Группа компаний "Эликс"

Отдел разработки интеллектуальных систем безопасности

Описание открытого текстового протокола конфигурирования настольного считывателя PW-Desktop BLE v2

2025

Оглавление

Сброс считывателя в заводское состояние

Нужно отправить в COM-порт последовательно три команды:

```
"do emerg_bypass\n"
"do dropdown\n"
"do reboot\n"
```

Считыватель сбросит себя в заводское состояние и перезагрузится. Должно помочь решить подавляющее большинство проблем.

Внимание! Со считывателя будут безвозвратно удалены все настройки и ячейки программирования! Абсолютно все! Абсолютно безвозвратно!

1. Описание работы считывателя по USB-CDC (COM-порт)

USB-CDC - это реализация в USB обмена данных, аналогичного COM-порту. При этом в компьютере настольный считыватель представляется как COM-порт и не требует никаких дополнительных драйверов.

Текстовый протокол открытый и предназначен для интеграции считывателя в другие системы, а так же ручного управления считывателем при помощи использования COM-порта. Отправляемые команды или внесение изменений в настройки считывателя вступают в силу моментально. Далее будем описывать именно текстовый протокол.

2. Структура команд и ответов текстового протокола

Текстовый протокол в своём составе имеет укороченные и полные команды. Укороченным командам посвящён один подраздел 2.1, а полные команды рассматриваются во всём остальном документе.

Также в этом разделе находится информация о том, в каком виде считыватель формирует ответы на команды, а так же сообщения, отправляемые считывателем при возникновении на нём событий.

2.1 Укороченные команды

Текстовый протокол имеет три укороченные команды. Укороченные команды завершаются знаком переноса строки (`'\n'`) или знаком возврата каретки (`'\r'`). Допустимо использовать оба знака в любой последовательности.

2.1.1 `i`

`i` - команда получения информации о считывателе. Считыватель вернёт четыре строки с названием считывателя, серийным номером и двумя версиями установленных прошивок. Ответ выглядит примерно так:

```
"PW-DesktopV2\r\nSERIAL 03:01:00:00:00:01\r\nVERSION 00.04\r\nBLE_VER 00.28\r\n"
```

То же, но в более читаемом виде:

```
PW-DesktopV2
SERIAL 03:01:00:00:00:01
VERSION 00.04
BLE_VER 00.28
```

2.1.2 r

r - команда для прочтения лежащей на считывателе карты. Возвращает одну строчку с информацией о карте. Ответ начинается с информации о физическом типе идентификатора (RFID_13.56, RFID125, MobileID, PWtag). В квадратных скобках указывается сам идентификатор в том виде, в котором он будет передан по интерфейсу Wiegand. Затем указывается Facility-код в стандартном виде. В конце уточняется тип идентификатора. Ответ выглядит примерно так:

```
"RFID_13.56[0000001216aae1] 022,43745 MIFARE_Classic_1K_4b_SL1\r\n"
```

Если на момент получения команды "r" на считывателе нет карты - считыватель ответит "No card!".

2.1.3 k

k - команда переключения режима клавиатуры. Возвращает одну строчку с текущим статусом режима клавиатуры. Ответы выглядят так:

```
"Keyboard mode ON\r\n"
"Keyboard mode OFF\r\n"
```

2.2 Полные команды

Полные команды текстового протокола состоят из заголовка, адреса, параметра и значения. У некоторых команд некоторых элементов может не быть. Элементы команды разделяются пробелами. Считыватель завершает ответы на команды парой символов "\r\n".

У текстового протокола три типа заголовков - get, set, do.

`get` - команды для получения настроек и ячеек программирования из считывателя.

`set` - команды для установки настроек и ячеек программирования в считыватель.

`do` - команды для выполнения действий считывателем.

Заголовок `get` дополняется адресом и параметром. Также существуют безадресные команды с заголовком `get`.

Заголовок `set` дополняется адресом, параметром и значением.

Заголовок `do` дополняется действием, и для некоторых действий дополняется значением.

В протоколе применяются 9 адресов - настройки непосредственно считывателя и 8 ячеек программирования: `config`, `prog1`, `prog2`, `prog3`, `prog4`, `prog5`, `prog6`, `prog7`, `prog8`.

Параметры будут описаны ниже, отдельно для каждого адреса.

Значения, устанавливаемые для параметров, могут быть различных типов. Двоичный (битовый) флаг (0 или 1), двоичная (битовая) маска (набор нулей и единиц), десятичное число (набор десятичных цифр), шестнадцатеричное число (набор шестнадцатеричных цифр), или строка (набор символов). Иногда значения бывают составными (состоящими из нескольких чисел, в том числе разных типов), подробности раскрыты в описании соответствующих команд.

Примеры команд:

```
"get config BLE_PWtag_active\n"
"set config beeper_active 1\r"
"get prog1 ident_type\r\n"
"set prog2 keyA_bytes 0123456789AB\n\r"
"do read_card\n"
"do write_card 1 C0FFEE\r"
"do reboot\n\r"
```

Для выполнения команд и настроек (кроме сброса считывателя в заводское состояние, описанного в начале документа) требуется провести авторизацию с ключом инженера. Авторизация описана в пункте 6.1.

2.3 Формирование ответа на полные команды

Ответы на полные команды формируются как повтор команды, добавление знака двоеточия, пробела, и указание непосредственно ответа.

В непосредственно ответе считыватель может прислать либо какие-то данные, либо статус. Тип данных в ответе соответствует типу данных, используемому данной командой при записи, либо это дополнительно указано в данном документе. Также считыватель может ответить статусом. Статусы описаны ниже.

`OK` - команда успешно выполнена.

`NOT_ALLOWED` - действие в команде запрещено к выполнению. Возможно, не была выполнена авторизация ключом инженера, или выполнение команды запрещено по соображениям безопасности.

`NOT_SUPPORTED` - действие в команде не поддерживается. Например, команда на включение интерфейса 125 кГц, в то время как он аппаратно отсутствует.

`WRONG_HEADER` - неверный заголовок команды.

`WRONG_ADDRESS` - неверный адрес команды.

`WRONG_PARAMETER` - неверный параметр команде.

`WRONG_ACTION` - неверное действие.

`WRONG_TYPE` - неверный тип данных в значении.

`WRONG_LEN` - неверная длина данных в значении.

`WRONG_VALUE` - неверное значение данных (например, выходящее за пределы допустимого диапазона).

`PROG_READ_ERROR` - ошибка чтения ячейки программирования.

`PROG_EMPTY` - ячейка программирования не запрограммирована.

NONE - пустой ответ. Например, если название ячейки программирования удалено или не указано.

UNKNOWN_ERROR - неизвестная ошибка.

2.4 Сообщения с информированием о событиях

При считывании идентификатора считыватель самостоятельно передаёт информацию о прочитанном идентификаторе в COM-порт. Формат выдачи:

`тип_идентификатора[данные] фасилити,код (уточнённый_идентификатор) (RSSI)`

Тип идентификатора - указывает на физический тип носителя идентификатора. Применяются: `RFID125`, `RFID1356`, `MobileID`, `PWtag`.

Данные - код идентификатора в шестнадцатеричном формате. Соответствуют данным, передаваемым по Wiegand.

Фасилити код - фасилити код. Трёхзначное десятичное число, запятая и многозначное десятичное число. Без пробелов.

Уточнённый идентификатор - уточняет тип носителя идентификатора. Для `RFID1356` указывается тип карты, для Mobile ID указывается тип устройства (Android или Apple).

RSSI - указывает на значение RSSI для идентификаторов типов Mobile ID и PW-Tag. Позволяет приблизительно оценить расстояние до идентификатора.

3. Безадресные команды get

Безадресные команды с заголовком get применяются для получения информации о считывателе в целом.

3.1 device_name

device_name - Название устройства. Ответ представляет из себя строку с названием устройства.

Примеры использования команды:

```
"get device_name\n"
```

Пример ответа считывателя:

```
"get device_name: PW_DesktopV2\r\n"
```

3.2 device_version

device_version - Версия прошивки устройства. Ответ представляет из себя строку с версией прошивки устройства.

Примеры использования команды:

```
"get device_version\n"
```

Пример ответа считывателя:

```
"get device_version: 1.0.0\r\n"
```

3.3 device_serial

`device_serial` - Серийный номер устройства. Ответ представляет из себя строку с серийным номером устройства.

Примеры использования команды:

```
"get device_serial\n"
```

Пример ответа считывателя:

```
"get device_serial: 03:01:00:00:00:01\r\n"
```

3.4 device_supported_idents

`device_supported_idents` - Идентификаторы, которые поддерживаются устройством. Возвращает восемь двоичных цифр, указывающих на поддержку или не поддержку идентификаторов заданного типа. По номерам цифр:

1) EM-Marine;
2) HID Prox;
3) INDALA;
4) Mifare/Mikron Ultralight/SL1/SL3 (RFID1356);
5) ICODE;
6) Mifare Desfire;
7) BLE (Mobile-ID, PW-Tag);
8) Банковские карты / NFC.

Команда:

```
"get device_supported_idents\r\n"
```

Пример ответа считывателя:

```
"get device_supported_idents: 00010110\r\n"   // Поддерживаются RFID1356
                              // Mifare Desfire и BLE (Mobile-ID, PW-Tag)
```

3.5 progs_used

progs_used - Запрашивает перечень занятых и свободных ячеек программирования. Команда возвращает восемь двоичных цифр, обозначающие занятость или свободность ячейки программирования. 1 - ячейка занята, 0 - ячейка свободна.

Команда:

```
"get progs_used\n"
```

Пример ответа считывателя:

```
"get progs_used: 11111100\r\n"   // Заняты ячейки 1, 2, 3, 4, 5 и 6.
                                 //                Свободны ячейки 7 и 8.
```

4. Описание параметров настроек считывателя (config)

Настройки непосредственно считывателя (config) имеют много параметров. Ниже они все описаны, представлены примеры их использования с комментариями и типичные ответы считывателя с комментариями.

4.1 BLE_PWtag_active

BLE_PWtag_active - Разрешено ли читать идентификаторы типа PW-Tag. Одна двоичная цифра: 0 - нет, 1 - да. Если установлено 1, есть настроенные ячейки программирования, но нет ячейки PW-Tag - чтение PW-Tag происходить не будет.

Примеры использования команды:

```
"get config BLE_PWtag_active\n"     // Проверяем разрешение PW-Tag
"set config BLE_PWtag_active 1\n"   // Разрешаем читать PW-Tag
"set config BLE_PWtag_active 0\n"   // Запрещаем читать PW-Tag
```

Примеры ответов считывателя:

```
"get config BLE_PWtag_active: 1\r\n"      // Разрешено читать PW-Tag
"set config BLE_PWtag_active 1: OK\r\n"   // Разрешили читать PW-Tag
"set config BLE_PWtag_active 0: OK\r\n"   // Запретили читать PW-Tag
```

4.2 BLE_Android_active

BLE_Android_active - Разрешено ли читать идентификаторы типа Mobile ID на устройствах с операционной системой Android. Одна двоичная

цифра: 0 - нет, 1 - да. Если установлено 1, есть настроенные ячейки программирования, но нет ячейки Mobile ID - чтение происходить не будет.

Примеры использования команды:

```
"get config BLE_Android_active\n"     // Проверяем разрешение Android
"set config BLE_Android_active 1\n"   // Разрешаем читать Android
"set config BLE_Android_active 0\n"   // Запрещаем читать Android
```

Примеры ответов считывателя:

```
"get config BLE_Android_active: 1\r\n"      // Разрешено читать Android
"set config BLE_Android_active 1: OK\r\n"   // Разрешили читать Android
"set config BLE_Android_active 0: OK\r\n"   // Запретили читать Android
```

4.3 BLE_Apple_active

`BLE_Apple_active` - Разрешено ли читать идентификаторы типа Mobile ID на яблофонах. Одна двоичная цифра: 0 - нет, 1 - да. Если установлено 1, есть настроенные ячейки программирования, но нет ячейки Mobile ID - чтение происходить не будет.

Примеры использования команды:

```
"get config BLE_Apple_active\n"     // Проверяем разрешение Apple
"set config BLE_Apple_active 1\n"   // Разрешаем читать Apple
"set config BLE_Apple_active 0\n"   // Запрещаем читать Apple
```

Примеры ответов считывателя:

```
"get config BLE_Apple_active: 1\r\n"      // Разрешено читать Apple
"set config BLE_Apple_active 1: OK\r\n"   // Разрешили читать Apple
"set config BLE_Apple_active 0: OK\r\n"   // Запретили читать Apple
```

4.4 RFID1356_active

`RFID1356_active` - Задействован ли интерфейс 13,56 МГц (карты и брелки Mifare, Infineon и Mikron). Данный параметр разрешает, либо

запрещает использовать аппаратный интерфейс 13,56 МГц. Одна двоичная цифра: 0 - нет, 1 - да. Если установлено 1, но нет настроенных ячеек программирования типа Mifare - чтение будет происходить с настройками по умолчанию.

Примеры использования команды:

```
"get config RFID1356_active\n"     // Проверяем интерфейс 13,56 МГц
"set config RFID1356_active 1\n"   // Включаем интерфейс 13,56 МГц
"set config RFID1356_active 0\n"   // Выключаем интерфейс 13,56 МГц
```

Примеры ответов считывателя:

```
"get config RFID1356_active: 1\r\n"      // Интерфейс 13,56 МГц включен
"set config RFID1356_active 1: OK\r\n"   // Включили интерфейс 13,56 МГц
"set config RFID1356_active 0: OK\r\n"   // Выключили интерфейс 13,56 МГц
```

4.5 RFID125_active

RFID125_active - Задействован ли интерфейс 125 кГц (карты и брелки EM-Marine, HID Prox, INDALA). Данный параметр разрешает, либо запрещает использовать аппаратный интерфейс 125 кГц. Одна двоичная цифра: 0 - нет, 1 - да. На текущий момент не поддерживается.

Примеры использования команды:

```
"get config RFID125_active\n"     // Проверяем интерфейс 125 кГц
"set config RFID125_active 1\n"   // Включаем интерфейс 125 кГц
"set config RFID125_active 0\n"   // Выключаем интерфейс 125 кГц
```

Примеры ответов считывателя:

```
"get config RFID125_active: 0\r\n"      // Интерфейс 125 кГц выключен
"set config RFID125_active 1: OK\r\n"   // Включили интерфейс 125 кГц
"set config RFID125_active 0: OK\r\n"   // Выключили интерфейс 125 кГц
"set config RFID125_active 1: NOT_SUPPORTED\r\n"
                      // Интерфейс 125 кГц не поддерживается устройством
```

4.6 beeper_volume

beeper_volume - Громкость бипера. Десятичное число от 0 до 255. При установке считыватель пискнет.

Примеры использования команды:

```
"get config beeper_volume\n"       // Проверяем громкость
"set config beeper_volume 32\n"    // Тихо пищать
"set config beeper_volume 64\n"    // Нормально пищать
"set config beeper_volume 255\n"   // Громко пищать
```

Примеры ответов считывателя:

```
"get config beeper_volume: 64\r\n"                 // Стоит громкость 64
"set config beeper_volume 32: OK\r\n"              // Поставили громкость
                                                   //                  32
"set config beeper_volume 256: WRONG_VALUE\r\n"    // 256 - неправильная
                                                   //            громкость
```

4.7 beeper_freq

beeper_freq - Частота бипера в герцах. Десятичное число. Рекомендуется от 1000 до 5000. Допустимо от 100 до 15 000. Рекомендуется ставить несколько отличающуюся частоту у считывателей, находящихся рядом, чтобы сотрудники могли на слух отличать их срабатывания. Разницы частот в 100 герц обычно достаточно. При установке считыватель пискнет на установленной частоте.

Примеры использования команды:

```
"get config beeper_freq\n"         // Проверяем частоту
"set config beeper_freq 100\n"     // Треск как у вибромоторчика
"set config beeper_freq 3900\n"    // Нормально пищать
"set config beeper_freq 10000\n"   // Теперь вы - летучая мышь
```

Примеры ответов считывателя:

```
"get config beeper_freq: 3900\r\n"            // Стоит частота 3900 Гц
"set config beeper_freq 100: OK\r\n"          // Поставили частоту 100
                                              //                    Гц
"set config beeper_freq 10: WRONG_VALUE\r\n"  // 10 Гц - неправильная
                                              //                частота
```

4.8 LED_brightness

`LED_brightness` - Яркость индикации. Десятичное число. Рекомендуется от 0 до 200. Допустимо до 255, но возможны искажения смешанных цветов. При установке считыватель моргнёт красным, зелёным и синим цветами.

Примеры использования команды:

```
"get config LED_brightness\n"       // Проверяем яркость
"set config LED_brightness 64\n"    // Ставим нормальную яркость
"set config LED_brightness 128\n"   // Ставим высокую яркость
```

Примеры ответов считывателя:

```
"get config LED_brightness: 16\r\n"             // Стоит яркость 16
"set config LED_brightness 32: OK\r\n"          // Поставили яркость
                                                //                32
"set config LED_brightness 256: WRONG_VALUE\r\n"  // 256 - неправильная
                                                //             яркость
```

4.9 RI_RGBB

`RI_RGBB` - Маска параметров для индикации прочтения идентификатора. Сразу после прочтения идентификатора считыватель выполняет RI (Read Indication, индикация прочтения). Индикация имеет в своём составе 4 двоичных флага - R (Red, красный), G (Green, зелёный), B (Blue, синий), B (Beeper, бипер). 0 - запрет использования, 1 - разрешение использования. Например, "1000" означает, что при прочтении идентификатора считыватель моргнёт красным цветом; "0101" означает, что при прочтении

идентификатора считыватель моргнёт зелёным цветом и пискнет. При установке считыватель продемонстрирует установленную индикацию.

Примеры использования команды:

```
"get config RI_RGBB\n"        // Проверяем индикацию прочтения
"set config RI_RGBB 1000\n"   // Устанавливаем моргание красным
"set config RI_RGBB 0101\n"   // Устанавливаем моргание зелёным и писк
```

Примеры ответов считывателя:

```
"get config RI_RGBB: 1000\r\n"            // Установлено моргание красным
"set config RI_RGBB 0101: OK\r\n"         // Установили моргание зелёным
                                                                  и писк
"set config RI_RGBB 100: WRONG_LEN\r\n"   // 100 - неправильная длина
```

4.10 RI_ms

RI_ms - Время, которое длится индикация прочтения идентификатора (в миллисекундах). Десятичное число. Значение по умолчанию - 100. Рекомендуется от 50 до 250. Допустимо от 0 до 1000.

Примеры использования команды:

```
"get config RI_ms\n"      // Проверяем время до включения
"set config RI_ms 80\n"   // Включать индикацию на 80 мс
"set config RI_ms 250\n"  // Включать индикацию на 250 мс
```

Примеры ответов считывателя:

```
"get config RI_ms: 100\r\n"     // Стоит длительность 100 мс
"set config RI_ms 80: OK\r\n"   // Установили длительность 80 мс
```

4.11 SI_mode

SI_mode - Метод индикации в режиме ожидания (Standby Indication mode). Одна десятичная цифра: 0 - индикация выключена, 1 - волна, 2 -

дыхание, 3 - плавное включение, 4 - резкое включение, 5 - быстрое резкое моргание, 6 - медленное резкое моргание, 7 - быстрое плавное моргание, 8 - медленное плавное моргание. При установке индикация будет изменена и включена сразу на один цикл, либо на 10 секунд если индикация не циклическая.

Примеры использования команды:

```
"get config SI_mode\n"     // Проверяем метод индикации
"set config SI_mode 1\n"   // Установить волну
"set config SI_mode 4\n"   // Установить резкое включение
```

Примеры ответов считывателя:

```
"get config SI_mode: 0\r\n"               // Индикация выключена
"set config SI_mode 2: OK\r\n"            // Установили дыхание
"set config SI_mode 9: WRONG_VALUE\r\n"   // 9 - неправильная индикация
```

4.12 SI_RGB

SI_RGB - Маска цвета индикации режима ожидания. Двоичная маска из трёх цифр 0 или 1, где 0 - неиспользование цвета, 1 - использование цвета. Порядок цифр соответствует обозначению RGB - красный, зелёный, синий. Например, "100" означает, что индикация будет красного цвета; "011" означает, что индикация будет бирюзовой (смесь зелёного и синего). Рекомендуется "001" (синий). При установке цвета, если считыватель не находился в режиме ожидания, индикация будет включена сразу на один цикл, либо на 10 секунд если индикация не циклическая. Если считыватель находился в режиме ожидания, цвет индикации будет изменён сразу.

Примеры использования команды:

```
"get config SI_RGB\n"       // Проверяем цвет индикации
"set config SI_RGB 100\n"   // Установить красный цвет
"set config SI_RGB 001\n"   // Установить синий цвет
```

Примеры ответов считывателя:

```
"get config SI_RGB: 001\r\n"            // Установлен синий цвет
"set config SI_RGB 100: OK\r\n"         // Установили красный цвет
"set config SI_RGB 10: WRONG_LEN\r\n"   // 10 - неправильная длина
```

4.13 SI_timeout

SI_timeout - Таймаут индикации режима ожидания. Через какое время бездействия включать индикацию режима ожидания (в миллисекундах). Десятичное число. Значение по умолчанию - 5000. Рекомендуется от 1000 до 10 000. Допустимо от 0 до 65 535. Числа менее 1000 будут игнорироваться.

Примеры использования команды:

```
"get config SI_timeout\n"        // Проверяем время до включения
"set config SI_timeout 1000\n"   // Включать индикацию через секунду
"set config SI_timeout 5000\n"   // Включать индикацию через 5 секунд
```

Примеры ответов считывателя:

```
"get config SI_timeout: 5000\r\n"      // Стоит таймаут 5 секунд
"set config SI_timeout 1000: OK\r\n"   // Установили таймаут 1 с
```

4.14 EI_R

EI_R - Настройка метода индикации красным цветом под внешним управлением (EI, External Indication, внешнее управление индикацией) (используется, если SI_mode настроен как 5 - внешнее управление). 0 - никогда не светится, 1 - всегда светится, 2 - прямое внешнее управление (при затягивании провода RedLED (коричневый) на землю индикация будет включаться), 3 - инвертированное внешнее управление (нормально индикация включена, при затягивании провода RedLED (коричневый) на землю индикация будет выключаться).

Актуально для настенных считывателей. Настольный считыватель игнорирует эту настройку, но хранит её и позволяет переносить её на настенные считыватели.

Примеры использования команды:

```
"get config EI_R\n"     // Проверяем метод индикации
"set config EI_R 2\n"   // Установить прямую индикацию
"set config EI_R 1\n"   // Установить постоянную индикацию
```

Примеры ответов считывателя:

```
"get config EI_R: 0\r\n"      // Установлена отсутствующая индикаций
"set config EI_R 2: OK\r\n"   // Установили прямую индикацию
"set config EI_R 3: OK\r\n"   // Установили инверсную индикацию
```

4.15 EI_G

`EI_G` - Настройка метода индикации зелёным цветом под внешним управлением (EI, External Indication, внешнее управление индикацией) (используется, если `SI_mode` настроен как 5 - внешнее управление). 0 - никогда не светится, 1 - всегда светится, 2 - прямое внешнее управление (при затягивании провода GrnLED (коричневый) на землю индикация будет включаться), 3 - инвертированное внешнее управление (нормально индикация включена, при затягивании провода GrnLED (коричневый) на землю индикация будет выключаться).

Актуально для настенных считывателей. Настольный считыватель игнорирует эту настройку, но хранит её и позволяет переносить её на настенные считыватели.

Примеры использования команды:

```
"get config EI_G\n"     // Проверяем метод индикации
"set config EI_G 2\n"   // Установить прямую индикацию
"set config EI_G 1\n"   // Установить постоянную индикацию
```

Примеры ответов считывателя:

```
"get config EI_G: 0\r\n"      // Установлена отсутствующая индикаций
"set config EI_G 2: OK\r\n"   // Установили прямую индикацию
"set config EI_G 3: OK\r\n"   // Установили инверсную индикацию
```

4.16 EI_B

`EI_B` - Настройка метода индикации синим цветом под внешним управлением (EI, External Indication, внешнее управление индикацией) (используется, если `SI_mode` настроен как 5 - внешнее управление). 0 - никогда не светится, 1 - всегда светится. Значения 2 и 3 (прямое и инвертированное внешнее управление) для синего цвета недоступны ввиду отсутствия провода управления индикацией синего цвета.

Актуально для настенных считывателей. Настольный считыватель игнорирует эту настройку, но хранит её и позволяет переносить её на настенные считыватели.

Примеры использования команды:

```
"get config EI_B\n"     // Проверяем метод индикации
"set config EI_B 0\n"   // Установить отсутствующую индикацию
"set config EI_B 1\n"   // Установить постоянную индикацию
```

Примеры ответов считывателя:

```
"get config EI_B: 0\r\n"      // Установлена отсутствующая индикаций
"set config EI_B 1: OK\r\n"   // Установили постоянную индикацию
"set config EI_B 0: OK\r\n"   // Установили отсутствующую индикацию
```

4.17 EI_Bz

`EI_Bz` - Настройка метода индикации бипером под внешним управлением (EI, External Indication, внешнее управление индикацией) (используется, если `SI_mode` настроен как 5 - внешнее управление). 0 -

никогда не пищит, 1 - всегда пищит, 2 - прямая (при затягивании провода BEEP (синий) на землю индикация будет включаться), 3 - инверсия (нормально индикация включена, при затягивании провода BEEP (синий) на землю индикация будет выключаться).

Актуально для настенных считывателей. Настольный считыватель игнорирует эту настройку, но хранит её и позволяет переносить её на настенные считыватели.

Примеры использования команды:

```
"get config EI_Bz\n"     // Проверяем метод индикации
"set config EI_Bz 2\n"   // Установить прямую индикацию
"set config EI_Bz 1\n"   // Установить постоянную индикацию
```

Примеры ответов считывателя:

```
"get config EI_Bz: 0\r\n"      // Установлена отсутствующая индикаций
"set config EI_Bz 2: OK\r\n"   // Установили прямую индикацию
"set config EI_Bz 3: OK\r\n"   // Установили инверсную индикацию
```

4.18 Wiegand_type

`Wiegand_type` - Тип вейганда. Трёхзначное десятичное число, либо строка. Используется, если в считывателе не настроены ячейки программирования, либо в ячейке программирования не установлен тип вейганда. В противном случае используются значения из ячеек программирования.

При задании типа вейганда числом - первые 2 знака числа показывают количество бит данных, а третий знак - количество бит чётности. Количество бит данных желательно устанавливать кратным 8. Количество бит чётности поддерживается 0 и 2.

В случае со строкой - поддерживается установка 13 стандартных интерфейсов:

1) Wiegand 24 - "W24";
2) Wiegand 26 - "W26";
3) Wiegand 32 - "W32";
4) Wiegand 34 - "W34";
5) Wiegand 37 - "W37";
6) Wiegand 40 - "W40";
7) Wiegand 42 - "W42";
8) Wiegand 56 - "W56";
9) Wiegand 58 - "W58";
10) Wiegand 64 - "W64";
11) Wiegand 66 - "W66";
12) Wiegand 80 - "W80";
13) Wiegand 82 - "W82".

Примеры использования команды:

```
"get prog1 Wiegand_type\n"       // Проверяем тип вейганда
"set prog1 Wiegand_type 242\n"   // 24 бита данных и 2 бита чётности
"set prog1 Wiegand_type W26\n"   // Wiegand 26
"set prog1 Wiegand_type 640\n"   // 64 бита данных и 0 бит чётности
"set prog1 Wiegand_type W64\n"   // Wiegand 64
```

Примеры ответов считывателя:

```
"get prog1 Wiegand_type: W58\r\n"          // Установлен Wiegand 58
"get prog1 Wiegand_type: 722\r\n"           // 72 бита данных и 2 бита
                                             //                  чётности
"set prog1 Wiegand_type 422: OK\r\n"          // Установили Wiegand 42
"set prog1 Wiegand_type W42: OK\r\n"           // Установили Wiegand 42
"set prog1 Wiegand_type 423: WRONG_VALUE\r\n"   // Неправильные данные
"set prog1 Wiegand_type W27: WRONG_VALUE\r\n"    // Неправильные данные
```

4.19 BLE_TX_power

BLE_TX_power - Мощность BLE-передатчика. Указывается в децибелах по милливатту (дБм). Десятичное число от -128 до 127. Считывателем

поддерживается ограниченный список мощностей от -14 дБм до 9 дБм включительно, поэтому реальная мощность устанавливается считывателем из списка доступных, ближайшая к указанному пользователем значению. Если указанная в команде мощность не поддерживается, в ответе будет указана реальная установленная мощность. Значение по умолчанию - 0 дБм.

Примеры значений:

```
"get config BLE_TX_power\n"         // Проверяем мощность
"set config BLE_TX_power -14\n"    // Установить минимальную мощность.
"set config BLE_TX_power 0\n"     // Минимальная мощность для максимальной
                                 //           дальности считывания PW-Tag.
"set config BLE_TX_power 9\n"      // Установить максимальную мощность,
                                  //    будет слышно в соседних кабинетах.
```

Примеры ответов считывателя:

```
"get config BLE_TX_power: 0\r\n"          // Установлена мощность 0 дБм
"set config BLE_TX_power -14: OK\r\n"     // Установили мощность -14 дБм
"set config BLE_TX_power -7: -6 OK\r\n"   // -7 не поддерживается,
                                          //   установили мощность -6 дБм
```

4.20 `startup_delay`

`startup_delay` - Дополнительная задержка включения при подаче питания. Рекомендуется установить её при возникновении проблем при работе совместно с контроллерами, которые долго запускаются. Настольному считывателю эта задержка не требуется и используется лишь для имитации запуска настенного считывателя, потому для настольного считывателя рекомендуется значение 0. Десятичное число в миллисекундах. Рекомендуется от 0 до 10 000. Допустимо от 0 до 65 535.

Примеры использования команды:

```
"get config startup_delay\n"         // Проверяем задержку
"set config startup_delay 0\n"       // Убираем дополнительную задержку
"set config startup_delay 10000\n"   // Устанавливаем задержку 10 секунд
```

Примеры ответов считывателя:

```
"get config startup_delay: 0\r\n"          // Установлена задержка 0 сек.
"set config startup_delay 10000: OK\r\n"   // Установили задержку 10 сек.
```

4.21 auto_keyboard_mode

auto_keyboard_mode - Используется ли при включении считывателя режим клавиатуры. Сразу после включения и запуска считыватель включит режим клавиатуры и начнёт передавать считанные идентификаторы в компьютер в режиме клавиатуры (имитируя нажатия на кнопки на клавиатуре). Одна цифра: 0 - нет, 1 - да.

Примеры использования команды:

```
"get config auto_keyboard_mode\n"     // Проверяем режим клавиатуры
"set config auto_keyboard_mode 0\n"   // Включаемся как обычно
"set config auto_keyboard_mode 1\n"   // Включаемся в режиме клавиатуры
```

Примеры ответов считывателя:

```
"get config auto_keyboard_mode: 0\n"      // Режим клавиатуры при
                                          //    включении не используется
"set config auto_keyboard_mode 1: OK\n"
              // Установили использование режима клавиатуры при включении
```

4.22 is_keyboard_mode

is_keyboard_mode - Работает ли на данный момент режим клавиатуры. Одна цифра: 0 - нет, 1 - да.

Примеры использования команды:

```
"get config is_keyboard_mode\n"     // Проверяем режим клавиатуры
"set config is_keyboard_mode 0\n"   // Выключаем режим клавиатуры
"set config is_keyboard_mode 1\n"   // Включаем режим клавиатуры
```

Примеры ответов считывателя:

```
"get config is_keyboard_mode: 0\r\n"      // Режим клавиатуры выключен
"set config is_keyboard_mode 0: OK\r\n"   // Выключили режим клавиатуры
"set config is_keyboard_mode 1: OK\r\n"   // Включили режим клавиатуры
```

4.23 DO_mode

DO_mode - Режим выдачи данных в режиме эмуляции клавиатуры (Data Output mode). Описывает, в каком виде нужно передавать данные в компьютер в режиме эмуляции клавиатуры.

Auto - Автоматический режим. Идентификатор будет передан в шестнадцатеричном формате той длины, которая установлена в ячейке программирования;

DEC - Весь идентификатор в десятичном формате;

DEC3 - Три последних байта идентификатора в десятичном формате;

Facility - Фасилити-код идентификатора;

HEX3 - 3 последних байта идентификатора в шестнадцатеричном формате;

HEX4 - 4 последних байта идентификатора в шестнадцатеричном формате;

HEX5 - 5 последних байт идентификатора в шестнадцатеричном формате;

HEX6 - 6 последних байт идентификатора в шестнадцатеричном формате;

HEX7 - 7 последних байт идентификатора в шестнадцатеричном формате;

HEX8 - 8 последних байт идентификатора в шестнадцатеричном формате.

Примеры использования команды:

```
"get config DO_mode\n"            // Проверяем режим выдачи данных
"set config DO_mode Auto\n"       // Ставим автоматический режим
"set config DO_mode Facility\n"   // Ставим фасилити-код
"set config DO_mode HEX3\n"       // Ставим три шестнадцатеричных байта
```

Примеры ответов считывателя:

```
"get config DO_mode: Auto\r\n"              // Стоит автоматический
                                            //                    режим
"set config DO_mode Auto: OK\r\n"           // Установили автоматический
                                            //                    режим
"set config DO_mode Facility: OK\r\n"       // Установили Фасилити-код
"set config DO_mode HEX2: WRONG_VALUE\r\n"  // HEX2 - неизвестный режим
```

4.24 keyboard_extra_char

keyboard_extra_char - Автоматически добавляемый символ к печатаемой строке в режиме эмуляции клавиатуры. Бывает полезным, если нужно перенести строку (перейти на ячейку ниже в Excel) или вставить пробел между идентификаторами. Символ устанавливается в шестнадцатеричном виде согласно таблице ASCII (www.asciitable.com). Установить "00" для выключения добавления символа.

Примеры использования команды:

```
"get config keyboard_extra_char\n"      // Проверяем доп.символ
"set config keyboard_extra_char 00\n"   // Убираем доп.символ
"set config keyboard_extra_char 0A\n"   // Перенос строки (Enter, '\n')
"set config keyboard_extra_char 20\n"   // Пробел (Space, ' ')
```

Примеры ответов считывателя:

```
"get config keyboard_extra_char: 00\r\n"   // Дополнительный символ не
                                           //                  установлен
"set config keyboard_extra_char 0A: OK\r\n"
       // Установили дополнительный символ переноса строки (Enter, '\n')
"set config keyboard_extra_char 100: WRONG_LEN\r\n"
                                              // 100 - неправильная длина
```

4.25 eng_key

eng_key - Ключ инженера. Ключ, используемый для получения доступа к конфигурированию считывателя. 6 десятичных цифр. На текущий момент по соображениям безопасности прочитать код инженера из конфигурации считывателя нельзя. Ключ инженера по умолчанию - 128512.

Примеры использования команды:

```
"get config eng_key\n"          // Проверяем ключ инженера (не получим)
"set config eng_key 128512\n"   // Ключ инженера по умолчанию
"set config eng_key 123456\n"   // Какой-то свой ключ инженера
```

Примеры ответов считывателя:

```
"get config eng_key: NOT_ALLOWED\r\n"   // Ключ инженера нельзя читать
"set config eng_key 128512: OK\r\n"     // Установили ключ инженера по
                                        //                        умолчанию
"set config eng_key 123456: OK\r\n"     // Установили какой-то свой ключ
                                        //                         инженера
```

4.26 write_start_ident

write_start_ident - Стартовый идентификатор, используемый при автоматическом выпуске рабочих карт. Записывается в виде чётного количества шестнадцатеричных цифр от 2 до 32 (от 1 до 16 байт). Общая длина записываемого числа должна соответствовать длине идентификатора.

Примеры использования команды:

```
"get config write_start_ident\n"     // Проверяем стартовый идентификатор
"set config write_start_ident 012345\n"
      // Установить стартовый идентификатор 012345 (6 символов, 3 байта)
"set config write_start_ident 0000006789ABCDEF\n"
                  // Установить стартовый идентификатор с нулями в начале
```

Примеры ответов считывателя:

```
"get config write_start_ident: 000000\n"
                                // Указан нулевой 3-байтовый идентификатор
"set config write_start_ident 012345: OK\n"
      // Установили стартовый идентификатор 012345 (6 символов, 3 байта)
"set config write_start_ident 000006789ABCDEF: WRONG_LEN\n"
                                    // Неправильная длина идентификатора
```

5. Описание параметров ячеек программирования (prog1 - prog8)

Тут описаны команды для настройки ячеек программирования. В примерах использования команд будет использована первая ячейка программирования, однако можно обратиться к любой из доступных восьми ячеек.

5.1 Общие для всех типов идентификаторов параметры

Некоторые параметры применяются для всех типов идентификаторов. Для идентификаторов типа EM-Marine, HID Prox, INDALA и ICODE это полный список параметров.

5.1.1 is_active

is_active - Активна ли данная ячейка программирования. Ячейка программирования может быть запрограммирована, но деактивирована. Тогда она не будет удалена из памяти считывателя, но и использоваться считывателем не будет.

Примеры использования команды:

```
"get prog1 is_active\n"     // Проверяем активность ячейки
"set prog1 is_active 0\n"   // Назначить ячейку неактивной
"set prog1 is_active 1\n"   // Назначить ячейку активной
```

Примеры ответов считывателя:

```
"get prog1 is_active: 1\r\n"      // Ячейка программирования активна
"set prog1 is_active 0: OK\r\n"   // Назначили ячейку неактивной
"set prog1 is_active 1: OK\r\n"   // Назначили ячейку активной
```

5.1.2 ident_type

ident_type - Тип идентификатора в ячейке программирования. Шестнадцатеричная цифра:

0 - Пустая ячейка;
1 - EM-Marine;
2 - HID Prox;
3 - INDALA;
4 - Mifare Ultralight;
5 - Mifare SL1;
6 - Mifare SL3;
7 - ICODE;
8 - Mifare Desfire;
9 - Mobile ID;
A - PW-Tag.

Данная команда при использовании заголовка set сбрасывает ячейку в заводское состояние, либо создаёт ячейку программирования, если ранее она не была запрограммирована. Использование нуля в качестве значения приведёт к удалению ячейки программирования. В заводском состоянии ячейка имеет в себе:

EM-Marine - ячейка активна, название ячейки "EM-Marine", Wiegand-26, инвертированный порядок передачи кода.

HID Prox - ячейка активна, название ячейки "HID Prox", Wiegand-26, инвертированный порядок передачи кода.

INDALA - ячейка активна, название ячейки "INDALA", Wiegand-26, инвертированный порядок передачи кода.

Mifare Ultralight - ячейка активна, название ячейки "Ultralight", Wiegand-26, инвертированный порядок передачи кода, передача UID, учитывается код производителя (нулевой байт у 7-байтных UID),

предустановлен блок 4 для чтения (но не задействован, поскольку установлена передача UID).

Mifare SL1 - ячейка активна, название ячейки "Mifare SL1", Wiegand-26, инвертированный порядок передачи кода, передача UID, учитывается код производителя (нулевой байт у 7-байтных UID), предустановлен блок 1 для чтения (но не задействован, поскольку установлена передача UID) и ключи A и B из 6 байт 0xFF (заводские ключи по умолчанию).

Mifare SL3 - ячейка активна, название ячейки "Mifare SL3", Wiegand-26, инвертированный порядок передачи кода, передача UID, учитывается код производителя (нулевой байт у 7-байтных UID), предустановлен блок 1 для чтения (но не задействован, поскольку установлена передача UID) и ключи A и B из 16 байт 0xFF (заводской ключ AES по умолчанию).

ICODE - ячейка активна, название ячейки "ICODE", Wiegand-26, инвертированный порядок передачи кода.

Mifare Desfire - ячейка активна, название ячейки "Desfire", Wiegand-26, инвертированный порядок передачи кода, передача UID, учитывается код производителя (нулевой байт у 7-байтных UID).

Mobile ID - ячейка активна, название ячейки "Mobile ID", Wiegand-26, прямой порядок передачи кода, минимальный RSSI для всех -60 (расстояние около 50 сантиметров), всем разрешено использовать все режимы работы и коррекцию RSSI.

PW-Tag - ячейка активна, название ячейки "PW-Tag", Wiegand-26, прямой порядок передачи кода, минимальный RSSI -60 (расстояние около 50 сантиметров).

Примеры использования команды:

```
"get prog1 ident_type\n"     // Проверяем тип идентификатора
"set prog1 ident_type 0\n"   // Назначить ячейку пустой
"set prog1 ident_type 5\n"   // Mifare SL1
"set prog1 ident_type A\n"   // PW-Tag
```

Примеры ответов считывателя:

```
"get prog1 ident_type: 9\r\n"               // Ячейка Mobile ID
"set prog1 ident_type 5: OK\r\n"            // Назначили ячейку Mifare
                                            //                         SL1
"set prog1 ident_type B: WRONG_VALUE\r\n"   // B - неизвестный тип
                                            //             идентификатора
```

5.1.3 name

`name` - Название ячейки программирования. Строка длиной до 16 знаков. Знаки только из набора ASCII. Можно указать пустое имя.

Примеры использования команды:

```
"get prog1 name\n"                    // Проверяем название ячейки
"set prog1 name\n"                    // Удалили название
"set prog1 name \n"                   // Тоже удалили название
"set prog1 name Domofon\n"            // Назвали ячейку "Domofon"
"set prog1 name BOSS_of_this_GYM\n"   // Максимальная длина - 16 знаков
```

Примеры ответов считывателя:

```
"get prog1 name: PW-Tag\r\n"       // Ячейка называется "PW-Tag"
"get prog1 name: NONE\r\n"         // Ячейка не имеет названия
"set prog1 name: OK\r\n"           // Назначили ячейке пустое название
"set prog1 name Domofon: OK\r\n"   // Назначили ячейке название "Domofon"
"set prog1 name BOSS_of_this_GYM!: WRONG_LEN\r\n"
           // "BOSS_of_this_GYM!" (17 знаков) - слишком длинное название
```

5.1.4 Wiegand_type

`Wiegand_type` - Тип вейганда. Трёхзначное десятичное число, либо строка.

При задании интерфейса числом - первые 2 знака числа показывают количество бит данных, а третий знак - количество бит чётности. Количество бит данных желательно устанавливать кратным 8. Количество бит чётности поддерживается 0 и 2.

В случае со строкой - поддерживается установка 13 стандартных интерфейсов:

1) Wiegand 24 - "W24";
2) Wiegand 26 - "W26";
3) Wiegand 32 - "W32";
4) Wiegand 34 - "W34";
5) Wiegand 37 - "W37";
6) Wiegand 40 - "W40";
7) Wiegand 42 - "W42";
8) Wiegand 56 - "W56";
9) Wiegand 58 - "W58";
10) Wiegand 64 - "W64";
11) Wiegand 66 - "W66";
12) Wiegand 80 - "W80";
13) Wiegand 82 - "W82".

Если данный параметр не установить - для передачи идентификатора будет использоваться тип вейганда, установленный в основных настройках считывателя. Для удаления этого параметра, чтобы считыватель использовал значение из основных настроек, нужно использовать кодовое слово "NONE".

Примеры использования команды:

```
"get prog1 Wiegand_type\n"        // Проверяем тип вейганда
"set prog1 Wiegand_type 242\n"    // 24 бита данных и 2 бита чётности
"set prog1 Wiegand_type W26\n"    // Wiegand 26
"set prog1 Wiegand_type 640\n"    // 64 бита данных и 0 бит чётности
"set prog1 Wiegand_type W64\n"    // Wiegand 64
"set prog1 Wiegand_type NONE\n"   // Удаляем параметр
```

Примеры ответов считывателя:

```
"get prog1 Wiegand_type: W58\r\n"         // Установлен Wiegand 58
"get prog1 Wiegand_type: 722\r\n"          // 72 бита данных и 2 бита
                                           //                   чётности
"get prog1 Wiegand_type: NONE\r\n"           // Wiegand не установлен
"set prog1 Wiegand_type 422: OK\r\n"          // Установили Wiegand 42
"set prog1 Wiegand_type W42: OK\r\n"           // Установили Wiegand 42
"set prog1 Wiegand_type 423: WRONG_VALUE\r\n"   // Неправильные данные
"set prog1 Wiegand_type W27: WRONG_VALUE\r\n"    // Неправильные данные
```

5.1.5 is_code_inverted

is_code_inverted - Указывает на инверсию (прямой или обратный) порядка передачи байт. 0 - прямой порядок байт (не инвертированный), 1 - обратный порядок байт (инвертированный).

Примеры использования команды:

```
"get prog1 is_code_inverted\n"     // Проверяем порядок передачи
"set prog1 is_code_inverted 0\n"   // Назначить прямой порядок передачи
"set prog1 is_code_inverted 1\n"   // Назначить обратный порядок передачи
```

Примеры ответов считывателя:

```
"get prog1 is_code_inverted: 1\r\n"      // Стоит обратный порядок
"set prog1 is_code_inverted 0: OK\r\n"   // Ставим прямой порядок
"set prog1 is_code_inverted 1: OK\r\n"   // Ставим обратный порядок
```

5.1.6 code_start_byte

code_start_byte - Стартовый байт для передачи данных. Указывает, с какого байта начать передавать данные. Десятичное число от 0 до 127.

Примеры использования команды:

```
"get prog1 code_start_byte\n"     // Проверяем стартовый байт
"set prog1 code_start_byte 0\n"   // Назначить нулевой байт стартовым
"set prog1 code_start_byte 3\n"   // Назначить третий байт стартовым
```

Примеры ответов считывателя:

```
"get prog1 code_start_byte: 0\r\n"      // Назначен стартовым байтом
                                        //                          нулевой
"set prog1 code_start_byte 0: OK\r\n"   // Назначаем стартовым байтом
                                        //                          нулевой
"set prog1 code_start_byte 3: OK\r\n"   // Назначаем стартовым байтом
                                        //                           третий
```

5.2 Специализированные параметры для Mifare Ultralight

Mifare Ultralight - незащищённые идентификаторы, такие как Mifare Ultralight, Mifare Ultralight C, NTAG-215, NTAG-216.

5.2.1 `tx_params`

`tx_params` - Параметры передачи данных. Десятичное число. 0 - передаём только UID, 1 - передаём UID после успешного чтения блока, 2 - передаём содержимое блока.

Примеры использования команды:

```
"get prog1 tx_params\n"     // Проверяем параметры передачи данных
"set prog1 tx_params 0\n"   // Передаём только UID
"set prog1 tx_params 2\n"   // Передаём содержимое блока
```

Примеры ответов считывателя:

```
"get prog1 tx_params: 0\r\n"      // Назначено передавать только UID
"set prog1 tx_params 2: OK\r\n"   // Назначили передавать содержимое
                                  //                              блока
```

5.2.2 `vendor_control`

`vendor_control` - Контроль производителя. Используется при чтении 7-байтных UID, на другие сценарии не влияет. 0 - игнорируется нулевой байт 7-байтных UID, 1 - 7-байтные UID передаются целиком.

Примеры использования команды:

```
"get prog1 vendor_control\n"     // Проверяем контроль производителя
"set prog1 vendor_control 0\n"   // Снять контроль производителя
"set prog1 vendor_control 1\n"   // Установить контроль производителя
```

Примеры ответов считывателя:

```
"get prog1 vendor_control: 0\r\n"      // Контроль производителя снят
"set prog1 vendor_control 1: OK\r\n"   // Установили контроль
                                       //                  производителя
```

5.2.3 prox_check

prox_check - Proximity check, защита от атаки "Человек между". Проверяет время ответов карты, и если оно превышает установленный предел - соединение с картой объявляется скомпрометированным и карта не считывается. 0 - Proximity check выключен, 1 - Proximity check включен. Пока не реализовано, появится с дальнейшими обновлениями прошивки.

Примеры использования команды:

```
"get prog1 prox_check\n"     // Проверяем Proximity check
"set prog1 prox_check 0\n"   // Выключить Proximity check
"set prog1 prox_check 1\n"   // Включить Proximity check
```

Примеры ответов считывателя:

```
"get prog1 prox_check: 0\r\n"      // Proximity check выключен
"set prog1 prox_check 1: OK\r\n"   // Включили Proximity check
```

5.2.4 block_num

block_num - Номер блока для чтения. Двузначное шестнадцатеричное число от 00 до FF. Блоки в картах данного типа имеют размер 4 байта, и за одно прочтение карта выдаёт 16 байт (4 блока).

Примеры использования команды:

```
"get prog1 block_num\n"      // Проверяем номер блока
"set prog1 block_num 01\n"   // Установить блок 0x01 (1)
"set prog1 block_num 1A\n"   // Установить блок 0x1A (26)
```

Примеры ответов считывателя:

```
"get prog1 block_num: 04\n"      // Установлен блок 0x04 (4)
"set prog1 block_num 1A: OK\n"   // Установили блок 0x1A (26)
```

5.3 Специализированные параметры для Mifare SL1

Идентификаторы типа Mifare SL1 включают себя импортные карты и брелки типа Mifare Classic, Mifare ID, Mifare Plus в режиме SL1, а так же российские карты и брелки типа Mikron 1K, Mikron 3K в режиме SL1.

Длина ключа - 6 байт. Метод шифрования проприетарный.

5.3.1 `tx_params`

`tx_params` - Параметры передачи данных. Десятичное число. 0 - передаём только UID, 1 - передаём UID после успешного чтения блока, 2 - передаём содержимое блока.

Примеры использования команды:

```
"get prog1 tx_params\n"     // Проверяем параметры передачи данных
"set prog1 tx_params 0\n"   // Передаём только UID
"set prog1 tx_params 2\n"   // Передаём содержимое блока
```

Примеры ответов считывателя:

```
"get prog1 tx_params: 0\r\n"      // Назначено передавать только UID
"set prog1 tx_params 2: OK\r\n"   // Назначили передавать содержимое
                                  //                                блока
```

5.3.2 `vendor_control`

`vendor_control` - Контроль производителя. Используется при чтении 7-байтных UID, на другие сценарии не влияет. 0 - игнорируется нулевой байт 7-байтных UID, 1 - 7-байтные UID передаются целиком.

Примеры использования команды:

```
"get prog1 vendor_control\n"     // Проверяем контроль производителя
"set prog1 vendor_control 0\n"   // Снять контроль производителя
"set prog1 vendor_control 1\n"   // Установить контроль производителя
```

Примеры ответов считывателя:

```
"get prog1 vendor_control: 0\r\n"      // Контроль производителя снят
"set prog1 vendor_control 1: OK\r\n"   // Установили контроль
                                       //                производителя
```

5.3.3 prox_check

prox_check - Proximity check, защита от атаки "Человек между". Проверяет время ответов карты, и если оно превышает установленный предел - соединение с картой объявляется скомпрометированным и карта не считывается. 0 - Proximity check выключен, 1 - Proximity check включен. Пока не реализовано, появится с дальнейшими обновлениями прошивки.

Примеры использования команды:

```
"get prog1 prox_check\n"     // Проверяем Proximity check
"set prog1 prox_check 0\n"   // Выключить Proximity check
"set prog1 prox_check 1\n"   // Включить Proximity check
```

Примеры ответов считывателя:

```
"get prog1 prox_check: 0\r\n"      // Proximity check выключен
"set prog1 prox_check 1: OK\r\n"   // Включили Proximity check
```

5.3.4 block_num

block_num - Номер блока для чтения. Двузначное шестнадцатеричное число от 00 до FF. Блоки в картах данного типа имеют размер 4 байта, и за одно прочтение карта выдаёт 16 байт (4 блока).

Примеры использования команды:

```
"get prog1 block_num\n"      // Проверяем номер блока
"set prog1 block_num 01\n"   // Установить блок 0x01 (1)
"set prog1 block_num 1A\n"   // Установить блок 0x1A (26)
```

Примеры ответов считывателя:

```
"get prog1 block_num: 04\n"      // Установлен блок 0x04 (4)
"set prog1 block_num 1A: OK\n"   // Установили блок 0x1A (26)
```

5.3.5 read_key_B

read_key_B - Выбор ключа, который будет использоваться для чтения карты. Двоичная цифра, 0 - ключ A, 1 - ключ B.

Примеры использования команды:

```
"get prog1 read_key_B\n"     // Проверяем ключ
"set prog1 read_key_B 0\n"   // Читать ключом A
"set prog1 read_key_B 1\n"   // Читать ключом B
```

Примеры ответов считывателя:

```
"get prog1 read_key_B: 0\r\n"      // Читаем ключом A
"set prog1 read_key_B 0: OK\r\n"   // Назначили чтение ключом A
"set prog1 read_key_B 1: OK\r\n"   // Назначили чтение ключом B
```

5.3.6 keyA_bytes

keyA_bytes - Байты непосредственно самого ключа A. Нужно установить, если настроено чтение карты ключом A или при необходимости выпуска карт. Ключ в картах SL1 имеет размер 6 байт. Ключ записывается в виде 12 шестнадцатеричных цифр. На текущий момент по соображениям безопасности прочитать ключи из ячейки программирования считывателя нельзя.

Примеры использования команды:

```
"get prog1 keyA_bytes\n"                // Проверяем ключ A (не получим)
"set prog1 keyA_bytes FFFFFFFFFFFF\n"   // Установить ключ A (стандартный
                                        //                   ключ)
"set prog1 keyA_bytes 010203040506\n"   // Установить ключ A (свой)
```

Примеры ответов считывателя:

```
"get prog1 keyA_bytes: NOT_ALLOWED\r\n"         // Запрещено читать ключи
"set prog1 keyA_bytes FFFFFFFFFFFF: OK\r\n"
                                  // Установили стандартный ключ на ключ A
"set prog1 keyA_bytes FFFFFFFFFFF: WRONG_LEN\r\n"   // Неправильная длина
```

5.3.7 keyB_bytes

keyB_bytes - Байты непосредственно самого ключа B. Нужно установить, если настроено чтение карты ключом B или при необходимости выпуска карт. Ключ в картах SL1 имеет размер 6 байт. Ключ записывается в виде 12 шестнадцатеричных цифр. На текущий момент по соображениям безопасности прочитать ключи из ячейки программирования считывателя нельзя.

Примеры использования команды:

```
"get prog1 keyB_bytes\n"                 // Проверяем ключ B (не получим)
"set prog1 keyB_bytes FFFFFFFFFFFF\n"    // Установить ключ B (стандартный
                                         //                          ключ)
"set prog1 keyB_bytes 010203040506\n"    // Установить ключ B (свой)
```

Примеры ответов считывателя:

```
"get prog1 keyB_bytes: NOT_ALLOWED\r\n"         // Запрещено читать
ключи
"set prog1 keyB_bytes FFFFFFFFFFFF: OK\r\n"
                                  // Установили стандартный ключ на ключ B
"set prog1 keyB_bytes FFFFFFFFFFF: WRONG_LEN\r\n"   // Неправильная длина
```

5.3.8 write_acc_bits

write_acc_bits - Биты доступа, записываемые на выпускаемые рабочие карты. Эти биты задают, какими ключами доступ к каким ресурсам на карте будет разрешён. Рекомендуется более подробно изучить этот вопрос по ссылкам ниже.

https://slebe.dev/mifarecalc/
http://calc.gmss.ru/Mifare1k/

Биты доступа записываются в виде 6 шестнадцатеричных цифр. Можно их не записывать, тогда будут использованы биты доступа 787788. Такие биты доступа позволяют читать содержимое карты обоими ключами, а записывать только ключом B, и являются обратимыми. Эти же биты доступа будут применены, если пользователь впишет некорректные или необратимые биты доступа.

При записи бит доступа считыватель проверяет их корректность и обратимость. При установке некорректных или необратимых бит доступа считыватель вернёт ошибку.

Примеры использования команды:

```
"get prog1 write_acc_bits\n"          // Проверяем установленные биты
                                      //                          доступа
"set prog1 write_acc_bits FF0780\n"   // Установить стандартные биты
                                      //                          доступа
"set prog1 write_acc_bits 787788\n"   // Установить свои биты доступа
```

Примеры ответов считывателя:

```
"get prog1 write_acc_bits: 787788\r\n"        // Установлены биты доступа
                                               //                  787788
"set prog1 write_acc_bits FF0780: OK\r\n"      // Установили стандартные
                                                //          биты доступа
"set prog1 write_acc_bits FF078: WRONG_LEN\r\n"   // Неправильная длина
```

5.3.9 `write_all_trailers`

`write_all_trailers` - Нужно ли перезаписывать все трейлеры карты (зашифровать все секторы карты). Полезно для защиты от некоторых типов атак, но существенно увеличивает время записи. Двоичная цифра, 0 - не нужно, 1 - нужно.

Примеры использования команды:

```
"get prog1 write_all_trailers\n"     // Проверяем потребность перезаписи
                                     //                    всех трейлеров
"set prog1 write_all_trailers 1\n"   // Установить потребность перезаписи
                                     //                    всех трейлеров
```

Примеры ответов считывателя:

```
"get prog1 write_all_trailers: 0\r\n"      // Потребность перезаписи всех
                                           //    трейлеров не установлена
"set prog1 write_all_trailers 1: OK\r\n"   // Установили потребность
                                           //   перезаписи всех трейлеров
```

5.4 Специализированные параметры для Mifare SL3

Идентификаторы типа Mifare SL3 включают себя импортные карты и брелки типа Mifare Plus в режиме SL3, а так же российские карты и брелки типа Mikron 3K в режиме SL3.

Длина ключа - 16 байт. Метод шифрования - AES.

Также идентификаторы типа SL3 поддерживают шифрование передаваемого трафика. Дополнительно для каждого ключа можно указать, требуется ли шифровать трафик при работе данным ключом.

5.4.1 `tx_params`

`tx_params` - Параметры передачи данных. Десятичное число. 0 - передаём только UID, 1 - передаём UID после успешного чтения блока, 2 - передаём содержимое блока.

Примеры использования команды:

```
"get prog1 tx_params\n"     // Проверяем параметры передачи данных
"set prog1 tx_params 0\n"   // передаём только UID
"set prog1 tx_params 2\n"   // передаём содержимое блока
```

Примеры ответов считывателя:

```
"get prog1 tx_params: 0\r\n"     // Назначено передавать только UID
"set prog1 tx_params 2: OK\r\n"  // Назначили передавать содержимое блока
```

5.4.2 `vendor_control`

`vendor_control` - Контроль производителя. Используется при чтении 7-байтных UID, на другие сценарии не влияет. 0 - игнорируется нулевой байт 7-байтных UID, 1 - 7-байтные UID передаются целиком.

Примеры использования команды:

```
"get prog1 vendor_control\n"     // Проверяем контроль производителя
"set prog1 vendor_control 0\n"   // Снять контроль производителя
"set prog1 vendor_control 1\n"   // Установить контроль производителя
```

Примеры ответов считывателя:

```
"get prog1 vendor_control: 0\r\n"      // Контроль производителя снят
"set prog1 vendor_control 1: OK\r\n"   // Установили контроль
                                       //                   производителя
```

5.4.3 prox_check

`prox_check` - Proximity check, защита от атаки "Человек между". Проверяет время ответов карты, и если оно превышает установленный предел - соединение с картой объявляется скомпрометированным и карта не считывается. 0 - Proximity check выключен, 1 - Proximity check включен. Пока не реализовано, появится с дальнейшими обновлениями прошивки.

Примеры использования команды:

```
"get prog1 prox_check\n"     // Проверяем Proximity check
"set prog1 prox_check 0\n"   // Выключить Proximity check
"set prog1 prox_check 1\n"   // Включить Proximity check
```

Примеры ответов считывателя:

```
"get prog1 prox_check: 0\r\n"      // Proximity check выключен
"set prog1 prox_check 1: OK\r\n"   // Включили Proximity check
```

5.4.4 block_num

`block_num` - Номер блока для чтения. Двузначное шестнадцатеричное число от 00 до FF. Блоки в картах данного типа имеют размер 4 байта, и за одно прочтение карта выдаёт 16 байт (4 блока).

Примеры использования команды:

```
"get prog1 block_num\n"      // Проверяем номер блока
"set prog1 block_num 01\n"   // Установить блок 0x01 (1)
"set prog1 block_num 1A\n"   // Установить блок 0x1A (26)
```

Примеры ответов считывателя:

```
"get prog1 block_num: 04\n"      // Установлен блок 0x04 (4)
"set prog1 block_num 1A: OK\n"   // Установили блок 0x1A (26)
```

5.4.5 read_key_B

`read_key_B` - Выбор ключа, который будет использоваться для чтения карты. Двоичная цифра: 0 - ключ A, 1 - ключ B.

Примеры использования команды:

```
"get prog1 read_key_B\n"     // Проверяем ключ
"set prog1 read_key_B 0\n"   // Читать ключом A
"set prog1 read_key_B 1\n"   // Читать ключом B
```

Примеры ответов считывателя:

```
"get prog1 read_key_B: 0\r\n"      // Читаем ключом A
"set prog1 read_key_B 0: OK\r\n"   // Назначили чтение ключом A
"set prog1 read_key_B 1: OK\r\n"   // Назначили чтение ключом B
```

5.4.6 keyA_bytes

`keyA_bytes` - Байты непосредственно самого ключа A. Нужно установить, если настроено чтение карты ключом A или при необходимости выпуска карт. Ключ в картах SL3 имеет размер 16 байт. Ключ записывается в виде 32 шестнадцатеричных цифр. На текущий момент по соображениям безопасности прочитать ключи из ячейки программирования считывателя нельзя.

Примеры использования команды:

```
"get prog1 keyA_bytes\n"                  // Проверяем ключ A (не получим)
"set prog1 keyA_bytes FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF\n"
                                  // Установить ключ A (стандартный ключ)
"set prog1 keyA_bytes 000102030405060708090A0B0C0D0E0F\n"
                                             // Установить ключ A (свой)
```

Примеры ответов считывателя:

```
"get prog1 keyA_bytes: NOT_ALLOWED\r\n"         // Запрещено читать ключи
"set prog1 keyA_bytes FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF: OK\r\n"
                                 // Установили стандартный ключ на ключ A
"set prog1 keyA_bytes FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF: WRONG_LEN\r\n"
                                                  // Неправильная длина
```

5.4.7 keyB_bytes

keyB_bytes - Байты непосредственно самого ключа B. Нужно установить, если настроено чтение карты ключом B или при необходимости выпуска карт. Ключ в картах SL3 имеет размер 16 байт. Ключ записывается в виде 32 шестнадцатеричных цифр. На текущий момент по соображениям безопасности прочитать ключи из ячейки программирования считывателя нельзя.

Примеры использования команды:

```
"get prog1 keyB_bytes\n"                  // Проверяем ключ A (не получим)
"set prog1 keyB_bytes FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF\n"
                                  // Установить ключ A (стандартный ключ)
"set prog1 keyA_bytes 000102030405060708090A0B0C0D0E0F\n"
                                             // Установить ключ A (свой)
```

Примеры ответов считывателя:

```
"get prog1 keyB_bytes: NOT_ALLOWED\r\n"         // Запрещено читать ключи
"set prog1 keyB_bytes FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF: OK\r\n"
                                 // Установили стандартный ключ на ключ A
"set prog1 keyB_bytes FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF: WRONG_LEN\r\n"
                                                  // Неправильная длина
```

5.4.8 crypt

`crypt` - Требуется ли при чтении карты шифровать трафик между считывателем и картой. Поможет сохранить секретность кода идентификатора при попытке анализа трафика, но несколько увеличивает время чтения карт. Двоичная цифра: 0 - не шифровать трафик, 1 - шифровать трафик. Пока не реализовано, появится с дальнейшими обновлениями.

Примеры использования команды:

```
"get prog1 crypt\n"     // Проверяем параметр
"set prog1 crypt 0\n"   // Не использовать шифрование трафика
"set prog1 crypt 1\n"   // Использовать шифрование трафика
```

Примеры ответов считывателя:

```
"get prog1 crypt: 0\r\n"      // Шифрование трафика не используется
"set prog1 crypt 1: OK\r\n"   // Установлено использование шифрования
```

5.4.9 write_acc_bits

`write_acc_bits` - Биты доступа, записываемые на выпускаемые рабочие карты. Эти биты управляют, какими ключами доступ к каким ресурсам будет разрешён. Рекомендуется более подробно изучить этот вопрос по ссылкам ниже, если вы не знаете что это такое.

https://slebe.dev/mifarecalc/
http://calc.gmss.ru/Mifare1k/

Биты доступа записывается в виде 6 шестнадцатеричных цифр. Можно их не записывать, тогда будут использованы биты доступа 787788. Эти же биты доступа будут применены, если пользователь впишет некорректные или необратимые биты доступа. Такие биты доступа позволяют читать содержимое карты обоими ключами, а записывать только ключом B, и являются обратимыми.

Примеры использования команды:

```
"get prog1 write_acc_bits\n"          // Проверяем установленные биты
                                      //                          доступа
"set prog1 write_acc_bits FF0780\n"   // Установить стандартные биты
                                      //                          доступа
"set prog1 write_acc_bits 787788\n"   // Установить свои биты доступа
```

Примеры ответов считывателя:

```
"get prog1 write_acc_bits: 787788\r\n"        // Установлены биты доступа
                                              //                  787788
"set prog1 write_acc_bits FF0780: OK\r\n"
                                    // Установили стандартные биты доступа
"set prog1 write_acc_bits FF078: WRONG_LEN\r\n"     // Неправильная длина
```

5.4.10 `write_all_trailers`

`write_all_trailers` - Нужно ли перезаписывать все трейлеры карты (зашифровать все секторы карты). Полезно для защиты от некоторых типов атак, но существенно увеличивает время записи. Двоичная цифра, 0 - не нужно, 1 - нужно.

Примеры использования команды:

```
"get prog1 write_all_trailers\n"     // Проверяем потребность перезаписи
                                     //                    всех трейлеров
"set prog1 write_all_trailers 1\n"   // Установить потребность перезаписи
                                     //                    всех трейлеров
```

Примеры ответов считывателя:

```
"get prog1 write_all_trailers: 0\r\n"      // Потребность перезаписи всех
                                           //    трейлеров не установлена
"set prog1 write_all_trailers 1: OK\r\n"   // Установили потребность
                                           //   перезаписи всех трейлеров
```

5.5 Специализированные параметры для Mifare Desfire

Идентификаторы типа Mifare Desfire включают в себя импортные карты и брелки Mifare Desfire.

Структура их памяти отличается от идентификаторов типов SL1 и SL3, потому команды конфигурации тоже отличаются. Здесь используется файловая система, в которой есть Application (приложения), по функциям схожие с папками, и File (файлы).

5.5.1 `tx_params`

`tx_params` - Параметры передачи данных. Десятичное число. 0 - передаём только UID, 1 - передаём UID после успешного чтения файла, 2 - передаём содержимое файла.

Примеры использования команды:

```
"get prog1 tx_params\n"     // Проверяем параметры передачи данных
"set prog1 tx_params 0\n"   // передаём только UID
"set prog1 tx_params 2\n"   // передаём содержимое файла
```

Примеры ответов считывателя:

```
"get prog1 tx_params: 0\r\n"      // Назначено передавать только UID
"set prog1 tx_params 2: OK\r\n"   // Назначили передавать содержимое
                                  //                              файла
```

5.5.2 `vendor_control`

`vendor_control` - Контроль производителя. Используется при чтении 7-байтных UID, на другие сценарии не влияет. 0 - игнорируется нулевой байт 7-байтных UID, 1 - 7-байтные UID передаются целиком.

Примеры использования команды:

```
"get prog1 vendor_control\n"     // Проверяем контроль производителя
"set prog1 vendor_control 0\n"   // Снять контроль производителя
"set prog1 vendor_control 1\n"   // Установить контроль производителя
```

Примеры ответов считывателя:

```
"get prog1 vendor_control: 0\r\n"      // Контроль производителя снят
"set prog1 vendor_control 1: OK\r\n"   // Установили контроль
                                       //                   производителя
```

5.5.3 prox_check

`prox_check` - Proximity check, защита от атаки "Человек между". Проверяет время ответов карты, и если оно превышает установленный предел - соединение с картой объявляется скомпрометированным и карта не считывается. 0 - Proximity check выключен, 1 - Proximity check включен. Пока не реализовано, появится с дальнейшими обновлениями прошивки.

Примеры использования команды:

```
"get prog1 prox_check\n"     // Проверяем Proximity check
"set prog1 prox_check 0\n"   // Выключить Proximity check
"set prog1 prox_check 1\n"   // Включить Proximity check
```

Примеры ответов считывателя:

```
"get prog1 prox_check: 0\r\n"      // Proximity check выключен
"set prog1 prox_check 1: OK\r\n"   // Включили Proximity check
```

5.5.4 AID

`AID` - Application ID, идентификационный номер приложения. Имеет размер три байта. Устанавливается как шестизначное шестнадцатеричное число.

Примеры использования команды:

```
"get prog1 AID\n"          // Проверяем номер приложения
"set prog1 AID 000000\n"   // Установить приложение 0x000000
"set prog1 AID 01020A\n"   // Установить приложение 0x01020A
```

Примеры ответов считывателя:

```
"get prog1 AID: 000000\r\n"            // Установлено приложение 0x000000
"set prog1 AID 01020A: OK\r\n"         // Установили приложение 0x01020A
"set prog1 AID 1020A: WRONG_LEN\r\n"   // Неправильная длина
```

5.5.5 FID

FID - File ID, идентификационный номер файла. Имеет размер один байт. Устанавливается как двухзначное шестнадцатеричное число.

Примеры использования команды:

```
"get prog1 FID\n"      // Проверяем номер файла
"set prog1 FID 00\n"   // Установить файл 0x00
"set prog1 FID F0\n"   // Установить файл 0xF0
```

Примеры ответов считывателя:

```
"get prog1 FID: 00\r\n"            // Установлен файл 0x00
"set prog1 FID F0: OK\r\n"         // Установили файл 0xF0
"set prog1 FID F: WRONG_LEN\r\n"   // Неправильная длина
```

5.5.6 crypt

crypt - Метод шифрования. Устанавливается одной десятичной цифрой. 0 - без шифрования, 1 - DES или 3DES, 2 - 3K3DES, 3 - AES.

Примеры использования команды:

```
"get prog1 crypt\n"     // Проверяем метод шифрования
"set prog1 crypt 0\n"   // Без шифрования
"set prog1 crypt 3\n"   // Шифрование AES
```

Примеры ответов считывателя:

```
"get prog1 crypt: 0\r\n"                // Установлено отсутствие
                                        //                      шифрования
"set prog1 crypt 3: OK\r\n"             // Установили шифрование AES
"set prog1 crypt 4: WRONG_VALUE\r\n"    // Неправильное значение
```

5.5.7 key

key - ключ, которым будет выполняться авторизация в приложение. Состоит из 16 байт для DES и AES, либо из 24 байт для 3K3DES. 16-байтный ключ вводится как 32 шестнадцатеричные цифры. 24-байтный ключ вводится как 48 шестнадцатеричных цифр. На текущий момент по соображениям безопасности прочитать ключ из ячейки программирования считывателя нельзя.

Примеры использования команды:

```
"get prog1 key\n"                           // Проверяем ключ (не получим)
"set prog1 key FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF\n"
                        // Установить ключ (стандартный 16-байтный ключ)
"set prog1 key FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF\n"
                        // Установить ключ (стандартный 24-байтный ключ)
```

Примеры ответов считывателя:

```
"get prog1 key: NOT_ALLOWED\n"   // Запрещено читать ключи
"set prog1 key FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF: OK\n"
                        // Установили ключ (стандартный 16-байтный ключ)
"set prog1 key FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF: WRONG_LEN\n"
                                                  // Неправильная длина
"set prog1 key FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF: OK\n"
                        // Установили ключ (стандартный 24-байтный ключ)
"set prog1 key FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF: WRONG_LEN\n"
                                                  // Неправильная длина
```

5.6 Специализированные параметры для Mobile ID

Идентификаторы типа Mobile ID - это идентификаторы, передаваемые на считыватель по BLE с мобильного телефона. Дальность передачи идентификаторов до 4-5 метров.

Приложения мобильной идентификации существуют для устройств с операционной системой Android и устройств Apple. В связи с наличием особенностей в этих группах устройств, их настройки выполняются также отдельно. Дальность считывания идентификаторов настраивается в диапазоне от 4-5 сантиметров до 4-5 метров путём установки минимального значения RSSI (минимальный уровень мощности принимаемого сигнала в децибелах по милливатту, или дБм) в диапазоне от -40 до -95. Изменение RSSI на 10 дБм изменяет дальность примерно в 3 раза. Также мобильные идентификаторы могут передаваться по кнопке, по включению экрана или в режиме "свободные руки". А ещё в настройках в приложении мобильной идентификации пользователь может корректировать значение RSSI для небольшого изменения дальности считывания.

5.6.1 `min_RSSI_Android`

`min_RSSI_Android` - Минимальное значение RSSI для Mobile ID на устройствах на операционной системе Android. Десятичное целое число со знаком. Рекомендуется от -90 до -50. Допустимо от -95 до -40. Чем больше число, тем меньше дальность. При -100 дальность максимальна, при -40 - минимальна.

Примеры использования команды:

```
"get prog1 min_RSSI_Android\n"       // Проверяем минимальный RSSI
"set prog1 min_RSSI_Android -90\n"   // Установить минимальный RSSI -90
"set prog1 min_RSSI_Android -50\n"   // Установить минимальный RSSI -50
```

Примеры ответов считывателя:

```
"get prog1 min_RSSI_Android: -80\r\n"      // Установлен миним. RSSI -80
"set prog1 min_RSSI_Android -60: OK\r\n"   // Установили миним. RSSI -60
```

5.6.2 min_RSSI_Apple

min_RSSI_Apple - Минимальное значение RSSI для Mobile ID на устройствах Apple. Десятичное целое число со знаком. Рекомендуется от -90 до -50. Допустимо от -95 до -40. Чем больше число, тем меньше дальность. При -95 дальность максимальна, при -40 - минимальна.

Примеры использования команды:

```
"get prog1 min_RSSI_Apple\n"       // Проверяем минимальный RSSI
"set prog1 min_RSSI_Apple -90\n"   // Установить минимальный RSSI -90
"set prog1 min_RSSI_Apple -50\n"   // Установить минимальный RSSI -50
```

Примеры ответов считывателя:

```
"get prog1 min_RSSI_Apple: -80\r\n"      // Установлен миним. RSSI -80
"set prog1 min_RSSI_Apple -60: OK\r\n"   // Установили миним. RSSI -60
```

5.6.3 byButton_Android

byButton_Android - Разрешено ли передавать идентификатор в режиме "По кнопке" устройствам с операционной системой Android. Двоичная цифра, где 0 - запрещено работать по кнопке, 1 - разрешено работать по кнопке.

Примеры использования команды:

```
"get prog1 byButton_Android\n"     // Проверяем разрешение
"set prog1 byButton_Android 0\n"   // Запретить работать по кнопке
"set prog1 byButton_Android 1\n"   // Разрешить работать по кнопке
```

Примеры ответов считывателя:

```
"get prog1 byButton_Android: 0\r\n"      // Запрещено работать по кнопке
"set prog1 byButton_Android 1: OK\r\n"   // Разрешили работать по кнопке
```

5.6.4 byScreen_Android

byScreen_Android - Разрешено ли передавать идентификатор в режиме "По включению экрана" устройствам с операционной системой Android. Двоичная цифра, где 0 - запрещено работать по включению экрана, 1 - разрешено работать по включению экрана.

Примеры использования команды:

```
"get prog1 byScreen_Android\n"     // Проверяем разрешение
"set prog1 byScreen_Android 0\n"   // Запретить работать по включению
                                   //                              экрана
"set prog1 byScreen_Android 1\n"   // Разрешить работать по включению
                                   //                              экрана
```

Примеры ответов считывателя:

```
"get prog1 byScreen_Android: 0\r\n"      // Запрещено работать по
                                         //              включению экрана
"set prog1 byScreen_Android 1: OK\r\n"   // Разрешили работать по
                                         //              включению экрана
```

5.6.5 byHF_Android

byHF_Android - Разрешено ли передавать идентификатор в режиме "Свободные руки" (Hands-Free) устройствам с операционной системой Android. Двоичная цифра, где 0 - запрещено работать в режиме "Свободные руки", 1 - разрешено работать в режиме "Свободные руки".

Примеры использования команды:

```
"get prog1 byHF_Android\n"     // Проверяем разрешение
"set prog1 byHF_Android 0\n"   // Запретить работать в режиме "Свободные
                               //                                  руки"
"set prog1 byHF_Android 1\n"   // Разрешить работать в режиме "Свободные
                               //                                  руки"
```

Примеры ответов считывателя:

```
"get prog1 byScreen_Android: 0\r\n"      // Запрещено работать в режиме
                                         //              "Свободные руки"
"set prog1 byScreen_Android 1: OK\r\n"   // Разрешили работать в режиме
                                         //              "Свободные руки"
```

5.6.6 byButton_Apple

byButton_Apple - Разрешено ли передавать идентификатор в режиме "По кнопке" устройствам Apple. Двоичная цифра, где 0 - запрещено работать по кнопке, 1 - разрешено работать по кнопке.

Примеры использования команды:

```
"get prog1 byButton_Apple\n"     // Проверяем разрешение
"set prog1 byButton_Apple 0\n"   // Запретить работать по кнопке
"set prog1 byButton_Apple 1\n"   // Разрешить работать по кнопке
```

Примеры ответов считывателя:

```
"get prog1 byButton_Apple: 0\r\n"      // Запрещено работать по кнопке
"set prog1 byButton_Apple 1: OK\r\n"   // Разрешили работать по кнопке
```

5.6.7 byScreen_Apple

byScreen_Apple - Разрешено ли передавать идентификатор в режиме "По включению экрана" устройствам Apple. Двоичная цифра, где 0 - запрещено работать по включению экрана, 1 - разрешено работать по включению экрана.

Примеры использования команды:

```
"get prog1 byScreen_Apple\n"     // Проверяем разрешение
"set prog1 byScreen_Apple 0\n"   // Запретить работать по включению
                                 //                              экрана
"set prog1 byScreen_Apple 1\n"   // Разрешить работать по включению
                                 //                              экрана
```

Примеры ответов считывателя:

```
"get prog1 byScreen_Apple: 0\r\n"      // Запрещено работать по включению
                                       //                              экрана
"set prog1 byScreen_Apple 1: OK\r\n"   // Разрешили работать по включению
                                       //                              экрана
```

5.6.8 byHF_Apple

byHF_Apple - Разрешено ли передавать идентификатор в режиме "Свободные руки" (Hands-Free) устройствам Apple. Двоичная цифра, где 0 - запрещено работать в режиме "Свободные руки", 1 - разрешено работать в режиме "Свободные руки".

Примеры использования команды:

```
"get prog1 byHF_Apple\n"     // Проверяем разрешение
"set prog1 byHF_Apple 0\n"   // Запретить работать в режиме "Свободные
                             //                                      руки"
"set prog1 byHF_Apple 1\n"   // Разрешить работать в режиме "Свободные
                             //                                      руки"
```

Примеры ответов считывателя:

```
"get prog1 byHF_Apple: 0\r\n"      // Запрещено работать в режиме
                                   //                    "Свободные руки"
"set prog1 byHF_Apple 1: OK\r\n"   // Разрешили работать в режиме
                                   //                    "Свободные руки"
```

5.6.9 RSSI_corr_Android

RSSI_corr_Android - Разрешено ли применять коррекцию RSSI для устройств с операционной системой Android. Двоичная цифра, где 0 - не применять коррекцию RSSI, 1 - применять коррекцию RSSI.

Примеры использования команды:

```
"get prog1 RSSI_corr_Android\n"     // Проверяем применение коррекции
"set prog1 RSSI_corr_Android 0\n"   // Не применять коррекцию RSSI
"set prog1 RSSI_corr_Android 1\n"   // Применять коррекцию RSSI
```

Примеры ответов считывателя:

```
"get prog1 RSSI_corr_Android: 0\r\n"      // Коррекция RSSI не
                                          //                  применяется
"set prog1 RSSI_corr_Android 1: OK\r\n"   // Применяем коррекцию RSSI
```

5.6.10 RSSI_corr_Apple

RSSI_corr_Apple - Разрешено ли применять коррекцию RSSI для устройств Apple. Двоичная цифра, где 0 - не применять коррекцию RSSI, 1 - применять коррекцию RSSI.

Примеры использования команды:

```
"get prog1 RSSI_corr_Apple\n"     // Проверяем применение коррекции
"set prog1 RSSI_corr_Apple 0\n"   // Не применять коррекцию RSSI
"set prog1 RSSI_corr_Apple 1\n"   // Применять коррекцию RSSI
```

Примеры ответов считывателя:

```
"get prog1 RSSI_corr_Apple: 0\r\n"      // Коррекция RSSI не применяется
"set prog1 RSSI_corr_Apple 1: OK\r\n"   // Применяем коррекцию RSSI
```

5.7 Специализированные параметры для PW-Tag

Идентификаторы типа PW-Tag - это брелки с активными BLE-метками с дальностью до 10-15 метров. Дальность считывания идентификаторов настраивается в диапазоне от 10-15 сантиметров до 10-15 метров путём установки минимального значения RSSI (минимальный уровень мощности принимаемого сигнала в децибелах по милливатту, или дБм) в диапазоне от -40 до -95. Изменение RSSI на 10 дБм изменяет дальность примерно в 3 раза.

5.7.1 `min_RSSI`

`min_RSSI` - Минимальное значение RSSI для PW-Tag. Десятичное целое число со знаком. Рекомендуется от -90 до -50. Допустимо от -95 до -40. Чем больше число, тем меньше дальность. При -95 дальность максимальна, при -40 - минимальна.

Примеры использования команды:

```
"get prog1 min_RSSI\n"       // Проверяем минимальный RSSI
"set prog1 min_RSSI -90\n"   // Установить минимальный RSSI -90
"set prog1 min_RSSI -50\n"   // Установить минимальный RSSI -50
```

Примеры ответов считывателя:

```
"get prog1 min_RSSI: -80\r\n"      // Установлен минимальный RSSI -80
"set prog1 min_RSSI -60: OK\r\n"   // Установили минимальный RSSI -60
```

6. Команды, выполняемые под заголовком do

Под заголовком do находятся исполняемые команды. Они предназначены для выполнения считывателем различных действий. В частности, под этим заголовком будет происходить авторизация ключом инженера для выполнения всех остальных действий со считывателем.

6.1 Команды авторизации

6.1.1 auth

auth - Команда авторизации ключом инженера в считыватель для выполнения всех остальных операций. Ключ инженера состоит из 6 десятичных цифр. Ключ инженера по умолчанию - 128512.

Пример использования команды:

```
"do auth 128512\n"   // Авторизация ключом инженера по умолчанию
```

Примеры ответов считывателя:

```
"do auth 128512: OK\r\n"            // Авторизация ключом инженера
                                    //                  выполнена успешно
"do auth 000001: WRONG_VALUE\r\n"   // Ключ инженера введён неверно
"do auth 00000: WRONG_LEN\r\n"      // Неверная длина ключа инженера
```

6.1.2 emerg_bypass

emerg_bypass - Команда аварийного обхода авторизации ключом инженера. При этом даётся доступ только к трём командам - сброс настроек в заводское состояние, сброс считывателя в заводское состояние и перезагрузка. Если ранее к считывателю были подключены бинарным протоколом или по USB, или были авторизованы ключом инженера в

текстовом протоколе - ранее установленное соединение будет разорвано с соответствующей индикацией.

Пример использования команды:

```
"do emerg_bypass\n"   // Аварийный обход авторизации
```

Примеры ответов считывателя:

```
"do emerg_bypass: OK\r\n"   // Аварийный обход авторизации выполнен
```

Типичное применение данной команды выглядит так:

```
"do emerg_bypass\n"    // Аварийный обход авторизации
"do dropdown\n"        // Сброс считывателя
"do reboot\n"          // Перезагрузка
```

6.1.3 `disconnect`

`disconnect` - Команда отключения от считывателя в текстовом протоколе. Разрешает другие подключения к считывателю.

Пример использования команды:

```
"do disconnect\n"   // Отключаемся
```

Пример ответа считывателя:

```
"do disconnect: OK\r\n"   // Отключились
```

6.2 Команды управления считывателем

6.2.1 dropdown

dropdown - Сбросить считыватель в заводское состояние.

Внимание! Все данные будут безвозвратно удалены со считывателя!

Пример использования команды:

```
"do dropdown\n"   // Сбросить считыватель
```

Пример ответа считывателя:

```
"do dropdown: OK\r\n"   // Сброс считывателя выполнен успешно
```

6.2.2 drop_progs

drop_progs - Сбросить ячейки программирования в значения по умолчанию. Ячейки программирования будут заполнены указанными ниже идентификаторами с настройками по умолчанию (см. 5.1.2 ident_type, стр. 32-34):

1) Mifare Ultralight;
2) Mifare SL1;
3) Mifare SL3;
4) Mifare Desfire;
5) Mobile ID;
6) PW-Tag.

Пример использования команды:

```
"do drop_progs\n"   // Сбросить ячейки программирования
```

Пример ответа считывателя:

```
"do drop_progs: OK\r\n"  // Сброс ячеек программирования выполнен успешно
```

6.2.3 clean_progs

clean_progs - Удалить все ячейки программирования прямо сейчас. Когда в считывателе нет ни одной запрограммированной ячейки программирования, он читает UID у всех поддерживаемых карт, читает PW-Tag с минимальным RSSI -60, читает Mobile ID с разрешением всех режимов с минимальным RSSI -60.

Пример использования команды:

```
"do clean_progs\n"   // Удалить все ячейки программирования
```

Пример ответа считывателя:

```
"do clean_progs: OK\r\n"   // Все ячейки программирования удалены успешно
```

6.2.4 reboot

reboot - Перезагрузка считывателя. Обычно применяется совместно со сбросом считывателя к заводскому состоянию. Также может использоваться как имитация отключения питания для наблюдения за поведением считывателя при запуске. При этом авторизация ключом инженера завершается, и для продолжения работы со считывателем потребуется авторизоваться снова.

Пример использования команды:

```
"do reboot\n"   // Перезагрузка
```

Пример ответа считывателя:

```
"do reboot: OK\r\n"   // Перезагрузка начинается
```

6.2.5 read_card

read_card - Команда чтения приложенной карты. Похожа на укороченную команду 'r', но ответ формируется по структуре полных команд. Данные идентификатора прочитанной карты отправляются после завершения чтения.

Пример использования команды:

```
"do read_card\n"   // Читаем приложенную карту
```

Примеры ответов считывателя:

```
"do read_card: NO_CARD\n"   // Карта не найдена
"do read_card: OK\r\n"      // Нашли карту, читаем, ждите результат
```

6.2.6 beep

beep - Команда писка. Считыватель пискнет для демонстрации текущей громкости и частоты.

Пример использования команды:

```
"do beep\n"   // Просим пискнуть
```

Пример ответа считывателя:

```
"do beep: OK\r\n"   // Считыватель подтвердил запрос и вскоре пискнет
```

6.2.7 blink

blink - Команда моргания. Считыватель моргнёт красным, зелёным и синим цветами для демонстрации текущей яркости.

Пример использования команды:

```
"do blink\n"   // Просим моргнуть
```

Пример ответа считывателя:

```
"do blink: OK\r\n"   // Считыватель подтвердил запрос и вскоре моргнёт
```

6.3 Команды записи карт

6.3.1 `write_card`

`write_card` - Команда записи карты. Для этого необходимо указать номер ячейки программирования, который будет использоваться в дальнейшем для чтения этой карты, а так же сам идентификатор в виде чётного количества шестнадцатеричных цифр от 2 до 32 (от 1 до 16 байт).

Важно - идентификатор записывается всегда в прямом порядке, в то время как при чтении карты передача прочитанного идентификатора традиционно происходит в обратном порядке (порядок передачи байт можно настроить в ячейке программирования).

При этом стартовый байт, указанный в ячейке программирования, учитывается. Например, если указан стартовый байт 3, то при записи первые три байта блока будут нулевыми, а следующие за ними байты будут содержать указанный идентификатор в прямом порядке.

Если карта не соответствует уровню защиты ячейки программирования (например, приложена карта в SL0, а ячейка программирования требует SL1 или SL3), считыватель попытается перевести карту на необходимый уровень защиты, если это возможно. В противном случае считыватель вернёт ошибку неверного типа карты или ошибку перевода на необходимый уровень защиты.

Команда применима только для ранее не записанных карт - если сектор карты повреждён или защищён ключом шифрования, отличным от стандартного (состоящего из байт `0xFF`), считыватель вернёт ошибку авторизации. Если карта защищена стандартным ключом шифрования, но в блоке уже что-то записано - считыватель вернёт ошибку, что карта уже была записана. Для перезаписи карты используйте команду `rewrite_card`.

Пример использования команды:

```
"do write_card 1 010203\n"
          // Записать в карту ячейкой программирования 1 данные 0x010203
```

Примеры ответов считывателя:

```
"do write_card 1 010203: OK\r\n"    // Карта записана и проверена успешно

"do write_card 1 010203: PROG_IS_NOT_USED\r\n"
                          // Ячейка программирования не запрограммирована

"do write_card 1 010203: PROG_READING_ERROR\r\n"
                                  // Ошибка чтения ячейки программирования

"do write_card 1 010203: PROG_IS_BROKEN\r\n"
                                     // Ячейка программирования повреждена

"do write_card 1 010203: PROG_HAS_AN_UNSCRIPTABLE_IDENTIFIER\r\n"
        // Ячейка программирования содержит незаписываемый идентификатор

"do write_card 1 010203: PROG_IS_NON_AUTH\r\n"
                       // Ячейка программирования не требует авторизацию

"do write_card 1 010203: WRONG_IDENTIFICATOR_TYPE\r\n"
          // Неправильный тип идентификатора в ячейке программирования -
          //        поддерживаются Mifare Ultralight, SL1, SL3 и Desfire

"do write_card 1 010203: CAN_NOT_SELECT_CARD\r\n"
               // Карта не найдена - карта не приложена, либо неисправна

"do write_card 1 010203: DIFFERENT_CARD_TYPE\r\n"
// Найденная карта имеет неверный тип (например, ячейка SL1, а карта SL3)

"do write_card 1: SL0_TO_SL1_CONVERSION ERROR\r\n"
                             // Ошибка перевода карты из режима SL0 в SL1

"do write_card 1: SL1_TO_SL3_CONVERSION ERROR\r\n"
                             // Ошибка перевода карты из режима SL1 в SL3

"do write_card 1 010203: CAN_NOT_AUTHORIZE_CARD\r\n"
                                  // Не получилось авторизоваться в карту

"do write_card 1 010203: CAN_NOT_READ_BLOCK\r\n"
                                           // Ошибка чтения блока карты

"do write_card 1 010203: CAN_NOT_WRITE_BLOCK\r\n"
                                           // Ошибка записи блока карты

"do write_card 1 010203: CAN_NOT_CHECK_BLOCK\r\n"
                                  // Ошибка при проверке записи блока карты
```

```
"do write_card 1 010203: BLOCK_CHECKED_WRONG_DATA\r\n"
       // Карта проверена - прочитанные данные не совпадают с записанными

"do write_card 1 010203: CARD_IS_ALREADY_WRITTEN\r\n"
                                          // Карта уже была записана ранее
```

6.3.2 write_card_auto

write_card_auto - Команда автоматической записи карты. Данную команду необходимо дополнить только номером ячейки программирования, а номер идентификатора будет взят из конфигурации считывателя. После успешной записи он будет автоматически инкрементирован, начиная с нулевого байта.

Например:

До инкрементации: [01 02 03]
После инкрементации: [02 02 03]

До инкрементации: [FF 02 03]
После инкрементации: [00 03 03]

До инкрементации: [FF FF FF]
После инкрементации: [00 00 00]

Важно - идентификатор записывается всегда в прямом порядке, в то время как при чтении карты передача прочитанного идентификатора традиционно происходит в обратном порядке (порядок передачи байт можно настроить в ячейке программирования).

При этом стартовый байт, указанный в ячейке программирования, учитывается. Например, если указан стартовый байт 3, то при записи первые три байта блока будут нулевыми, а следующие за ними байты будут содержать стартовый идентификатор в прямом порядке.

Если карта не соответствует уровню защиты ячейки программирования (например, приложена карта в SL0, а ячейка программирования требует SL1 или SL3), считыватель попытается перевести карту на необходимый уровень защиты, если это возможно. В противном случае считыватель вернёт ошибку неверного типа карты или ошибку перевода на необходимый уровень защиты.

Команда применима только для ранее не записанных карт - если сектор карты повреждён или защищён ключом шифрования, отличным от стандартного (состоящего из байт 0xFF), считыватель вернёт ошибку авторизации. Если карта защищена стандартным ключом шифрования, но в блоке уже что-то записано - считыватель вернёт ошибку, что карта уже была записана. Для перезаписи карты используйте команду `rewrite_card_auto`.

Пример использования команды:

```
"do write_card_auto 1\n"
              // Автоматически записать карту ячейкой программирования 1
```

Примеры ответов считывателя:

```
"do write_card_auto 1: OK\r\n"      // Карта записана и проверена успешно

"do write_card_auto 1: PROG_IS_NOT_USED\r\n"
                          // Ячейка программирования не запрограммирована

"do write_card_auto 1: PROG_READING_ERROR\r\n"
                                  // Ошибка чтения ячейки программирования

"do write_card_auto 1: START_IDENT_NOT_SET\r\n"
                                       // Стартовый идентификатор не задан

"do write_card_auto 1: PROG_IS_BROKEN\r\n"
                                     // Ячейка программирования повреждена

"do write_card_auto 1: PROG_HAS_AN_UNSCRIPTABLE_IDENTIFIER\r\n"
        // Ячейка программирования содержит незаписываемый идентификатор

"do write_card_auto 1: PROG_IS_NON_AUTH\r\n"
                        // Ячейка программирования не требует авторизацию
```

```
"do write_card_auto 1: WRONG_IDENTIFICATOR_TYPE\r\n"
          // Неправильный тип идентификатора в ячейке программирования -
          //        поддерживаются Mifare Ultralight, SL1, SL3 и Desfire

"do write_card_auto 1: CAN_NOT_SELECT_CARD\r\n"
               // Карта не найдена - карта не приложена, либо неисправна

"do write_card_auto 1: DIFFERENT_CARD_TYPE\r\n"
// Найденная карта имеет неверный тип (например, ячейка SL1, а карта SL3)

"do write_card_auto 1: SL0_TO_SL1_CONVERSION ERROR\r\n"
                            // Ошибка перевода карты из режима SL0 в SL1

"do write_card_auto 1: SL1_TO_SL3_CONVERSION ERROR\r\n"
                            // Ошибка перевода карты из режима SL1 в SL3

"do write_card_auto 1: CAN_NOT_AUTHORIZE_CARD\r\n"
                                  // Не получилось авторизоваться в карту

"do write_card_auto 1: CAN_NOT_READ_BLOCK\r\n"
                                              // Ошибка чтения блока карты

"do write_card_auto 1: CAN_NOT_WRITE_BLOCK\r\n"
                                              // Ошибка записи блока карты

"do write_card_auto 1: CAN_NOT_CHECK_BLOCK\r\n"
                                    // Ошибка при проверке записи блока карты

"do write_card_auto 1: BLOCK_CHECKED_WRONG_DATA\r\n"
      // Карта проверена - прочитанные данные не совпадают с записанными

"do write_card_auto 1: CARD_IS_ALREADY_WRITTEN\r\n"
                                          // Карта уже была записана ранее
```

6.3.3 rewrite_card

`rewrite_card` - Команда перезаписи карты. Для этого необходимо указать номер ячейки программирования, который будет использоваться в дальнейшем для чтения этой карты, а так же сам идентификатор в виде чётного количества шестнадцатеричных цифр от 2 до 32 (от 1 до 16 байт).

Важно - идентификатор записывается всегда в прямом порядке, в то время как при чтении карты передача прочитанного идентификатора

традиционно происходит в обратном порядке (порядок передачи байт можно настроить в ячейке программирования).

При этом стартовый байт, указанный в ячейке программирования, учитывается. Например, если указан стартовый байт 3, то при записи первые три байта блока будут нулевыми, а следующие за ними байты будут содержать указанный идентификатор в прямом порядке.

Если карта не соответствует уровню защиты ячейки программирования (например, приложена карта в SL0, а ячейка программирования требует SL1 или SL3), считыватель вернёт ошибку неверного типа карты.

Команда применима как для чистых карт, так и для ранее записанных этой же ячейкой программирования. Если есть опасения, что будет перезаписана ранее записанная рабочая карта - воспользуйтесь командой `write_card`.

Пример использования команды:

```
"do rewrite_card 1 010203\n"
        // Перезаписать карту ячейкой программирования 1 данными 0x010203
```

Примеры ответов считывателя:

```
"do rewrite_card 1 010203: OK\r\n"
                                    // Карта записана и проверена успешно

"do rewrite_card 1 010203: PROG_IS_NOT_USED\r\n"
                            // Ячейка программирования не запрограммирована

"do rewrite_card 1 010203: PROG_READING_ERROR\r\n"
                                  // Ошибка чтения ячейки программирования

"do rewrite_card 1 010203: PROG_IS_BROKEN\r\n"
                                    // Ячейка программирования повреждена

"do rewrite_card 1 010203: PROG_HAS_AN_UNSCRIPTABLE_IDENTIFIER\r\n"
         // Ячейка программирования содержит незаписываемый идентификатор

"do rewrite_card 1 010203: PROG_IS_NON_AUTH\r\n"
                         // Ячейка программирования не требует авторизацию
```

```
"do rewrite_card 1 010203: WRONG_IDENTIFICATOR_TYPE\r\n"
          // Неправильный тип идентификатора в ячейке программирования -
          //        поддерживаются Mifare Ultralight, SL1, SL3 и Desfire

"do rewrite_card 1 010203: CAN_NOT_SELECT_CARD\r\n"
               // Карта не найдена - карта не приложена, либо неисправна

"do rewrite_card 1 010203: DIFFERENT_CARD_TYPE\r\n"
// Найденная карта имеет неверный тип (например, ячейка SL1, а карта SL3)

"do rewrite_card 1: SL0_TO_SL1_CONVERSION ERROR\r\n"
                             // Ошибка перевода карты из режима SL0 в SL1

"do rewrite_card 1: SL1_TO_SL3_CONVERSION ERROR\r\n"
                             // Ошибка перевода карты из режима SL1 в SL3

"do rewrite_card 1 010203: CAN_NOT_AUTHORIZE_CARD\r\n"
                                  // Не получилось авторизоваться в карту

"do rewrite_card 1 010203: CAN_NOT_READ_BLOCK\r\n"
                                          // Ошибка чтения блока карты

"do rewrite_card 1 010203: CAN_NOT_WRITE_BLOCK\r\n"
                                          // Ошибка записи блока карты

"do rewrite_card 1 010203: CAN_NOT_CHECK_BLOCK\r\n"
                                    // Ошибка при проверке записи блока карты

"do rewrite_card 1 010203: BLOCK_CHECKED_WRONG_DATA\r\n"
      // Карта проверена - прочитанные данные не совпадают с записанными
```

6.3.4 rewrite_card_auto

rewrite_card_auto - Команда автоматической перезаписи карты. Данную команду необходимо дополнить только номером ячейки программирования, а номер идентификатора будет взят из конфигурации считывателя. После успешной записи он будет автоматически инкрементирован, начиная с нулевого байта.

Например:

До инкрементации: `[01 02 03]`
После инкрементации: `[02 02 03]`

До инкрементации: `[FF 02 03]`
После инкрементации: `[00 03 03]`

До инкрементации: `[FF FF FF]`
После инкрементации: `[00 00 00]`

Важно - идентификатор записывается всегда в прямом порядке, в то время как при чтении карты передача прочитанного идентификатора традиционно происходит в обратном порядке (порядок передачи байт можно настроить в ячейке программирования).

При этом стартовый байт, указанный в ячейке программирования, учитывается. Например, если указан стартовый байт 3, то при записи первые три байта блока будут нулевыми, а следующие за ними байты будут содержать стартовый идентификатор в прямом порядке.

Если карта не соответствует уровню защиты ячейки программирования (например, приложена карта в SL0, а ячейка программирования требует SL1 или SL3), считыватель вернёт ошибку неверного типа карты.

Команда применима как для чистых карт, так и для ранее записанных. Если есть опасения, что будет перезаписана ранее записанная рабочая карта - воспользуйтесь командой `write_card_auto`.

Пример использования команды:

```
"do rewrite_card_auto 1\n"
          // Автоматически перезаписать карту ячейкой программирования 1
```

Примеры ответов считывателя:

```
"do rewrite_card_auto 1: OK\r\n"    // Карта записана и проверена успешно

"do rewrite_card_auto 1: PROG_IS_NOT_USED\r\n"
                          // Ячейка программирования не запрограммирована

"do rewrite_card_auto 1: PROG_READING_ERROR\r\n"
                                // Ошибка чтения ячейки программирования

"do rewrite_card_auto 1: START_IDENT_NOT_SET\r\n"
                                     // Стартовый идентификатор не задан

"do rewrite_card_auto 1: PROG_IS_BROKEN\r\n"
                                   // Ячейка программирования повреждена

"do rewrite_card_auto 1: PROG_HAS_AN_UNSCRIPTABLE_IDENTIFIER\r\n"
        // Ячейка программирования содержит незаписываемый идентификатор

"do rewrite_card_auto 1: PROG_IS_NON_AUTH\r\n"
                       // Ячейка программирования не требует авторизацию

"do rewrite_card_auto 1: WRONG_IDENTIFICATOR_TYPE\r\n"
          // Неправильный тип идентификатора в ячейке программирования -
          //        поддерживаются Mifare Ultralight, SL1, SL3 и Desfire

"do rewrite_card_auto 1: CAN_NOT_SELECT_CARD\r\n"
                // Карта не найдена - карта не приложена, либо неисправна

"do rewrite_card_auto 1: DIFFERENT_CARD_TYPE\r\n"
// Найденная карта имеет неверный тип (например, ячейка SL1, а карта SL3)

"do rewrite_card_auto 1: SL0_TO_SL1_CONVERSION ERROR\r\n"
                            // Ошибка перевода карты из режима SL0 в SL1

"do rewrite_card_auto 1: SL1_TO_SL3_CONVERSION ERROR\r\n"
                            // Ошибка перевода карты из режима SL1 в SL3

"do rewrite_card_auto 1: CAN_NOT_AUTHORIZE_CARD\r\n"
                                  // Не получилось авторизоваться в карту

"do rewrite_card_auto 1: CAN_NOT_READ_BLOCK\r\n"
                                              // Ошибка чтения блока карты

"do rewrite_card_auto 1: CAN_NOT_WRITE_BLOCK\r\n"
                                              // Ошибка записи блока карты

"do rewrite_card_auto 1: CAN_NOT_CHECK_BLOCK\r\n"
                                 // Ошибка при проверке записи блока карты
```

```
"do rewrite_card_auto 1: BLOCK_CHECKED_WRONG_DATA\r\n"
      // Карта проверена - прочитанные данные не совпадают с записанными
```

6.3.5 flush_card

flush_card - Команда очистки карты. Очищает один сектор карты (используемый для хранения идентификатора), а также меняет ключи, биты доступа и пользовательский байт на стандартные. Если в ячейке программирования для записи идентификаторов указана необходимость перезаписи всех трейлеров карты (write_all_trailers) - будет очищена вся карта целиком (займёт больше времени). Для этого необходимо указать номер ячейки программирования, которой эта карта ранее была запрограммирована. Повреждённые секторы, а так же секторы с неизвестными ключами очищены не будут!

Пример использования команды:

```
"do flush_card 1\n"          // Очистить карту ячейкой программирования 1
```

Примеры ответов считывателя:

```
"do flush_card 1: OK\r\n"                          // Карта очищена успешно

"do flush_card 1: PROG_IS_NOT_USED\r\n"
                          // Ячейка программирования не запрограммирована

"do flush_card 1: PROG_READING_ERROR\r\n"
                                  // Ошибка чтения ячейки программирования

"do flush_card 1: PROG_IS_BROKEN\r\n"
                                     // Ячейка программирования повреждена

"do flush_card 1: PROG_HAS_AN_UNSCRIPTABLE_IDENTIFIER\r\n"
        // Ячейка программирования содержит незаписываемый идентификатор

"do flush_card 1: PROG_IS_NON_AUTH\r\n"
                        // Ячейка программирования не требует авторизацию

"do flush_card 1: WRONG_IDENTIFICATOR_TYPE\r\n"
          // Неправильный тип идентификатора в ячейке программирования -
          //        поддерживаются Mifare Ultralight, SL1, SL3 и Desfire
```

```
"do flush_card 1: CAN_NOT_SELECT_CARD\r\n"
                // Карта не найдена - карта не приложена, либо неисправна

"do flush_card 1: DIFFERENT_CARD_TYPE\r\n"
// Найденная карта имеет неверный тип (например, ячейка SL1, а карта SL3)

"do flush_card 1: CAN_NOT_AUTHORIZE_CARD\r\n"
                                  // Не получилось авторизоваться в карту

"do flush_card 1: CAN_NOT_WRITE_BLOCK\r\n"   // Ошибка записи блока карты
```

6.3.6 flush_card_all

flush_card_all - Команда принудительной полной очистки карты. Очищает все секторы карты, а также меняет ключи, биты доступа и пользовательский байт на стандартные, независимо от установок в ячейке программирования для записи идентификаторов. Для этого необходимо указать номер ячейки программирования, которой эта карта ранее была запрограммирована. Повреждённые секторы, а так же секторы с неизвестными ключами очищены не будут!

Пример использования команды:

```
"do flush_card_all 1\r\n"
                 // Полностью очистить карту ячейкой программирования 1
```

Примеры ответов считывателя:

```
"do flush_card_all 1: OK\r\n"          // Карта полностью очищена успешно

"do flush_card_all 1: PROG_IS_NOT_USED\r\n"
                         // Ячейка программирования не запрограммирована

"do flush_card_all 1: PROG_READING_ERROR\r\n"
                                 // Ошибка чтения ячейки программирования

"do flush_card_all 1: PROG_IS_BROKEN\r\n"
                                   // Ячейка программирования повреждена

"do flush_card_all 1: PROG_HAS_AN_UNSCRIPTABLE_IDENTIFIER\r\n"
        // Ячейка программирования содержит незаписываемый идентификатор
```

```
"do flush_card_all 1: PROG_IS_NON_AUTH\r\n"
                        // Ячейка программирования не требует авторизацию

"do flush_card_all 1: WRONG_IDENTIFICATOR_TYPE\r\n"
          // Неправильный тип идентификатора в ячейке программирования -
          //        поддерживаются Mifare Ultralight, SL1, SL3 и Desfire

"do flush_card_all 1: CAN_NOT_SELECT_CARD\r\n"
                // Карта не найдена - карта не приложена, либо неисправна

"do flush_card_all 1: DIFFERENT_CARD_TYPE\r\n"
// Найденная карта имеет неверный тип (например, ячейка SL1, а карта SL3)

"do flush_card_all 1: CAN_NOT_AUTHORIZE_CARD\r\n"
                                  // Не получилось авторизоваться в карту

"do flush_card_all 1: CAN_NOT_WRITE_BLOCK\r\n"
                                              // Ошибка записи блока карты
```

6.3.7 `flush_card_auto`

`flush_card_auto` - Команда автоматической полной очистки карты. Очищает все секторы карты, а также меняет ключи, биты доступа и пользовательский байт на стандартные, независимо от установок в ячейке программирования для записи идентификаторов. При этом будут использованы стандартные ключи, а так же все имеющиеся на считывателе ключи из всех ячеек программирования, тип идентификатора которых соответствует типу приложенной карты. Повреждённые секторы, а так же секторы с неизвестными ключами очищены не будут!

Если известны ключи шифрования, но в ячейках программирования они не указаны - можно добавить новую ячейку программирования и занести эти ключи туда, либо добавить ключи в ранее созданные ячейки программирования.

Пример использования команды:

```
"do flush_card_auto\n"                          // Полностью очистить карту
```

Примеры ответов считывателя:

```
"do flush_card_auto: OK\r\n"           // Карта полностью очищена успешно

"do flush_card_auto: WRONG_IDENTIFICATOR_TYPE\r\n"
          // Неправильный тип идентификатора в ячейке программирования -
          //        поддерживаются Mifare Ultralight, SL1, SL3 и Desfire

"do flush_card_auto: CAN_NOT_SELECT_CARD\r\n"
               // Карта не найдена - карта не приложена, либо неисправна

"do flush_card_auto: CAN_NOT_AUTHORIZE_CARD\r\n"
                                  // Не получилось авторизоваться в карту

"do flush_card_auto: CAN_NOT_WRITE_BLOCK\r\n"
                                             // Ошибка записи блока карты
```