Оглавление

Типы

Перечисления

Название	Описание
E_RG_ENDPOINT_TYPE	Возможные типы подключения, используемого для взаимодействия со считывателем (*целое, 8 бит, без знака*).
E_RG_DEVICE_TYPE	Список моделей считывателей (*целое, 8 бит, без знака*).
E_RG_CAPABILITIES	Набор флагов, позволяющих определить функциональные возможности конкретного считывателя (*целое, 32 бита, без знака*).
E_RG_STATUS_TYPE	Возможные статусы устройства (*целое, 8 бит, без знака*)
E_RG_CARD_FAMILY_CODE	Список семейств карт, с которыми может работать считыватель. Используется для установки маски считываемых типов карт (*целое, 8 бит, без знака).*
E_RG_CARD_TYPE_CODE	Список типов карт, с которыми может работать считыватель. Позволяет определить тип карты, которая в настоящий момент находится в поле считывателя (*целое, 8 бит, без знака).*
E_RG_CARD_AUTH_FLAGS	Набор флагов, определяющих механизм авторизации карт Mifare, работы с мобильным приложением или Apple/Google Pay. Используется при записи профилей в память считывателя (*целое, 8 бит, без знака).*
E_RG_CODOGRAMM_PRIORITY	Список вариантов выполнения кодограмм (световой/звуковой индикации) (*целое, 8 бит, без знака).*
E_RG_DEVICE_EVENT_TYPE	Набор флагов, определяющих типы возможных событий (приложена/убрана карта, изменилось состояние входа/выхода и т.д.) устройства (*целое, 32 бита, без знака*).

Перечисление E_RG_ENDPOINT_TYPE (целое, 8 бит, без знака) (список)

Перечисление используется для указания типа подключения, которое будет использоваться для обмена данными со считывателем. Применяется в структурах RG_ENDPOINT и RG_ENDPOINT_INFO. От типа подключения зависит формат написания адреса подключения.

Поле	Значение	Описание
ET_UNKNOWN	0 (00h)	Значения используется для указания неизвестного или неподдерживаемого типа подключения. Использование его вызовет ошибку.
ET_USBHID	1 (01h)	Обмен со считывателем будет производиться посредством USB HID (только для считывателей, который подключаются по USB). В качестве адреса подключения задается серийный номер устройства, например: *«01023FE6»*
ET_SERIAL	2 (02h)	Обмен со считывателем будет производиться посредством обмена через последовательный порт. В качестве адреса указывается имя порта, в зависимости от платформы: • Windows: “*COM1*”, “\\\\.*COM1*” • Linux: “*/dev/ttyS1*”, “*/dev/ttyACM0*”

Перечисление E_RG_DEVICE_TYPE (целое, 8 бит, без знака) (список)

Содержит список кодов для определения конкретной модели считывателя. Определить модель устройства можно сравнив соответствующее поле структуры RG_DEVICE_INFO_EXT или одним из значений перечисления.

Поле	Значение	Описание
DTE_RDR202	4 (04h)	Мульти форматный считыватель RDR-202
DTE_R10EHT	10 (0Ah)	Считыватель R10-EHT
DTE_R10MF	11 (0Bh)	Считыватель R10-MF
DTE_R15MULTI	15 (0Fh)	Считыватель R15 Multi
DTE_R5USBMULTI	16 (10h)	USB считыватель R5-USB Multi
DRTE_R5USBMULTIPROF	17 (11h)	USB считыватель R5-USB Multi Prof
DTE_RDR204EH	12 (0Ch)	Считыватель RDR204-EH Key
DTE_RDR204MF	13 (0Dh)	Считыватель RDR204-MF Key
DTE_RDR102	14 (0Eh)	Считыватель RDR-102

Перечисление E_RG_CAPABILITIES (целое, 32 бит, без знака) (список)

Содержит набор флагов для определения функциональных возможностей считывателя. Определить, поддерживается та или иная функция конкретным считывателем, можно выполнив «поразрядное И» одного из значений перечисления со значением соответствующего поля структуры RG_DEVICE_INFO_EXT.

Поле	Значение	Описание
CFE_SUPPORT_HID_EM	1 (01h)	Поддерживается работа с картами HID/EM-Marine
CFE_SUPPORT_TEMIC	2 (02h)	Поддерживается работа с картами Temic
CFE_SUPPORT_COTAG	4 (04h)	Поддерживается работа с картами COTAG
CFE_SUPPORT_MIFARE	8 (08h)	Поддерживается работа с картами Mifare
CFE_SUPPORT_INF	16 (10h)	
CFE_SUPPORT_NFC	32 (20h)	Поддерживается работа с мобильным приложением RusGuard
CFE_SUPPORT_NFC_PAY	64 (40h)	Поддерживается работа с Apple/Google Pay
CFE_SUPPORT_BLE	128 (80h)	Поддерживается работа с BLE (Bluetooth Low Energy)
CFE_SUPPORT_TM_W	256 (100h)	
CFE_SUPPORT_RBUS	512 (200h)	Возможность работы по RBUS
CFE_SUPPORT_RS485	1024 (400h)	Возможность подключения по RS-485
CFE_SUPPORT_USB	2048 (800h)	Возможность подключения по USB
CFE_HAS_MEMORY	4096 (1000h)	Имеет встроенную память
CFE_HAS_KEYBOARD	8192 (2000h)	Имеется встроенная кодонаборная панель
CFE_HAS_CLOCK	16384 (4000h)	Имеются встроенные часы реального времени
CFE_HAS_TERMO	32768 (8000h)	Имеется встроенный датчик температуры
CFE_HAS_RELAY	65536 (10000h)	Имеется встроенное реле для управления замком
CFE_READER	131072 (20000h)	Может работать в режиме считывателя
CFE_CONTROLLER	262144 (40000h)	Может работать в режиме автономного контроллера
CFE_SUPPORT_ODSP	524288 (80000h)	Поддерживает работу по протоколу OSDP

Перечисление E_RG_STATUS_TYPE (целое, 8 бит, без знака) (список)

Содержит список значений, позволяющих определить текущий статус устройства. Определить статус можно путем сравнения значения, которое возвращается через аргумент *pStatusType* функции RG_GetStatus, с одним из полей перечисления.

Поле	Значение	Описание
STE_UNKNOWN	0 (00h)	Неопределенный статус устройства
STE_NO_CARD	1 (01h)	В поле считывателя нет карты
STE_CARD	9 (09h)	В поле считывателя: • есть карты без памяти; • карта с памятью, но в памяти считывателя нет профилей.
STE_CARD_NO_AUTH	10 (0Ah)	В поле считывателя есть карта с памятью. В памяти считывателя есть профили авторизации, но не по одному из них авторизация не проходит.
STE_CARD_AUTH	26 (1Ah)	В поле считывателя есть карта с памятью. По одному из профилей успешно прошла авторизация.

Перечисление E_RG_CARD_FAMILY_CODE (целое, 8 бит, без знака) (список)

Содержит список флагов, определяющих семейства карт, которые будут считываться устройством. Для конфигурирования считывателя используется функция RG_SetCardsMask. Для объединения нескольких значений используется «поразрядное И».

Поле	Значение	Описание
CF_PIN	1 (01h)	PIN-код, ввод с клавиатуры. Необходима поддержка CFE_HAS_KEYBOARD.
CF_TEMIC	2 (02h)	Разрешает считывание карт TEMIC.
CF_HID	4 (04h)	Разрешает считывание карт HID
CF_EM	8 (08h)	Разрешает считывание карт EM-Marine
CF_INDALA	16 (10h)	Разрешает считывание карт Indala
CF_COTAG	32 (20h)	Разрешает считывание карт COTAG
CF_MIFARE	64 (40h)	Разрешает считывание карт Mifare
CF_INDALA_MT	128 (80h)	-
CF_ALL	255 (FFh)	Разрешает считывание всех поддерживаемых типов карт.

Перечисление E_RG_CARD_TYPE_CODE (целое, 32 бит, без знака) (список)

Содержит список значений, позволяющих определить тип карты в поле считывателя при запросе статуса устройства через RG_GetStatus. Информация о типе карты и идентификаторе возвращается через аргумент *pCardInfo*, который является указателем на структуру информации о карте RG_CARD_INFO.

Поле	Значение	Описание
CTC_PIN	0 (00h)	PIN-код набранный на клавиатуре. Необходима поддержка CFE_HAS_KEYBOARD.
CTC_TEMIC	1 (01h)	Карта Temic
CTC_HID	2 (02h)	Карта HID
CTC_EM	3 (03h)	Карта EM-Marine
CTC_INDALA	4 (04h)	Карта Indala
CTC_COTAG	5 (05h)	Карта Cotag
CTC_MF_DESFIRE	6 (06h)	Карта MIFARE DESFire EV1
CTC_MF_UL	7 (07h)	Карта MIFARE Ultralight
CTC_MF_MINI	8 (08h)	Карта MIFARE Mini

Поле	Значение	Описание
CTC_MF_CL1K_PL2K	9 (09h)	Карта MIFARE Classic 1K / MIFARE Plus EV1 2K SL1
CTC_MF_CL4K_PL4K	10 (0Ah)	Карта MIFARE Classic 4K / MIFARE Plus EV1 4K SL1
CTC_MF_PL2K_SL2	11 (0Bh)	Карта MIFARE Plus 2K SL2
CTC_MF_PL4K_SL2	12 (0Ch)	Карта MIFARE Plus 4K SL2
CTC_MF_SL3	13 (0Dh)	Карта MIFARE Plus SL3
CTC_SMX4K	14 (0Eh)	Карта SmartMX 4K
CTC_SMX1K	15 (0Fh)	Карта SmartMX 1K
CTC_PAY	253 (FDh)	Apple/Google Pay
CTC_MOBILE	254 (FEh)	Смартфон с приложением RusGuard

Перечисление E_RG_CARD_AUTH_FLAGS (целое, 8 бит, без знака) (список)

Содержит набор флагов, определяющих механизм авторизации карт Mifare, работы с мобильным приложением или Apple/Google Pay. Значения данного перечисления применяются при формировании профиля авторизации, который в дальнейшем записывается в память считывателя. Флаги делятся на 2 группы, первая определяет типы карт, вторая - параметры авторизации и разрешенные типы ключей.

Флаги, определяющие тип носителя ключа (мобильное приложение, Apple/Google Pay или карта Mifare):

Поле	Значение	Описание
CAF_USE_APP	32 (20h)	Позволяет считывателю работать с мобильным приложением RusGuard Key (необходим смартфон с поддержкой NFC). Взаимоисключающий с CAF_USE_PAY.
CAF_USE_PAY	64(40h)	Позволяет считывателю работать с Apple/Google Pay (необходим смартфон с поддержкой NFC). Взаимоисключающий с CAF_USE_APP.
-	0 (00h)	Если не установлен ни один и этих флагов, профиль будет ориентирован на работу с картами Mifare.

Флаги, отвечающие за параметры авторизации и разрешенные типы ключей:

Поле(я)	Значение	Описание
CAF_CLASSIC_KEY_B *CAF_GENERATED_KEY*	1 (01h)	В зависимости от типа носителя ключа: • Mifare Classic: авторизация по ключу «B» если флаг установлен, иначе – по ключу «A»; • Приложение, Apple/Google Pay: разрешает использование типа ключей «сгенерированный».
CAF_PLUS_KEY_B *CAF_PRESENT_KEY*	2 (02h)	В зависимости от типа носителя ключа: • Mifare Plus: авторизация по ключу «B» если флаг установлен, иначе – по ключу «A»; • Приложение, Apple/Google Pay: разрешает использование типа ключей «заданный».
CAF_PLUS_SL3 *CAF_EMITED_KEY*	4 (04h)	В зависимости от типа носителя ключа: • Mifare: использовать механизм авторизации карт SL3 если флаг установлен, иначе – механизм авторизации SL1/Classic; • Приложение, Apple/Google Pay: разрешает использование типа ключей «эмитированный».

Перечисление E_RG_CODOGRAMM_PRIORITY (целое, 8 бит, без знака) (список)

Содержит набор значений, определяющих приоритет (уровень) выполнения кодограммы, т. е. световой и звуковой индикации. Наибольшее значение соответствует наивысшему приоритету выполнения. Всего есть 5 приоритетов (уровней):

Поле(я)	Значение	Описание
CPE_BACKGROUND	0 (00h)	Кодограмма выполняется в фоновом режиме.
CPE_CYCLIC_LO	1 (01h)	Кодограмма выполняет циклически
CPE_CYCLIC_HI	2 (02h)	Кодограмма выполняет циклически с более высоким приоритетом
CPE_ONCE_LO	3 (03h)	Кодограмма выполняется разово
CPE_ONCE_HI	4 (04h)	Кодограмма выполняется разово с более высоким приоритетом

По завершении или отмене выполнения кодограммы на конкретном уровне, индикация переходит к менее приоритетному уровню, для которого назначена кодограмма.

Перечисление E_RG_DEVICE_EVENT_TYPE (целое, 8 бит, без знака) (список)

Содержит набор флагов, определяющих конкретный тип события считывателя (приложена карта, изменилось состояние входа и т. д.). Определить конкретный тип события, можно выполнив «поразрядное И» одного из значений данного перечисления со значением, возвращаемым через аргумент *pEventType* функции RG_PollEvents.

Поле(я)	Значение	Описание
DET_UNKNOWN_EVENT	0 (00h)	Неверный тип события.
DET_CARD_PLACED_EVENT	2 (02h)	Была приложена карта.
DET_CARD_REMOVED_EVENT	4 (04h)	Была убрана карта.
DET_RELAY_STATE_CHANGED	8 (08h)	Изменилось состояние релейного выхода.
DET_TAMPER_STATE_CHANGED	16 (10h)	Изменилось состояние тампера.
DET_BUTTON_STATE_CHANGED	32 (20h)	Изменилось состояние кнопки выхода.
DET_DOOR_SENSOR_STATE_CHANGED	64 (40h)	Изменилось состояние датчика двери.
DET_POLL_ERROR	128 (80h)	Событие, сообщающее об ошибке в потоке опроса.

Структуры

Название	Описание
RG_ENDPOINT	Структура конечной точки подключения. Содержит тип подключения и адрес подключения согласно типу.
RG_ENDPOINT_INFO	Структура информации о конечной точке. Помимо типа и адреса подключения, содержит строку с системным именем устройства.
RG_DEVICE_INFO_SHORT	Структура информации о считывателе.
RG_DEVICE_INFO_EXT	Структура расширенной информации о считывателе.
RG_CARD_INFO	Структура с информацией о карте, находящейся в поле считывателя.
RG_CARD_MEMORY	Структура с данными блока памяти карты Mifare.
RG_CARD_AUTH_PARAMS	Структура с параметрами авторизации и ключами для карт Mifare.
RG_CODOGRAMM	Структура с параметрами кодограммы (параметрами индикации).

Структура RG_ENDPOINT (список)

```
typedef struct _RG_ENDPOINT {
	uint8_t		type;
	const char*	address;
} RG_ENDPOINT;
```

Структура определяет параметры подключения, такие как тип и адрес. Заполняется пользователем и используется для адресации команд конкретным устройствам, находящимся на одном подключении (на одной физической линии).

Поле	Тип	Описание
type	*uint8_t*	Тип подключения. Задается одним из значений перечисления E_RG_ENDPOINT_TYPE. Значение ET_UNKNOWN является недопустимым и приведет к ошибке.
address	*const char**	Строка ASCII, содержащая адрес подключения согласно указанному типу подключения. Формат адреса подключения варьируется как от типа подключения, так и от платформы.

Как отмечалось в описании E_RG_ENDPOINT_TYPE, есть два допустимых типа подключения: ET_SERIAL и ET_USBHID. Строка адреса для типа ET_USBHID имеет единый формат вне зависимости от платформы (Windows/Linux) и содержит серийный номер устройства. Строка адреса для типа ET_SERIAL имеет различный формат на разных платформах:

- Windows: любое, допустимое для операционной системы, написание адреса последовательного порта, т.е. “COM1” или “\\\\.\\COM10”;
- Linux: системный путь определяемый типом последовательного порта. Например: “/dev/ttyS1”, “/dev/ttyACM0” или “/dev/ttyUSB0”;

Структура RG_ENDPOINT_INFO (список)

```
typedef struct _RG_ENDPOINT_INFO {
	uint8_t		type;
	char		address[64];
	char		friendly_name[128];
} RG_ENDPOINT_INFO;
```

Структура содержит информацию о конечной точке, т. е. данные, которые можно использовать в качестве параметров подключения. Возвращается пользователю в качестве результата выполнения функции RG_FindEndPoints.

Поле	Тип	Описание
type	*uint8_t*	Тип подключения. Содержит одно из значений перечисления E_RG_ENDPOINT_TYPE. Значение ET_UNKNOWN является допустимым и говорит о том, что данные других полей в структуре значения не имеют.
address	*char[64]*	Строка из 64 ASCII символов (включая завершающий 0) содержащая адрес подключения согласно типу подключения. Значение поля *type* должно отличаться от ET_UNKNOWN.
friendly_name	*char[128]*	Строка из 128 символов (включая завершающий 0) содержащая имя устройства, предоставляемое целевой системой, например – как в диспетчере устройств Windows. Значение поля *type* должно отличаться от ET_UNKNOWN.

Структура RG_DEVICE_INFO_SHORT (список)

```
typedef struct _RG_DEVICE_INFO_SHORT {
	uint8_t		address;
	uint8_t		type;
	uint8_t		firmware;
	uint8_t		codogramms;
} RG_DEVICE_INFO_SHORT, *PRG_DEVICE_INFO_SHORT;
```

Структура содержит базовую информацию о конкретном считывателе. Возвращает пользователю в качестве результата вызова функции RG_GetInfo.

Поле	Тип	Описание
address	*uint8_t*	Адрес считывателя. Может принимать значения от 0 до 3 включительно.
type	*uint8_t*	Тип (модель) считывателя, согласно значениям перечисления E_RG_DEVICE_TYPE.
firmware	*uint8_t*	Версия прошивки считывателя.
codogramms	*uint8_t*	Количество кодограмм в памяти считывателя.

Структура RG_DEVICE_INFO_EXT (список)

```
typedef struct _RG_DEVICE_INFO_EXT {
	uint8_t		address;
	uint32_t	serial;
	uint8_t		firmwareUpdateLock;
	uint8_t		type;
	uint16_t	firmware;
	uint32_t	capabilities;
} RG_DEVICE_INFO_EXT, *PRG_DEVICE_INFO_EXT;
```

Структура содержит расширенную информацию о конкретном считывателе. Возвращает пользователю в качестве результата вызова функции RG_GetInfoExt.

Поле	Тип	Описание
address	*uint8_t*	Адрес считывателя. Может принимать значения от 0 до 3 включительно.
serial	*uint32_t*	Серийный номер считывателя.
firmwareUpdateLock	*uint8_t*	Флаг для проверки на запрет прошивки.
type	*uint8_t*	Тип (модель) считывателя, согласно значениям перечисления E_RG_DEVICE_TYPE.
firmware	*uint16_t*	Версия прошивки считывателя.
capabilities	*uint32_t*	Флаги функциональных возможностей устройства, согласно значениям перечисления E_RG_CAPABILITIES.

Структура RG_CARD_INFO (список)

```
typedef struct _RG_CARD_INFO {
	uint8_t		type;
	uint8_t		uid[7];
} RG_CARD_INFO, *PRG_CARD_INFO;
```

Структура содержит информацию о карте в поле считывателя, её типе и идентификаторе (UID).

Поле	Тип	Описание
type	*uint8_t*	Тип карты, согласно значениям перечисления E_RG_CARD_TYPE_CODE.
uid	*uint8_t[7]*	Идентификатор (UID) карты, 7 байт в Little Endian. Количество значащих байт зависит от типа карты.

Структура RG_CARD_MEMORY (список)

```
typedef struct _RG_CARD_MEMORY {
	uint8_t		profile_block;
	uint8_t		block_data[16];
} RG_CARD_MEMORY, *PRG_CARD_MEMORY;
```

Структура с данными конкретного блока памяти карты Mifare. Используется как для чтения данных из карты, так и для записи данных на карту.

Поле	Тип	Описание
profile_block	*uint8_t*	Номер блока карты или номер профиля, в зависимости от сценария использования.
block_data	*uint8_t[16]*	16 байт данных блока памяти карты Mifare Plus.

Структура RG_CARD_AUTH_PARAMS (список)

```
typedef struct _RG_CARD_AUTH_PARAMS {
	uint8_t		flags;
	uint8_t		classicKey[6];
	uint8_t		plusKey[16];
} RG_CARD_AUTH_PARAMS, *PRG_CARD_AUTH_PARAMS;
```

Структура параметров авторизации карт Mifare. Используется при формировании профилей, чтении и записи данных на карту.

Поле	Тип	Описание
flags	*uint8_t*	Флаги авторизации. Формируется из значений перечисления E_RG_CARD_AUTH_FLAGS.
classicKey	*uint8_t[6]*	6 байт ключа авторизации для карт Mifare Classic.
plusKey	*uint8_t[16]*	16 байт ключа авторизации для карт Mifare Plus.

Структура RG_CODOGRAMM (список)

```
typedef struct _RG_CODOGRAMM {
	uint8_t		length;
	uint32_t	body;
} RG_CODOGRAMM, *PRG_CODOGRAMM;
```

Структура параметров кодограммы, содержит данные о длине (продолжительности) и схеме индикации.

Поле	Тип	Описание
length	*uint8_t*	Длина кодограммы в битах. Может принимать значения от 0 до 32 включительно. 1 бит = 100мс индикации.
body	*uint32_t*	Тело кодограммы. Представляет собой набор из “*length*” бит, определяющих уровень (вкл./выкл.) канала индикации в конкретный промежуток времени: 1 - высокий уровень (вкл.), 0 – низки уровень (выкл.).

Длина кодограммы задается в битах, от 0 до 32 включительно. 1 бит равняется 100мс времени индикации, следовательно максимальное время индикации одной кодограммы равняется 3,2 сек. Тело кодограммы считывается от младшего бита к старшему.

Допустим нам требуется кодограмма (схема индикации) со следующими параметрами: пауза 0.5 сек, индикация 0.5 сек. В этом случае длина кодограммы будет 10 (помним, что один бит – это 100 мс), а тело кодограммы будет 992 (1111100000b или 3E0h):

Биты										
31…10	9									0
0	1	1	1	1	1	0	0	0	0	0
-	Индикация: 100мс * 5 = 500мс = 0.5сек					Пауза: 100мс * 5 = 500мс = 0.5сек				

Константы

Коды ошибок

Имя	Значение	Описание
EC_OK	0	Успешное выполнение, нет ошибок
EC_FAIL	1	Ошибка
EC_NOT_IMPLEMENTED	2	Функция не реализована
EC_BAD_ARGUMENT	3	Неверно задан один из аргументов
EC_INVALID_HANDLE	4	Используется неправильный дескриптор ресурса или ресурс был закрыт
EC_INVALID_RESOURCE	5	Используется дескриптор несовместимого с операцией ресурса
EC_INVALID_CONNECTION_TYPE	6	Указан неподдерживаемый тип подключения
EC_INVALID_CONNECTION_ADDRESS	7	Адрес подключения задан неверно или не соответствует типу подключения
EC_INVALID_DEVICE_ADDRESS	8	Неверно задан Адрес считывателя
EC_DEVICE_OPERATION_UNSUPPORTED	9	Функция не поддерживается считывателем
EC_DEVICE_NOT_CONNECTED	10	Считыватель был отключен (USB)
EC_DEVICE_NO_RESPOND	11	Считыватель не отвечает на команды
EC_DEVICE_COMM_FAILURE	12	Произошла ошибка в процессе обмена данными со считывателем
EC_DEVICE_PROTOCOL_FAILURE	13	Произошла ошибка при обработке сообщений протокола
EC_POLL_NO_EVENTS	14	Очередь событий считывателя пуста
EC_POLL_QUEUE_CLOSED	15	Очередь событий считывателя закрыта
EC_CALL_INIT	16	Требуется вызов функции инициализации
EC_DEVICE_INVALID_COMMAND	17	Считыватель не смог обработать команду
EC_DEVICE_INVALID_PARAM	18	Указан недопустимый параметр команды
EC_DEVICE_INVALID_PIN	19	Неверный PIN-код (для команд настройки)
EC_DEVICE_COMMAND_TIMEOUT	20	Истекло время ожидания на получение считывателем следующей команды (для команд настройки)
EC_DEVICE_NO_CARD	21	Нет карты на считывателе
EC_DEVICE_UNKNOWN_CARD	22	Считыватель не смог распознать тип карты
EC_DEVICE_INCOMPATIBLE_CARD	23	Тип карты не подходит для выполнения запрошенной операции
EC_DEVICE_AUTH_FAIL	24	Ошибка авторизации карты Mifare
EC_DEVICE_PROFILE_FAIL	25	Используется неподходящий профиль
EC_DEVICE_RW_FAIL	26	Ошибка авторизации карты Mifare для чтения/записи
EC_IO_OPEN_FAIL	27	Ошибка при открытии соединения
EC_IO_CLOSE_FAIL	28	Ошибка при закрытии соединения
EC_IO_READ_FAIL	29	Ошибка при выполнении операции чтения данных
EC_IO_WRITE_FAIL	30	Ошибка при выполнении операции записи данных
EC_IO_CLOSED	31	Операция прервана, соединение было закрыто

Функции

Общие функции и инициализация

Название	Описание
RG_GetVersion	Возвращает версию библиотеки.
RG_InitializeLib	Выполняет инициализацию внутренних механизмов библиотеки.
RG_Uninitialize	Завершает работу библиотеки.
RG_CloseResource	Уничтожает ресурс и освобождает выделенную память.

Функция RG_GetVersion (список)

```
API_METHOD uint32_t CALL_CONV RG_GetVersion();
```

Описание:
Возвращает версию библиотеки.

Возвращаемое значение:
32 битное целое число, где два старших байта хранят основной номер версии, а два младших – дополнительный.

Функция RG_InitializeLib (список)

```
API_METHOD api_error CALL_CONV RG_InitializeLib();
```

Описание:
Выполняет инициализацию и настройку внутренних механизмов библиотеки.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_Uninitialize (список)

```
API_METHOD api_error CALL_CONV RG_Uninitialize ();
```

Описание:
Завершает работу библиотеки.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_CloseResource (список)

```
API_METHOD api_error CALL_CONV RG_CloseResource(void* handle);
```

Описание:
Уничтожает ресурс и освобождает выделенную память.

Аргументы:

Имя	Тип	Описание
handle	*void**	Дескриптор ресурса, полученный ранее вызовом функций: • RG_FindEndPoints • RG_FindDevices • RG_Subscribe

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Поиск устройств и возможных точек подключения

Название	Описание
RG_FindEndPoints	Выполняет поиск доступных точек подключения и возвращает их список
RG_GetFoundEndPointInfo	Извлекает информацию о точке подключения из списка, сформированного функцией RG_FindEndPoints
RG_FindDevices	Выполняет поиск считывателей и возвращает список найденных
RG_GetFoundDeviceInfo	Извлекает информацию о считывателе из списка, сформированного функцией RG_FindDevices

Функция RG_FindEndPoints (список)

```
API_METHOD api_error CALL_CONV RG_FindEndPoints(
	void** pEndPointListHandle,
	uint8_t enpointTypeMask,
	uint32_t* pCount);
```

Описание:
Выполняет поиск доступных точек подключения и возвращает их список и количество.

Аргументы:

Имя	Тип	Описание
pEndPointListHandle	*void***	Указатель на void* (на значение указателя), по которому будет сохранен дескриптор списка найденных точек подключения.
enpointTypeMask	*uint8_t*	Маска типов подключения. На основе значений E_RG_ENDPOINT_TYPE.
pCount	*uint32_t**	Указатель на переменную, в которою будет записано количество найденных точек подключения.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_GetFoundEndPointInfo (список)

```
API_METHOD api_error CALL_CONV RG_GetFoundEndPointInfo(
	void* endPointListHandle,
	uint32_t listIndex,
	PRG_ENDPOINT_INFO pEndpointInfo);
```

Описание:
Извлекает из списка результатов, полученного с помощью вызова функции RG_FindEndPoints, информацию о точке подключения по индексу.

Аргументы:

Имя	Тип	Описание
endPointListHandle	*void**	Дескриптор списка найденных точек подключения, полученный через вызов функции RG_FindEndPoints.
listIndex	*uint32_t*	Индекс конкретного элемента в списке. От *0* до *pCount - 1*
pEndpointInfo	*PRG_ENDPOINT_INFO*	Указатель на структуру информации о конечной точке RG_ENDPOINT_INFO, в которую будут скопированы данные элемента *listIndex* списка *endPointListHandle*.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_FindDevices (список)

```
API_METHOD api_error CALL_CONV RG_FindDevices(
	void** pDevicesListHandle,
	uint8_t endpointTypeMask,
	uint32_t* pCount);
```

Описание:
Выполняет поиск подключенных считывателей.

Аргументы:

Имя	Тип	Описание
pDevicesListHandle	*void***	Указатель на void* (на значение указателя), по которому будет сохранен дескриптор списка найденных устройств.
enpointTypeMask	*uint8_t*	Маска типов подключения. На основе значений E_RG_ENDPOINT_TYPE.
pCount	*uint32_t**	Указатель на переменную, в которую будет записано количество найденных считывателей.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_GetFoundDeviceInfo (список)

```
API_METHOD api_error CALL_CONV RG_GetFoundDeviceInfo(
	void* deviceListHandle,
	uint32_t ListIndex,
	PRG_ENDPOINT_INFO pEndpointInfo,
	PRG_DEVICE_INFO_EXT pDeviceInfoExt);
```

Описание:
Извлекает из списка результатов, полученного с помощью вызова функции RG_FindDevices, информацию о подключенном считывателе и его точке подключения.

Аргументы:

Имя	Тип	Описание
pDevicesListHandle	*void**	Дескриптор списка считывателей, полученный через вызов функции RG_FindDevices.
ListIndex	*uint32_t*	Индекс конкретного элемента в списке. От *0* до *pCount - 1*
pEndpointInfo	*PRG_ENDPOINT_INFO*	Указатель на структуру информации о конечной точке RG_ENDPOINT_INFO, в которую будут скопированы данные элемента *ListIndex* списка *pDevicesListHandle*.
pDeviceInfoExt	*PRG_DEVICE_INFO_EXT*	Указатель на структуру расширенной информации о считывателе RG_DEVICE_INFO_EXT, в которую будут скопированы данные элемента *ListIndex* списка *pDevicesListHandle*.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Инициализация считывателей, конфигурирование

Название	Описание
RG_InitDevice	Выполняет инициализацию считывателя
RG_CloseDevice	Выполняет деинициализацию считывателя, очистку ресурсов
RG_GetInfo	Возвращает краткую информацию о считывателе
RG_GetInfoExt	Возвращает расширенную информацию о считывателе
RG_SetCardsMask	Задает маску типов карт, с которыми будет работать считыватель.
RG_ClearProfiles	Выполняет стирание всех профилей Mifare из памяти считывателя.
RG_WriteProfile	Записывает профиль Mifare в память считывателя.
RG_WriteCodogramm	Записывает кодограмму в память считывателя.
RG_StartCodogramm	Запускает выполнение кодограммы.

Функция RG_InitDevice (список)

```
API_METHOD api_error CALL_CONV RG_InitDevice(
       PRG_ENDPOINT pEndPoint,
       uint8_t address);
```

Описание:
Выполняет инициализацию считывателя. Вызов данной функции является обязательным перед началом работы с считывателем.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_CloseDevice (список)

```
API_METHOD api_error CALL_CONV RG_CloseDevice(
	PRG_ENDPOINT pEndPoint,
	uint8_t address);
```

Описание:
Выполняет деинициализацию, завершает работу со считывателем, освобождает ресурсы. Для дальнейшего взаимодействия со считывателем необходимо повторно вызвать RG_InitDevice.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_GetInfo (список)

```
API_METHOD api_error CALL_CONV RG_GetInfo(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	PRG_DEVICE_INFO_SHORT pDeviceInfo);
```

Описание:
Запрашивает и возвращает краткую информацию о считывателе.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
pDeviceInfo	*PRG_DEVICE_INFO_SHORT*	Указатель на инициализированную структуру краткой информации о считывателе RG_DEVICE_INFO_SHORT.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_GetInfoExt (список)

```
API_METHOD api_error CALL_CONV RG_GetInfoExt(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	PRG_DEVICE_INFO_EXT pDeviceInfo);
```

Описание:
Запрашивает и возвращает расширенную информацию о считывателе.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
pDeviceInfo	*PRG_DEVICE_INFO_EXT*	Указатель на инициализированную структуру расширенной информации о считывателе RG_DEVICE_INFO_EXT.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_SetCardMask (список)

```
API_METHOD api_error CALL_CONV RG_SetCardMask(
       PRG_ENDPOINT pEndPoint,
       uint8_t address,
       uint8_t mCardsFamilyMask);
```

Описание:
Задает маску типов карт, с которыми будет работать считыватель.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
mCardsFamilyMask	*uint8_t*	Маска типов карт на основе одного или комбинации нескольких значений перечисления E_RG_CARD_FAMILY_CODE.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_ClearProfiles (список)

```
API_METHOD api_error CALL_CONV RG_ClearProfiles(
       PRG_ENDPOINT pEndPoint,
       uint8_t address);
```

Описание:
Удаляет все профили из памяти считывателя.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_WriteProfile (список)

```
API_METHOD api_error CALL_CONV RG_Writeprofile(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	uint8_t profileNum,
	uint8_t blockNum,
	PRG_CARD_AUTH_PARAMS pCardAuthParams);
```

Описание:
Записывает профиль в память считывателя.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
profileNum	*uint8_t*	Порядковый номер профиля, под которым он будет записан в память считывателя. От 0 до 4 включительно.
blockNum	*uint8_t*	Номер блока памяти карты Mifare из которого будут прочитан данные при удачной авторизации.
pCardAuthParams	*PRG_CARD_AUTH_PARAMS*	Указатель инициализированную на структуру параметров авторизации RG_CARD_AUTH_PARAMS.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_WriteCodogramm (список)

```
API_METHOD api_error CALL_CONV RG_WriteCodogramm(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	uint8_t codogrammNum,
	PRG_CODOGRAMM pCodogramm);
```

Описание:
Удаляет все профили из памяти считывателя.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
codogrammNum	*uint8_t*	Порядковый номер кодограммы, под которым он будет записан в память считывателя. От 1 до 20 включительно.
pCodogramm	*PRG_CODOGRAMM*	Указатель на инициализированную структуру параметров кодограммы RG_CODOGRAMM.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_StartCodogramm (список)

```
API_METHOD api_error CALL_CONV RG_StartCodogramm(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	uint8_t priority,
	uint8_t sChannelNum,
	uint8_t rChannelNum,
	uint8_t gChannelNum, uint8_t bChannelNum);
```

Описание:
Запускает выполнение кодограмм на каналах индикации.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
priority	*uint8_t*	Приоритет (уровень), на котором будет запущена кодограмма. Согласно значениям перечисления E_RG_CODOGRAMM_PRIORITY.
sChannelNum	*uint8_t*	Номер кодограммы, которая будет запущена на указанном уровне priority для звукового канала индикации.
rChannelNum	*uint8_t*	Номер кодограммы, которая будет запущена на указанном уровне priority для канала индикации «красный».
gChannelNum	*uint8_t*	Номер кодограммы, которая будет запущена на указанном уровне priority для канала индикации «зеленый».
bChannelNum	*uint8_t*	Номер кодограммы, которая будет запущена на указанном уровне priority для канала индикации «синий».

Примечание:
Значение «0» в качестве номера кодограммы для какого-либо канала индикации отключает индикацию на данном канале для указанного уровня. При этом индикация возвращается к менее приоритетному уровню, для которого назначена кодограмма. Допустим, что для всех каналов назначена фоновая индикация по схеме (кодограмме) под номером «1». Для этого мы вызвали функцию RG_StartCodogramm, указав «1» в качестве номера кодограммы для всех каналов и CPE_BACKGROUND в качестве значения аргумента priority. Индикация начинается по следующей схеме:

Приоритет	Номер кодограммы			
	Звуковой канал	Канал «красный»	Канал «зеленый»	Канал «синий»
CPE_BACKGROUND	1	1	1	1
CPE_CYCLIC_LO	-	-	-	-
CPE_CYCLIC_HI	-	-	-	-
CPE_ONCE_LO	-	-	-	-
CPE_ONCE_HI	-	-	-	-

Теперь мы хотим разово мигнуть всеми светодиодами и издать короткий звуковой сигнал, допустим, что за это отвечает кодограмма под номером «2» длительностью в 1 секунду. Вызываем функцию RG_StartCodogramm , указав «1» в качестве номера кодограммы для всех каналов и «CPE_ONCE_LO» в качестве значения аргумента priority. Индикация начинается по следующей схеме:

Приоритет	Номер кодограммы			
	Звуковой канал	Канал «красный»	Канал «зеленый»	Канал «синий»
CPE_BACKGROUND	1	1	1	1
CPE_CYCLIC_LO	-	-	-	-

CPE_CYCLIC_HI	-	-	-	-
CPE_ONCE_LO	2	2	2	2
CPE_ONCE_HI	-	-	-	-

По окончании индикации кодограммы под номером «2» (т.е. через 1 секунду), индикация по всем каналам вернется к уровню CPE_BACKGROUND, т.к. кодограммы для уровней CPE_CYCLIC_HI и CPE_CYCLIC_LO не назначены.

Приоритет	**Номер кодограммы**			
	Звуковой канал	**Канал «красный»**	**Канал «зеленый»**	**Канал «синий»**
CPE_BACKGROUND	1	1	1	1
CPE_CYCLIC_LO	-	-	-	-
CPE_CYCLIC_HI	-	-	-	-
CPE_ONCE_LO	-	-	-	-
CPE_ONCE_HI	-	-	-	-

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Получение статуса, работа с картами и входами, управление реле

Название	**Описание**
RG_GetStatus	Возвращает текущий статус считывателя. Данные о типе карты в поле, её UID, данные блока памяти и состояния входов/выходов.
RG_WriteBlock	Выполняет запись данных в блок памяти карты Mifare. Авторизация карты производится на основе профиля в памяти считывателя, записанного функцией RG_WriteProfile
RG_WriteBlockDirect	Выполняет запись данных в блок памяти карты Mifare. Авторизация карты производится на основе параметров RG_CARD_AUTH_PARAMS
RG_ReadBlockDirect	Выполняет чтение данных из блока памяти карты Mifare. Авторизация карты производится на основе параметров RG_CARD_AUTH_PARAMS
RG_SetControlOutputState	Задает состояние управляющего выхода

Функция RG_GetStatus (список)

```
API_METHOD api_error CALL_CONV RG_GetStatus(
       PRG_ENDPOINT pEndPoint,
       uint8_t address,
       uint8_t* pStatusType,
       uint8_t* pPinStates,
       PRG_CARD_INFO pCardInfo,
       PRG_CARD_MEMORY pMemory);
```

Описание:
Удаляет все профили из памяти считывателя.

Аргументы:

Имя	**Тип**	**Описание**
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.

Имя	Тип	Описание
pStatusType	*uint8_t**	Указатель на переменную, в которой будет сохранено значение статуса согласно E_RG_STATUS_TYPE.
pPinStates	*uint8_t**	Указатель на переменную, в которой будет сохранено значение, отражающее состояние входов/выходов, тампера и реле.
pCardInfo	*PRG_CARD_INFO*	Указатель на инициализированную структуру информации о карте RG_CARD_INFO.
pMemory	*PRG_CARD_MEMORY*	Указатель на инициализированную структуру RG_CARD_MEMORY, в которую будут записаны номер профиля/блока памяти и данные блока памяти карты (при успешной авторизации по профилю).

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_WriteBlock (список)

```
API_METHOD api_error CALL_CONV RG_WriteBlock(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	PRG_CARD_MEMORY pMemory);
```

Описание:
Записывает данные в блок памяти карты Mifare по указанному номеру профиля, ранее сохраненному в память считывателя.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
pMemory	*PRG_CARD_MEMORY*	Указатель на инициализированную структуру RG_CARD_MEMORY.

Примечание:
Поле profile_block структуры RG_CARD_MEMORY должно содержать **номер профиля**, по которому необходимо проводить авторизацию карты, а поле block_data должно содержать данные, которые необходимо записать в блок, определяемый профилем. Профиль должен быть предварительно записан в память считывателя.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_WriteBlockDirect (список)

```
API_METHOD api_error CALL_CONV RG_WriteBlockDirect(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	PRG_CARD_MEMORY pMemory,
	PRG_CARD_AUTH_PARAMS pCardAuthParams);
```

Описание:
Записывает данные в блок памяти карты Mifare используя указанные параметры авторизации.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
pMemory	*PRG_CARD_MEMORY*	Указатель на инициализированную структуру RG_CARD_MEMORY.
pCardAuthParams	*PRG_CARD_AUTH_PARAMS*	Указатель на инициализированную структуру RG_CARD_AUTH_PARAMS.

Примечание:
Поле profile_block структуры RG_CARD_MEMORY должно содержать **номер блока** карты, в который необходимо записать данные, а поле block_data должно содержать данные, которые необходимо записать в блок. Параметры авторизации карты определяются структурой RG_CARD_AUTH_PARAMS, указатель на экземпляр которой передаётся в функцию через аргумент pCardAuthParams.
Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_ReadBlockDirect (список)

```
API_METHOD api_error CALL_CONV RG_ReadBlockDirect(
        PRG_ENDPOINT pEndPoint,
        uint8_t address,
        PRG_CARD_MEMORY pMemory,
        PRG_CARD_AUTH_PARAMS pCardAuthParams);
```

Описание:
Считывает данные из указанного блок памяти карты Mifare используя указанные параметры авторизации.

Аргументы:

Имя	Тип	Описание
pEndPoint	*RG_ENDPOINT**	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
pMemory	*RG_CARD_MEMORY**	Указатель на инициализированную структуру RG_CARD_MEMORY.
pCardAuthParams	*RG_CARD_AUTH_PARAMS**	Указатель на инициализированную структуру RG_CARD_AUTH_PARAMS.

Примечание:
Поле profile_block структуры RG_CARD_MEMORY должно содержать **номер блока** памяти карты, из которого необходимо прочитать данные. При успешной авторизации поле block_data будет содержать данные, считанные из указанного блока памяти карты. Параметры авторизации карты определяются структурой RG_CARD_AUTH_PARAMS, указатель на экземпляр которой передаётся в функцию через аргумент pCardAuthParams.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_SetControlOutputState (список)

```
API_METHOD api_error CALL_CONV RG_SetControlOutputState(
        PRG_ENDPOINT pEndPoint,
        uint8_t address,
        uint8_t outputNum,
        bool state,
        uint8_t timeSec);
```

Описание:
Задает состояние управляющего выхода/порта по его номеру.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
outputNum	*uint8_t*	Номер управляющего выхода/порта.
state	*bool*	Новое состояние.
timeSec	*uint8_t*	Время переключения в секундах, от 0 до 255 включительно (0 - постоянно).

Примечание:
Если значение аргумента timeSec задано отличным от 0, то по истечении указанного времени выход переключится в предыдущее состояние.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Команды ISO, работа с картами семейства Mifare

Название	Описание
RG_ResetField	Сбрасывает поле на время, достаточное для переключения карты из одного режима в другой (например, при переключении из SL1 в SL3) и других операций.
RG_Iso_Ras	Выполняет запрос RAS (Request + Anticollision + Select).
RG_Iso_Rats	Выполняет запрос RATS (Request for Answer To Select). Переводит карту в режим обмена по ISO 14443-4.
RG_Iso_Exchange	Выполняет произвольную передачу данных по протоколу ISO 14443-4.
RG_Iso_Auth	Выполняет авторизацию в сектор карты Mifare Plus.
RG_MF_AuthorizeClassic	Выполняет авторизацию в сектор карты Mifare Classic.
RG_MF_ReadBlock	Выполняет чтение блока памяти карты Mifare.
RG_MF_WriteBlock	Выполняет запись блока памяти карты Mifare.

Функция RG_ResetField (список)

```
API_METHOD api_error CALL_CONV RG_ResetField(
	PRG_ENDPOINT pEndPoint,
	uint8_t address);
```

Описание:
Сбрасывает поле на время, достаточное для переключения карты из одного режима в другой (например, при переключении из SL1 в SL3) и других операций.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_Iso_Ras (список)

```
API_METHOD api_error CALL_CONV RG_Iso_Ras(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	uint16_t* pSak,
	uint16_t* pAtqa,
	void* pUidBuf,
	int32_t uidBufSize);
```

Описание:
Выполняет запрос RATS (Request for Answer To Select). Переводит карту в режим обмена по ISO 14443-4.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
pSak	*uint16_t**	Указатель на переменную, в которую будет сохранен код SAK (Select Acknowledge).
pAtqa	*uint16_t**	Указатель на переменную, в которую будет сохранен код ATQA (Answer To Request).
pUidBuf	*void**	Указатель на блок памяти, в которую будет сохранен UID карты.
uidBufSize	*Int32_t*	Размер блока памяти по указателю pUidBuf.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_Iso_Rats (список)

```
API_METHOD api_error CALL_CONV RG_Iso_Rats(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	void* pAtsBuf,
	int32_t atsBufSize,
	int32_t* pAtsSize);
```

Описание:
Выполняет запрос RATS (Request for Answer To Select). Переводит карту в режим обмена по ISO 14443-4.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
pAtsBuf	*void**	Указатель на блок памяти, в которую будет сохранен ответ ATS (Answer To Select).
atsBufSize	*Int32_t*	Размер блока памяти по указателю pAtsBuf.
pAtsSize	*Int32_t**	Указатель на переменную, в которую будет сохранен размер ответа ATS.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_Iso_Exchange (список)

```
API_METHOD api_error CALL_CONV RG_Iso_Exchange(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	const void* pDataBuf,
	int32_t dataBufSize,
	void* pRespBuf,
	int32_t respBufSize,
	int32_t* pRespSize);
```

Описание:
Выполняет произвольную передачу данных по протоколу ISO 14443-4.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
pDataBuf	*const void**	Указатель на блок памяти с данными запроса.
dataBufSize	*Int32_t*	Размер данных в блоке памяти по указателю pDataBuf.
pRespBuf	*void**	Указатель на блок памяти, в которую будет сохранен ответ.
respBufSize	*int32_t*	Размер блока памяти по указателю pRespBuf.
pRespSize	*Int32_t**	Указатель на переменную, в которую будет сохранен размер данных ответа.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_Iso_Auth (список)

```
API_METHOD api_error CALL_CONV RG_Iso_Auth(
       PRG_ENDPOINT pEndPoint,
       uint8_t address,
       uint16_t blockNum,
       uint8_t* pKeyBuf,
       int32_t keyBufSize);
```

Описание:
Выполняет авторизацию сектора карты Mifare Plus.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
blockNum	*uint16_t*	Номер блока, соответствующий сектору, в который производится авторизация.
pKeyBuf	*uint8_t**	Указатель на блок памяти, в которой хранится ключ авторизации.
keyBufSize	*Int32_t*	Размер данных ключа в блоке памяти по указателю pKeyBuf.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_MF_AuthorizeClassic (список)

```
API_METHOD api_error CALL_CONV RG_MF_AuthorizeClassic(
       PRG_ENDPOINT pEndPoint,
       uint8_t address,
       uint8_t blockNum,
       uint8_t keyType,
       uint8_t* pKeyBuf,
       int32_t keyBufSize);
```

Описание:
Выполняет авторизацию сектора карты Mifare .

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
blockNum	*uint8_t*	Номер блока, соответствующий сектору, в который производится авторизация.
keyType	*uint8_t*	Тип ключа (A/B).
pKeyBuf	*uint8_t**	Указатель на блок памяти, в которой хранится ключ авторизации.

Имя	Тип	Описание
keyBufSize	*int32_t*	Размер данных ключа в блоке памяти по указателю pKeyBuf.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_MF_ReadBlock (список)

```
API_METHOD api_error CALL_CONV RG_MF_ReadBlock(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	uint8_t blockNum,
	uint8_t flags,
	uint8_t* pDataBuf,
	uint32_t dataBufSize);
```

Описание:
Выполняет чтение данных из блока памяти карты Mifare Classic/Plus. Требуется предварительная авторизация в сектор вызовом RG_Iso_Auth или RG_MF_AuthorizeClassic (в зависимости от типа карты).

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
blockNum	*uint8_t*	Номер блока, в который требуется записать данные.
flags	*uint8_t*	Флаги, определяющие тип карты Mifare (Classic/Plus).
pDataBuf	*uint8_t**	Указатель на блок памяти, содержащий данные, которые требуется записать на карту.
dataBufSize	*int32_t*	Размер данных по указателю pDataBuf.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_MF_WriteBlock (список)

```
API_METHOD api_error CALL_CONV RG_MF_WriteBlock(
	PRG_ENDPOINT pEndPoint,
	uint8_t address,
	uint8_t blockNum,
	uint8_t flags,
	uint8_t* pDataBuf,
	uint32_t dataBufSize);
```

Описание:
Выполняет запись данных в блок памяти карты Mifare Classic/Plus. Требуется предварительная авторизация в сектор вызовом RG_Iso_Auth или RG_MF_AuthorizeClassic (в зависимости от типа карты).

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.

Имя	Тип	Описание
address	*uint8_t*	Адрес считывателя.
blockNum	*uint8_t*	Номер блока, в который требуется записать данные.
flags	*uint8_t*	Флаги, определяющие тип карты Mifare (Classic/Plus).
pDataBuf	*uint8_t**	Указатель на блок памяти, содержащий данные, которые требуется записать на карту.
dataBufSize	*int32_t*	Размер данных по указателю pDataBuf.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Подписка на события, управление опросом

Название	Описание
RG_Subscribe	Запускает циклический фоновый опрос считывателя и выполняет подписку на события, согласно маске типов событий E_RG_DEVICE_EVENT_TYPE. Возвращает дескриптор очереди событий для текущего считывателя.
RG_PollEvents	Извлекает доступные события из очереди событий, полученной вызовом функции RG_Subscribe
RG_IsolateDevice	Приостанавливает фоновый опрос устройства для выполнения иных операций.
RG_ResetIsolation	Восстанавливает фоновый опрос устройства, остановленный ранее вызовом RG_IsolateDevice

Функция RG_Subscribe (список)

```
API_METHOD api_error CALL_CONV RG_Subscribe(
        void** pDeviceEventsQueue,
        PRG_ENDPOINT pEndPoint,
        uint8_t address,
        uint32_t eventsMask,
        void* systemEventHandle);
```

Описание:
Выполняет подписку на указанные события и запускает фоновый опрос считывателя.

Аргументы:

Имя	Тип	Описание
pDeviceEventsQueue	*void***	Указатель на дескриптор очереди событий.
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.
eventMask	*uint32_t*	Маска типов событий, на которые следует подписаться.
systemEventHandle	*void**	(WINDOWS) Дескриптор системного объекта-события.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_PollEvents (список)

```
API_METHOD api_error CALL_CONV RG_PollEvents(
	void* deviceEventsQueue,
	uint32_t* pEventType,
	void** pEventMem,
	uint32_t pollWaitTimeoutMs);
```

Описание:
Извлекает доступные события из очереди событий, полученной вызовом функции RG_Subscribe.

Аргументы:

Имя	Тип	Описание
deviceEventsQueue	*void**	Дескриптор очереди событий, полученный вызовом функции RG_Subscribe.
pEventType	*uint32_t**	Указатель на переменную, которой будет присвоено значение, соответствующее типу события E_RG_DEVICE_EVENT_TYPE, извлеченного из очереди.
pEventMem	*void***	Указатель на значение указателя, которому будет назначен адрес данных события в памяти.
pollWaitTimeoutMs	*uint32_t*	Время ожидания события в очереди, в миллисекундах.

Примечание:
Значение, находящееся по указателю pEventMem, остается актуальным только до следующего вызова функции RG_PollEvents.

```
void* eventDataPtr = nullptr;
uint32_t eventType = E_RG_DEVICE_EVENT_TYPE::DET_UNKNOWN_EVENT;
RG_PollEvents(eventQueue, &eventType, &eventDataPtr, 100);

// тут можем работать с eventDataPtr и eventDataTypedPtr будет актуальным (uint32_t для примера)
uint32_t* eventDataTypedPtr = reinterpret_cast<uint32_t*>(eventDataPtr);
if(*eventDataTypedPtr == 0 /*...*/) {
	// ... тут eventDataTypedPtr имеет актуальное значение
}
// делаем второй вызов
RG_PollEvents(eventQueue, &eventType, &eventDataPtr, 100);
if(*eventDataTypedPtr == 0 /*...*/) {
	// ... теперь использование eventDataTypedPtr приведет к Undefined Behavior
	// т.к. eventDataTypedPtr указывает на освобожденную память, которая может быть
	// занята другими данными
}
```

В зависимости от типа события, по указателю, переданному через аргумент pEventMem, могут находиться данные различного формата:

Тип события	Тип данных	Описание
DET_UNKNOWN_EVENT	-	Событие без данных.
DET_CARD_PLACED_EVENT	*RG_CARD_INFO**	В данных события – указатель на структуру RG_CARD_INFO содержащей сведения о карте.
DET_CARD_REMOVED_EVENT	-	Событие без данных.
DET_RELAY_STATE_CHANGED	*uint8_t**	Указатель на целое число, 8 бит, без знака. Содержит данные о новом и предыдущем состоянии реле: • 0 бит – новое состояние реле; • 1 бит – предыдущее состояние реле.

Тип события	Тип данных	Описание
DET_TAMPER_STATE_CHANGED	*uint8_t**	Указатель на целое число, 8 бит, без знака. Содержит данные о новом и предыдущем состоянии тампера: • 0 бит – новое состояние тампера; • 1 бит – предыдущее состояние тампера.
DET_BUTTON_STATE_CHANGED	*uint8_t**	Указатель на целое число, 8 бит, без знака. Содержит данные о новом и предыдущем состоянии кнопки: • 0 бит – новое состояние кнопки; • 1 бит – предыдущее состояние кнопки.
DET_DOOR_SENSOR_STATE_CHANGED	*uint8_t**	Указатель на целое число, 8 бит, без знака. Содержит данные о новом и предыдущем состоянии датчика двери: • 0 бит – новое состояние датчика двери; • 1 бит – предыдущее состояние датчика двери.
DET_POLL_ERROR	*uint32_t**	Указатель на целое число, 32 бита, без знака. Содержит код ошибки согласно таблице кодов ошибок.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, если в очереди нет событий, то возвращается код EC_POLL_NO_EVENTS, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_IsolateDevice (список)

```
API_METHOD api_error CALL_CONV RG_IsolateDevice(
       PRG_ENDPOINT pEndPoint,
       uint8_t address);
```

Описание:
Приостанавливает фоновый опрос считывателя для выполнения иных операций.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.

Функция RG_ResetIsolation (список)

```
API_METHOD api_error CALL_CONV RG_ResetIsolation(
       PRG_ENDPOINT pEndPoint,
       uint8_t address);
```

Описание:
Возобновляет фоновый опрос считывателя, остановленный ранее вызовом RG_IsolateDevice.

Аргументы:

Имя	Тип	Описание
pEndPoint	*PRG_ENDPOINT*	Указатель на инициализированную структуру параметров подключения RG_ENDPOINT.
address	*uint8_t*	Адрес считывателя.

Возвращаемое значение:
При успешном выполнении функция возвращает код EC_OK, в ином случае – код ошибки согласно таблице кодов ошибок.