 5 секунд с последней попытки, пропускаем
(чтобы не спамить при ошибках)
        if time_since_last_attempt < 5.0:
            # Не логируем, чтобы не засорять логи
            pass
        else:
            if enabled == 1:
                # Перед взводом нужно разрешить дистанционный взвод
                if not remote_close_allowed.get(dev.name, False):
                    rem_api.log("Устройство {}: разрешаем
дистанционный взвод".format(dev.name))
                    if send_remote_command(dev, modbus_address,
CMD_REMOTE_CLOSE_ALLOWED):
                        rem_api.log("Дистанционный взвод разрешен")
                        remote_close_allowed[dev.name] = True
```

```
                        time.sleep(0.2)  # Небольшая задержка перед
следующей командой
                    else:
                        rem_api.log("Не удалось разрешить
дистанционный взвод, пропускаем команду взвода")
                        return last_log_time, last_enabled_state,
last_command_attempt_time, remote_close_allowed, error_count


                rem_api.log("Устройство {}: enabled=1, взводим
автомат".format(dev.name))
                last_command_attempt_time[dev.name] = current_time
                if send_remote_command(dev, modbus_address,
CMD_REMOTE_CLOSE):
                    rem_api.log("Автомат успешно взведен")
                    last_enabled_state[dev.name] = 1
                else:
                    # Не обновляем last_enabled_state при ошибке,
чтобы попробовать снова через 5 секунд
                    pass
            elif enabled == 0:
                rem_api.log("Устройство {}: enabled=0, отключаем
автомат".format(dev.name))
                last_command_attempt_time[dev.name] = current_time
                if send_remote_command(dev, modbus_address,
CMD_REMOTE_OPEN):
                    rem_api.log("Автомат успешно отключен")
                    last_enabled_state[dev.name] = 0
                else:
                    # Не обновляем last_enabled_state при ошибке,
чтобы попробовать снова через 5 секунд
                    pass


    # Читаем данные из устройства
    result = read_device_data(dev, modbus_address)


    # Обрабатываем результат (может быть кортеж (data, error_info) или
просто data для обратной совместимости)
    if isinstance(result, tuple):
        data, error_info = result
    else:
        data = result
        error_info = None


    if data is None:
        # Ошибка коммуникации
        error_count[dev.name] = error_count.get(dev.name, 0) + 1


        if error_info:
            rem_api.log("Ошибка коммуникации с устройством {}: {}
(ошибок подряд: {})".format(
                dev.name, error_info, error_count[dev.name]))
```

```
        else:
            rem_api.log("Ошибка коммуникации с устройством {}: нет
ответа от устройства (ошибок подряд: {})".format(
                dev.name, error_count[dev.name]))


        tasks = dev.create_tasks()


        # Если 10 ошибок подряд - устанавливаем статус ALMAJ и
сообщение "No connection"
        if error_count[dev.name] >= 10:
            tasks.set_message("No connection")
            tasks.set_measure("N/A")
            tasks.set_status(rem_devices.STATUS_ALMAJ)
        else:
            tasks.set_message("Ошибка связи")
            tasks.set_measure("N/A")
            tasks.set_status(rem_devices.STATUS_ALMIN)


        try:
            dev.perform(tasks)
        except Exception as e:
            error_msg = str(e)
            rem_api.log("Ошибка API при установке статуса устройства
{}: {}".format(dev.name, error_msg))
        return last_log_time, last_enabled_state,
last_command_attempt_time, remote_close_allowed, error_count


    # Проверяем полноту данных - если нет working_status или
alarm_status, считаем что данные неполные.
    # Это мягкая ошибка: не считаем связь потерянной, не трогаем
error_count, только ограниченно логируем.
    if 'working_status' not in data or 'alarm_status' not in data:
        # Логируем только первый и затем каждый 10-й случай, чтобы не
заспамить логи
        if error_count.get(dev.name, 0) == 0 or
(error_count.get(dev.name, 0) % 10 == 0):
            rem_api.log("Устройство {}: получены неполные данные,
пропускаем обработку".format(dev.name))
        return last_log_time, last_enabled_state,
last_command_attempt_time, remote_close_allowed, error_count


    # Полноценное чтение данных - сбрасываем счетчик жестких ошибок
(data is None), если он был
    if error_count.get(dev.name, 0) > 0:
        rem_api.log("Устройство {}: связь
восстановлена".format(dev.name))
        error_count[dev.name] = 0


    # Формируем список токов для логирования
    currents = []
    if 'current_l1' in data:
        currents.append(('L1', data['current_l1']))
```

```
    if 'current_l2' in data:
        currents.append(('L2', data['current_l2']))
    if 'current_l3' in data:
        currents.append(('L3', data['current_l3']))

    # Формируем сообщение и измерение
    message = format_message(data)
    measure = format_measure(data)
    status = compute_status(data)

    # Логируем аварии, если они есть (но не "Open" или "Close" - это
не аварии)
    if message not in ["Normal", "Open", "Close"]:
        rem_api.log("Устройство {}: обнаружены аварии -
{}".format(dev.name, message))

    # Логируем параметры питания каждые 5 минут
    current_time = time.time()
    if current_time - last_log_time >= POWER_LOG_INTERVAL:
        log_msg = "Параметры питания - "
        log_parts = []

        if currents:
            for phase, current_ma in currents:
                log_parts.append("{}: {:.1f}A".format(phase,
current_ma / 1000.0))

        voltages = []
        if 'voltage_l1' in data:
            voltages.append(('L1', data['voltage_l1']))
        if 'voltage_l2' in data:
            voltages.append(('L2', data['voltage_l2']))
        if 'voltage_l3' in data:
            voltages.append(('L3', data['voltage_l3']))

        if voltages:
            for phase, voltage_001v in voltages:
                log_parts.append("{}: {:.0f}B".format(phase,
voltage_001v / 100.0))

        powers = []
        if 'power_l1' in data:
            powers.append(('L1', data['power_l1']))
        if 'power_l2' in data:
            powers.append(('L2', data['power_l2']))
        if 'power_l3' in data:
            powers.append(('L3', data['power_l3']))
```

```
        if powers:
            total_power_w = sum(abs(p[1]) for p in powers) / 100.0
            log_parts.append("Всего: {:.2f}кВт".format(total_power_w /
1000.0))

        if log_parts:
            rem_api.log(log_msg + ", ".join(log_parts))

        last_log_time = current_time

    # Отправляем message/measure/status на устройство
    tasks = dev.create_tasks()

    tasks.set_message(message)
    tasks.set_measure(measure)
    tasks.set_status(status)

    try:
        dev.perform(tasks)
    except Exception as e:
        error_msg = str(e)
        rem_api.log("Ошибка API при отправке данных устройства {}:
{}".format(dev.name, error_msg))

    return last_log_time, last_enabled_state,
last_command_attempt_time, remote_close_allowed, error_count



def main():
    rem_api.init()
    rem_api.log("Скрипт для работы с 3-фазным УЗО Chint NB2-80ZT
запущен!")

    devices = []
    try:
        devices = find_mpydev_devices()
        if len(devices) == 0:
            rem_api.log("Не найдено устройство mpydev с параметром
script_target='3phase' на шине RS485")
            return

        rem_api.log("Найдено устройство с script_target='3phase':
{}".format(len(devices)))

        # Получаем адреса устройств из конфигурации
        device_addresses = {}
        for dev in devices:
            # Получаем свежее устройство из конфигурации
            all_devices = rem_devices.get_all()
```

```
            current_dev = None
            for d in all_devices:
                if d.name == dev.name:
                    current_dev = d
                    break

            if current_dev is None:
                rem_api.log("Устройство {} не найдено в
конфигурации".format(dev.name))
                continue

            # Получаем адрес Modbus из конфигурации устройства (по
умолчанию 3)
            modbus_address = current_dev.id if hasattr(current_dev,
'id') else DEFAULT_MODBUS_ADDRESS
            device_addresses[dev.name] = modbus_address
            rem_api.log("Устройство {}: Modbus адрес =
{}".format(dev.name, modbus_address))

        # Словарь для отслеживания времени последнего логирования для
каждого устройства
        last_log_times = {dev.name: time.time() for dev in devices}
        # Словарь для отслеживания последнего состояния enabled для
каждого устройства
        last_enabled_state = {}
        # Словарь для отслеживания времени последней попытки отправки
команды
        last_command_attempt_time = {}
        # Словарь для отслеживания состояния разрешения дистанционного
взвода
        remote_close_allowed = {}
        # Словарь для отслеживания количества последовательных ошибок
связи
        error_count = {}

        # При запуске разрешаем дистанционный взвод для всех устройств
        rem_api.log("Разрешаем дистанционный взвод для всех
устройств...")
        for dev in devices:
            modbus_address = device_addresses.get(dev.name,
DEFAULT_MODBUS_ADDRESS)
            rem_api.log("Устройство {}: отправляем команду разрешения
дистанционного взвода".format(dev.name))
            if send_remote_command(dev, modbus_address,
CMD_REMOTE_CLOSE_ALLOWED):
                rem_api.log("Устройство {}: дистанционный взвод
разрешен".format(dev.name))
                remote_close_allowed[dev.name] = True
            else:
                rem_api.log("Устройство {}: не удалось разрешить
дистанционный взвод".format(dev.name))
                remote_close_allowed[dev.name] = False
            time.sleep(0.5)  # Задержка между командами для
```

```
стабильности API


        # Основной цикл
        while True:
            for dev in devices:
                try:
                    modbus_address = device_addresses.get(dev.name,
DEFAULT_MODBUS_ADDRESS)
                    last_log_times[dev.name], last_enabled_state,
last_command_attempt_time, remote_close_allowed, error_count =
process_device(
                        dev, modbus_address, last_log_times[dev.name],
last_enabled_state, last_command_attempt_time, remote_close_allowed,
error_count)
                except Exception as e:
                    rem_api.log("Ошибка при обработке устройства {}:
{}".format(dev.name, str(e)))


            time.sleep(1.5)  # Опрос каждые 1.5 секунды для
стабильности API


    finally:
        rem_api.deinit()



if __name__ == '__main__':
    main()
```

8. Модуль rem_devices – работа с DoorHub

В модуле rem_devices есть отдельный тип объектов, который описывает сущности, связанные с DoorHub-ом: двери, датчики 1-Wire, RST-1, NTC. Кроме того, есть и отдельные функции, с помощью которых на DoorHub можно отослать команду. Ниже в таблице 8.1 описаны эти функции. Они вызываются от объекта типа device.

Таблица 8.1 – функции rem_devices, связанные с DoorHub

Номер	Название	Аргументы	Описание
1.	set_power_status	status_obj – константа статуса (описано выше в пункте 6)	Устанавливает у DoorHub-а индикацию статуса питания

Ниже в таблице 8.2 показаны атрибуты объекта dh_item. Получить сам объект, можно с помощью соответствующих атрибутов объекта device, описано выше в пункте 6.

Таблица 8.2 – описание объекта dh_item

Номер	Название	Тип	Описание
1.	name	string	Имя объекта
2.	enabled	bool	Если True, то датчик включен
3.	monitor		Если True, то датчик отображается в панели мониторинга
4.	send_emails		Если True, то датчик будет отсылать Email-уведомления при изменении состояния
5.	send_traps		Если True, то датчик будет отсылать trap-уведомления при изменении состояния
6.	group	int	Группа датчика: GROUP_24_7 – 24/7

Номер	Название	Тип	Описание
			GROUP_GUARD – Охрана GROUP_INFO – Информационная GROUP_ENTRANCE – Входная
7.	status		Статус датчика: STATUS_NORMAL – Норма STATUS_ALMIN – Незначительная авария STATUS_ALMAJ – Критическая авария
8.	uri	int	Уникальный идентификатор датчика
9.	door_template	int	Шаблон двери: DOORHUB_DOOR_NC – нормально закрытая дверь DOORHUB_DOOR_NO – нормально открытая дверь DOORHUB_DOOR_UNKNOWN – неизвестный шаблон
10.	open	bool	Показывает открыта ли дверь
11.	lost	bool	Показывает потеряна ли связь с датчиком
12.	sensor_template	int	Шаблон датчика: DOORHUB_SENSOR_NTC – датчик NTC DOORHUB_SENSOR_RSHT1 – датчик RSHT-1 DOORHUB_SENSOR_1WIRE – датчик 1-Wire DOORHUB_SENSOR_UNKNOWN – неизвестный шаблон
13.	temp	float	Температура датчика, ℃
14.	humid		Влажность датчика, %
15.	th_state	int	Статус влажности: STATUS_NORMAL – Норма STATUS_ALMIN – Незначительная авария (предупреждение) STATUS_ALMAJ – Критическая авария
16.	temp_state		Статус температуры: STATUS_NORMAL – Норма STATUS_ALMIN – Незначительная авария (предупреждение) STATUS_ALMAJ – Критическая авария
17.	almin_l		Настройки датчиков температуры: нижний порог предупреждения, ℃
18.	almin_h		Настройки датчиков температуры: верхний порог предупреждения, ℃
19.	almaj_l		Настройки датчиков температуры: нижний порог аварии, ℃
20.	almaj_h		Настройки датчиков температуры: верхний порог аварии, ℃
21.	hyst		Настройки датчиков температуры: гистерезис, ℃
22.	temp_alarm_high	float	Настройки датчиков RSHT-1: верхний порог аварии (температура), ℃
23.	temp_alarm_low		Настройки датчиков RSHT-1: нижний порог аварии (температура), ℃
24.	temp_warn_high		Настройки датчиков RSHT-1: верхний порог предупреждения (температура), ℃
25.	temp_warn_low		Настройки датчиков RSHT-1: нижний порог предупреждения (температура), ℃
26.	humid_alarm_high		Настройки датчиков RSHT-1: верхний порог аварии (влажность), %
27.	humid_alarm_low		Настройки датчиков RSHT-1: нижний порог аварии (влажность), %
28.	humid_warn_high		Настройки датчиков RSHT-1: верхний порог предупреждения (влажность), %
29.	humid_warn_low		Настройки датчиков RSHT-1: нижний порог предупреждения (влажность), %
30.	temp_hyst		Настройки датчиков RSHT-1: гистерезис температуры, ℃
31.	warn_hyst		Настройки датчиков RSHT-1: гистерезис влажности, %

Пример использования:

Ниже приведен скрипт, который выводит в лог настройки DoorHub и статусы датчиков.

```
"""
Тестовый скрипт для doorhub

Сценарий:
```

```
- Находит первый doorhub в конфиге
- Печатает базовые параметры и состояния подузлов
- (Опционально) запускает поиск RS-HT1 и выставляет индикацию питания

"""

import rem_api
import rem_devices

rem_api.init()

try:
    devices = rem_devices.get_all()
    doorhubs = []
    for dev in devices:
        if dev.type == rem_devices.DOORHUB:
            doorhubs.append(dev)

    if len(doorhubs) == 0:
        rem_api.log("Doorhub devices not found")
    else:
        dev = doorhubs[0]
        rem_api.log("Doorhub: {}".format(dev.name))
        rem_api.log("id={} baud={} lock_addr={}
display_addr={}".format(
            dev.id, dev.baud, dev.lock_addr, dev.display_addr
        ))

        if not dev.state_loaded:
            rem_api.log("Warning: device state not loaded, report
fields may be empty")
        rem_api.log("indication: ACS={} CLIMATE={} POWER={}".format(
            dev.acs_status, dev.climat_status, dev.power_status
        ))

        door = dev.door
        rem_api.log("Door {}: open={} status={} lost={}".format(
            door.name, door.open, door.status, door.lost
        ))

        ntc = dev.internal_ntc
        rem_api.log("Internal NTC {}: temp={} state={}
lost={}".format(
            ntc.name, ntc.temp, ntc.temp_state, ntc.lost
        ))

        for i in range(rem_devices.DOORHUB_SENSOR_NUM):
            sensor = dev.sensors[i]
```

```
            rem_api.log("Sensor {} {}: template={} temp={} humid={}
th_state={} lost={}".format(
                i, sensor.name, sensor.sensor_template, sensor.temp,
sensor.humid, sensor.th_state, sensor.lost
            ))

        onewire = dev.onewire
        rem_api.log("1-Wire {}: temp={} state={} lost={}".format(
            onewire.name, onewire.temp, onewire.temp_state,
onewire.lost
        ))

        run_commands = False
        if run_commands:
            dev.rst1_search()
            dev.set_power_status(rem_devices.STATUS_NORMAL)

finally:
    rem_api.deinit()
```

9. Модуль rem_fs

Модуль используется для работы с файлами и файловой системой Контроллера. Ограничивают работу с ФС в выделенном каждому скрипту пространстве. Имена файлов указываются по относительному пути (только само имя файла). Описание функций представлено в таблице 9.1.

Таблица 9.1 – Описание функций модуля rem_api

Номер	Название	Аргументы	Описание
1.	exists()	file_name_obj – строка, имя файла	Возвращает True или False – существует ли файл с указанным именем
2.	size()	file_name_obj – строка, имя файла	Возвращает число – длину файла
3.	read(…)	file_name_obj – строка, имя файла	Возвращает строку – данные, вычитанные из файла
4.	write(…)	file_name_obj – строка, имя файла data_obj – строка, данные для записи	Записывает данные в файл. Возвращает объект none, в случае успеха, либо ошибку записи. Размер файла ограничен 1 МБ.
5.	remove(…)	file_name_obj – строка, имя файла	Удаляет файл.
6.	clear(…)	file_name_obj – строка, имя файла	Очищает содержимое файла.
7.	touch(…)	file_name_obj – строка, имя файла	Создает файл.
8.	storage_read(…)	storage_idx_obj – число, индекс накопителя file_name_obj – строка, имя файла	Считывает файл с накопителя, возвращает строку. Индекс накопителя начинается с 0, узнать его можно с помощью команды “storage” в CLI (подробнее в РЭ).

		file_size_obj – число, длина файла для чтения	
9.	storage_write(…)	storage_idx_obj – число, индекс накопителя file_name_obj – строка, имя файла data_obj – строка, данные для записи. Длина определяется автоматически	Записывает данные в файл на накопителе.
10.	storage_remove(…)	storage_idx_obj – число, индекс накопителя file_name_obj – строка, имя файла	Удаляет файл с накопителя.
11.	storage_copy(…)	storage_idx_obj – число, индекс накопителя file_name_obj – строка, имя файла	Копирует файл с ФС Контроллера на накопитель.
12.	storage_stats(…)	storage_idx_obj – число, индекс накопителя	Возвращает данные об объеме накопителя. Возвращается объект tuple: 1й элемент – число байт (полное); 2й элемент – число использованных байт; 3й элемент – число свободных байт; 4й элемент – размер блоков записи.

Пример использования:

```
"""
Пример rem_fs: работа с накопителем (USB/Storage).


Функции накопителя (требуют индекс накопителя, обычно 0 для первого
USB):
  - storage_stats(idx)        — (bytesTotal, bytesUsed, bytesFree,
blockSize)
  - storage_read(idx, name, size) — прочитать до 16 КБ из файла на
накопителе
  - storage_write(idx, name, data) — записать файл на накопитель
  - storage_remove(idx, name) — удалить файл на накопителе
  - storage_copy(idx, name)   — скопировать файл из папки скрипта на
накопитель


Сценарий: сохраняем лог-файл скрипта в локальный файл, затем копируем
на USB для выгрузки.
"""


import rem_api
import rem_fs


STORAGE_INDEX = 0  # первый накопитель (USB)
LOG_FILE = "last_run.log"
BACKUP_NAME = "script_backup.log"
```

```
rem_api.init()


try:
    # Пишем что-то в локальный файл скрипта
    rem_fs.write(LOG_FILE, "Run at ... (here could be timestamp)\n")
    rem_api.log("Локальный лог записан: {}
байт".format(rem_fs.size(LOG_FILE)))


    # Проверяем место на накопителе
    try:
        total, used, free, block = rem_fs.storage_stats(STORAGE_INDEX)
        rem_api.log("Накопитель {}: free={}
bytes".format(STORAGE_INDEX, free))
    except OSError as e:
        rem_api.log("Накопитель недоступен: {}".format(e))
        raise


    # Копируем файл из папки скрипта на накопитель
    rem_fs.storage_copy(STORAGE_INDEX, LOG_FILE)
    rem_api.log("Файл {} скопирован на накопитель".format(LOG_FILE))


    # Альтернатива: записать на накопитель свои данные напрямую
    rem_fs.storage_write(STORAGE_INDEX, BACKUP_NAME, "backup data line
1\nline 2\n")
    rem_api.log("Создан файл на накопителе: {}".format(BACKUP_NAME))


    # Прочитать обратно (макс. 16384 байт)
    data = rem_fs.storage_read(STORAGE_INDEX, BACKUP_NAME, 256)
    rem_api.log("Прочитано с накопителя: {} bytes".format(len(data)))
finally:
    rem_api.deinit()
```

10. Модуль rem_inputs

Модуль предоставляет возможность считывать состояния внутренних устройств, то есть цифровых входов DIN, аналоговых входов AIN и датчика удара (акселерометра). Кроме того, доступны для изменения их настройки.

Описание функций представлено в таблице 10.1.

Таблица 10.1 – Описание функций модуля rem_inputs

Номер	Название	Аргументы	Описание
1.	get_all(…)	subsystem_idx_obj – номер подсистемы	Получает список объектов input, представляющих датчики подсистемы subsystem. Номера подсистем: ONBOARD_SENSORS – Встроенные в плату сенсоры (датчик удара) DINS – Дискретные входы AINS – Аналоговые входы

Номер	Название	Аргументы	Описание
			Описание объекта input приведено в таблице **Ошибка! Источник ссылки не найден.**
2.	get(…)	1. subsystem_idx_obj – номер подсистемы 2. input_idx_obj – номер входа (начинается с 0)	Получает объект input. Соответствует внутреннему устройству с заданным индексом input_idx из указанной подсистемой.
3.	save(…)	input_obj – объект, хранящий настройки внутреннего устройства	Функция принимает в качестве аргумента объект input и применяет в контроллере настройки соответствующего этому объекту внутреннего устройства.

Ниже в таблице 10.2 описаны атрибуты объекта input, который можно получить с помощью функций выше.

Таблица 10.2 – Описание объекта input

Номер	Название	Тип	Описание
1.	name	string	Название внутреннего устройства
2.	subsystem	integer	Тип устройства: ONBOARD_SENSORS – Встроенные в плату сенсоры (датчик удара) DINS – Дискретные входы AINS – Аналоговые входы
3.	type	integer	Шаблон устройства. ***Для встроенных сенсоров:*** HIT_DETECTOR – Датчик удара ***Для дискретного входа:*** DIN_DOOR_NC – Дверь, НЗ DIN_DOOR_NO – Дверь, НО DIN_SENSOR_IR – Датчик движения DIN_SENSOR_SMOKE – Датчик дыма DIN_PULSE_COUNTER – Счетчик импульсов DIN_SENSOR_FLOOD_NO – Датчик протечки, НО DIN_SENSOR_FLOOD_NC – Датчик протечки, НЗ DIN_SENSOR_RSHT1 – Датчик температуры и влажности DIN_INPUT_NO – Вход, НО DIN_INPUT_NC – Вход, НЗ ***Для аналогового входа:*** AIN_SENSOR_SMOKE – Датчик дыма AIN_NAMUR_DOOR – NAMUR дверь AIN_SENSOR_FLOOD – Датчик протечки, аналоговый AIN_BLOCKER – Аналоговый ключ работы электронного замка
4.	enabled	boolean	Если True, то устройство включено
5.	monitor	boolean	Если True, то устройство отображается в панели мониторинга
6.	send_emails	boolean	Если True, то устройство будет отсылать Email-уведомления при изменении состояния
7.	send_traps	boolean	Если True, то устройство будет отсылать trap-уведомления при изменении состояния
8.	group	integer	Группа устройства: GROUP_24_7 – 24/7 GROUP_GUARD – Охрана GROUP_INFO – Информационная GROUP_ENTRANCE – Входная
9.	status	integer	Статус устройства: STATUS_NORMAL – Норма STATUS_ALMIN – Незначительная авария STATUS_ALMAJ – Критическая авария
10.	settings_loaded	boolean	Если True, то настройки устройства загружены в объект. Не у всех датчиков есть настройки (см. ниже)

Номер	Название	Тип	Описание
11.	state_loaded	boolean	Если True, то состояние устройства загружено в объект. Состояние отображается только для включенных в настройках датчиков
12.	impact_duration	integer	Настройки датчика удара: длительность удара, с
13.	impact_strength	integer	Настройки датчика удара: сила удара, усл. ед.
14.	temp_alarm_high	float	Настройки датчика температуры и влажности: верхняя граница аварийной температуры, ℃
	temp_alarm_low	float	Настройки датчика температуры и влажности: нижняя граница аварийной температуры, ℃
	temp_warn_high	float	Настройки датчика температуры и влажности: верхняя граница температуры предупреждения, ℃
	temp_warn_low	float	Настройки датчика температуры и влажности: нижняя граница температуры предупреждения, ℃
	humid_alarm_high	float	Настройки датчика температуры и влажности: верхняя граница аварийной влажности, %
	humid_alarm_low	float	Настройки датчика температуры и влажности: нижняя граница аварийной влажности,
	humid_warn_high	float	Настройки датчика температуры и влажности: верхняя граница влажности предупреждения, %
	humid_warn_low	float	Настройки датчика температуры и влажности: нижняя граница влажности предупреждения, %
	temp_hyst	float	Настройки датчика температуры и влажности: гистерезис температуры, ℃
	humid_hyst	float	Настройки датчика температуры и влажности: гистерезис влажности, %
15.	value	float	Значение, считываемое аналоговым входом
16.	triggered	boolean	Если True, то состояние дискретного входа было изменено
17.	hit_detected	boolean	Если True, то датчик удара зафиксировал удар
18.	temp	float	Значение температуры, измеряемое датчиком температуры и влажности, ℃
	hyst	float	Значение влажности, измеряемое датчиком температуры и влажности, %
19.	dout_enabled	boolean	Если True, то дискретный вход начинает работать как дискретный выход

Пример использования:

Скрипт контролирует состояние датчика дыма, датчика удара и датчика температуры и влажности. При превышении порога по температуре, которая получена с датчика RSHT1, выключается розетка, в которую включен нагреватель. Если температура падает ниже другого порога, розетка, наоборот, включается.

Если в течении 5 проходов цикла замечен удар на датчике удара, то выводится сообщение о том, что началось землетрясение.

Также скрипт включает устройство пожаротушения на розетке 5, если аналоговый датчик дыма обнаружил задымление, и отключает его, если задымления нет.

```
import time
import rem_api
import rem_inputs
import rem_sockets

rem_api.init()
rem_api.log("script 1 started")

counter = 0
fire_detector_threshold = 20 #mA
```

```
while True:


   hit_detector = rem_inputs.get(rem_inputs.ONBOARD_SENSORS, 1)
   if isinstance(hit_detector, rem_inputs.input):
         if hit_detector.type == rem_inputs.HIT_DETECTOR and
hit_detector.enabled and hit_detector.state_loaded:
               if hit_detector.hit_detected and
hit_detector.impact_strength > 10 and hit_detector_counter > 5:
                     rem_api.log("Earthquake started!")
               elif hit_detector.hit_detected and
hit_detector.impact_strength > 10:
                     hit_detector_counter += 1
               elif
                     hit_detector_counter = 0


   rsht1 = rem_inputs.get(rem_inputs.DINS, 1)
   if isinstance(rsht1, rem_inputs.input):
         socket = rem_sockets.get(0, 0)
         if rsht1.type == rem_inputs. DIN_SENSOR_RSHT1 and
rsht1.enabled and rsht1.state_loaded:
               if rsht1.temp > 45.0:
                     # Отключаем розетку 1, в которую включен
нагреватель
                     socket.enabled = False
               elif rsht1.temp < 20.0:
                     # Включаем розетку 1, в которую включен
нагреватель
                     socket.enabled = True


   fire_detector = rem_inputs.get(rem_inputs.AINS, 2)
   if isinstance(fire_detector, rem_inputs.input):
         if fire_detector.type == rem_inputs.AIN_SENSOR_SMOKE and
fire_detector.enabled and fire_detector.state_loaded:
               socket = rem_sockets.get(0, 4)
               if fire_detector.value < fire_detector_threshold:
                     # Включаем условное устройство пожаротушения на
пятой розетке
                     socket.enabled = True
                     rem_api.log("Fire started at facility!")
               else:
                     socket.enabled = False
                     rem_api.log("Fire was extinguished!")
   time.sleep(5)
```

Второй пример:

Скрипт демонстрирует работу с дискретными выходами Контроллера. Можно использовать для управления подключенными к ним реле и датчиками.

```
import rem_api
import rem_inputs
import time
```

```
rem_api.init()
rem_api.log("dout test script launched")

toggleBack = False

# Получаем цифровой вход
din = rem_inputs.get(rem_inputs.DINS, 0)

# Проверяем что DIN выключен
if din.enabled:
    # Выключаем DIN
    din.enabled = False
    rem_inputs.save(din)
    toggleBack = True
    time.sleep(5)

# Теперь можем управлять DOUT
rem_api.log("dout 1 enabled")
din.dout_enabled = True  # Включить DOUT

time.sleep(30)

rem_api.log("dout 1 disabled")
din.dout_enabled = False  # Выключить DOUT

# ВАЖНО: При включении DIN обратно, DOUT автоматически переключается
# в режим питания для DIN и ручное управление прекращается
if toggleBack:
    din.enabled = True
    rem_inputs.save(din)
```

11. Модуль rem_pdu

Модуль нужен для получения доступа к измерениям электропитания PDU, а также разных данных и настроек Контроллера. Описание функций представлено в таблице 11.1. Описание объектов модуля приведены в таблицах 11.2, 11.3, 11.4.

Таблица 11.1 – Описание функций модуля rem_pdu

Номер	Название	Аргументы	Описание
1.	get()	Нет	Возвращает объект pdu, который содержит данные о контроллере: серийный номер, аппаратные и программные версии, модель, статус, сетевые настройки, статус электропитания и потребления.
2.	broadcast(…)	msg_obj – строка, которую необходимо вывести в WEB-интерфейсе или CLI как уведомление	Отправляет уведомление в WEB и CLI.

Таблица 11.2 – Описание объекта pdu

Номер	Название	Тип	Описание
1.	serial_number	string	Строка с серийным номером PDU (Контроллера)
2.	mac_address	string	MAC-адрес первого интерфейса Контроллера
3.	sw_version	string	Версия ПО Контроллера
4.	hw_version	string	Аппаратная версия Контроллера
5.	model	string	Модель Контроллера
6.	phase_number	int	Число фаз Контроллера
7.	bank_number	int	Число контуров Контроллера
8.	status	int	Статус Контроллера: STATUS_NORMAL – нормальный статус; STATUS_ALMIN – предупреждение; STATUS_ALMAJ – авария.
9.	name	string	Имя Контроллера
10.	address	string	Адрес установки Контроллера
11.	owner	string	Владелец Контроллера
12.	responsible	string	Ответственный за Контроллер
13.	installer	string	Установщик Контроллера
14.	ipv4	string	IPv4-адрес первого LAN-интерфейса Контроллера
15.	ipv4_mask	string	IPv4-маска первого LAN-интерфейса Контроллера
16.	ipv4_gateway	string	IPv4-шлюз первого LAN-интерфейса Контроллера
17.	dhcp_enabled	bool	True, если включено получение сетевых настроек по DHCP
18.	ipv6_enabled	bool	True, если включена поддержка IPv6-адреса
19.	ipv6	string	IPv6-адрес первого LAN-интерфейса Контроллера
20.	ipv6_mask	int	IPv6-маска первого LAN-интерфейса Контроллера, задается числом
21.	ipv6_gateway	string	IPv4-шлюз первого LAN-интерфейса Контроллера
22.	ipv6_dhcp	bool	True, если включено получение сетевых настроек по DHCP
23.	phases	list[dict]	Объект, представляющий измерения по фазам Контроллера
24.	banks	list[dict]	Объект, представляющий измерения по контурам Контроллера

Таблица 11.3 – Описание объекта phases

Номер	Название	Тип	Описание
1.	status	int	Статус фазы (доступ по индексу)
2.	voltage	float	Напряжение на фазе, В
3.	current	float	Ток фазы, А
4.	power	float	Мощность на фазе, кВт
5.	energy	int	Накопленная энергия на фазе, Вт*ч

Таблица 11.4 – Описание объекта banks

Номер	Название	Тип	Описание
1.	status	int	Статус контура (доступ по индексу)
2.	voltage	float	Напряжение на контуре, В
3.	current	float	Ток контуре, А
4.	power	float	Мощность на контуре, кВт
5.	energy	int	Накопленная энергия на контуре, Вт*ч

Пример использования:

```
import rem_pdu
import time

while True:
    try:
        pdu = rem_pdu.get()

        # Проверка статусов
        for phase in pdu.phases:
```

```
            if phase.status != rem_pdu.STATUS_NORMAL:
                rem_pdu.broadcast(f"Внимание: аварийный статус фазы!")

        for i, phase in pdu.phases:
            rem_api.log(f"Фаза {i+1}: {phase.voltage}V,
{phase.current}A")

        rem_api.log(f"Серийный номер: {pdu.serial_number}")
        rem_api.log(f"MAC адрес: {pdu.mac_address}")
        rem_api.log(f"Версия ПО: {pdu.sw_version}")
        rem_api.log(f"Версия АО: {pdu.hw_version}")
        rem_api.log(f"Модель: {pdu.model}")
        rem_api.log(f"Фаз: {pdu.phase_number}, Банков:
{pdu.bank_number}")
        rem_api.log(f"Название: {pdu.name}")
        rem_api.log(f"IP адрес: {pdu.ipv4}")
        time.sleep(5)

    except Exception as e:
        rem_api.log(f"Ошибка: {e}")
        time.sleep(10)
```

12. Модуль rem_scripts

Модуль позволяет работать с загруженными скриптами, загружать скрипты, отключать, переименовывать их и удалять.

Описание функций представлено в таблице 12.1. Описание объекта script, полученного функциями ниже можно увидеть в таблице 12.2.

Таблица 12.1 – Функции модуля rem_scripts

Номер	Название	Аргументы	Описание
1.	get_all()	Нет	Получает список объектов script, описание которого приведено в таблице **Ошибка! Источник ссылки не найден.**.
2.	get(…)	script_idx_obj – номер скрипта в настройках (начинается с 0)	Получает объект script. Соответствует скрипту с заданным индексом.
3.	save(…)	script_obj – объект типа script	Сохраняет настройки соответствующего скрипта в соответствии со значениями, переданными с объектом script
4.	remove(…)	script_obj – объект типа script	Удаляет скрипт из конфигурации и с файловой системы
5.	load_tftp(…)	script_name_obj – имя скрипта на TFTP-сервере	Загружаем скрипт под указанным именем с TFTP-сервера и добавляет его в конфигурацию
6.	load_storage(…)	1 – script_name_obj – имя скрипта на накопителе 2 – storage_idx_obj – индекс накопителя (начинается с нуля)	Загружаем скрипт под указанным именем со внешнего накопителя под индексом storage_idx и добавляет его в конфигурацию

Таблица 12.2 – Описание объекта script модуля rem_scripts

Номер	Название	Тип	Описание
1.	name	String	Имя скрипта.

Номер	Название	Тип	Описание
2.	enabled	boolean	Настройка, отвечающая за включение и выключение скрипта. Применяется сразу, нет необходимости вызывать save()
3.	comment	String	Комментарий к скрипту, максимальная длина – 20 символов.

Пример использования:

```
import rem_scripts
import rem_sockets
import rem_api
import time

# Инициализируем API
rem_api.init()

# Конфигурация
TARGET_SCRIPT = "traffic_light.py"
FALLBACK_SCRIPT = "cleanup.py"
CHECK_INTERVAL = 60

# Функция получает скрипт по имени
def get_script(name):
    scripts = rem_scripts.get_all()
    for script in scripts:
        if script.name == name:
            return script
    return None

# Функция включает все розетки контроллера
def enable_all_sockets():
    try:
        sockets = rem_sockets.get_all(0)
        for socket in sockets:
              if socket.has_control and not socket.enabled:
                    rem_api.log(f"Enabling socket {socket.name}")
                    socket.enabled = True
                    rem_sockets.save(socket)
    except Exception as e:
        rem_api.log(f"Error enabling sockets: {str(e)}")

def main():
    last_state = None

    while True:
        try:
            # Проверяем статус нужного скрипта
            target_script = get_script(TARGET_SCRIPT)
            fallback_script = get_script(FALLBACK_SCRIPT)
```

```
            if target_script is None:
                rem_api.log(f"Warning: Target script {TARGET_SCRIPT}
not found")
            else:
                current_state = not target_script.enabled

                # Если статус нужного скрипта изменился
                if current_state != last_state:
                    if current_state:
                        # Этот скрипт выключили
                        rem_api.log(f"Target script {TARGET_SCRIPT} is
disabled, activating fallback")

                        # Включаем другой скрипт
                        if fallback_script and not
fallback_script.enabled:
                            fallback_script.enabled = True
                            rem_api.log(f"Enabled fallback script
{FALLBACK_SCRIPT}")

                        # Включаем все розетки
                        enable_all_sockets()
                    else:
                        # Скрипт все еще работает
                        rem_api.log(f"Target script {TARGET_SCRIPT} is
enabled, returning to normal operation")

                    last_state = current_state

            time.sleep(CHECK_INTERVAL)

        except Exception as e:
            rem_api.log(f"Error in main loop: {str(e)}")
            time.sleep(CHECK_INTERVAL)

if __name__ == "__main__":
    rem_api.log("Starting script monitor")
    main()
```

13. Модуль rem_sockets

Модуль предоставляет возможность настраивать розетки PDU и CPDU, а также считывать их состояния и показания измерителей.

Описание функций модуля представлено в таблице 13.1.

Таблица 13.1 – Описаний функций модуля rem_sockets

Номер	Название	Аргументы	Описание
1.	get_all(…)	pdu_idx_obj – номер PDU	Получает список объектов socket, описание которого приведено в таблице **Ошибка! Источник ссылки не найден.**. Элементы этого списка – все объекты,

Номер	Название	Аргументы	Описание
			представляющие розетки Контроллера (если pdu_idx == 0) или CPDU с индексом pdu_idx (pdu_idx > 0).
2.	get(…)	1. pdu_idx_obj – номер PDU 2. socket_idx_obj – номер розетки	Возвращает объект socket, представляющий розетку Контроллера (CPDU) по индексу pdu_idx. Номер розетки начинается с 0.
3.	save(…)	Объект socket_obj, с которым ведется работа	Сохраняет настройки розетки. Возвращает: -1 – Если тип переданного в функцию объекта не является типом «socket» -2 – Если по какой-то причине не удалось сохранить настройки розетки (внутренняя ошибка) 0 – Успешное сохранение

Объект socket содержит отчет о состоянии розетки (измеренные ток, напряжение и мощность) и ее настройки. Подробное описание в таблице 13.2.

Таблица 13.2 – Описание объекта socket

Номер	Название	Тип	Описание
1.	name	String	Имя розетки (максимум 20 символов)
2.	enabled	boolean	Настройка, отвечающая за включение и выключение розетки
3.	restart_time_sec	Integer	Время перезапуска розетки во время работы Ping Watchdog, указывается в секундах (максимальное значение – 255)
4.	watchdog_enabled	boolean	Настройка, отвечающая за включение и исключение розетки в мониторинге Ping Watchdog
5.	status	Integer	Статус розетки: STATUS_NORMAL – Норма STATUS_ALMIN – Незначительная авария STATUS_ALMAJ – Критическая авария
6.	has_control	boolean	Флаг, показывающий можно ли управлять (включать/выключать) розеткой
7.	no_connection	boolean	Флаг, показывающий есть ли связь с розеткой. Если флаг True, связь с розеткой потеряна
8.	current	float	Ток на розетке, А
9.	voltage	float	Напряжение на розетке, В
10.	power	float	Мощность на розетке, кВт

Пример использования:

```
"""
Пример для rem_sockets + rem_inputs:


Отслеживание датчика DIN1 с шаблоном RSHT1 и управление Розеткой 1.


Логика:
- если у RSHT1 установился статус "Авария" (PDU_STATUS_ALMAJ),
  проверяем, что эта авария вызвана именно температурой (а не влажностью),
  и включаем розетку 1.
- авария ушла (по внутреннему гистерезису RSHT1) -> выключаем розетку 1.


Примечания:
- Нумерация DIN/розеток в UI обычно 1..N, а индексы API 0..N-1.
- Этот пример использует DIN_NUM=1 и SOCKET_NUM=1 и переводит в индексы автоматически.
```

```
"""

import time

import rem_api
import rem_inputs
import rem_sockets


# --- Настройки под контроллер ---
DIN_NUM = 1          # DIN1
PDU_IDX = 0          # 0 = основной PDU
SOCKET_NUM = 1       # Розетка 1
POLL_SEC = 1.0       # период опроса, сек

DIN_IDX = DIN_NUM - 1
SOCKET_IDX = SOCKET_NUM - 1


def _setup_din1_as_rsht1():
    """
    Инициализация при загрузке:
    - переводим DIN1 в шаблон RSHT-1
    - выставляем пороги:
      влажность аварии: 1%
      влажность предупреждения: 5%
      температура аварии: 35°C
      температура предупреждения: 30°C
    - имя: "RSHT-1"
    - включаем вход и сохраняем конфигурацию
    """
    din = rem_inputs.get(rem_inputs.DINS, DIN_IDX)
    rem_api.log("Инициализация DIN{} как RSHT-1".format(DIN_NUM))

    din.type = rem_inputs.DIN_SENSOR_RSHT1
    din.name = "RSHT-1"
    din.enabled = True
    din.monitor = True

    # Пороги по влажности (низкие значения)
    din.humid_alarm_low = 1.0
    din.humid_warn_low = 5.0

    # Пороги по температуре (высокие значения)
    din.temp_alarm_high = 35.0
    din.temp_warn_high = 30.0
```

```
    rem_inputs.save(din)
    rem_api.log("DIN{} настроен как RSHT-1 и конфигурация
сохранена".format(DIN_NUM))



def _rsht_value_status_get(cur_status, value, alarm_high, warn_high,
alarm_low, warn_low, hyst):
    """
    Повторяет логику pdu-service-inputs для RSHT1 (см. din.c:
rshtValueStatusGet()).
    Возвращает один из rem_inputs.STATUS_* (PDU_STATUS_*).
    """
    # ALMAJ с учетом гистерезиса
    if (
        (value > alarm_high)
        or (cur_status == rem_inputs.STATUS_ALMAJ and value >
(alarm_high - hyst))
        or (value < alarm_low)
        or (cur_status == rem_inputs.STATUS_ALMAJ and value <
(alarm_low + hyst))
    ):
        return rem_inputs.STATUS_ALMAJ

    # ALMIN с учетом гистерезиса
    if (
        (value > warn_high)
        or (cur_status == rem_inputs.STATUS_ALMIN and value >
(warn_high - hyst))
        or (value < warn_low)
        or (cur_status == rem_inputs.STATUS_ALMIN and value <
(warn_low + hyst))
    ):
        return rem_inputs.STATUS_ALMIN

    return rem_inputs.STATUS_NORMAL



def _status_to_str(status):
    """
    Преобразование enum `rem_inputs.STATUS_*` в читаемую строку для
логов.
    """
    if status == rem_inputs.STATUS_NORMAL:
        return "НОРМА"
    if status == rem_inputs.STATUS_ALMIN:
        return "ПРЕДУПРЕЖДЕНИЕ"
    if status == rem_inputs.STATUS_ALMAJ:
        return "АВАРИЯ"
    return "НЕИЗВЕСТНО({})".format(status)
```

```
def main():
    rem_api.init()
    rem_api.log("Скрипт мониторинга климата запущен!")

    # Ведем свой temp_status, чтобы корректно учитывать гистерезис при
определении причины аварии
    temp_status = rem_inputs.STATUS_NORMAL

    try:
        _setup_din1_as_rsht1()
        while True:
            din = rem_inputs.get(rem_inputs.DINS, DIN_IDX)

            # Проверяем что это действительно RSHT1
            if din.type != rem_inputs.DIN_SENSOR_RSHT1:
                time.sleep(POLL_SEC)
                continue

            # Если нет данных/настроек, не предпринимаем действий
            if not din.state_loaded or not din.settings_loaded:
                time.sleep(POLL_SEC)
                continue

            # Если связь потеряна, это тоже может быть ALMAJ, но нам
нужна авария именно по температуре
            if din.state == rem_inputs.DIN_CONNECTION_LOST:
                should_on = False
            else:
                t = float(din.temp)

                # Уставки температуры (°C)
                t_alarm_high = float(din.temp_alarm_high)
                t_alarm_low = float(din.temp_alarm_low)
                t_warn_high = float(din.temp_warn_high)
                t_warn_low = float(din.temp_warn_low)
                t_hyst = float(din.temp_hyst)

                # Обновляем оценку temp_status с учетом гистерезиса
                temp_status = _rsht_value_status_get(
                    temp_status, t, t_alarm_high, t_warn_high,
t_alarm_low, t_warn_low, t_hyst
                )

                # Включаем розетку только если:
                # - общий статус входа = ALMAJ ("Авария")
                # - и именно температура сейчас в ALMAJ по тем же
правилам
```

```
                should_on = (din.status == rem_inputs.STATUS_ALMAJ)
and (temp_status == rem_inputs.STATUS_ALMAJ)

            sock = rem_sockets.get(PDU_IDX, SOCKET_IDX)
            if not sock.has_control:
                rem_api.log(
                    "Группа розеток недоступна для управления"
                )
                time.sleep(POLL_SEC)
                continue

            if bool(sock.enabled) != bool(should_on):
                sock.enabled = bool(should_on)
                rem_api.log(
                    "Розетка {} -> {} (DIN{} status={},
temp_status={})".format(
                        SOCKET_NUM,
                        "ВКЛ" if should_on else "ВЫКЛ",
                        DIN_NUM,
                        _status_to_str(din.status),
                        _status_to_str(temp_status),
                    )
                )

            time.sleep(POLL_SEC)

    finally:
        rem_api.log("Скрипт мониторинга климата остановлен")
        rem_api.deinit()

main()
```