Комплект для разработки программного обеспечения для работы с настольными и сетевыми считывателями Iron Logic

SDK Readers v4.0.10

Руководство пользователя

Оглавление

ВВЕДЕНИЕ

SDK Readers – это библиотека, предназначенная для работы с настольными и сетевыми считывателями Iron Logic.

Комплектация:

- bin\ — папка инструментария разработчика, содержит dll:
  - x86\ — 32-битная версия библиотеки ILReaders.dll;
  - x64\ — 64-битная версия библиотеки ILReaders.dll;
- demo\ — папка с программой Демо и её исходниками на Delphi;
  - Delphi\ — исходники на Delphi программы Демо;
  - Release_x64 — программа Демо;
- doc\ — папка с документацией;
  - ILReaders.chm — документация на русском;
- include\ — папка с заголовочными файлами;
  - C++\ — папка с заголовочными файлами на языке C++;
  - Delphi\ — папка с заголовочными файлами на языке Delphi;
  - C#\ — папка с заголовочными файлами на языке C#;
- lib\ — библиотеки для подключения разработчиками;
  - C++_Builder_x64\, C++_Builder_x86\ — библиотеки для подключения к проекту на C++ Builder;
  - Visual_C++_x64\, Visual_C++_x86\ — библиотеки для подключения к проекту на Visual C++;
- samples\ — папка с примерами использования SDK;
  - C++_Builder\ — примеры на языке C++ Builder;
  - Delphi\ — примеры на языке Delphi;
  - Visual_C++\ — примеры на языке Visual C++;
  - Visual_C#\ — примеры на языке C#.

Библиотеки SDK Readers написаны на языке Visual C++ в MSVS2022. Библиотека "ILReaders.dll" не является COM объектом, её не нужно регистрировать в ОС.

Основные возможности

- Чтение номеров карт с помощью RFID-считывателя (поддерживаемые модели);
- Возможность чтения/записи памяти ключей:
  - Mifare Ultralight (считыватели: Z-2 RD_ALL, Z-2 USB MF, Z-2 MF-I, Z-2 MF CCID, Matrix III Rd-All, Matrix III Net, CP-Z-2 MF-I);
  - Mifare Classic (считыватели: Z-2 USB MF, Z-2 MF-I, Z-2 MF CCID, Matrix III Net, CP-Z-2 MF-I);
  - Mifare Plus SL3 (считыватели: Z-2 MF-I);
  - Temic (считыватели: Z-2 RD_ALL, Z-2 EHR);
- Получение событий о нахождении/потери считывателя;
- Получение событий о поднесении/удалении карты.

Системные требования

ОС: Windows® 7/8/10/11 (32- или 64- битная версия)

Компиляторы и среды разработки:

Среда	Компилятор/Язык
Embarcadero RAD Studio XE5	Delphi XE5, C++Builder XE5
Visual Studio 2022	Visual C++ 2022, C# 2022

Требования к RFID-считывателю:

Поддерживаемые модели:

- подключаемые по USB: Z-2 Rd All, Z-2 USB MF, Z-2 EHR, Z-2 Base, RF-1996, Z-2 MF-I, Z-2 MF CCID
- подключаемые через конвертер (USB или IP): Matrix III Rd-All, Matrix III Net, Matrix V, CP-Z-2 MF-I, Matrix-VI (мод. NFC K Net)
    - поддерживаемые модели конвертеров: Z-397, Z-397 Guard (в режиме Normal), Z-397 Web (режим Server), Z-397 IP (режим Server)

Прошивки считывателей: только заводские версии

Драйвер: для моделей считывателей и конвертеров, подключаемых по USB, нужен драйвер (доступен на сайте www.ironlogic.ru)

Для устройств с чипом FTDI: считывателей Z-2 (мод. RD_ALL) (артикул 7716), Z-2 (мод. MF), Z-2 (мод. E HTZ RF) / Z-2 EHR, Z-2 (мод. E HT Hotel) / Z-2 RF-1996 и конвертеров Z-397, Z-397 Guard нужно установить стандартный PID = 0x6001 и специальное описание устройства:

- В ОС Windows скачайте и установите программу FT_Prog для изменения PID по ссылке https://ironlogic.ru/il_new.nsf/file/ru_ftdi-ft_prog.zip/$FILE/ftdi-ft_prog.zip или https://ftdichip.com/utilities/#ft_prog.
- Подключите устройство, в котором хотим сменить PID или описание. Если PID не равен 0x6001, то установите драйверы, которые шли в комплекте или скачайте с сайта https://ironlogic.ru/ на странице считывателя. Инструкция по установке драйверов находится в файле архива драйвера.
- Во избежание ошибок отключите "лишние" устройства (если есть).
- Запустите FT_Prog. В меню "Devices" выберите "Scan and Parse" (или нажмите F5). Появляется список найденных устройств.
- Для нужного устройства выберите слева пункт "USB_Device_Descriptor".
- Затем справа в свойствах "Custom VID/PID" выбираем "FTDI_Default".
- Выберите слева пункт "USB_String_Descriptors", затем справа в поле "Manufacturer:" установите "ILogic", и в поле "Product Description:" установите описание, соответствующее модели считывателя:

Модель	Описание
Z-2 (мод. RD_ALL)	USB IronLogic RFID Adapter
Z-2 (мод. MF)	USB IL Mifare Adapter
Z-2 (мод. E HTZ RF) / Z-2 EHR	USB IronLogic Adapter Z-2 EHR

Модель	Описание
Z-2 (мод. E HT Hotel) / Z-2 RF-1996	USB IL RF-96 Adapter
Z-397	USB <-> RS-485/422
Z-397 Guard	Z397 GUARD Converter

- В меню "Devices" выберите "Program" (или нажмите Ctrl+P).
- Появляется окно записи. В окне установите галочку напротив устройства в списке "Device List".
- Внизу снимите галочку с "Only Program Blank Device".
- Нажмите кнопку "Program".
- Отключите устройство от USB. При следующем подключении устройство будет находится в SDK как устройство IronLogic.

API

API состоит из 3-х функций и 6 интерфейсов. Главная функция ILR_GetInterface возвращает главный интерфейс IILR, который даёт интерфейс поиска считывателей IILRSearch и интерфейс подключения к считывателю IILReader. Методы интерфейсов блокирующие, т.е. пока не завершится команда управление не возвращается вызывающему потоку. Для работы в неблокирующем (асинхронном) режиме предназначены интерфейсы IILRSearchAsync и IILReaderAsync, а также IILRAsyncCommand, который представляет собой интерфейс команды, которую запускает метод интерфейсов IILRSearchAsync и IILReaderAsync.

Интерфейс IILReader позволяет получать событие поднесения/удаления карты к считывателю, а также позволяет читать/писать данные карт Mifare Ultralight, Mifare Classic, Mifare Plus и Temic (если поддерживается считывателем).

Функции

- ILR_GetVersion – возвращает номер версии библиотеки "ILReaders.dll".
- ILR_GetErrorText – возвращает описание ошибки по её коду.
- ILR_GetInterface – возвращает главный интерфейс библиотеки.

Функция ILR_GetVersion

Возвращает номер текущей версии библиотеки ILReaders.dll. Позволяет определить совместимость заголовочного файла с dll.

C#

```
[DllImport(DllName, CallingConvention =
CallingConvention.StdCall, EntryPoint = "ILR_GetVersion")]
public static extern UInt32 ILR_GetVersion();
```

C++

```
DWORD __stdcall ILR_GetVersion();
```

Delphi

```
function ILR_GetVersion(): Cardinal; stdcall;
```

Параметры:

Эта функция не имеет параметров.

Возвращаемое значение:

Возвращает DWORD значение, включающее в себя основной (в младшем байте) и дополнительный (остальные байты) номер версии SDK.

Примечание:

Версия, которую возвращает эта функция, должна совпадать со значением константы ILR_SDK_VERSION. Если не совпадает, то нельзя использовать ILReaders.dll вместе с заголовочным файлом с константой ILR_SDK_VERSION (=4).

Функция ILR_GetErrorText

Возвращает текст ошибки по коду ошибки.

C#

```
[DllImport(DllName, CharSet = CharSet.Unicode, CallingConvention
= CallingConvention.StdCall, EntryPoint = "ILR_GetErrorText")]
public static extern UInt32 ILR_GetErrorText(
  Int32 errorCode,
  [MarshalAs(UnmanagedType.BStr)] out String text);
```

C++

```
HRESULT __stdcall ILR_GetErrorText(HRESULT nErrorCode, OUT BSTR
*pText);
```

Delphi

```
function ILR_GetErrorText(AErrorCode: HResult; out VText:
TBStr): HResult; stdcall;
```

Параметры:

ErrorCode

[in] Код ошибки.

Text

[out] Текст ошибки. Для освобождения памяти используйте SysFreeString(Text).

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда Text = null.
E_OUTOFMEMORY	Недостаточно памяти.
Другой код ошибки	Код ошибки функции FormatMessage.

Примечание

Вместо этой функции можно использовать функцию FormatMessage. Если код ошибки возвращает метод интерфейса и код ошибки не стандартный (системный), то можно использовать функцию GetErrorInfo.

Функция ILR_GetInterface

Возвращает главный интерфейс библиотеки.

C#

```
[DllImport(DllName, CallingConvention =
CallingConvention.StdCall, EntryPoint = "ILR_GetInterface")]
public static extern Int32 ILR_GetInterface(
  [MarshalAs(UnmanagedType.Interface)] out IILR obj,
  UInt32 versionRequested = SDK_VERSION);
```

C++

```
HRESULT __stdcall ILR_GetInterface(IILR **ppObj, DWORD
nVersionRequested = ILR_SDK_VERSION);
```

Delphi

```
function ILR_GetInterface(out VObj: IILR;
    AVersionRequested: Cardinal = ILR_SDK_VERSION): HResult;
stdcall;
```

Параметры:

Obj

[out] Главный интерфейс IILR.

VersionRequested

[in] Требуемая версия SDK.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppObj= null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.
E_UNEXPECTED	Неожиданная ошибка.
ILR_E_WRONG_SDK_VERSION	Неправильная версия Sdk Readers.

Интерфейсы

- **IILR** – главный интерфейс SDK.
- **IILRSearch** – интерфейс поиска считывателей.
- **IILReader** – интерфейс подключения к считывателю.
- Интерфейсы для асинхронного режима:
  - **IILRAsyncCommand** – интерфейс асинхронной команды.
  - **IILRSearchAsync** – интерфейс с асинхронными функциями поиска считывателей.
  - **IILReaderAsync** – интерфейс с асинхронными функциями подключения к считывателю.

Интерфейс IILR

Главный интерфейс, создаёт интерфейсы IILRSearch, IILReader.

Методы:

- **SetFilterPortCallback** – устанавливает функцию обратного вызова для исключения портов.
- **SetLogPath** – устанавливает путь к файлу лога отладки.
- **GetLogPath** – возвращает путь к файлу лога отладки.
- **SetLogLevel** – устанавливает уровень лога отладки.
- **GetLogLevel** – возвращает уровень лога отладки.
- **ClearLog** – очищает лог отладки.
- **SetStopBits** – устанавливает количество стоповых бит для COM-порта.
- **GetStopBits** – возвращает количество стоповых бит для COM-порта.

- SetRequestTimeout – устанавливает тайм-аут запроса.
- GetRequestTimeout – возвращает тайм-аут запроса.
- SetRequestAttempts – устанавливает количество попыток запроса.
- GetRequestAttempts – возвращает количество попыток запроса.
- GetSearch – возвращает интерфейс поиска считывателей.
- GetReader – возвращает интерфейс подключения к считывателю.

Метод IILR.SetFilterPortCallback

Устанавливает функцию обратного вызова для исключения портов. Позволяет фильтровать порты, опрашиваемые при поиске считывателей.

C#

```
void
SetFilterPortCallback([MarshalAs(UnmanagedType.FunctionPtr)]
FilterPortProc callback, IntPtr userData);
```

C++

```
STDMETHOD(SetFilterPortCallback)(FilterPortProc pCallback, void
*pUserData) PURE;
```

Delphi

```
procedure SetFilterPortCallback(ACallback: TFilterPortProc;
AUserData: Pointer); safecall;
```

Параметры:

Callback

[in] Функция обратного вызова.

UserData

[in] Ссылка на данные пользователя, которая передаётся в функцию обратного вызова.

Возвращаемое значение:

S_OK – функция выполнена успешно.

Метод IILR.SetLogPath

Устанавливает путь к файлу лога отладки.

C#

```
void SetLogPath(String path);
```

C++

```
STDMETHOD(SetLogPath)(LPCWSTR pszPath) PURE;
```

Delphi

```
procedure SetLogPath(APath: LPCWSTR); safecall;
```

Параметры:

Path

[in] Путь к файлу лога отладки.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

E_OUTOFMEMORY	Недостаточно памяти.

Метод IILR.GetLogPath

Возвращает путь к файлу лога отладки.

C#

```
String GetLogPath();
```

C++

```
STDMETHOD(GetLogPath)(OUT BSTR *pPath) PURE;
```

Delphi

```
procedure GetLogPath(out VPath: TBStr); safecall;
```

Параметры:

Path

[out] Путь к файлу лога отладки. Строка должна быть освобождена функцией SysFreeString.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pPath = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILR.SetLogLevel

Устанавливает уровень лога отладки. Запись лога активируется если уровень лога не равен Disabled, и путь к лог файлу, установленный методом SetLogPath, не пустой.

C#

```
void SetLogLevel(LogLevel level);
```

C++

```
STDMETHOD(SetLogLevel)(LogLevel nLevel) PURE;
```

Delphi

```
procedure SetLogLevel(ALevel: TLogLevel); safecall;
```

Параметры:

Level

[in] Уровень лога.

Константа	Описание
Disabled	Лог выключен
Assert	Критические ошибки
Error	Ошибки
Warning	Предупреждения. Показывает возможные проблемы, которые не являются ошибками
Info	Уведомления. Показывает полезную информацию, в основном успехи
Debug	Отладочные сообщения. Показвает шаги программы, получаемые и отправляемые данные

Константа	Описание
Verbose	Подробные отладочные сообщения. Показывает каждую мелочь

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILR.GetLogLevel

Возвращает уровень лога отладки.

C#

```
LogLevel GetLogLevel();
```

C++

```
STDMETHOD(GetLogLevel)(LogLevel *pLevel) PURE;
```

Delphi

```
function GetLogLevel(): TLogLevel; safecall;
```

Параметры:

Level

[out] Уровень лога.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pLevel = null (актуально для C++).

Метод IILR.ClearLog

Очищает лог отладки. Устанавливает размер лог файла равный нулю.

C#

```
[PreserveSig]int ClearLog();
```

C++

```
STDMETHOD(ClearLog)() PURE;
```

Delphi

```
function ClearLog(): HResult; stdcall;
```

Параметры:

Эта функция не имеет параметров.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
Другая ошибка	Код системной ошибки.

Метод IILR.SetStopBits

Устанавливает количество стоповых бит для COM-порта. Применяется при подключении к считывателю. При установке нового значения автоматически не переподключается к считывателям.

C#

```
void SetStopBits(Byte stopBits);
```

C++

```
STDMETHOD(SetStopBits)(BYTE nStopBits) PURE;
```

Delphi

```
procedure SetStopBits(AStopBits: Byte); safecall;
```

Параметры:

StopBits

[in] Количество стоповых бит.

Константа	Значение	Описание
ONESTOPBIT	0	Один стоповый бит.
ONE5STOPBITS	1	Полтора стоповых бита.
TWOSTOPBITS	2	Два стоповых бита.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILR.GetStopBits

Возвращает количество стоповых бит для COM-порта.

C#

```
Byte GetStopBits();
```

C++

```
STDMETHOD(GetStopBits)(BYTE* pStopBits) PURE;
```

Delphi

```
function GetStopBits(): Byte; safecall;
```

Параметры:

StopBits

[out] Количество стоповых бит.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pStopBits = null (актуально для C++).

Метод IILR.SetRequestTimeout

Устанавливает тайм-аут запроса.

C#

```
void SetRequestTimeout(uint ms);
```

C++

```
STDMETHOD(SetRequestTimeout)(DWORD nMs) PURE;
```

Delphi

```
procedure SetRequestTimeout(AMs: Cardinal); safecall;
```

Параметры:

Ms

[in] Количество миллисекунд.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Ms = 0 или Ms = 0xffffffff.

Метод IILR.GetRequestTimeout

Возвращает тайм-аут запроса.

C#

```
uint GetRequestTimeout();
```

C++

```
STDMETHOD(GetRequestTimeout)(DWORD* pMs) PURE;
```

Delphi

```
function GetRequestTimeout(): Cardinal; safecall;
```

Параметры:

Ms

[out] Количество миллисекунд.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pMs = null (актуально для C++).

Метод IILR.SetRequestAttempts

Устанавливает количество попыток запроса.

C#

```
void SetRequestAttempts(int attempts);
```

C++

```
STDMETHOD(SetRequestAttempts)(INT nAttempts) PURE;
```

Delphi

```
procedure SetRequestAttempts(AAttempts: Integer); safecall;
```

Параметры:

Attempts

[in] Количество попыток.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Attempts < 1.

Метод IILR.GetRequestAttempts

Возвращает количество попыток запроса.

C#

```
int GetRequestAttempts();
```

C++

```
STDMETHOD(GetRequestAttempts)(INT* pAttempts) PURE;
```

Delphi

```
function GetRequestAttempts(): Integer; safecall;
```

Параметры:

Attempts

[out] Количество попыток.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pAttempts = null (актуально для C++).

Метод IILR.GetSearch

Возвращает интерфейс поиска считывателей. Создаёт новый объект интерфейса и создаёт поток для поиска считывателей если ещё не создан (синглтон).

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRSearch GetSearch();
```

C++

```
STDMETHOD(GetSearch)(IILRSearch **ppObj) PURE;
```

Delphi

```
function GetSearch(): IILRSearch; safecall;
```

Параметры:

Obj

[out] Интерфейс поиска считывателей.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppObj = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_OUT_OF_RESOURCES	Недостаточно ресурсов. Когда функция CreateEvent вернула ошибку.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда GetSearch вызвана из функции обратного вызова, установленной методом SetFilterPortCallback или методом IILRSearch.SetNotifyCallback.

Метод IILR.GetReader

Возвращает интерфейс подключения к считывателю. Создаёт новый объект интерфейса и создаёт поток для указанного порта если ещё не создан.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILReader GetReader(PortType portType, string portName);
```

C++

```
STDMETHOD(GetReader)(PortType nPortType, LPCWSTR pszPortName,
  IILReader **ppObj) PURE;
```

Delphi

```
function GetReader(APortType: TPortType; APortName: LPCWSTR):
IILReader; safecall;
```

Параметры:

PortType

[in] Тип порта считывателя.

Константа	Описание
ComPort	Имя виртуального COM порта считывателя
CCID	Имя CCID считывателя (Smart Cards).
Server	Адрес конвертера в режиме "Сервер" (например, 10.0.0.2:1000).

PortName

[in] Имя порта считывателя.

Obj

[out] Интерфейс подключения к считывателю.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppObj = null (актуально для C++).
E_INVALIDARG	Неправильные параметры. Когда PortName = null.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_OUT_OF_RESOURCES	Недостаточно ресурсов. Когда функция CreateEvent вернула ошибку.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда GetReader вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.

Интерфейс IILRSearch

Ищет считыватели (по запросу или в фоне), уведомляет о найденных/потерянных считывателях. Интерфейс IILRSearch можно получить с помощью главного интерфейса IILR методом GetSearch.

Методы:

- **SetNotifyCallback** – устанавливает функцию обратного вызова для уведомлений.
- **EnableMsgQueue** – включает/выключает очередь сообщений (для синхронизации).
- **GetMessage** – извлекает следующее сообщение из очереди.
- **SetReaderTypes** – устанавливает типы считывателей, которые нужно искать.
- **GetReaderTypes** – возвращает типы считывателей, которые нужно искать.
- **SetUdpScanPeriod** – устанавливает период опроса IP конвертеров по UDP в миллисекундах.
- **GetUdpScanPeriod** – возвращает период опроса IP конвертеров по UDP в миллисекундах.

- SetUdpRequestTimeout – устанавливает тайм-аут запроса по UDP для поиска IP конвертеров.
- GetUdpRequestTimeout – возвращает тайм-аут запроса по UDP в миллисекундах.
- SetUdpRequestAttempts – устанавливает количество попыток запроса по UDP для поиска IP конвертеров.
- GetUdpRequestAttempts – возвращает количество попыток запроса по UDP.
- SetUdpCvtAddresses – устанавливает IP адреса конвертеров для опроса по UDP, которые не находятся автоматически.
- GetUdpCvtAddresses – возвращает IP адреса конвертеров для опроса по UDP.
- SetListenTcpPorts – устанавливает список TCP-портов для прослушки конвертеров в режиме "Клиент".
- GetListenTcpPorts – возвращает список TCP-портов для прослушки конвертеров в режиме "Клиент".
- SetOpenListenerPeriod – устанавливает период между попытками открыть TCP порт для прослушки Клиентов.
- Scan – ищет считыватели.
- GetReaderCount – возвращает количество найденных считывателей.
- GetReaderInfo – возвращает инфо о найденном считывателе.
- EnableAutoScan – включает/выключает режим автоматического поиска считывателей.
- GetAutoScanEnabled – возвращает True если автоматический поиск включен.
- OpenPort – открывает порт.
- ClosePort – закрывает порт.

Метод IILRSearch.SetNotifyCallback

Устанавливает функцию обратного вызова для получения уведомлений об обнаружении/потери считывателей.

C#

```
void SetNotifyCallback([MarshalAs(UnmanagedType.FunctionPtr)]
SearchNotifyProc callback, IntPtr userData);
```

C++

```
STDMETHOD(SetNotifyCallback)(SearchNotifyProc pCallback, void
*pUserData) PURE;
```

Delphi

```
procedure SetNotifyCallback(ACallback: TSearchNotifyProc;
AUserData: Pointer); safecall;
```

Параметры:

Callback

[in] Ссылка на функцию обратного вызова.

UserData

[in] Ссылка на данные пользователя, которая будет передаваться в функцию обратного вызова.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILRSearch.EnableMsgQueue

Включает/выключает очередь сообщений (для синхронизации). Если очередь включена, то уведомления добавляются в список, чтобы извлечь уведомление из очереди нужно вызвать GetMessage.

C#

```
void EnableMsgQueue(
  [MarshalAs(UnmanagedType.Bool)] bool enable = true);
```

C++

```
STDMETHOD(EnableMsgQueue)(BOOL fEnable = TRUE) PURE;
```

Delphi

```
procedure EnableMsgQueue(AEnable: LongBool = True); safecall;
```

Параметры:

Enable

[in] True включает очередь.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILRSearch.GetMessage

Извлекает следующее сообщение из очереди. Очередь сообщений активируется методом EnableMsgQueue.

C#

```
[return: MarshalAs(UnmanagedType.Bool)]
bool GetMessage(out SearchMsg msgType, out IntPtr msgData);
```

C++

```
STDMETHOD(GetMessage)(SearchMsg *pMsg, LPCVOID *pMsgData, BOOL
*pFound) PURE;
```

Delphi

```
function GetMessage(out VMsg: TSearchMsg; out VMsgData:
Pointer): LongBool; safecall;
```

Параметры:

Msg

[out] Тип сообщения.

MsgData

[out] Ссылка на данные сообщения.

Found

[out] True, сообщение извлечено, иначе - нет сообщений.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pMsg = null, или pMsgData = null, или pFound = null (актуально для C++).

Метод IILRSearch.SetReaderTypes

Устанавливает типы считывателей, которые нужно искать. Автоматически запускает поиск считывателей.

C#

```
void SetReaderTypes(RDTYPEF types);
```

C++

```
STDMETHOD(SetReaderTypes)(RDTYPEF nTypes) PURE;
```

Delphi

```
procedure SetReaderTypes(ATypes: Cardinal); safecall;
```

Параметры:

Types

[in] Биты типов считывателей.

Флаг	Значение	Описание
RTF_ILUSB	0x01	USB считыватели Iron logic
RTF_TPUSB	0x02	USB считыватели сторонних производителей
RTF_CCID	0x04	Считыватели SmartCards
RTF_SERVER	0x08	IP конвертеры в режиме "Сервер" (поиск по UDP)

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILRSearch.GetReaderTypes

Возвращает типы считывателей, которые нужно искать.

C#

```
Int32 GetReaderTypes();
```

C++

```
STDMETHOD(GetReaderTypes)(RDTYPEF *pTypes) PURE;
```

Delphi

```
function GetReaderTypes(): Cardinal; safecall;
```

Параметры:

Types

[out] Биты типов считывателей.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pTypes = null (актуально для C++).

Метод IILRSearch.SetUdpScanPeriod

Устанавливает период опроса IP конвертеров по UDP в миллисекундах.

C#

```
void SetUdpScanPeriod(uint ms);
```

C++

```
STDMETHOD(SetUdpScanPeriod)(DWORD nMs) PURE;
```

Delphi

```
procedure SetUdpScanPeriod(AMs: Cardinal); safecall;
```

Параметры:

Ms

[in] Количество миллисекунд.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILRSearch.GetUdpScanPeriod

Возвращает период опроса IP конвертеров по UDP в миллисекундах.

C#

```
uint GetUdpScanPeriod();
```

C++

```
STDMETHOD(GetUdpScanPeriod)(DWORD* pMs) PURE;
```

Delphi

```
function GetUdpScanPeriod(): Cardinal; safecall;
```

Параметры:

Ms

[out] Количество миллисекунд.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pMs = null (актуально для C++).

Метод IILRSearch.SetUdpRequestTimeout

Устанавливает тайм-аут запроса по UDP для поиска IP конвертеров.

C#

```
void SetUdpRequestTimeout(uint ms);
```

C++

```
STDMETHOD(SetUdpRequestTimeout)(DWORD nMs) PURE;
```

Delphi

```
procedure SetUdpRequestTimeout(AMs: Cardinal); safecall;
```

Параметры:

Ms

[in] Количество миллисекунд.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILRSearch.GetUdpRequestTimeout

Возвращает тайм-аут запроса по UDP в миллисекундах.

C#

```
uint GetUdpRequestTimeout();
```

C++

```
STDMETHOD(GetUdpRequestTimeout)(DWORD* pMs) PURE;
```

Delphi

```
function GetUdpRequestTimeout(): Cardinal; safecall;
```

Параметры:

Ms

[out] Количество миллисекунд.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pMs = null (актуально для C++).

Метод IILRSearch.SetUdpRequestAttempts

Устанавливает количество попыток запроса по UDP (поиск IP конвертеров).

C#

```
void SetUdpRequestAttempts(int attempts);
```

C++

```
STDMETHOD(SetUdpRequestAttempts)(INT nAttempts) PURE;
```

Delphi

```
procedure SetUdpRequestAttempts(AAttempts: Integer); safecall;
```

Параметры:

Attempts

[in] Количество попыток запроса.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILRSearch.GetUdpRequestAttempts

Возвращает количество попыток запроса по UDP (поиск IP конвертеров).

C#

```
int GetUdpRequestAttempts();
```

C++

```
STDMETHOD(GetUdpRequestAttempts)(INT* pAttempts) PURE;
```

Delphi

```
function GetUdpRequestAttempts(): Integer; safecall;
```

Параметры:

Attempts

[out] Количество попыток запроса.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pAttempts = null (актуально для C++).

Метод IILRSearch.SetUdpCvtAddresses

Устанавливает IP адреса конвертеров для опроса по UDP, которые не находятся автоматически.

C#

```
void SetUdpCvtAddresses(string addresses);
```

C++

```
STDMETHOD(SetUdpCvtAddresses)(LPCWSTR pszAddresses) PURE;
```

Delphi

```
procedure SetUdpCvtAddresses(AAddresses: PWideChar); safecall;
```

Параметры:

Addresses

[in] Список адресов, разделённых символом ';'.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILRSearch.GetUdpCvtAddresses

Возвращает IP адреса конвертеров для опроса по UDP.

C#

```
string GetUdpCvtAddresses();
```

C++

```
STDMETHOD(GetUdpCvtAddresses)(OUT BSTR * pAddresses) PURE;
```

Delphi

```
procedure GetUdpCvtAddresses(out Addresses: TBStr); safecall;
```

Параметры:

Addresses

[out] Список адресов, разделённых символом ';'. Строка должна быть освобождена функцией SysFreeString.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILRSearch.SetListenTcpPorts

Устанавливает список TCP-портов для прослушки конвертеров в режиме "Клиент".

C#

```
void SetListenTcpPorts(
  [In, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 1)]
UInt16 ports,
  int count);
```

C++

```
STDMETHOD(SetListenTcpPorts)(const WORD* pPorts, INT nCount)
PURE;
```

Delphi

```
procedure SetListenTcpPorts(APorts: PWord; ACount: Integer);
safecall;
```

Параметры:

Ports

[in] Список номеров TCP-портов.

Count

[in] Количество TCP-портов.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILRSearch.GetListenTcpPorts

Возвращает список TCP-портов для прослушки конвертеров в режиме "Клиент".

C#

```
int GetListenTcpPorts(
  [Out, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 1)]
UInt16 ports,
  int count);
```

C++

```
STDMETHOD(GetListenTcpPorts)(WORD* pBuf, INT nCount, INT*
pRCount) PURE;
```

Delphi

```
function GetListenTcpPorts(VBuf: PWord; ACount: Integer):
Integer; safecall;
```

Параметры:

Ports

[out] Буфер для списка номеров TCP-портов.

Count

[in] Размер буфера (количество портов).

RCount

[out] Полученное количество TCP-портов.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
ILG_E_BUFFER_TOO_SMALL	Размер буфера слишком мал. Когда Count меньше количества номеров TCP-портов в списке.

Метод IILRSearch.SetOpenListenerPeriod

Устанавливает период между попытками открыть TCP порт для прослушки Клиентов.

C#

```
void SetOpenListenerPeriod(UInt32 ms);
```

C++

```
STDMETHOD(SetOpenListenerPeriod)(DWORD nMs) PURE;
```

Delphi

```
procedure SetOpenListenerPeriod(AMs: Cardinal); safecall;
```

Параметры:

Ms

[in] Количество миллисекунд.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILRSearch.Scan

Ищет считыватели.

C#

```
void Scan([MarshalAs(UnmanagedType.Bool)] bool reset = false);
```

C++

```
STDMETHOD(Scan)(BOOL fReset = FALSE) PURE;
```

Delphi

```
procedure Scan(AReset: LongBool = False); safecall;
```

Параметры:

Reset

[in] True, очистить список найденных перед поиском.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда Scan вызвана из функции обратного вызова, установленной методом SetFilterPortCallback или методом IILRSearch.SetNotifyCallback.

Метод IILRSearch.GetReaderCount

Возвращает количество найденных считывателей.

C#

```
int GetReaderCount();
```

C++

```
STDMETHOD(GetReaderCount)(INT *pCount) PURE;
```

Delphi

```
function GetReaderCount(): Integer; safecall;
```

Параметры:

Count

[out] Количество найденных считывателей.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pCount = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILRSearch.GetReaderInfo

Возвращает информацию о найденном считывателе.

C#

```
void GetReaderInfo(int idx, out ReaderInfo info);
```

C++

```
STDMETHOD(GetReaderInfo)(INT nIdx, ReaderInfo *pInfo) PURE;
```

Delphi

```
procedure GetReaderInfo(AIdx: Integer; out VInfo: TReaderInfo);
safecall;
```

Параметры:

Idx

[in] Позиция в списке найденных считывателей.

Info

[out] Информация о считывателе.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pInfo = null (актуально для C++).
E_BOUNDS	Индекс вне диапазона. Когда позиция Idx вне диапазона списка.

Метод IILRSearch.EnableAutoScan

Включает/выключает режим автоматического поиска считывателей.

C#

```
void EnableAutoScan(
  [MarshalAs(UnmanagedType.Bool)] bool enable = true,
```

```
  [MarshalAs(UnmanagedType.Bool)] bool wait = true);
```

C++

```
STDMETHOD(EnableAutoScan)(BOOL fEnable = TRUE, BOOL fWait =
TRUE) PURE;
```

Delphi

```
procedure EnableAutoScan(AEnable: LongBool = True; AWait:
LongBool = True); safecall;
```

Параметры:

Enable

[in] True, включает поиск в реальном времени, иначе - выключает.

Wait

[in] True, ждать завершение операции.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда EnableAutoScan с установленным флагом Wait = True вызвана из функции обратного вызова, установленной методом SetFilterPortCallback или методом IILRSearch.SetNotifyCallback.

Метод IILRSearch.GetAutoScanEnabled

Возвращает True если автоматический поиск включен. Для включения/выключения используйте метод EnableAutoScan.

C#

```
[return: MarshalAs(UnmanagedType.Bool)]
bool GetAutoScanEnabled();
```

C++

```
STDMETHOD(GetAutoScanEnabled)(BOOL *pEnabled) PURE;
```

Delphi

```
function GetAutoScanEnabled(): LongBool; safecall;
```

Параметры:

Enable

[out] True, автоматический поиск включен.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pEnabled = null (актуально для C++).

Метод IILRSearch.OpenPort

Открывает порт и возвращает дескриптор COM порта или дескриптор сокета.

C#

```
IntPtr OpenPort(PortType Type, string Name,
```

```
  out ReaderInfo Info);
```

C++

```
STDMETHOD(OpenPort)(PortType nPortType, LPCTSTR pszPortName,
  ReaderInfo* pInfo, HANDLE* pPort) PURE;
```

Delphi

```
function OpenPort(APortType: TPortType; APortName: PWideChar;
  out VInfo: TReaderInfo): THandle; safecall;
```

Параметры:

Type

[in] Тип порта.

Name

[in] Имя порта.

Info

[out] Информация о считывателе (если найден).

Port

[out] Дескриптор порта.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (PortType = ptUnknownPort) или (PortName = null) или (Port = null).
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда OpenPort вызвана из функции обратного вызова, установленной методом SetFilterPortCallback или методом IILGSearch.SetNotifyCallback.

Метод IILRSearch.ClosePort

Закрывает порт.

C#

```
void ClosePort(PortType Type, string Name, IntPtr hPort);
```

C++

```
STDMETHOD(ClosePort)(PortType nPortType, LPCTSTR pszPortName,
  HANDLE hPort) PURE;
```

Delphi

```
procedure ClosePort(APortType: TPortType; APortName: PWideChar;
  APort: THandle); safecall;
```

Параметры:

Type

[in] Тип порта.

Name

[in] Имя порта.

Port

[in] Дескриптор порта.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда ClosePort вызвана из функции обратного вызова, установленной методом SetFilterPortCallback или методом IILGSearch.SetNotifyCallback.

Интерфейс IILReader

Подключается к считывателю, ищет карты (по запросу или в фоне), уведомляет о найденных/потерянных картах, позволяет читать/писать данные некоторых карт (Mifare, Temic). Интерфейс IILReader можно получить с помощью главного интерфейса IILR методом GetReader.

Методы:

- **SetNotifyCallback** – устанавливает функцию обратного вызова для уведомлений.
- **EnableMsgQueue** – включает/выключает очередь сообщений (для синхронизации).
- **GetMessage** – извлекает следующее сообщение из очереди.
- **SetModelToConnect** – устанавливает модель для подключения.
- **GetModelToConnect** – возвращает модель для подключения.
- **Connect** – подключается к считывателю.
- **Disconnect** – отключается от считывателя.
- **GetConnectionStatus** – возвращает состояние подключения к считывателю.
- **GetReaderInfo** – возвращает информацию о считывателе.
- **Scan** – ищет карту в поле считывателя.
- **GetCardInfo** – возвращает информацию о карте в поле считывателя.
- **EnableAutoScan** – включает/выключает автоматическое сканирование карт.
- **GetAutoScanEnabled** – возвращает True если авто сканирование карт включено.
- **SetHoldCardTypes** – устанавливает типы карт, при обнаружении которых сканирование приостанавливается. Чтобы возобновить сканирование карт нужно вызвать EnableAutoScan(True).
- **GetHoldCardTypes** – возвращает типы карт, при обнаружении которых автоматически приостанавливается сканирование.
- **Для Mifare Ultralight:**
  - **ReadMfUlralight** – читает данные карты Mifare Ultralight.
  - **WriteMfUlralight** – пишет данные карты Mifare Ultralight.
- **Для Mifare Classic/Plus:**
  - **LoadMfAuthKey** – загружает ключ для авторизации сектора Mifare Classic / Plus SL1.
  - **LoadMfPlusAuthKey** – загружает ключ для авторизации сектора Mifare Plus SL3.
  - **AuthMfCard** – авторизует сектор карты Mifare Classic / Plus, используя ключ, загруженный методом LoadMfAuthKey / LoadMfPlusAuthKey.
  - **AuthMfCardByRdKeys** – авторизует сектор карты Mifare Classic / Plus, используя ключи считывателя.
  - **ReadMfClassic** – читает данные карты Mifare Classic или Mifare Plus SL1.

- **WriteMfClassic** – пишет данные карты Mifare Classic или Mifare Plus SL1.
- **ReadMfPlus** – читает данные карты Mifare Plus SL3.
- **WriteMfPlus** – пишет данные карты Mifare Plus SL3.
- **MfIncrement** – увеличивает содержимое блока-значения карты Mifare и сохраняет результат во временном регистре данных.
- **MfDecrement** – уменьшает содержимое блока-значения карты Mifare и сохраняет результат во временном регистре данных.
- **MfTransfer** – записывает содержимое во временном регистре данных в блок-значение.
- **MfRestore** – перемещает содержимое блока в регистр данных Mifare.
- **MfPowerOff** – выключает RF поле считывателя.
- **MfRAS** – R+A+S(Request+Anticollision+Select).
- **MfRR** – R+R(Request+Reselect(по известному номеру)).
- **MfHalt** – Halt.
- **MfRATS** – Переходит на ISO 14443-4.
- **MfWritePerso** – записывает ключи AES и всех блоков.
- **MfCommitPerso** – переключает Mifare Plus в SL1 или SL3(если SL1 нет).
- **WriteMfAuthKeyToReader** – записывает ключи аутентификации Mifare Classic в память считывателя.
- **WriteMfPlusAuthKeyToReader** – записывает ключи аутентификации Mifare Plus в память считывателя.

- **Для Temic:**
  - **LoadTemicPassword** – загружает пароль Temic в память объекта считывателя.
  - **ScanTemic** – ищет карту Temic в поле считывателя.
  - **EnableAutoScanTemic** – включает/выключает сканирование карт Temic (для Z-2 Rd-All и Z-2 EHR).
  - **GetAutoScanTemicEnabled** – возвращает True если авто сканирование Temic включено.
  - **ReadTemic** – читает данные из карты Temic.
  - **WriteTemic** – пишет данные в карту Temic.
  - **ResetTemic** – сброс TRES.
  - **EncodeTemicEmMarine** – шифрует данные для эмуляции Em-Marine, для записи в блоки 0..2.
  - **DecodeTemicEmMarine** – дешифрует номер Em-Marine из данных блоков 0..2 карты Temic.
  - **EncodeTemicHID** – шифрует данные для эмуляции HID, для записи в блоки 0..3.
  - **DecodeTemicHID** – дешифрует номер HID из данных блоков 0..3 карты Temic.

Метод IILReader.SetNotifyCallback

Устанавливает функцию обратного вызова для получения уведомлений о обнаружении/потери карты и других.

C#

```
void SetNotifyCallback(
  [MarshalAs(UnmanagedType.FunctionPtr)] ReaderNotifyProc
callback,
  IntPtr userData);
```

C++

```
STDMETHOD(SetNotifyCallback)(ReaderNotifyProc pCallback, void
*pUserData) PURE;
```

Delphi

```
procedure SetNotifyCallback(ACallback: TReaderNotifyProc;
AUserData: Pointer); safecall;
```

Параметры:

Callback

[in] Ссылка на функцию обратного вызова.

UserData

[in] Ссылка на данные пользователя, которая будет передаваться в функцию обратного вызова.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILReader.EnableMsgQueue

Включает/выключает очередь сообщений (для синхронизации).

C#

```
void EnableMsgQueue([MarshalAs(UnmanagedType.Bool)] bool enable
= true);
```

C++

```
STDMETHOD(EnableMsgQueue)(BOOL fEnable = TRUE) PURE;
```

Delphi

```
procedure EnableMsgQueue(AEnable: LongBool = True); safecall;
```

Параметры:

Enable

[in] True, включает очередь, иначе - выключает.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILReader.GetMessage

Извлекает следующее сообщение из очереди. Очередь сообщений активируется методом EnableMsgQueue.

C#

```
[return: MarshalAs(UnmanagedType.Bool)]
bool GetMessage(out ReaderMsg msgType, out IntPtr msgData);
```

C++

```
STDMETHOD(GetMessage)(ReaderMsg *pMsg, LPCVOID *pMsgData, BOOL
*pFound) PURE;
```

Delphi

```
function GetMessage(out VMsg: TReaderMsg; out VMsgData:
Pointer): LongBool; safecall;
```

Параметры:

Msg

[out] Тип сообщения.

MsgData

[out] Ссылка на данные сообщения.

Found

[out] True, сообщение извлечено, иначе - нет сообщений.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pMsg = null, или pMsgData = null, или pFound = null (актуально для C++).

Метод IILReader.SetModelToConnect

Устанавливает модель для подключения.

C#

```
void SetModelToConnect(ReaderModel model);
```

C++

```
STDMETHOD(SetModelToConnect)(ReaderModel nModel) PURE;
```

Delphi

```
procedure SetModelToConnect(AModel: TReaderModel); safecall;
```

Параметры:

Model

[in] Модель считывателя, к которому нужно подключиться.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILReader.GetModelToConnect

Возвращает модель для подключения.

C#

```
ReaderModel GetModelToConnect();
```

C++

```
STDMETHOD(GetModelToConnect)(ReaderModel* pModel) PURE;
```

Delphi

```
function GetModelToConnect(): TReaderModel; safecall;
```

Параметры:

Model

[out] Модель считывателя, к которому которому подключаемся.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pModel = null (актуально для C++).

Метод ILReader.Connect

Подключается к считывателю.

C#

```
void Connect(
  [MarshalAs(UnmanagedType.Bool)] bool reconnect = false);
```

C++

```
STDMETHOD(Connect)(BOOL fReconnect = FALSE) PURE;
```

Delphi

```
procedure Connect(AReconnect: LongBool = False); safecall;
```

Параметры:

Reconnect

[in] True, переподключиться.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда Connect вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.

Метод IILReader.Disconnect

Отключается от считывателя.

C#

```
void Disconnect();
```

C++

```
STDMETHOD(Disconnect)() PURE;
```

Delphi

```
procedure Disconnect(); safecall;
```

Параметры:

Эта функция не имеет параметров.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_OUTOFMEMORY	Недостаточно памяти.

ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда Disconnect вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.

Метод IILReader.GetConnectionStatus

Возвращает состояние подключения к считывателю.

C#

```
ConnectionStatus GetConnectionStatus();
```

C++

```
STDMETHOD(GetConnectionStatus)(ConnectionStatus *pStatus) PURE;
```

Delphi

```
function GetConnectionStatus(): TConnectionStatus; safecall;
```

Параметры:

Status

[out] Состояние подключения.

Константа	Описание
Disconnected	Отключён
Connected	Подключён
Connecting	Подключение

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pStatus = null (актуально для C++).

Метод IILReader.GetReaderInfo

Возвращает информацию о считывателе.

C#

```
void GetReaderInfo(out ReaderInfo info);
```

C++

```
STDMETHOD(GetReaderInfo)(ReaderInfo *pInfo) PURE;
```

Delphi

```
procedure GetReaderInfo(out VInfo: TReaderInfo); safecall;
```

Параметры:

Info

[out] Информация о считывателе.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pInfo = null (актуально для C++).

Метод IILReader.Scan

Ищет карту в поле считывателя.

C#

```
void Scan(
  [MarshalAs(UnmanagedType.Bool)] bool reset = false,
  [MarshalAs(UnmanagedType.Bool)] bool powerOff = true);
```

C++

```
STDMETHOD(Scan)(
  BOOL fReset = FALSE,
  BOOL fPowerOff = TRUE) PURE;
```

Delphi

```
procedure Scan(
  AReset: LongBool = False;
  APowerOff: LongBool = True); safecall;
```

Параметры:

Reset

[in] True, сбросить старые результаты поиска.

PowerOff

[in] True, выключает RF поле после сканирования.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.

Метод IILReader.GetCardInfo

Возвращает информацию о карте в поле считывателя.

C#

```
void GetCardInfo(out CardInfo info);
```

C++

```
STDMETHOD(GetCardInfo)(CardInfo *pInfo) PURE;
```

Delphi

```
procedure GetCardInfo(out VInfo: TCardInfo); safecall;
```

Параметры:

Info

[out] Информация о карте.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pInfo = null (актуально для C++).

Метод IILReader.EnableAutoScan

Включает/выключает автоматическое сканирование карт.

C#

```
void EnableAutoScan(
  [MarshalAs(UnmanagedType.Bool)] bool enable = true,
  [MarshalAs(UnmanagedType.Bool)] bool wait = true);
```

C++

```
STDMETHOD(EnableAutoScan)(BOOL fEnable = TRUE, BOOL fWait =
TRUE) PURE;
```

Delphi

```
procedure EnableAutoScan(AEnable: LongBool = True; AWait:
LongBool = True); safecall;
```

Параметры:

Enable

[in] True, включает сканирование карт.

Wait

[in] True, не возвращает управление пока операция не будет выполнена.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции IILReader.SetNotifyCallback.

Метод IILReader.GetAutoScanEnabled

Возвращает True если автоматический поиск включен. Для включения/выключения используйте метод EnableAutoScan.

C#

```
[return: MarshalAs(UnmanagedType.Bool)]
bool GetAutoScanEnabled();
```

C++

```
STDMETHOD(GetAutoScanEnabled)(BOOL *pEnabled) PURE;
```

Delphi

```
function GetAutoScanEnabled(): LongBool; safecall;
```

Параметры:

Enabled

[out] True, автоматический поиск включен.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pEnabled = null (актуально для C++).

Метод IILReader.SetHoldCardTypes

Устанавливает типы карт, при обнаружении которых сканирование приостанавливается. Чтобы возобновить сканирование карт нужно вызвать EnableAutoScan(True).

C#

```
void SetHoldCardTypes(RWCTF types);
```

C++

```
STDMETHOD(SetHoldCardTypes)(RWCTF nTypes) PURE;
```

Delphi

```
procedure SetHoldCardTypes(ATypes: Cardinal); safecall;
```

Параметры:

Types

[in] Типы карт, при нахождении которых автоматическое сканирование выключается.

Константа	Значение	Описание
RWCTF_MFULTRALIGHT	0x01	Mifare Ultralight
RWCTF_MFCLASSIC	0x02	Mifare Classic
RWCTF_MFPPLUS	0x04	Mifare Plus
RWCTF_TEMIC	0x08	Temic

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILReader.GetHoldCardTypes

Возвращает типы карт, при обнаружении которых автоматически приостанавливается сканирование.

C#

```
RWCTF GetHoldCardTypes();
```

C++

```
STDMETHOD(GetHoldCardTypes)(RWCTF* pTypes) PURE;
```

Delphi

```
function GetHoldCardTypes(): Cardinal; safecall;
```

Параметры:

Types

[out] Типы карт, при нахождении которых автоматическое сканирование выключается.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILReader.ReadMfUlralight

Читает данные карты Mifare Ultralight.

C#

```
void ReadMfUlralight(int pageIdx,
  [Out, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
UInt32[] buf,
  int pageCount, out int read);
```

C++

```
STDMETHOD(ReadMfUlralight)(INT nPageIdx,
  DWORD *pBuf,
  INT nPageCount, INT *pRead = NULL) PURE;
```

Delphi

```
procedure ReadMfUlralight(APageIdx: Integer;
  VBuf: PCardinal;
  APageCount: Integer; VRead: PInteger = nil); safecall;
```

Параметры:

PageIdx

[in] Номер первой читаемой страницы (0..15).

Buf

[out] Буфер для прочитанных страниц.

PageCount

[in] Количество страниц, которые нужно прочитать.

Read

[out] Количество прочитанных страниц. Если функция завершается успешно, то Read = PageCount.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (PageIdx < 0) или (PageCount <= 0) или ((PageIdx + PageCount) > 16).
E_POINTER	Неправильный указатель. Когда Buf = null.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Z-2 Rd-All, Matrix III Rd-All, Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I, Z-2 MF CCID.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.
ILR_E_SCARD_ERROR	Ошибка функции SmartCards (актуально для CCID считывателя).

Метод IILReader.WriteMfUlralight

Пишет данные карты Mifare Ultralight.

C#

```
void WriteMfUlralight(int pageIdx,
  [In, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
UInt32[] data,
  int pageCount, out int written);
```

C++

```
STDMETHOD(WriteMfUlralight)(INT nPageIdx,
  const DWORD *pData,
  INT nPageCount, INT *pWritten = NULL) PURE;
```

Delphi

```
procedure WriteMfUlralight(APageIdx: Integer;
  const AData: PCardinal;
  APageCount: Integer; VWritten: PInteger = nil); safecall;
```

Параметры:

PageIdx

[in] Номер первой записываемой страницы (0..15).

Data

[in] Данные страниц.

PageCount

[in] Количество страниц, которые нужно записать.

Written

[out] Количество записанных страниц. Если функция завершается успешно, то Written = PageCount.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (PageIdx < 0) или (PageCount <= 0) или ((PageIdx + PageCount) > 16).
E_OUTOFMEMORY	Недостаточно памяти.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Z-2 Rd-All, Matrix III Rd-All, Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I, Z-2 MF CCID.
E_ABORT	Операция прервана.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_PAGE_LOCK	Страница карты Mifare Ultralight заблокирована.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

ILR_E_SCARD_ERROR	Ошибка функции SmartCards (актуально для CCID считывателя).

Метод IILReader.LoadMfAuthKey

Загружает ключ для авторизации сектора Mifare Classic / Plus SL1.

C#

```
void LoadMfAuthKey(Int64 key);
```

C++

```
STDMETHOD(LoadMfAuthKey)(const MfClassicKey nKey) PURE;
```

Delphi

```
procedure LoadMfAuthKey(const AKey: TMfClassicKey); safecall;
```

Параметры:

Key

[in] Ключ аутентификации Mifare Classic.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Key < 0) или (Key > 0xffffffffffff).

Метод IILReader.LoadMfPlusAuthKey

Загружает ключ для авторизации сектора Mifare Plus SL3.

C#

```
void LoadMfPlusAuthKey(ref MfPlusKey key);
```

C++

```
STDMETHOD(LoadMfPlusAuthKey)(const MfPlusKey &rKey) PURE;
```

Delphi

```
procedure LoadMfPlusAuthKey(const [Ref] AKey: TMfPlusKey);
safecall;
```

Параметры:

Key

[in] Ключ аутентификации Mifare Plus.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILReader.AuthMfCard

Авторизует сектор карты Mifare Classic / Plus, используя ключ, загруженный функцией LoadMfAuthKey / LoadMfPlusAuthKey.

C#

```
[return: MarshalAs(UnmanagedType.Bool)]
bool AuthMfCard(uint address,
  [MarshalAs(UnmanagedType.Bool)] bool keyB);
```

C++

```
STDMETHOD(AuthMfCard)(UINT nAddress, BOOL fKeyB, BOOL *pAuthOk)
PURE;
```

Delphi

```
function AuthMfCard(AAddress: Cardinal; AKeyB: LongBool):
LongBool; safecall;
```

Параметры:

Address

[in] Номер блока (0..255) или адрес Mifare Plus.

KeyB

[in] True, авторизовать по ключу Б, иначе - по ключу А.

AuthOk

[out] True, сектор успешно авторизован.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Address > 0xffff).
E_POINTER	Неправильный указатель. Когда pAuthOk = null (актуально для C++).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I, Z-2 MF CCID.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_MIFARE_ADDRESS	Номер блока вне диапазона блоков карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.
ILR_E_SCARD_ERROR	Ошибка функции SmartCards (актуально для CCID считывателя).

Метод IILReader.AuthMfCardByRdKeys

Авторизует сектор карты Mifare Classic / Plus по ключам в памяти считывателя. Записать ключ в память считывателя можно методом WriteMfAuthKeyToReader / WriteMfPlusAuthKeyToReader.

C#

```
int AuthMfCardByRdKeys(uint address,
  [MarshalAs(UnmanagedType.Bool)] bool keyB,  UInt32 rdKeys);
```

C++

```
STDMETHOD(AuthMfCardByRdKeys)(UINT nAddress,
  BOOL fKeyB,
  DWORD nRdKeys = 0xFFFF,
```

```
  INT *pRdKeyIdx = NULL) PURE;
```

Delphi

```
function AuthMfCardByRdKeys(AAddress: Cardinal;
  AKeyB: LongBool;
  ARdKeys: Cardinal = $FFFF): Integer; safecall;
```

Параметры:

Address

[in] Номер блока (0..255) или адрес Mifare Plus.

KeyB

[in] True, авторизовать по ключу Б, иначе - по ключу А.

RdKeys

[in] Биты (0..15) ключей в памяти считывателя.

RdKeyIdx

[out] Номер найденного ключа или -1 если ключ не найден.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Address > 0xffff).
E_POINTER	Неправильный указатель. Когда pRdKeyIdx = null (актуально для C++).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_BOUNDS	Номер блока вне диапазона блоков карты.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.ReadMfClassic

Читает данные карты Mifare Classic или Mifare Plus SL1. Перед чтением данных из сектора нужно авторизовать сектор методом AuthMfCard или AuthMfCardByRdKeys, параметры авторизации, переданные в эти функции, будут использованы для всех секторов (нет необходимости снова вызывать AuthMfCard для каждого сектора если параметры авторизации не изменились).

C#

```
void ReadMfClassic(int blockIdx,
  [Out, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
MfBlockData[] buf,
  int blockCount, out int read);
```

C++

```
STDMETHOD(ReadMfClassic)(INT nBlockIdx,
  MfBlockData *pBuf,
  INT nBlockCount, INT *pRead = NULL) PURE;
```

Delphi

```
procedure ReadMfClassic(ABlockIdx: Integer;
  VBuf: PMfBlockData;
  ABlockCount: Integer; VRead: PInteger = nil); safecall;
```

Параметры:

BlockIdx

[in] Номер первого читаемого блока (0..255).

Buf

[out] Буфер для прочитанных блоков.

BlockCount

[in] Количество блоков, которые нужно прочитать.

Read

[out] Количество прочитанных блоков. Если функция завершается успешно, то Read = BlockCount.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockCount <= 0) или ((BlockIdx + BlockCount) > 0xff).
E_POINTER	Неправильный указатель. Когда Buf = null (актуально для C++).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I, Z-2 MF CCID.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.
ILR_E_SCARD_ERROR	Ошибка функции SmartCards (актуально для CCID считывателя).

Метод IILReader.WriteMfClassic

Пишет данные карты Mifare Classic или Mifare Plus SL1. Перед записью данных в сектор нужно авторизовать сектор методом AuthMfCard или AuthMfCardByRdKeys, параметры авторизации, переданные в эти функции, будут использованы для всех секторов (нет необходимости снова вызывать AuthMfCard для каждого сектора если параметры авторизации не изменились).

C#

```
void WriteMfClassic(int blockIdx, [In,
MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
MfBlockData[] data, int blockCount, out int written);
```

C++

```
STDMETHOD(WriteMfClassic)(INT nBlockIdx,
  const MfBlockData *pData,
  INT nBlockCount, INT *pWritten = NULL) PURE;
```

Delphi

```
procedure WriteMfClassic(ABlockIdx: Integer;
  const AData: PMfBlockData;
  ABlockCount: Integer; VWritten: PInteger = nil); safecall;
```

Параметры:

BlockIdx

[in] Номер первого записываемого блока (0..255).

Data

[in] Данные записываемых блоков.

BlockCount

[in] Количество блоков, которые нужно записать.

Written

[out] Количество записанных блоков. Если функция завершается успешно, то Written = BlockCount.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockCount <= 0) или ((BlockIdx + BlockCount) > 0xff) или (Data = null).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I, Z-2 MF CCID.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.
ILR_E_SCARD_ERROR	Ошибка функции SmartCards (актуально для CCID считывателя).

Метод IILReader.ReadMfPlus

Читает данные карты Mifare Plus SL3. Перед чтением данных из сектора нужно авторизовать сектор методом AuthMfCard или AuthMfCardByRdKeys, параметры авторизации, переданные в эти функции, будут использованы для всех секторов (нет необходимости снова вызывать AuthMfCard для каждого сектора если параметры авторизации не изменились).

C#

```
void ReadMfPlus(uint address,
  [Out, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
MfBlockData[] buf,
  int blockCount,  [MarshalAs(UnmanagedType.Bool)] bool openText
/*= true*/,
  out int read);
```

C++

```
STDMETHOD(ReadMfPlus)(UINT nAddress,
  MfBlockData *pBuf,
  INT nBlockCount, BOOL fOpenText = TRUE,
  INT *pRead = NULL) PURE;
```

Delphi

```
procedure ReadMfPlus(AAddress: Cardinal;
  VBuf: PMfBlockData;
  ABlockCount: Integer; AOpenText: LongBool = True;
  VRead: PInteger = nil); safecall;
```

Параметры:

Address

[in] Номер первого читаемого блока (0..255).

Buf

[out] Буфер для прочитанных блоков.

BlockCount

[in] Количество блоков, которые нужно прочитать.

OpenText

[in] True, открытая передача, иначе - зашифрованная.

Read

[out] Количество прочитанных блоков. Если функция завершается успешно, то Read = BlockCount.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Address > 0xffff) или (BlockCount <= 0).
E_POINTER	Неправильный указатель. Когда Buf = null (актуально для C++).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.

ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.WriteMfPlus

Пишет данные карты Mifare Plus SL3. Перед записью данных в сектор нужно авторизовать сектор методом AuthMfCard или AuthMfCardByRdKeys, параметры авторизации, переданные в эти функции, будут использованы для всех секторов (нет необходимости снова вызывать AuthMfCard для каждого сектора если параметры авторизации не изменились).

C#

```
void WriteMfPlus(uint address,
  [In, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
MfBlockData[] data,
  int blockCount,
  [MarshalAs(UnmanagedType.Bool)] bool openText /*= true*/,
  out int written);
```

C++

```
STDMETHOD(WriteMfPlus)(UINT nAddress,
  const MfBlockData *pData,
  INT nBlockCount, BOOL fOpenText = TRUE,
  INT *pWritten = NULL) PURE;
```

Delphi

```
procedure WriteMfPlus(AAddress: Cardinal;
  const AData: PMfBlockData;
  ABlockCount: Integer; AOpenText: LongBool = True;
  VWritten: PInteger = nil); safecall;
```

Параметры:

Address

[in] Номер первого записываемого блока (0..255) или адрес Mifare Plus.

Data

[in] Данные записываемых блоков.

BlockCount

[in] Количество блоков, которые нужно записать.

OpenText

[in] True, открытая передача, иначе - зашифрованная.

Written

[out] Количество записанных блоков. Если функция завершается успешно, то Written = BlockCount.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Address > 0xffff) или (BlockCount <= 0) или (Data = null).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.MfIncrement

Увеличивает содержимое блока-значения карты Mifare и сохраняет результат во временном регистре данных.
Перед изменением данных в секторе нужно авторизовать сектор методом AuthMfCard или AuthMfCardByRdKeys, параметры авторизации, переданные в эти функции, будут использованы для всех секторов (нет необходимости снова вызывать AuthMfCard для каждого сектора если параметры авторизации не изменились).

C#

```
void MfIncrement(int blockIdx, uint value);
```

C++

```
STDMETHOD(MfIncrement)(INT nBlockIdx, DWORD nValue) PURE;
```

Delphi

```
procedure MfIncrement(ABlockIdx: Integer; Value: Cardinal);
safecall;
```

Параметры:

BlockIdx

[in] Номер блока (0..255).

Value

[in] Величина инкремента.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockIdx > 0xff).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.

ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.MfDecrement

Уменьшает содержимое блока-значения карты Mifare и сохраняет результат во временном регистре данных.
Перед изменением данных в секторе нужно авторизовать сектор методом AuthMfCard или AuthMfCardByRdKeys, параметры авторизации, переданные в эти функции, будут использованы для всех секторов (нет необходимости снова вызывать AuthMfCard для каждого сектора если параметры авторизации не изменились).

C#

```
void MfDecrement(int blockIdx, uint value);
```

C++

```
STDMETHOD(MfDecrement)(INT nBlockIdx, DWORD nValue) PURE;
```

Delphi

```
procedure MfDecrement(ABlockIdx: Integer; Value: Cardinal);
safecall;
```

Параметры:

BlockIdx

[in] Номер блока (0..255).

Value

[in] Величина декремента.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockIdx > 0xff).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.MfTransfer

Записывает содержимое во временном регистре данных в блок-значение.
Перед изменением данных в секторе нужно авторизовать сектор методом AuthMfCard или AuthMfCardByRdKeys, параметры авторизации, переданные в эти функции, будут использованы для всех секторов (нет необходимости снова вызывать AuthMfCard для каждого сектора если параметры авторизации не изменились).

C#

```
void MfTransfer(int blockIdx);
```

C++

```
STDMETHOD(MfTransfer)(INT nBlockIdx) PURE;
```

Delphi

```
procedure MfTransfer(ABlockIdx: Integer); safecall;
```

Параметры:

BlockIdx

[in] Номер блока (0..255).

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockIdx > 0xff).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.MfRestore

Перемещает содержимое блока в регистр данных Mifare.
Перед изменением данных в секторе нужно авторизовать сектор методом AuthMfCard или AuthMfCardByRdKeys, параметры авторизации, переданные в эти функции, будут использованы для всех секторов (нет необходимости снова вызывать AuthMfCard для каждого сектора если параметры авторизации не изменились).

C#

```
void MfRestore(int blockIdx);
```

C++

```
STDMETHOD(MfRestore)(INT nBlockIdx) PURE;
```

Delphi

```
procedure MfRestore(ABlockIdx: Integer); safecall;
```

Параметры:

BlockIdx

[in] Номер блока (0..255).

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockIdx > 0xff).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.MfPowerOff

Выключает RF поле считывателя. После выключения нужно подождать 10 мс, чтобы карты отключились. Поле включается автоматически при запросе к карте.

C#

```
void MfPowerOff();
```

C++

```
STDMETHOD(MfPowerOff)() PURE;
```

Delphi

```
procedure MfPowerOff; safecall;
```

Параметры:

Эта функция не имеет параметров.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.MfRAS

R+A+S(Request+Anticollision+Select). Запрос поиска карты в поле считывателя.

C#

```
[return: MarshalAs(UnmanagedType.Bool)]bool
MfRAS([MarshalAs(UnmanagedType.Bool)] bool wakeUp,
  out Byte SAK, out Int16 ATQ, out CardUID uID);
```

C++

```
STDMETHOD(MfRAS)(BOOL fWakeUp,
  BYTE *pSAK, WORD *pATQ, CardUID *pUID, BOOL* pFound) PURE;
```

Delphi

```
function MfRAS(AWakeUp: LongBool;
  out VSAK: Byte; out VATQ: Word; out VUID: TCardUID): LongBool;
safecall;
```

Параметры:

WakeUp

[in] True, разбудить карту.

SAK

[out] Код SAK.

ATQ

[out] Код ATQ.

UID

[out] Номер карты.

Возвращаемое значение:

True, карта найдена, иначе - не найдена.

S_OK	Функция выполнена успешно.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.MfRR

R+R(Request+Reselect. Запрос поиска определённой карты в поле считывателя.

C#

```
[return: MarshalAs(UnmanagedType.Bool)]bool
MfRR([MarshalAs(UnmanagedType.Bool)] bool wakeUp,
```

```
  [In] CardUID uID);
```

C++

```
STDMETHOD(MfRR)(BOOL fWakeUp, const CardUID & rUID,
  BOOL* pFound) PURE;
```

Delphi

```
function MfRR(AWakeUp: LongBool;
  const [Ref] AUID: TCardUID): LongBool; safecall;
```

Параметры:

WakeUp

[in] True, разбудить карту.

UID

[in] Номер карты.

Возвращаемое значение:

True, карта найдена, иначе - не найдена.

S_OK	Функция выполнена успешно.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.MfHalt

Halt. Усыпляет последнюю найденную карту в поле считывателя.

C#

```
void MfHalt();
```

C++

```
STDMETHOD(MfHalt)() PURE;
```

Delphi

```
procedure MfHalt(); safecall;
```

Параметры:

Эта функция не имеет параметров.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.MfRATS

Переходит на ISO 14443-4.

C#

```
void MfRATS([Out, MarshalAs(UnmanagedType.LPArray,
SizeParamIndex = 1)] Byte[] atsBuf, uint bufSize, out uint
requiredSize);
```

C++

```
STDMETHOD(MfRATS)(BYTE *pAtsBuf = NULL,
  DWORD nBufSize = 0, DWORD *pRequiredSize = NULL) PURE;
```

Delphi

```
procedure MfRATS(VAtsBuf: PByte;
  ABufSize: Cardinal; out VRequiredSize: Cardinal); safecall;
```

Параметры:

AtsBuf

[out] Буфер для данных ATS.

BufSize

[in] Размер буфера. Обычно нужно 12 байт.

RequiredSize

[out] Требуемый размер.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.

ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.MfWritePerso

Записывает ключи AES и всех блоков.

C#

```
void MfWritePerso(uint address, ref MfPlusKey rey);
```

C++

```
STDMETHOD(MfWritePerso)(UINT nAddress, const MfPlusKey &rKey)
PURE;
```

Delphi

```
procedure MfWritePerso(AAddress: Cardinal;
  const [Ref] AKey: TMfPlusKey); safecall;
```

Параметры:

Address

[in] Адрес ключа.

Key

[in] Значение ключа.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.MfCommitPerso

Переключает Mifare Plus в SL1 или SL3(если SL1 нет).

C#

```
void MfCommitPerso();
```

C++

```
STDMETHOD(MfCommitPerso)() PURE;
```

Delphi

```
procedure MfCommitPerso(); safecall;
```

Параметры:

Эта функция не имеет параметров.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.WriteMfAuthKeyToReader

Записывает ключи аутентификации Mifare Classic в память считывателя. Считыватели Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I имеют 16 ячеек для хранения ключей А и 16 ячеек - для ключей Б. Авторизовать сектор карты Mifare Classic этими ключами можно с помощью метода AuthMfCardByRdKeys. Прочитать ключи из памяти считывателя нельзя.

C#

```
void WriteMfAuthKeyToReader(int idx,
  [MarshalAs(UnmanagedType.Bool)] bool keyB,
  [In, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 3)]
Int64[] keys,
  int nCount, out int written);
```

C++

```
STDMETHOD(WriteMfAuthKeyToReader)(INT nIdx,
  BOOL fKeyB,
  const MfClassicKey *pKeys,
  INT nCount, INT *pWritten = NULL) PURE;
```

Delphi

```
procedure WriteMfAuthKeyToReader(AIdx: Integer;
  AKeyB: LongBool;
  const AKeys: PMfClassicKey;
  ACount: Integer; VWritten: PInteger = nil); safecall;
```

Параметры:

Idx

[in] Номер ячейки (0..15).

KeyB

[in] True, ключ Б, иначе - ключ А.

Keys

[in] Список записываемых ключей.

Count

[in] Количество ключей, которые нужно записать.

Written

[in] Количество записанных ключей. Если функция завершается успешно, то Written = Count.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Idx < 0) или (Count <= 0) или ((Idx + Count) > 16).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.WriteMfPlusAuthKeyToReader

Записывает ключи аутентификации Mifare Plus в память считывателя. Считыватель Z-2 MF-I имеет 16 ячеек для хранения ключей А и 16 ячеек - для ключей Б. Авторизовать сектор карты Mifare Plus SL3 этими ключами можно с помощью метода AuthMfCardByRdKeys. Прочитать ключи из памяти считывателя нельзя.

C#

```
void WriteMfPlusAuthKeyToReader(int idx,
  [MarshalAs(UnmanagedType.Bool)] bool keyB,
  [In, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 3)]
MfPlusKey[] keys,
  int nCount, out int written);
```

C++

```
STDMETHOD(WriteMfPlusAuthKeyToReader)(INT nIdx,
  BOOL fKeyB,
  const MfPlusKey *pKeys,
  INT nCount, INT *pWritten = NULL) PURE;
```

Delphi

```
procedure WriteMfPlusAuthKeyToReader(AIdx: Integer;
  AKeyB: LongBool;
  const AKeys: PMfPlusKey;
  ACount: Integer; VWritten: PInteger = nil); safecall;
```

Параметры:

Idx

[in] Номер ячейки (0..15).

KeyB

[in] True, ключ Б, иначе - ключ А.

Keys

[in] Список записываемых ключей.

Count

[in] Количество ключей, которые нужно записать.

Written

[in] Количество записанных ключей. Если функция завершается успешно, то Written = Count.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Idx < 0) или (Count <= 0) или ((Idx + Count) > 16).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.LoadTemicPassword

Загружает пароль Temic в память DLL. Пароль используется при автоматическом сканировании карт Temic, и используется методами ReadTemic, WriteTemic, Begin_ReadTemic, Begin_WriteTemic.

C#

```
void LoadTemicPassword(Int64 password);
```

C++

```
STDMETHOD(LoadTemicPassword)(const INT64 nPassword) PURE;
```

Delphi

```
procedure LoadTemicPassword(const APassword: Int64); safecall;
```

Параметры:

Password

[in] Пароль Temic. Если =-1, то нет пароля.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Key < -1) или (Key > 0xffffffff).

Метод IILReader.ScanTemic

Ищет карту Temic в поле считывателя (Z-2 Rd-All, Z-2 EHR).

C#

```
void ScanTemic(int scanParam = -1);
```

C++

```
STDMETHOD(ScanTemic)(INT nScanParam = -1) PURE;
```

Delphi

```
procedure ScanTemic(AScanParam: Integer = -1); safecall;
```

Параметры:

ScanParam

[in] Параметры сканирования Temic. Если равно -1, авто определение.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Z-2 Rd-All, Z-2 EHR.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.EnableAutoScanTemic

Включает/выключает сканирование карт Temic (для Z-2 Rd-All и Z-2 EHR).

C#

```
void EnableAutoScanTemic(
  [MarshalAs(UnmanagedType.Bool)] bool enable = true);
```

C++

```
STDMETHOD(EnableAutoScanTemic)(BOOL fEnable) PURE;
```

Delphi

```
procedure EnableAutoScanTemic(AEnable: LongBool); safecall;
```

Параметры:

Enable

[in] True, включает сканирование Temic.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILReader.GetAutoScanTemicEnabled

Возвращает True если авто сканирование Temic включено.

C#

```
[return: MarshalAs(UnmanagedType.Bool)]
bool GetAutoScanTemicEnabled();
```

C++

```
STDMETHOD(GetAutoScanTemicEnabled)(BOOL* pEnabled) PURE;
```

Delphi

```
function GetAutoScanTemicEnabled(): LongBool; safecall;
```

Параметры:

Enabled

[out] True, автоматический поиск Temic включен.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pEnabled = null (актуально для C++).

Метод IILReader.ReadTemic

Читает данные из карты Temic. Пароль для доступа к карте загружается методом LoadTemicPassword.

C#

```
void ReadTemic(int blockN,
  [Out, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
UInt32[] buf,
  int blockCount, int scanParam /*= -1*/, out int read);
```

C++

```
STDMETHOD(ReadTemic)(INT nBlockIdx,
  DWORD *pBuf,
  INT nBlockCount, INT nScanParam = -1, INT *pRead = NULL) PURE;
```

Delphi

```
procedure ReadTemic(ABlockIdx: Integer;
  VBuf: PCardinal;
  ABlockCount: Integer; AScanParam: Integer = -1;
  VRead: PInteger = nil); safecall;
```

Параметры:

BlockIdx

[in] Номер первого блока, который нужно причитать (0..9).

Buf

[out] Буфер для прочитанных данных.

BlockCount

[in] Количество блоков, которые нужно прочитать.

ScanParam

[in] Параметры сканирования Temic. Если равно -1, авто определение.

Read

[out] Количество прочитанных блоков. Если функция завершается успешно, то Read = BlockCount.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockCount <= 0) или ((BlockIdx + BlockCount) > 10).
E_POINTER	Неправильный указатель. Когда Buf = null.

E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Z-2 Rd-All, Z-2 EHR.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.WriteTemic

Пишет данные в карту Temic. Пароль для доступа к карте загружается методом LoadTemicPassword.

C#

```
void WriteTemic(int blockN,
  [In, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
UInt32[] data,
  int blockCount,
  [MarshalAs(UnmanagedType.Bool)] bool Lock /*= false*/,
  int scanParam /*= -1*/, out int written);
```

C++

```
STDMETHOD(WriteTemic)(INT nBlockIdx,
  const DWORD *pData, INT nBlockCount,
  BOOL fLock = FALSE, INT nScanParam = -1,
  INT *pWritten = NULL) PURE;
```

Delphi

```
procedure WriteTemic(ABlockIdx: Integer;
  const AData: PCardinal; ABlockCount: Integer;
  ALock: LongBool = False; AScanParam: Integer = -1;
  VWritten: PInteger = nil); safecall;
```

Параметры:

BlockIdx

[in] Номер первого блока, в который нужно записать (0..7).

Data

[in] Данные блоков для записи.

BlockCount

[in] Количество блоков, которые нужно записать.

ScanParam

[in] Параметры сканирования Temic. Если равно -1, авто определение.

Written

[out] Количество записанных блоков. Если функция завершается успешно, то Written = BlockCount.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockCount <= 0) или ((BlockIdx + BlockCount) > 8) или (Data = null).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Z-2 Rd-All, Z-2 EHR.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_WRITE_T57	Не удалось записать на Temic. Вероятно, данные заблокированы от перезаписи.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.ResetTemic

Сброс TRES.

C#

```
void ResetTemic(
  [MarshalAs(UnmanagedType.Bool)] bool wait = true);
```

C++

```
STDMETHOD(ResetTemic)(BOOL fWait = TRUE) PURE;
```

Delphi

```
procedure ResetTemic(); safecall;
```

Параметры:

Wait

[in] True, ждать завершения команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Z-2 Rd-All, Z-2 EHR.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_BLOCKING_CALL_NOT_ALLOWED	Блокирующий вызов функции из обработчика не разрешен. Когда эта функция вызвана из функции обратного вызова, установленной методом IILReader.SetNotifyCallback.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReader.EncodeTemicEmMarine

Шифрует данные для эмуляции Em-Marine, для записи в блоки 0..2.

C#

```
void EncodeTemicEmMarine(CardUID uid,
  [In, Out, MarshalAs(UnmanagedType.LPArray, SizeParamIndex =
2)] uint[] buf3,
  int blockCount);
```

C++

```
STDMETHOD(EncodeTemicEmMarine)(const CardUID &rUID,
  DWORD *pBuf3,
  INT nBlockCount) PURE;
```

Delphi

```
procedure EncodeTemicEmMarine(const AUID: TCardUID;
  VBuf3: PCardinal;
  ABlockCount: Integer); safecall;
```

Параметры:

UID

[in] Номер Em-Marine.

Buf3

[out] Буфер для зашифрованных данных.

BlockCount

[in] Размер буфера в блоках. Должен быть не менее 3.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда UID.nLength не равен 5.
E_POINTER	Неправильный указатель. Когда Buf3 = null.
ILR_E_BUFFER_TOO_SMALL	Размер буфера слишком мал. Когда BlockCount меньше 3.

Метод IILReader.DecodeTemicEmMarine

Дешифрует номер Em-Marine из данных блоков 0..2 карты Temic.

C#

```
void DecodeTemicEmMarine([In, MarshalAs(UnmanagedType.LPArray,
SizeParamIndex = 1)] uint[] data3, int blockCount, out CardUID
uid, [MarshalAs(UnmanagedType.Bool)] out bool configOk);
```

C++

```
STDMETHOD(DecodeTemicEmMarine)(const DWORD *pData3, INT
nBlockCount, CardUID *pUID, BOOL *pConfigOk = NULL) PURE;
```

Delphi

```
procedure DecodeTemicEmMarine(const AData3: PCardinal;
  ABlockCount: Integer; out VUID: TCardUID;
  out VConfigOk: Boolean); safecall;
```

Параметры:

Data3

[in] Данные блоков 0..2.

BlockCount

[in] Количество блоков. Должно быть не меньше 3.

UID

[out] Номер Em-Marine. Если Em-Marine не обнаружен, то пустой номер.

ConfigOk

[out] True, конфигурация Temic для эмуляции Em-Marine правильная.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Data3 = null) или (BlockCount < 3).
E_POINTER	Неправильный указатель. Когда UID = null.

Метод IILReader.EncodeTemicHID

Шифрует данные для эмуляции HID, для записи в блоки 0..3.

C#

```
void EncodeTemicHID(CardUID uid, [In, Out,
MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)] uint[]
buf4, int blockCount, int wiegand);
```

C++

```
STDMETHOD(EncodeTemicHID)(const CardUID &rUID,
  DWORD *pBuf4, INT nBlockCount, INT nWiegand) PURE;
```

Delphi

```
procedure EncodeTemicHID(const AUID: TCardUID;
  VBuf4: PCardinal;
  ABlockCount: Integer; AWiegand: Integer); safecall;
```

Параметры:

UID

[in] Номер HID.

Buf4

[out] Буфер для зашифрованных данных.

BlockCount

[in] Размер буфера в блоках. Должен быть не менее 4.

Wiegand

[in] Номер кодировки Wiegand 18..37.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (UID.nLength = 0) или (Wiegand < 18) или (Wiegand > 37).
E_POINTER	Неправильный указатель. Когда Buf4 = null.
ILR_E_BUFFER_TOO_SMALL	Размер буфера слишком мал. Когда BlockCount меньше 4.

Метод IILReader.DecodeTemicHID

Дешифрует номер HID из данных блоков 0..3 карты Temic.

C#

```
void DecodeTemicHID([In, MarshalAs(UnmanagedType.LPArray,
SizeParamIndex = 1)] uint[] data4, int blockCount, out CardUID
uid, out int wiegand, [MarshalAs(UnmanagedType.Bool)] out bool
configOk);
```

C++

```
STDMETHOD(DecodeTemicHID)(const DWORD *pData4,
  INT nBlockCount, CardUID *pUID, INT *pWiegand,
  BOOL *pConfigOk = NULL) PURE;
```

Delphi

```
procedure DecodeTemicHID(const AData4: PCardinal;
  ABlockCount: Integer;
  out VUID: TCardUID; out VWiegand: Integer;
  out VConfigOk: Boolean); safecall;
```

Параметры:

Data4

[in] Данные блоков 0..3.

BlockCount

[in] Количество блоков. Должно быть не меньше 3.

UID

[out] Номер HID. Если HID не обнаружен, то пустой номер.

Wiegand

[out] Номер кодировки Wiegand.

ConfigOk

[out] True, конфигурация Temic для эмуляции HID правильная.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Data4 = null) или (BlockCount < 4).
E_POINTER	Неправильный указатель. Когда UID = null.

Интерфейс IILRAsyncCommand

Позволяет отслеживать прогресс выполнения асинхронной команды, позволяет отменить команду. Интерфейс IILRAsyncCommand можно получить с помощью интерфейсов IILRSearchAsync и IILReaderAsync методами "Begin_...".

Методы:

- **Cancel** – отменяет команду. Устанавливает статус E_ABORT.
- **GetStatus** – возвращает состояние команды.
- **GetProgress** – возвращает состояние прогресса выполнения команды.

Метод IILRAsyncCommand.Cancel

Отменяет команду. Устанавливает статус E_ABORT.

C#

```
void Cancel();
```

C++

```
STDMETHOD(Cancel)() PURE;
```

Delphi

```
procedure Cancel(); safecall;
```

Параметры:
Эта функция не имеет параметров.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

Метод IILRAsyncCommand.GetStatus

Возвращает состояние команды.

C#

```
Int32 GetStatus();
```

C++

```
STDMETHOD(GetStatus)(HRESULT *pStatus) PURE;
```

Delphi

```
function GetStatus(): HResult; safecall;
```

Параметры:

Status

[out] Состояние команды. Если =E_PENDING, команда ещё выполняется, иначе - завершена.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда pStatus = null (актуально для C++).

Метод IILRAsyncCommand.GetProgress

Возвращает прогресс выполнения команды.

C#

```
void GetProgress(out int current, out int total);
```

C++

```
STDMETHOD(GetProgress)(INT *pCurrent, INT *pTotal) PURE;
```

Delphi

```
procedure GetProgress(out VCurrent, VTotal: Integer); safecall;
```

Параметры:

Current

[out] Текущая позиция прогресса.

Total

[out] Максимальная позиция прогресса.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

E_POINTER	Неправильный указатель. Когда (pCurrent = null) или (pTotal = null) (актуально для C++).

Интерфейс IILRSearchAsync

Вызывает функции поиска считывателей в асинхронном режиме. Интерфейс IILRSearchAsync можно получить с помощью IILRSearch методом QueryInterface.

Методы:

- **Begin_Scan** – запускает асинхронную команду поиска считывателей.
- **Begin_EnableAutoScan** – запускает асинхронную команду включения/выключения режима автоматического поиска считывателей.
- **Begin_OpenPort** – запускает асинхронную команду открытия порта.
- **End_OpenPort** – возвращает результат открытия порта.
- **Begin_ClosePort** – запускает асинхронную команду закрытия порта.

Метод IILRSearchAsync.Begin_Scan

Запускает асинхронную команду поиска считывателей.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_Scan(
  [MarshalAs(UnmanagedType.Bool)] bool reset = false);
```

C++

```
STDMETHOD(Begin_Scan)(BOOL fReset, IILRAsyncCommand **ppCmd)
PURE;
```

Delphi

```
function Begin_Scan(AReset: LongBool): IILRAsyncCommand;
safecall;
```

Параметры:

Reset

[in] True, очистить список найденных перед поиском.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILRSearchAsync.Begin_EnableAutoScan

Запускает асинхронную команду включения/выключения режим автоматического поиска считывателей.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_EnableAutoScan(
  [MarshalAs(UnmanagedType.Bool)] bool enable = true);
```

C++

```
STDMETHOD(Begin_EnableAutoScan)(BOOL fEnable, IILRAsyncCommand
**ppCmd) PURE;
```

Delphi

```
function Begin_EnableAutoScan(AEnable: LongBool):
IILRAsyncCommand; safecall;
```

Параметры:

Enable

[in] True, включает поиск в реальном времени, иначе - выключает.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILRSearchAsync.Begin_OpenPort

Запускает асинхронную команду открытия порта.

C#

```
IILGAsyncCommand Begin_OpenPort(PortType portType, string
portName);
```

C++

```
STDMETHOD(Begin_OpenPort)(PortType nPortType, LPCTSTR
pszPortName,
  ILGAsyncCommand** ppCmd) PURE;
```

Delphi

```
function Begin_OpenPort(APortType: TPortType; APortName:
PWideChar
  ): IILGAsyncCommand; safecall;
```

Параметры:

PortType

[in] Тип порта.

PortName

[in] Имя порта.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда PortName = null.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILRSearchAsync.End_OpenPort

Возвращает результат открытия порта.

C#

```
void End_OpenPort(
  [In, MarshalAs(UnmanagedType.Interface)] IILGAsyncCommand cmd,
  out IntPtr hPort, out ReaderInfo info);
```

C++

```
STDMETHOD(End_OpenPort)(IILGAsyncCommand* pCmd, HANDLE* pPort,
  ReaderInfo* pInfo) PURE;
```

Delphi

```
procedure End_OpenPort(ACmd: IILGAsyncCommand; out VPort:
THandle;
  out VInfo: TReaderInfo); safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_OpenPort.

Port

[out] Дескриптор порта.

Info

[out] Информация о считывателе (если известно).

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Cmd = null) или (PortHandle = null).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILRSearchAsync.Begin_ClosePort

Запускает асинхронную команду закрытия порта.

C#

```
IILGAsyncCommand Begin_ClosePort(PortType portType, string
portName,
  IntPtr hPort);
```

C++

```
STDMETHOD(Begin_ClosePort)(PortType nPortType, LPCTSTR
pszPortName,
  HANDLE hPort, IILGAsyncCommand** ppCmd) PURE;
```

Delphi

```
function Begin_ClosePort(APortType: TPortType; APortName:
PWideChar;
  APort: THandle): IILGAsyncCommand; safecall;
```

Параметры:

PortType

[in] Тип порта.

PortName

[in] Имя порта.

Port

[in] Дескриптор порта.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда PortType = ptUnknownPort.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Интерфейс IILReaderAsync

Вызывает функции работы со считывателем в асинхронном режиме. Интерфейс IILReaderAsync можно получить с помощью IILReader методом QueryInterface.

Методы:

- **Begin_Connect** – запускает асинхронную команду подключения к считывателю.
- **Begin_Disconnect** – запускает асинхронную команду отключения от считывателя.
- **Begin_Scan** – запускает асинхронную команду поиска карты.
- **Begin_EnableAutoScan** – запускает асинхронную команду включения/выключения автоматического сканирования карт.
- **Для Mifare Ultralight:**
  - **Begin_ReadMfUlralight** – запускает асинхронную команду чтения данных из карты Mifare Ultralight.
  - **End_ReadMfUlralight** – возвращает результат чтения данных из карты Mifare Ultralight.
  - **Begin_WriteMfUlralight** – запускает асинхронную команду записи данных в карту Mifare Ultralight.
  - **End_WriteMfUlralight** – возвращает результат записи данных в карту Mifare Ultralight.
- **Для Mifare Classic/Plus:**
  - **Begin_AuthMfCard** – запускает асинхронную команду авторизации сектора карты Mifare Classic / Plus используя ключ, загруженный методом LoadMfAuthKey / LoadMfPlusAuthKey.
  - **End_AuthMfCard** – возвращает результат авторизации сектора карты.
  - **Begin_AuthMfCardByRdKeys** – запускает асинхронную команду авторизации сектора карты Mifare Classic / Plus используя ключи в памяти считывателя.
  - **End_AuthMfCardByRdKeys** – возвращает результат авторизации сектора карты.
  - **Begin_ReadMfClassic** – запускает асинхронную команду чтения данных из карты Mifare Classic или Mifare Plus SL1.
  - **End_ReadMfClassic** – возвращает результат чтения данных из карты Mifare Classic или Mifare Plus SL1.
  - **Begin_WriteMfClassic** – запускает асинхронную команду записи данных в карту Mifare Classic или Mifare Plus SL1.
  - **End_WriteMfClassic** – возвращает результат записи данных в карту Mifare Classic или Mifare Plus SL1.

- Begin_ReadMfPlus – запускает асинхронную команду чтения данных из карты Mifare Plus SL3.
- End_ReadMfPlus – возвращает результат чтения данных из карты Mifare Plus SL3.
- Begin_WriteMfPlus – запускает асинхронную команду записи данных в карту Mifare Plus SL3.
- End_WriteMfPlus – возвращает результат записи данных в карту Mifare Plus SL3.
- Begin_MfIncrement – увеличивает содержимое блока-значения карты Mifare и сохраняет результат во временном регистре данных.
- Begin_MfDecrement – уменьшает содержимое блока-значения карты Mifare и сохраняет результат во временном регистре данных.
- Begin_MfTransfer – записывает содержимое во временном регистре данных в блок-значение.
- Begin_MfRestore – перемещает содержимое блока в регистр данных Mifare.
- Begin_WriteMfAuthKeyToReader – запускает асинхронную команду записи ключей аутентификации Mifare Classic в память считывателя.
- End_WriteMfAuthKeyToReader – возвращает результат записи ключей аутентификации Mifare Classic в память считывателя.
- Begin_WriteMfPlusAuthKeyToReader – запускает асинхронную команду записи ключей аутентификации Mifare Plus в память считывателя.
- End_WriteMfPlusAuthKeyToReader – возвращает результат записи ключей аутентификации Mifare Plus в память считывателя.

- **Для Temic:**
  - Begin_ScanTemic – запускает асинхронную команду поиска карты Temic в поле считывателя.
  - Begin_ReadTemic – запускает асинхронную команду чтения данных из карты Temic.
  - End_ReadTemic – возвращает результат чтения данных из карты Temic.
  - Begin_WriteTemic – запускает асинхронную команду записи данных в карту Temic.
  - End_WriteTemic – возвращает результат записи данных в карту Temic.
  - Begin_ResetTemic – запускает асинхронную команду сброса TRES.

Метод IILReaderAsync.Begin_Connect

Запускает асинхронную команду подключения к считывателю.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_Connect(
  [MarshalAs(UnmanagedType.Bool)] bool reconnect = false);
```

C++

```
STDMETHOD(Begin_Connect)(BOOL fReconnect /*= FALSE*/,
  IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_Connect(AReconnect: LongBool = False
  ): IILRAsyncCommand; safecall;
```

Параметры:

Reconnect

[in] True, переподключиться.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.Begin_Disconnect

Запускает асинхронную команду отключения от считывателя.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_Disconnect();
```

C++

```
STDMETHOD(Begin_Disconnect)(IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_Disconnect(): IILRAsyncCommand; safecall;
```

Параметры:

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.Begin_Scan

Запускает асинхронную команду поиска карты.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_Scan(
  [MarshalAs(UnmanagedType.Bool)] bool reset = false,
  [MarshalAs(UnmanagedType.Bool)] bool powerOff = true);
```

C++

```
STDMETHOD(Begin_Scan)(
  BOOL fReset /*= FALSE*/,
  BOOL fPowerOff, IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_Scan(
  AReset: LongBool = False;
  APowerOff: LongBool = True): IILRAsyncCommand; safecall;
```

Параметры:

Reset

[in] True, сбросить старые результаты поиска.

PowerOff

[in] True, выключает RF поле после сканирования.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.Begin_EnableAutoScan

Запускает асинхронную команду включения/выключения автоматического сканирования карт.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_EnableAutoScan(
  [MarshalAs(UnmanagedType.Bool)] bool enable = true);
```

C++

```
STDMETHOD(Begin_EnableAutoScan)(BOOL fEnable /*= TRUE*/,
IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_EnableAutoScan(AEnable: LongBool):
IILRAsyncCommand; safecall;
```

Параметры:

Enable

[in] True, включает сканирование карт.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.Begin_ReadMfUlralight

Запускает асинхронную команду чтения данных из карты Mifare Ultralight.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_ReadMfUlralight(int pageIdx, int
pageCount);
```

C++

```
STDMETHOD(Begin_ReadMfUlralight)(INT nPageIdx, INT nPageCount,
IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_ReadMfUlralight(APageIdx, APageCount: Integer):
IILRAsyncCommand; safecall;
```

Параметры:

PageIdx

[in] Номер первой читаемой страницы (0..15).

PageCount

[in] Количиство страниц, которые нужно прочитать.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.
E_INVALIDARG	Неправильные параметры. Когда (PageIdx < 0) или (PageCount <= 0) или ((PageIdx + PageCount) > 16).

Метод IILReaderAsync.End_ReadMfUlralight

Возвращает результат чтения данных из карты Mifare Ultralight.

C#

```
void End_ReadMfUlralight(
  [In, MarshalAs(UnmanagedType.Interface)] IILRAsyncCommand cmd,
  [Out, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
UInt32[] buf,
  int bufSize, out int read);
```

C++

```
STDMETHOD(End_ReadMfUlralight)(IILRAsyncCommand *pCmd,
  DWORD *pBuf, INT nBufSize, INT *pRead) PURE;
```

Delphi

```
procedure End_ReadMfUlralight(ACmd: IILRAsyncCommand; VBuf:
PCardinal;
  ABufSize: Integer; out VRead: Integer); safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_ReadMfUlralight.

Buf

[out] Буфер для прочитанных страниц.

BufSize

[in] Размер буфера в страницах.

Read

[out] Количество прочитанных страниц.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Cmd = null) или (BufSize < 0).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).

Метод IILReaderAsync.Begin_WriteMfUlralight

Запускает асинхронную команду записи данных в карту Mifare Ultralight.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_WriteMfUlralight(int pageIdx,
  [In, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
UInt32[] data,
  int pageCount);
```

C++

```
STDMETHOD(Begin_WriteMfUlralight)(INT nPageIdx,
  const DWORD *pData, INT nPageCount, IILRAsyncCommand **ppCmd)
PURE;
```

Delphi

```
function Begin_WriteMfUlralight(APageIdx: Integer;
  const AData: PCardinal; APageCount: Integer):
IILRAsyncCommand; safecall;
```

Параметры:

PageIdx

[in] Номер первой записываемой страницы (начиная от 0).

Data

[in] Данные страниц.

PageCount

[in] Количество записываемых страниц.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (PageIdx < 0) или (PageCount <= 0) или ((PageIdx + PageCount) > 16).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.End_WriteMfUlralight

Возвращает результат записи данных в карту Mifare Ultralight.

C#

```
void End_WriteMfUlralight(
  [In, MarshalAs(UnmanagedType.Interface)] IILRAsyncCommand cmd,
  out int written);
```

C++

```
STDMETHOD(End_WriteMfUlralight)(IILRAsyncCommand *pCmd, INT
*pWritten) PURE;
```

Delphi

```
procedure End_WriteMfUlralight(ACmd: IILRAsyncCommand;
```

```
  out VWritten: Integer); safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_WriteMfUlralight.

Written

[out] Количество записанных страниц.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Cmd = null.

Метод IILReaderAsync.Begin_AuthMfCard

Запускает асинхронную команду авторизации сектора карты Mifare Classic / Plus используя ключ, загруженный функцией LoadMfAuthKey / LoadMfPlusAuthKey.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_AuthMfCard(uint address,
  [MarshalAs(UnmanagedType.Bool)] bool keyB);
```

C++

```
STDMETHOD(Begin_AuthMfCard)(UINT nAddress,
  BOOL fKeyB, IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_AuthMfCard(AAddress: Cardinal;
  AKeyB: LongBool): IILRAsyncCommand; safecall;
```

Параметры:

Address

[in] Номер блока (0..255) или адрес Mifare Plus.

KeyB

[in] True, авторизовать по ключу Б, иначе - по ключу А.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Address > 0xffff.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.End_AuthMfCard

Возвращает результат авторизации сектора карты.

C#

```
void End_AuthMfCard(
  [In, MarshalAs(UnmanagedType.Interface)] IILRAsyncCommand cmd,
  [MarshalAs(UnmanagedType.Bool)] out bool authOk);
```

C++

```
STDMETHOD(End_AuthMfCard)(IILRAsyncCommand *pCmd, BOOL *pAuthOk)
PURE;
```

Delphi

```
procedure End_AuthMfCard(ACmd: IILRAsyncCommand;
  out VAuthOk: LongBool); safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_AuthMfCard.

AuthOk

[out] True, сектор успешно авторизован.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Cmd = null.
E_POINTER	Неправильный указатель. Когда AuthOk = null (актуально для C++).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I, Z-2 MF CCID.
E_ABORT	Операция прервана.
E_BOUNDS	Номер блока вне диапазона блоков карты.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.
ILR_E_SCARD_ERROR	Ошибка функции SmartCards (актуально для CCID считывателя).

Метод IILReaderAsync.Begin_AuthMfCardByRdKeys

Запускает асинхронную команду авторизации сектора карты Mifare Classic / Plus используя ключи в памяти считывателя.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_AuthMfCardByRdKeys(uint address,
  [MarshalAs(UnmanagedType.Bool)] bool keyB,
  uint rdKeys = 0xffff);
```

C++

```
STDMETHOD(Begin_AuthMfCardByRdKeys)(UINT nAddress,
  BOOL fKeyB, DWORD nRdKeys /*= 0xFFFF*/,
  IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_AuthMfCardByRdKeys(AAddress: Cardinal;
  AKeyB: LongBool;
  ARdKeys: Cardinal = $ffff): IILRAsyncCommand; safecall;
```

Параметры:

Address

[in] Номер блока (0..255) или адрес Mifare Plus.

KeyB

[in] True, авторизовать по ключу Б, иначе - по ключу А.

RdKeys

[in] Биты (0..15) ключей в памяти считывателя.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Address > 0xffff.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.End_AuthMfCardByRdKeys

Возвращает результат авторизации сектора карты.

C#

```
void End_AuthMfCardByRdKeys(
  [In, MarshalAs(UnmanagedType.Interface)] IILRAsyncCommand cmd,
  out int keyIdx);
```

C++

```
STDMETHOD(End_AuthMfCardByRdKeys)(IILRAsyncCommand *pCmd, INT
*pKeyIdx) PURE;
```

Delphi

```
procedure End_AuthMfCardByRdKeys(ACmd: IILRAsyncCommand; out
VKeyIdx: Integer); safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_AuthMfCardByRdKeys.

KeyIdx

[out] Позиция найденного ключа в памяти считывателя. Если =-1, ключ не найден.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Cmd = null.
E_POINTER	Неправильный указатель. Когда KeyIdx = null (актуально для C++).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I, Z-2 MF CCID.
E_ABORT	Операция прервана.
E_BOUNDS	Номер блока вне диапазона блоков карты.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReaderAsync.Begin_ReadMfClassic

Запускает асинхронную команду чтения данных из карты Mifare Classic или Mifare Plus SL1.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_ReadMfClassic(int blockIdx, int
blockCount);
```

C++

```
STDMETHOD(Begin_ReadMfClassic)(INT nBlockIdx, INT nBlockCount,
IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_ReadMfClassic(ABlockIdx, ABlockCount: Integer):
IILRAsyncCommand; safecall;
```

Параметры:

BlockIdx

[in] Номер первого читаемого блока (0..255).

BlockCount

[in] Количество блоков, которые нужно прочитать.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockIdx <= 0) или ((BlockIdx + BlockCount) > 0xff).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.End_ReadMfClassic

Возвращает результат чтения данных из карты Mifare Classic или Mifare Plus SL1.

C#

```
void End_ReadMfClassic(
  [In, MarshalAs(UnmanagedType.Interface)] IILRAsyncCommand cmd,
  [Out, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
MfBlockData[] buf,
  int bufSize, out int read);
```

C++

```
STDMETHOD(End_ReadMfClassic)(IILRAsyncCommand *pCmd,
  MfBlockData *pBuf, INT nBufSize, INT *pRead) PURE;
```

Delphi

```
procedure End_ReadMfClassic(ACmd: IILRAsyncCommand;
  VBuf: PMfBlockData; ABufSize: Integer; out VRead: Integer);
safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_ReadMfClassic.

Buf

[out] Буфер для прочитанных блоков.

BufSize

[in] Размер буфера в блоках.

Read

[out] Количество прочитанных блоков.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Cmd = null) или (BufSize < 0).
E_POINTER	Неправильный указатель. Когда (Buf = null) или (Read = null).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I, Z-2 MF CCID.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.
ILR_E_SCARD_ERROR	Ошибка функции SmartCards (актуально для CCID считывателя).

Метод IILReaderAsync.Begin_WriteMfClassic

Запускает асинхронную команду записи данных в карту Mifare Classic или Mifare Plus SL1.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_WriteMfClassic(int blockIdx,
  [In, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
MfBlockData[] data,
  int blockCount);
```

C++

```
STDMETHOD(Begin_WriteMfClassic)(INT nBlockIdx,
  const MfBlockData *pData, INT nBlockCount, IILRAsyncCommand
**ppCmd) PURE;
```

Delphi

```
function Begin_WriteMfClassic(ABlockIdx: Integer;
  const AData: PMfBlockData; ABlockCount: Integer):
IILRAsyncCommand; safecall;
```

Параметры:

BlockIdx

[in] Номер первого записываемого блока (0..255).

Data

[in] Данные записываемых блоков.

BlockCount

[in] Количество блоков, которые нужно записать.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockCount <= 0) или ((BlockIdx + BlockCount) > 0xff) или (Data = null).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.End_WriteMfClassic

Возвращает результат записи данных в карту Mifare Classic или Mifare Plus SL1.

C#

```
void End_WriteMfClassic(
  [In, MarshalAs(UnmanagedType.Interface)] IILRAsyncCommand cmd,
  out int written);
```

C++

```
STDMETHOD(End_WriteMfClassic)(IILRAsyncCommand *pCmd, INT
*pWritten) PURE;
```

Delphi

```
procedure End_WriteMfClassic(ACmd: IILRAsyncCommand;
  out VWritten: Integer); safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_WriteMfClassic.

Written

[out] Количество записанных блоков.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Cmd = null.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I, Z-2 MF CCID.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.
ILR_E_SCARD_ERROR	Ошибка функции SmartCards (актуально для CCID считывателя).

Метод IILReaderAsync.Begin_ReadMfPlus

Запускает асинхронную команду чтения данных из карты Mifare Plus SL3.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_ReadMfPlus(uint address, int blockCount,
  [MarshalAs(UnmanagedType.Bool)] bool openText);
```

C++

```
STDMETHOD(Begin_ReadMfPlus)(UINT nAddress, INT nBlockCount,
  BOOL fOpenText, IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_ReadMfPlus(AAddress: Cardinal; ABlockCount:
Integer;
  AOpenText: LongBool): IILRAsyncCommand; safecall;
```

Параметры:

Address

[in] Номер первого читаемого блока (0..255).

BlockCount

[in] Количество блоков, которые нужно прочитать.

OpenText

[in] True, открытая передача, иначе - зашифрованная.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Address > 0xffff) или (BlockCount <= 0).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.End_ReadMfPlus

Возвращает результат чтения данных из карты Mifare Plus SL3.

C#

```
void End_ReadMfPlus([In, MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand cmd, [Out, MarshalAs(UnmanagedType.LPArray,
SizeParamIndex = 2)] MfBlockData[] buf, int bufSize, out int
read);
```

C++

```
STDMETHOD(End_ReadMfPlus)(IILRAsyncCommand *pCmd,
  MfBlockData *pBuf, INT nBufSize, INT *pRead) PURE;
```

Delphi

```
procedure End_ReadMfPlus(ACmd: IILRAsyncCommand;
  VBuf: PMfBlockData; ABufSize: Integer; out VRead: Integer);
safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_ReadMfPlus.

Buf

[out] Буфер для прочитанных блоков.

BufSize

[in] Размер буфера в блоках.

Read

[out] Количество прочитанных блоков.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Cmd = null) или (BufSize < 0).
E_POINTER	Неправильный указатель. Когда (Buf = null) или (Read = null).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReaderAsync.Begin_WriteMfPlus

Запускает асинхронную команду записи данных в карту Mifare Plus SL3.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_WriteMfPlus(uint address,
  [In, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
MfBlockData[] data,
  int blockCount, [MarshalAs(UnmanagedType.Bool)] bool
openText);
```

C++

```
STDMETHOD(Begin_WriteMfPlus)(UINT nAddress,
  const MfBlockData *pData, INT nBlockCount,
  BOOL fOpenText, IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_WriteMfPlus(AAddress: Cardinal;
  const AData: PMfBlockData; ABlockCount: Integer;
  AOpenText: LongBool): IILRAsyncCommand; safecall;
```

Параметры:

Address

[in] Номер первого записываемого блока (0..255) или адрес Mifare Plus.

Data

[in] Данные записываемых блоков.

BlockCount

[in] Количество блоков, которые нужно записать.

OpenText

[in] True, открытая передача, иначе - зашифрованная.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Address > 0xffff) или (BlockCount <= 0) или (Data = null).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.End_WriteMfPlus

Возвращает результат записи данных в карту Mifare Plus SL3.

C#

```
void End_WriteMfPlus(
  [In, MarshalAs(UnmanagedType.Interface)] IILRAsyncCommand cmd,
  out int written);
```

C++

```
STDMETHOD(End_WriteMfPlus)(IILRAsyncCommand *pCmd, INT
*pWritten) PURE;
```

Delphi

```
procedure End_WriteMfPlus(ACmd: IILRAsyncCommand;
  out VWritten: Integer); safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_WriteMfPlus.

Written

[out] Количество записанных блоков.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Cmd = null.
E_POINTER	Неправильный указатель. Когда Written = null.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_CARD_AUTHORIZE	Ошибка авторизации карты.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReaderAsync.Begin_MfIncrement

Увеличивает содержимое блока-значения карты Mifare и сохраняет результат во временном регистре данных.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_MfIncrement(int blockIdx, uint value);
```

C++

```
STDMETHOD(Begin_MfIncrement)(INT nBlockIdx, DWORD nValue,
  IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_MfIncrement(ABlockIdx: Integer;
  AValue: Cardinal): IILRAsyncCommand; safecall;
```

Параметры:

BlockIdx

[in] Номер блока (0..255).

Value

[in] Величина инкремента.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockIdx > 0xff).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.Begin_MfDecrement

Уменьшает содержимое блока-значения карты Mifare и сохраняет результат во временном регистре данных.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_MfDecrement(int blockIdx, uint value);
```

C++

```
STDMETHOD(Begin_MfDecrement)(INT nBlockIdx, DWORD nValue,
  IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_MfDecrement(ABlockIdx: Integer;
  AValue: Cardinal): IILRAsyncCommand; safecall;
```

Параметры:

BlockIdx

[in] Номер блока (0..255).

Value

[in] Величина декремента.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockIdx > 0xff).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.Begin_MfTransfer

Записывает содержимое во временном регистре данных в блок-значение.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_MfTransfer(int blockIdx);
```

C++

```
STDMETHOD(Begin_MfTransfer)(INT nBlockIdx, IILRAsyncCommand
**ppCmd) PURE;
```

Delphi

```
function Begin_MfTransfer(ABlockIdx: Integer): IILRAsyncCommand;
safecall;
```

Параметры:

BlockIdx

[in] Номер блока (0..255).

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockIdx > 0xff).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.Begin_MfRestore

Перемещает содержимое блока в регистр данных Mifare.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_MfRestore(int blockIdx);
```

C++

```
STDMETHOD(Begin_MfRestore)(INT nBlockIdx, IILRAsyncCommand
**ppCmd) PURE;
```

Delphi

```
function Begin_MfRestore(ABlockIdx: Integer): IILRAsyncCommand;
safecall;
```

Параметры:

BlockIdx

[in] Номер блока (0..255).

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockIdx > 0xff).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.Begin_WriteMfAuthKeyToReader

Запускает асинхронную команду записи ключей аутентификации Mifare Classic в память считывателя.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_WriteMfAuthKeyToReader(int idx,
  [MarshalAs(UnmanagedType.Bool)] bool keyB,
  [In, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 3)]
Int64[] keys,
  int count);
```

C++

```
STDMETHOD(Begin_WriteMfAuthKeyToReader)(INT nIdx, BOOL fKeyB,
  const MfClassicKey *pKeys, INT nCount, IILRAsyncCommand
**ppCmd) PURE;
```

Delphi

```
function Begin_WriteMfAuthKeyToReader(AIdx: Integer; AKeyB:
LongBool;
  const AKeys: PMfClassicKey; ACount: Integer):
IILRAsyncCommand; safecall;
```

Параметры:

Idx

[in] Номер ячейки (0..15).

KeyB

[in] True, ключ Б, иначе - ключ А.

Keys

[in] Список записываемых ключей.

Count

[in] Количество ключей, которые нужно записать.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.

E_INVALIDARG	Неправильные параметры. Когда (Idx < 0) или (Count <= 0) или ((Idx + Count) > 16) или (Keys = null).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.End_WriteMfAuthKeyToReader

Возвращает результат записи ключей аутентификации Mifare Classic в память считывателя.

C#

```
void End_WriteMfAuthKeyToReader(
  [In, MarshalAs(UnmanagedType.Interface)] IILRAsyncCommand cmd,
  out int written);
```

C++

```
STDMETHOD(End_WriteMfAuthKeyToReader)(IILRAsyncCommand *pCmd,
INT *pWritten) PURE;
```

Delphi

```
procedure End_WriteMfAuthKeyToReader(ACmd: IILRAsyncCommand;
  out VWritten: Integer); safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_WriteMfAuthKeyToReader.

Written

[out] Количество записанных ключей.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Cmd = null.
E_POINTER	Неправильный указатель. Когда Written = null.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Matrix III Net, CP-Z2-MF, Z-2 USB MF, Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReaderAsync.Begin_WriteMfPlusAuthKeyToReader

Запускает асинхронную команду записи ключей аутентификации Mifare Plus в память считывателя.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_WriteMfPlusAuthKeyToReader(int idx,
  [MarshalAs(UnmanagedType.Bool)] bool keyB,
  [In, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 3)]
MfPlusKey[] keys,
  int count);
```

C++

```
STDMETHOD(Begin_WriteMfPlusAuthKeyToReader)(INT nIdx, BOOL
fKeyB,
  const MfPlusKey *pKeys, INT nCount, IILRAsyncCommand **ppCmd)
PURE;
```

Delphi

```
function Begin_WriteMfPlusAuthKeyToReader(AIdx: Integer; AKeyB:
LongBool;
  const AKeys: PMfPlusKey; ACount: Integer): IILRAsyncCommand;
safecall;
```

Параметры:

Idx

[in] Номер ячейки (0..15).

KeyB

[in] True, ключ Б, иначе - ключ А.

Keys

[in] Список записываемых ключей.

Count

[in] Количество ключей, которые нужно записать.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Idx < 0) или (Count <= 0) или ((Idx + Count) > 16) или (Keys = null).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.End_WriteMfPlusAuthKeyToReader

Возвращает результат записи ключей аутентификации Mifare Plus в память считывателя.

C#

```
void End_WriteMfPlusAuthKeyToReader(
  [In, MarshalAs(UnmanagedType.Interface)] IILRAsyncCommand cmd,
  out int written);
```

C++

```
STDMETHOD(End_WriteMfPlusAuthKeyToReader)(IILRAsyncCommand
*pCmd, INT *pWritten) PURE;
```

Delphi

```
procedure End_WriteMfPlusAuthKeyToReader(ACmd: IILRAsyncCommand;
  out VWritten: Integer); safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_WriteMfPlusAuthKeyToReader.

Written

[out] Количество записанных ключей.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Cmd = null.
E_POINTER	Неправильный указатель. Когда Written = null.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не Z-2 MF-I.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReaderAsync.Begin_ScanTemic

Запускает асинхронную команду поиска карты Temic в поле считывателя (Z-2 Rd-All, Z-2 EHR).

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_ScanTemic(int scanParam = -1);
```

C++

```
STDMETHOD(Begin_ScanTemic)(INT nScanParam /*= -1*/,
IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_ScanTemic(AScanParam: Integer = -1):
IILRAsyncCommand; safecall;
```

Параметры:

ScanParam

[in] Параметры сканирования Temic. Если равно -1, авто определение.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.Begin_ReadTemic

Запускает асинхронную команду чтения данных из карты Temic. Пароль для доступа к карте загружается методом LoadTemicPassword интерфейса IILReader.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_ReadTemic(int blockIdx,
  int blockCount, int scanParam = -1);
```

C++

```
STDMETHOD(Begin_ReadTemic)(INT nBlockIdx, INT nBlockCount,
```

```
  INT nScanParam, IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_ReadTemic(ABlockIdx, ABlockCount: Integer;
  AScanParam: Integer = -1): IILRAsyncCommand; safecall;
```

Параметры:

BlockIdx

[in] Номер первого блока, который нужно причитать (0..9).

BlockCount

[in] Количество блоков, которые нужно прочитать.

ScanParam

[in] Параметры сканирования Temic. Если равно -1, авто определение.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockIdx <= 0) или ((BlockIdx + BlockCount) > 10).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.End_ReadTemic

Возвращает результат чтения данных из карты Temic.

C#

```
void End_ReadTemic(
  [In, MarshalAs(UnmanagedType.Interface)] IILRAsyncCommand cmd,
  [Out, MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)]
uint[] buf,
  int blockSize, out int read);
```

C++

```
STDMETHOD(End_ReadTemic)(IILRAsyncCommand *pCmd,
  DWORD *pBuf, INT nBufSize, INT *pRead) PURE;
```

Delphi

```
procedure End_ReadTemic(ACmd: IILRAsyncCommand; VBuf: PCardinal;
ABufSize: Integer; out VRead: Integer); safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_ReadTemic.

Buf

[out] Буфер для прочитанных блоков.

BufSize

[in] Размер буфера в блоках.

Read

[out] Количество прочитанных блоков.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (Cmd = null) или (BufSize < 0).
E_POINTER	Неправильный указатель. Когда (Buf = null) или (Read = null).
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Z-2 Rd-All, Z-2 EHR.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReaderAsync.Begin_WriteTemic

Запускает асинхронную команду записи данных в карту Temic. Пароль для доступа к карте загружается методом LoadTemicPassword интерфейса IILReader.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_WriteTemic(int blockIdx, [In,
MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 2)] uint[]
data, int blockCount, [MarshalAs(UnmanagedType.Bool)] bool Lock
= false, int scanParam = -1);
```

C++

```
STDMETHOD(Begin_WriteTemic)(INT nBlockIdx, const DWORD *pData,
INT nBlockCount,
  BOOL fLock, INT nScanParam, IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_WriteTemic(ABlockIdx: Integer; const AData:
PCardinal;
  ABlockCount: Integer; ALock: LongBool = False;
  AScanParam: Integer = -1): IILRAsyncCommand; safecall;
```

Параметры:

BlockIdx

[in] Номер первого блока, в который нужно записать (0..7).

Data

[in] Данные блоков для записи.

BlockCount

[in] Количество блоков, которые нужно записать.

ScanParam

[in] Параметры сканирования Temic. Если равно -1, авто определение.

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда (BlockIdx < 0) или (BlockCount <= 0) или ((BlockIdx + BlockCount) > 8) или (Data = null).
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Метод IILReaderAsync.End_WriteTemic

Возвращает результат записи данных в карту Temic.

C#

```
void End_WriteTemic(
  [In, MarshalAs(UnmanagedType.Interface)] IILRAsyncCommand cmd,
  out int written);
```

C++

```
STDMETHOD(End_WriteTemic)(IILRAsyncCommand *pCmd, INT *pWritten)
PURE;
```

Delphi

```
procedure End_WriteTemic(ACmd: IILRAsyncCommand;
  out VWritten: Integer); safecall;
```

Параметры:

Cmd

[in] Команда, которую вернул метод Begin_WriteTemic.

Written

[out] Количество записанных блоков.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_INVALIDARG	Неправильные параметры. Когда Cmd = null.
E_NOTIMPL	Команда не поддерживается считывателем. Когда модель считывателя не одна из: Z-2 Rd-All, Z-2 EHR.
E_ABORT	Операция прервана.
E_OUTOFMEMORY	Недостаточно памяти.
ILR_E_NO_CARD	Нет карты в поле считывателя.
ILR_E_WRITE_T57	Не удалось записать на Temic. Вероятно, данные заблокированы от перезаписи.
ILR_E_READER_ERROR	Неизвестная ошибка считывателя.
ILR_E_BAD_RESPONSE	Не распознан ответ считывателя.
ILR_E_REQUEST_TIMEOUT	Тайм-аут запроса к считывателю.

Метод IILReaderAsync.Begin_ResetTemic

Запускает асинхронную команду сброса TRES.

C#

```
[return: MarshalAs(UnmanagedType.Interface)]
IILRAsyncCommand Begin_ResetTemic();
```

C++

```
STDMETHOD(Begin_ResetTemic)(IILRAsyncCommand **ppCmd) PURE;
```

Delphi

```
function Begin_ResetTemic(): IILRAsyncCommand; safecall;
```

Параметры:

Cmd

[out] Интерфейс команды.

Возвращаемое значение:

S_OK	Функция выполнена успешно.
E_POINTER	Неправильный указатель. Когда ppCmd = null (актуально для C++).
E_OUTOFMEMORY	Недостаточно памяти.

Карты

- Mifare Ultralight
- Mifare Classic 4K
- Mifare Plus
- Temic (T5557, T5577)

Mifare Ultralight

Phillips разработал карты Mifare Ultralight для использования со считывателями стандарта ISO/IEC14443A. Радиоинтерфейс (MIFARE RF) соответствует частям 2 и 3 стандарта ISO/IEC14443A. В основном Mifare Ultralight разработан для применения в сфере транспортных услуг в качестве бесконтактных билетов на одну поездку.

Основные возможности

1. Радиочастотный интерфейс Mifare (ISO/IEC 14443A)
    - Бесконтактная передача данных и питание по радиоканалу (не требует батарей)
    - Рабочее расстояние до 100 мм (зависит от геометрических параметров антенны)
    - Рабочая частота: 13,56 МГц
    - Быстрая передача данных: 106 Кбит/сек
    - Высокая надежность передачи (16-битовая CRC, проверка на четность...)
    - Настоящая антиколлизия (поддержка нескольких карт в поле одновременно)
    - 7-байтовый уникальный серийный номер
    - Время типовой транзакции менее 35 мс
    - Быстрая транзакция счетчика: менее 10 мс
2. Память
    - Размер 512 бит, организована в виде 16 страниц, по 4 байта каждая
    - Возможность программирования постраничной блокировки записи
    - 32-битовая область "одноразового" программирования (для нужд пользователя) - OTP (One Time Programmable)
    - 384-битовая (48 байт) область чтения-записи (12 страниц)
    - Срок хранения данных - 5 лет
    - 10 000 циклов записи
3. Защита
    - Уникальный 7-байтовый серийный номер каждой карты
    - 32-битовая область OTP
    - Функция блокировки записи отдельных страниц

Уникальный 7-байтовый UID однократно программируется в каждую карту в процессе производства и не может быть изменен в последствии, что является эффективной защитой от клонирования. Он может быть использован для организации криптозащиты хранимых данных.

32-битовая область OTP (одноразового программирования) обеспечивает возможность однократной записи (т.е. данные, записанные в нее, не могут быть изменены впоследствии).

Поле программируемой функции запрета записи соответствующих страниц. Эта функция позволяет уникально запрограммировать карту для специализированного применения.

Организация памяти

512 бит перепрограммируемой памяти организовано в виде 16 страниц по 4 байта каждая.

80 бит зарезервировано под данные изготовителя.

16 бит предназначено для механизма блокировки, делающий доступ только для чтения.

32 бита доступны как OTP область.

384 бита используются для программирования памяти для чтения/записи.

В стертом состоянии ячейки читаются, как логический ноль, в записанном - как 1.

Byte Number	0	1	2	3	Page
Serial Number	SN0	SN1	SN2	BCC0	0
Serial Number	SN3	SN4	SN5	SN6	1
Internal / Lock	BCC1	Internal	Lock0	Lock1	2
OTP	OTP0	OTP1	OTP2	OTP3	3
Data read/write	Data0	Data1	Data2	Data3	4
Data read/write	Data4	Data5	Data6	Data7	5
Data read/write	Data8	Data9	Data10	Data11	6
Data read/write	Data12	Data13	Data14	Data15	7
Data read/write	Data16	Data17	Data18	Data19	8
Data read/write	Data20	Data21	Data22	Data23	9
Data read/write	Data24	Data25	Data26	Data27	10
Data read/write	Data28	Data29	Data30	Data31	11
Data read/write	Data32	Data33	Data34	Data35	12
Data read/write	Data36	Data37	Data38	Data39	13
Data read/write	Data40	Data41	Data42	Data43	14
Data read/write	Data44	Data45	Data46	Data47	15

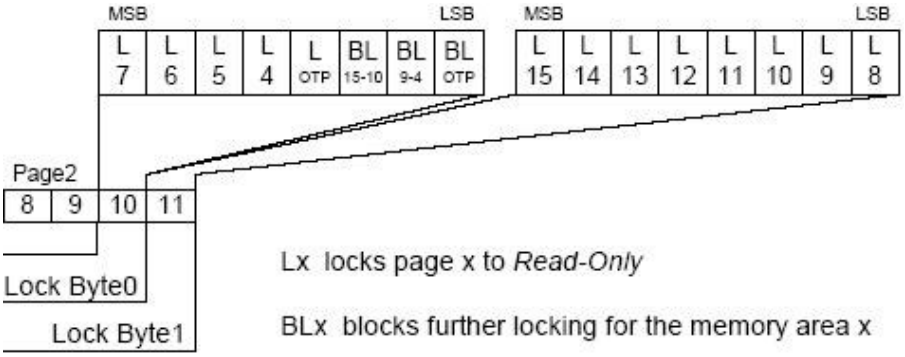
Жирная рамка указывает на пользовательскую память.

1. UID/Серийный номер (SN0, SN1, SN2, SN3, SN4, SN5, SN6)

Уникальный 7-байтовый серийный номер и два байта контрольной суммы запрограммированы в первые 9 байт памяти. Поэтому он занимает страницы 0,1 и первый байт страницы 2. Второй байт страницы 2 зарезервирован для внутреннего данных. Для обеспечения безопасности и системных требований эти байты защищены от записи после того, как были запрограммированы изготовителем при производстве.

2. Блокирующие байты (Lock0, Lock1)

Биты байтов 2 и 3 страницы 2 являются средством запрета записи соответствующих страниц. Каждая страница X из диапазона 3..15, может быть индивидуально заблокирована для записи путем установки соответствующего бита Lx в 1. После блокировки, страница становится доступна только для чтения.



Биты BLx нужны для того, чтобы блокировать дальнейшую несанкционированную блокировку областей памяти. Например, если установить BL15-10 в 1, то биты L15..L10 больше нельзя будет изменить.

Биты блокировки устанавливаются стандартной командой записи во вторую страницу. Байты 2 и 3 в команде записи и фактическое значение lock-байтов логически складываются (т.е. если какой-либо бит в Lock-байте был установлен в 1, он больше не может быть сброшен!).

3. OTP область (байты для одноразовой записи) (OTP0, OTP1, OTP2, OTP3)

Страница 3 представляет собой область одноразовой записи. При установке 1 в каком-либо бите области OTP, его становится невозможно сбросить в ноль. По умолчанию (с завода) байты OTP установлены в 0.
Примечание: область OTP может быть использована как одноразовый счетчик до 32.

4. Страницы данных (Data0, Data1, ..., Data47)

Страницы 4..15 могут использоваться по усмотрению пользователя для чтения-записи. После производства страницы данных инициализированы ко всем "0".

Mifare Classic 4K

Phillips разработал карты Mifare 4K для использования со считывателями стандарта ISO/IEC14443A. Радиоинтерфейс (MIFARE RF) соответствует частям 2 и 3 стандарта ISO/IEC14443A. Слой безопасности поддерживает доказанный для поля шифр потока CRYPTO1 для безопасного обмена данными mifare классического семейства.

Основные возможности

1. Радиочастотный интерфейс Mifare (ISO/IEC 14443A)
    - Бесконтактная передача данных и питание по радиоканалу (не требует батарей)
    - Рабочее расстояние до 100мм (зависит от геометрических параметров антенны)
    - Рабочая частота: 13,56МГц
    - Быстрая передача данных: 106Кбит/сек
    - Высокая надежность передачи (16-битовая CRC, проверка на четность...)
    - Настоящая антиколлизия (поддержка нескольких карт в поле одновременно)
    - Время типовой транзакции менее 100мс (в том числе резервное управление)
2. Память
    - Размер 4КБайт, разбита на 32 сектора по 4 блока и на 8 секоторов по 16 блоков. Один блок состоит из 16 байт.
    - Определяемые пользователем условия доступа для каждого блока памяти
    - Срок хранения данных - 10 лет
    - 100 000 циклов записи
3. Защита
    - Настоящие три прохода аутентификации (ISO/IEC DIS9798-2)
    - Шифрование данных в радиоканале

- Индивидуальная пара ключей для каждого сектора
- Уникальный 4-байтовый серийный номер для каждой карты
- Транспортный ключ защищает доступ к EEPROM при поставке чипа

Организация памяти

4 КБайта перепрограммируемой памяти разбиты на 32 сектора по 4 блока и на 8 секторов по 16 блоков. Один блок состоит из 16 байт.

В стертом состоянии ячейки читаются, как логический ноль, в записанном - как 1.

		Номер байта в блоке														
Сектор	**Блок**	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	**Описание**
39	15	Key A						Access Bits				Key B						Sector Trailer 39
	14																	Data
	13																	Data
	..																	..
	..																	..
	2																	Data
	1																	Data
	0																	Data
	..																	..
	..																	..
	..																	..
32	15	Key A						Access Bits				Key B						Sector Trailer 32
	14																	Data
	13																	Data
	..																	..
	..																	..
	2																	Data
	1																	Data
	0																	Data
31	3	Key A						Access Bits				Key B						Sector Trailer 31
	2																	Data
	1																	Data
	0																	Data
	..																	..
	..																	..
	..																	..
0	3	Key A						Access Bits				Key B						Sector Trailer 0
	2																	Data
	1																	Data
	0																	Manufacturer Data

1. Блок производителя (Manufacturer Block)

Нулевой блок хранит данные производителя. Уникальный (гарантировано Philips) ID, или серийный номер карты - байт 0..3. Четвертый байт - контрольная сумма номера. Блок данных производителя доступен только для чтения.

2. Блоки данных (Data)

Сектора 0..31 содержат по 3 блока данных, а сектора 32..39 - по 15 блоков данных. Блоки данных могут быть сконфигурированы с помощью битов доступа (access bits) для чтения-записи или для хранения значения (value).

Блоки значения (value)

Блоки значения позволяют выполнять команды read (чтение), write(запись), increment (увеличение), decrement (уменьшение), restore (восстановление) и transfer (сохранение). Блок значения имеет фиксированный формат, позволяющий обнаружение и исправление ошибок. Блок значения может быть сгенерирован только командой записи.

Номер байта	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Описание	значение (value)				НЕ значение (not value)				значение (value)				Адр.	НЕ адр.	Адр.	НЕ адр.

Значение (value) - 4 байтовое знаковое целое число. (not value) - 4 байтовое инверсное к value знаковое целое число. Адрес (адр) - 1-байтовый адрес, который может быть использован для реализации функции бэкапа. Изменяется только командой записи.

3. Прицеп сектора (Sector trailer)

Каждый сектор имеет "прицеп", расположенный в блоке №3 каждого сектора для первых 2х Килобайт и в блоке №15 каждого сектора - для старших 2х Килобайт. Каждый прицеп хранит секретные ключи А и Б, условия доступа для всех блоков в секторе (байты 6..9).

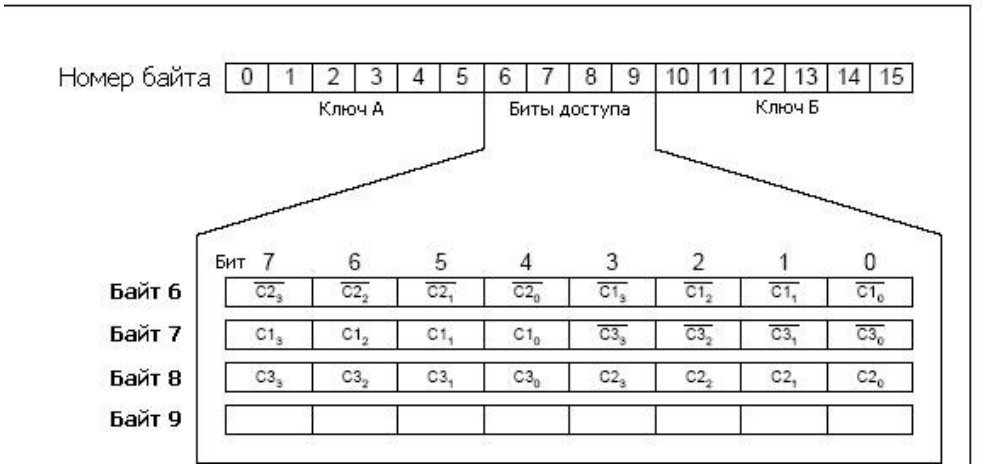
Номер байта	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Описание	Ключ А (key A)						Биты доступа				Ключ Б (key B)					

Работа с памятью

Условия доступа

Условия доступа к областям данных и к прицепу задаются тремя битами, которые хранятся в прицепе. Биты доступа задают, какие ключи необходимы для операций над областью.

Биты доступа	Допустимые команды			Описание
$C1_3\ C2_3\ C3_3$	read, write	→	3	прицеп сектора
$C1_2\ C2_2\ C3_2$	read, write, increment, decrement, transfer, restore	→	2	область данных
$C1_1\ C2_1\ C3_1$	read, write, increment, decrement, transfer, restore	→	1	область данных
$C1_0\ C2_0\ C3_0$	read, write, increment, decrement, transfer, restore	→	0	область данных



Примечание: для младших 2 Кбайт, область данных равна 1 блоку (16 байт). Для старших 2 Кбайт область данных = 5 блокам. Т.е. в старших 2 Кб, доступ задается не к каждому блоку индивидуально, а к областям по 5 блоков.

ВНИМАНИЕ: если биты доступа записаны неверно, то сектор становится недоступен - то есть необратимо блокируется и использовать его в дальнейшем НЕ сможет никто!!!

Условия доступа для прицепа

В зависимости от значения битов доступа, прицеп сектора может быть сконфигурирован для чтения/записи, как "никогда", "ключ А", "ключ Б" или "ключ А или Б". Для новых карт, ключ А задается производителем (для Филипса: FF FF FF FF FF FF).

ВНИМАНИЕ: так как биты доступа могут блокировать доступ к самим себе, следует быть внимательным при разметке новой карты!

биты доступа			**условия доступа для...**						**примечания**
			ключ А		биты доступа		ключ Б		
C1	C2	C3	чтение	запись	чтение	запись	чтение	запись	
0	0	0	никогда	ключ А	ключ А	никогда	ключ А	ключ А	ключ Б может быть прочитан
0	1	0	никогда	никогда	ключ А	никогда	ключ А	никогда	ключ Б может быть прочитан
1	0	0	никогда	ключ Б	ключ А или Б	никогда	никогда	ключ Б	
1	1	0	никогда	никогда	ключ А или Б	никогда	никогда	никогда	
0	0	1	никогда	ключ А	ключ А	ключ А	ключ А	ключ А	ключ Б может быть прочитан (новая карта)
0	1	1	никогда	ключ Б	ключ А или Б	ключ Б	никогда	ключ Б	
1	0	1	никогда	никогда	ключ А или Б	ключ Б	никогда	никогда	
1	1	1	никогда	никогда	ключ А или Б	никогда	никогда	никогда	

Условия доступа для областей данных

В зависимости от значений битов доступа, данные могут быть доступны для чтения/записи: "никогда", по ключу А, по ключу Б или по ключу А или Б. Установка битов доступа определяет допустимые команды и применение карты. блок чтения-записи: доступен и на чтение, и на запись блок значения (value): дополнительно позволяет операции increment, decrement, transfer и restore. В случае единицы (001) только чтение и декремент допустимы (для "не пополняемой" карты). В случае 110, "пополнение" возможно с помощью ключа Б. блок данных производителя всегда доступен на чтение, вне зависимости от битов доступа.

биты доступа			**условия доступа для...**				**применение**
C1	C2	C3	чтение	запись	инкремент	декремнт, transfer, restore	
0	0	0	ключ А или Б*	ключ А или Б*	ключ А или Б*	ключ А или Б*	новая карта*

0	1	0	ключ А или Б*	никогда	никогда	никогда	блок для чтения-записи*
1	0	0	ключ А или Б*	ключ Б*	никогда	никогда	блок для чтения-записи
1	1	0	ключ А или Б*	ключ Б*	ключ Б*	ключ А или Б*	блок значения
0	0	1	ключ А или Б*	никогда	никогда	ключ А или Б*	блок значения*
0	1	1	ключ Б*	ключ Б*	никогда	никогда	блок для чтения-записи
1	0	1	ключ Б*	никогда	никогда	никогда	блок для чтения-записи
1	1	1	никогда	никогда	никогда	никогда	блок для чтения-записи

* если ключ Б может быть прочитан (из соответствующего прицепа), он не может служить для авторизации.

При попытке авторизоваться ключом Б, карта будет отвечать отказом в любом последующем доступе.

Mifare Plus

Смарт-карты Mifare Plus - улучшенная версия Mifare Classic, призваны повысить существующий уровень безопасности при использовании бесконтактных смарт-карт карт в различных прикладных системах (оплата на транспорте, системы доступа, электронный сбор платы, парковки, интернет-кафе, программы лояльности и т.п.).

Карты Mifare Plus могут легко интегрироваться в существующие системы, где уже используются карты Mifare Classic. Уровень защищенности карт Mifare Plus может быть повышен в любой момент по мере развития системы путем активизации алгоритма AES-128 (Advanced Encryption Standard), обеспечивающего высокий уровень безопасности, целостности данных, аутентификации и шифрования.

Mifare Plus основана на глобальных открытых стандартах как по интерфейсу, так и по криптографии.

Mifare Plus имеет две версии:

- Mifare Plus S – это "Slim" версия, для быстрого перехода от Mifare Classis (может переключаться в режим SL1, SL3);
- Mifare Plus X – это "eXpert" версия, обеспечивающая большую гибкость, защищенность и функциональность (может переключаться в режим SL1, SL2, SL3).

Основные возможности

- 2 или 4 килобайта памяти (EEPROM);
- Простая структура фиксированной памяти, совместимая с Mifare Classic 1K и Mifare Classic 4K;
- Структура памяти идентична Mifare Classic 4K (сектора, блоки);
- Условия доступа свободно конфигурируемые;
- Поддержка ISO/IEC 14443-31 UID (4-байтовый UID, 4-байтовый NUID, 7-байтовый UID), опциональная поддержка случайных идентификаторов;

- Многосекторная аутентификация, многоблочное чтение и запись;
- AES-128 используется для обеспечения аутентификации и целостности;
- Анти-разрушающий механизм для записи ключей AES;
- Ключи могут храниться как ключи MIFARE CRYPTO1 (2 x 48 бит на сектор), так и как AES ключи (2 x 128 бит на сектор);
- Полная поддержка концепции виртуальной карты;
- Проверка близости (Proximity check);
- Скорость связи до 848 кбит/с;
- Количество операций одиночной записи: 200000 циклов (типичный);
- Сертификация по общим критериям: EAL4+ (Common Criteria Certification: EAL4+).

Уровни безопасности (Security level)

Карты, работающие на одном уровне безопасности, могут быть в любой момент переведены на более высокий уровень безопасности.

В прикладной системе карты Mifare Plus могут находиться только на одном конкретном уровне безопасности: SL1, SL2 или SL3. С завода-изготовителя чипы Mifare Plus (в картах, метках, браслетах и т.п.) поступают на уровне безопасности SL0.

Использовать в прикладной системе карты на уровне SL0 нельзя, чип Mifare Plus должен быть проинициализирован, т.е. переведен на уровень SL1, SL2 или SL3.

В прикладной системе можно последовательно повышать уровень безопасности, т.е. из SL1 переводить в SL2, а из SL2 переводит в SL3. Обратный перевод, с более высокого на более низкий уровень защиты, невозможен.

Уровень безопасности SL0

Начальный уровень. В этом состоянии чип Mifare Plus не хранит в себе никаких ключей. Из состояния SL-0 чип MIFARE Plus должен быть переведен на любой из трех более высоких уровней SL-1, SL-2 млм SL-3. На уровне SL-0 чипы предварительно персонализируются по ключам, соответствующим Mifare Classic с использованием крипто-алгоритма CRYPTO1, и по ключам AES для работы с памятью.

Уровень безопасности SL1

На этом уровне карты Mifare Plus имеют полную совместимость с Mifare Classic.

Уровень безопасности SL2

Аутентификация по AES является обязательной. Для защиты данных используется CRYPTO1.

Уровень безопасности SL3

Для аутентификации, обмена и шифрования данных, для работы с памятью, а также для выявления удаленных атак по радиоканалу используется крипто-алгоритм AES.

Работа с памятью

В основном структура памяти Mifare Plus SL3 идентична Mifare Classic (сектора, блоки), отличается структура прицепов (trailer), способ установки ключа авторизации и других ключей.

Распределение памяти

Trailer Mifare Classic 1(4)K, Mifare Plus SL1(SL2):

Байт	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Key A									69	Key B					

Trailer Mifare Plus SL3:

Байт	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						En										

где En - биты доступа (по умолчанию = 0x0F всё открыто):

Биты En	Блоки младших секторов	Блоки старших секторов	Описание
7	Invert (block 3)	Invert (block 15)	Если 0: Открытая передача данных.
6	Invert (block 2)	Invert (block 10–14)	
5	Invert (block 1)	Invert (block 5-9)	
4	Invert (block 0)	Invert (block 0-4)	
3	Block 3	Block 15	Если 1: Открытая передача данных.
2	Block 2	Block 10–14	
1	Block 1	Block 5-9	
0	Block 0	Block 0-4	

En =0x0F всё открыто.

En = 0xF0 все закрыто.

Режиме SL3 ключ A,B в трейлер секторе не используется, а 6-ой(байт 5) байт ключа A(который не используется) является флагом(4 флага) так же как и биты доступа нужно шифровать данные при записи и дешифровать при чтении.

При переходе в режим SL3 байт из configuration block копируется во все трейлер блоки(байт 5 от нуля) всех секторов. Уже в режиме SL3 их можно менять отдельно.

Вся память Mifare Plus

№	Название	Адрес	Описание
1	Данные	0000 – 007F	Секторы 0 - 31
2	Данные	0080 – 00FF	Секторы 32 - 39
3	MP Configuration block	B000	
4	Installation identifier	B001	
5	ATS information	B002	Заполнен правильно. Не менять.
6	Field Configuration block	B003	
7	AES Sector Keys	4000 – 404F	Ключи авторизации секторов: В четном адресе ключ A(4000, 4002,.,.). В нечетном адресе ключ B(4001, 4003,.,.)

№	Название	Адрес	Описание
8	Originality Key	8000	
9	Card Master Key	9000	
10	Card Configuration Key	9001	
11	Level 2 Switch Key	9002	Ключ для перевода карты на уровень SL2. Для S карт нет. Для L3 нет
12	Level 3 Switch Key	9003	Ключ для перевода карты на уровень SL3. Для L3 нет
13	Sl1 Card Authentication Key	9004	Для L3 нет
14	Select VC Key	A000	
15	Proximity Check Key	A001	
16	VC Polling ENC Key	A080	Фиксированный – записывается, но без разнообразия(diversification)
17	VC Polling MAC Key	A081	Фиксированный

Подробности выдаются NXP Semiconductors под NDA.

Temic (T5557, T5577)

Применяется для записи или копирования уникального номера (может имитировать EM Marine, HID Prox II и другие стандарты, работающие на частоте 100-150Khz). А также для создания систем с использованием криптоалгоритмов в работе карты (гостиницы, временные пропуски и т.п.). При использовании наклейки с липким слоем (паутч), можно произвести персонализацию.

Основные возможности

1. Бесконтактная передача данных (чтение/запись)
2. Радио частоты от 100 кГц до 150 кГц
3. Режим совместимости с e5550 или расширенный режим T5557
4. 7 x 32-бита EEPROM памяти данных в том числе 32-битный пароль
5. Отдельные 64-бита памяти для отслеживания происхождения данных
6. 32-битный регистр конфигурации в EEPROM для установки:
    - Скорость передачи данных
        - RF/2 - RF/128, двоичный выбор или
        - Фиксированные e5550 скорости передачи данных
    - Модуляции / кодирования
        - FSK, PSK, Манчестер, Biphase, NRZ
    - Другие опции
        - Режим пароля
        - Max Block функция
        - Режим Ответ-на-запрос (AOP)
        - Обратный вывод данных
        - Режим прямого доступа
        - Последовательность Терминатор(ов)
        - Защита от записи (через Лок-бит в блоке)
        - Метод быстрой записи (5 кбит/с по сравнению с 2 кбит/с)
        - OTP Функциональность
        - POR задержки до 67 мс

T5557 является бесконтактным R/W идентификатором IC (IDIC) для применения в 125 кГц частотном диапазоне. 330-битная EEPROM (10 блоков, 33 бит каждый), позволяет читать и писать блоки. Блок 0 зарезервирован для настройки режимов работы T5557 метки. Блок 7 может содержать пароль для предотвращения несанкционированной записи.

Организация памяти

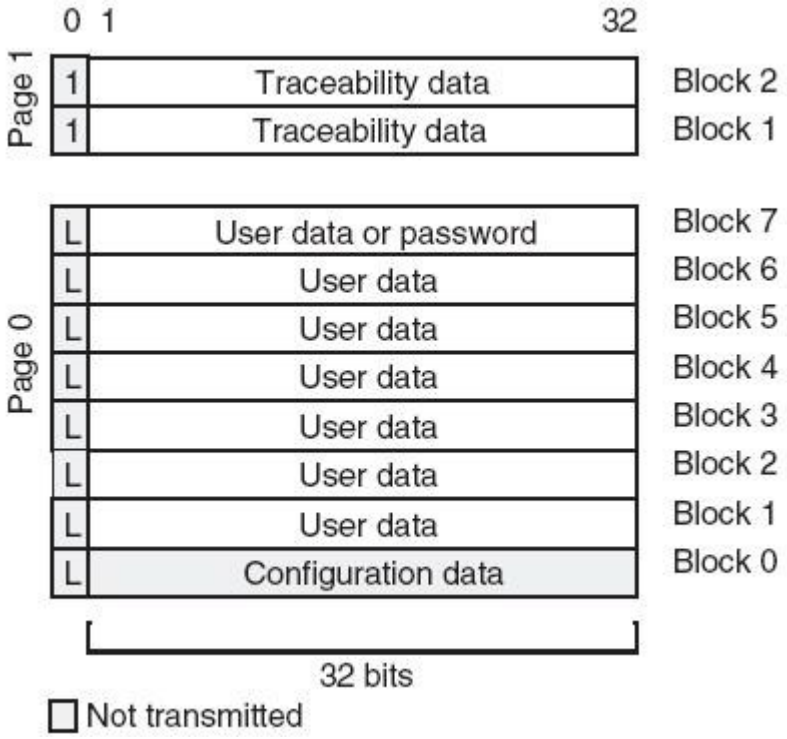
330 бит перепрограммируемой памяти организовано в виде 10 блоков по 4 байта каждый.

64 бит зарезервировано под данные производителя.

32 бит предназначено для настройки конфигурации T5557.

32 бита могут использоваться как пароль для предотвращения несанкционированной записи.

234(202 исключая пароль) бита используются для программирования памяти для чтения/записи.



1. Traceability data/Данные производителя (Page 1: block 1, block 2)

Данные страницы 1 содержат уникальный 5-байтовый серийный номер и защищены от записи производителем.

2. User data or password (Page 0: block 7)

Если T5557 в режиме пароля, то блок 7 содержит пароль для защиты от несанкционированной записи.

3. Блоки данных (Page 0: block 1 .. block6 (block 7))

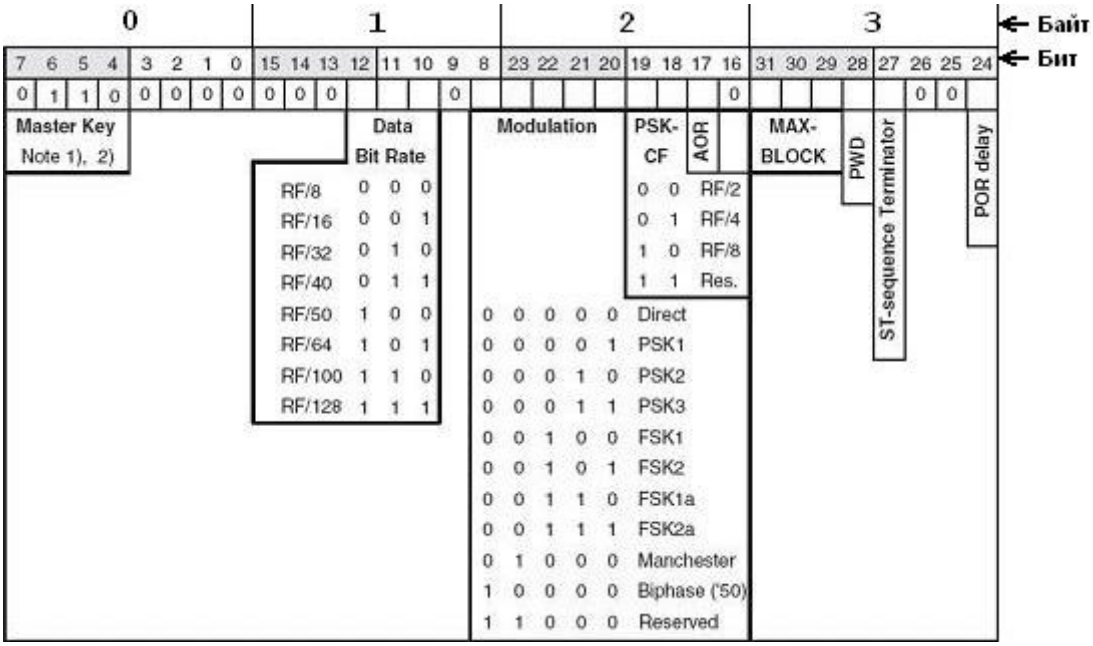
Блоки 1..6 могут использоваться по усмотрению пользователя для чтения-записи. После производства страницы данных инициализированы "0".

4. Конфигурация (Page 0: block 0)

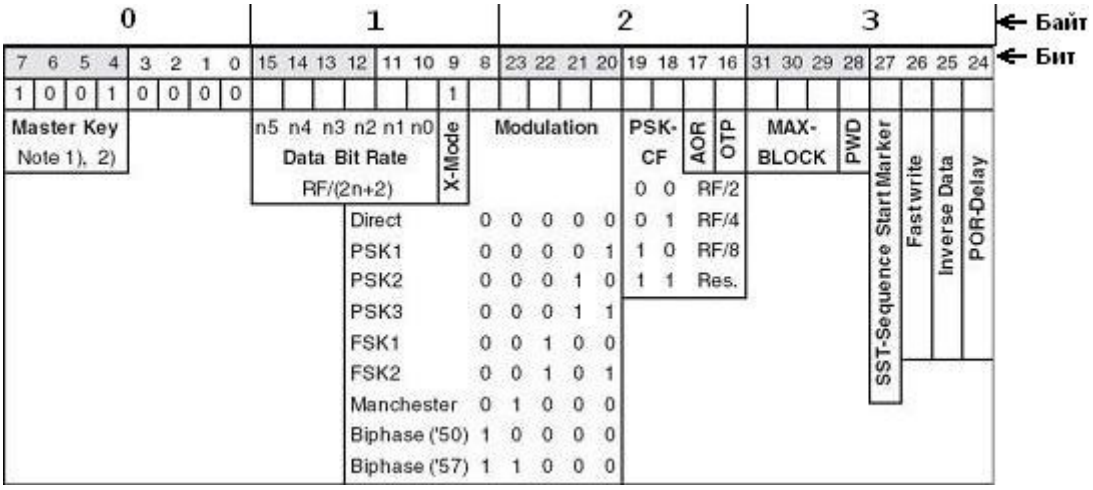
Блок 0 представляет собой регистр конфигурации T5557.

Конфигурация может быть настроена в режиме совместимости с e5550 или в расширенном режиме (X-Mode).

Конфигурация в режиме совместимости с e5550



Конфигурация в расширенном режиме (X-Mode)



Демо

Демо (Demo.exe) – программа, предназначенная для демонстрации возможностей SDK Readers. Исходный код Демо написан на языке Delphi, и расположен в папке «Demo».

После запуска «Demo.exe» появится главное окно Демо:

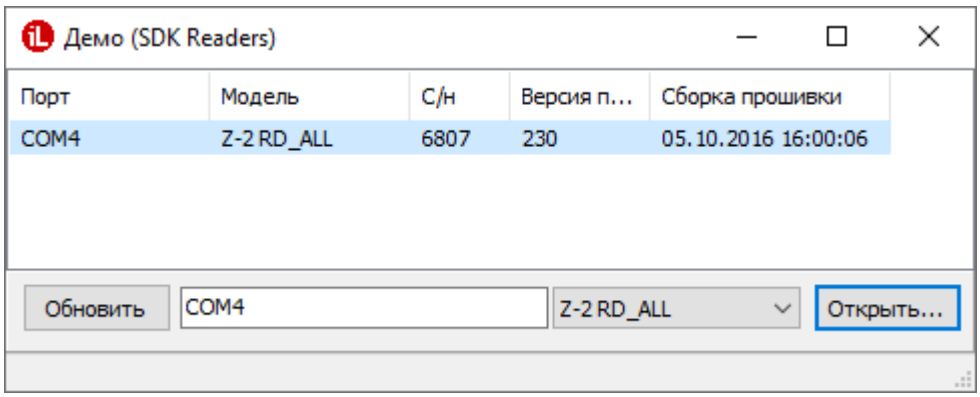


Рисунок 1. Главное окно Демо

В главном окне отображается список найденных считывателей (список обновляется автоматически).

Кнопка «Открыть…» позволяет подключиться к считывателю, имя порта которого указано слева от этой кнопки, после нажатия кнопки появится окно считывателя.

Поиск считывателей

В главном окне отображается список найденных считывателей (список обновляется автоматически).

Для поиска считывателей Демо запрашивает у SDK интерфейс IILRSearch с помощью метода IILR.GetSearch, и активирует фоновый поиск считывателей с помощью метода EnableAutoScan. В контекстном меню списка можно выбрать типы считывателей, которые нужно искать, типы устанавливаются методом SetReaderTypes. Кнопка «Обновить» очищает список и ищет считыватели с помощью интерфейса IILRSearchAsync и метода Begin_Scan, этот интерфейс позволяет выполнять команды поиска в фоновом режиме (без блокирования основного потока).

Подключение к считывателю

Кнопка «Открыть…» позволяет подключиться к считывателю, имя порта которого указано слева от этой кнопки, после нажатия кнопки появится окно считывателя. Вид окна считывателя зависит от модели считывателя:

- Окно с поддержкой чтения/записи карт Temic (Z-2 (мод. RD_ALL), Z-2 (мод. E HTZ RF))
- Окно с поддержкой чтения/записи карт Mifare Classic (Z-2 (мод. MF), Z-2 (мод. MF-I), Matrix-III (мод. MF K Net), CP-Z-2 (мод. MF-I))
- Простое окно, в котором доступно чтение номеров карт (все остальные считыватели)

Для подключения к считывателю нужно в главном окне в поле ввода ввести имя порта считывателя и нажать кнопку «Открыть…», появится окно считывателя. Для подключения Демо запрашивает у SDK интерфейс IILReader с помощью метода IILR.GetReader, и подключается к считывателю методом IILReader.Connect.

Подключение к Temic-считывателю

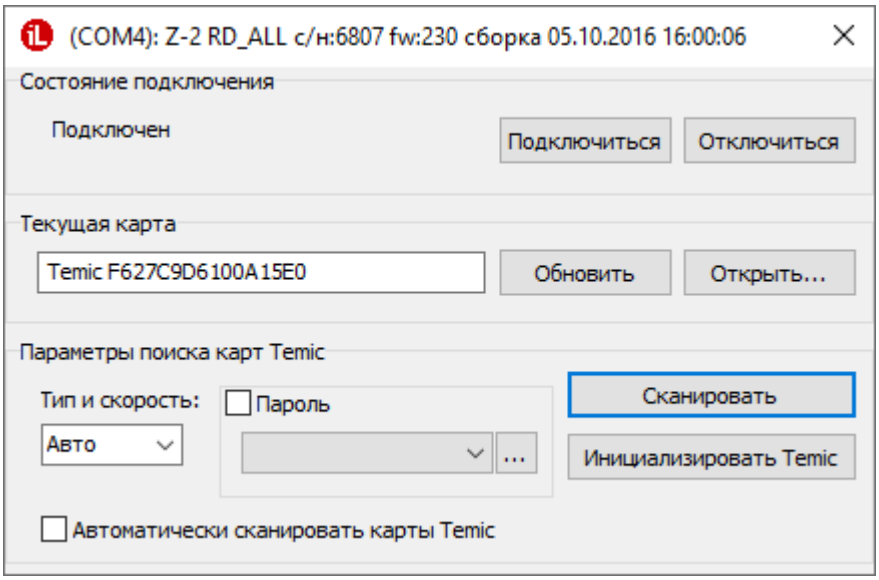


Рисунок 2. Окно считывателя с поддержкой чтения/записи карт Temic

В группе «Состояние подключения» показывается одно из состояний: 1) Подключён, 2) Подключение..., 3) Отключён, используя метод GetConnectionStatus. Кнопки «Подключиться» и «Отключиться» позволяют подключиться к / отключиться от считывателя, используя методы Connect и Disconnect.

В группе «Текущая карта» отображается тип и номер карты в поле считывателя, или надпись «Нет карты» если в поле нет карты, которую способен увидеть считыватель, используя метод GetCardInfo. Кнопка «Обновить» сканирует карту в поле считывателя, используя метод Scan. Кнопка «Открыть» открывает окно для редактирования данных карты Temic или окно для Mifare Ultralight.

Группа «Параметры поиска карт Temic» позволяет настроить сканирование карт Temic. Кнопка «Сканировать» ищет карту Temic, используя метод ScanTemic. Выбор определённого типа и скорости (кроме «Авто») позволяет быстрее находить Temic когда эти параметры соответствуют конфигурации Temic. Если карта не находится, то её нужно инициализировать кнопкой «Инициализировать Temic», при этом в конфигурационный блок карты записывается стандартная конфигурация методом WriteTemic. Чтобы активировать фоновый поиск Temic нужно установить флаг «Автоматически сканировать карты Temic», для этого используется метод EnableAutoScanTemic. Если у карты Temic установлен пароль, то нужно в группе «Пароль» установить флаг и выбрать пароль, если нужного пароля в списке нет, то добавить его с помощью кнопки «...» справа от выпадающего списка паролей, появится окно «Пароли Temic».

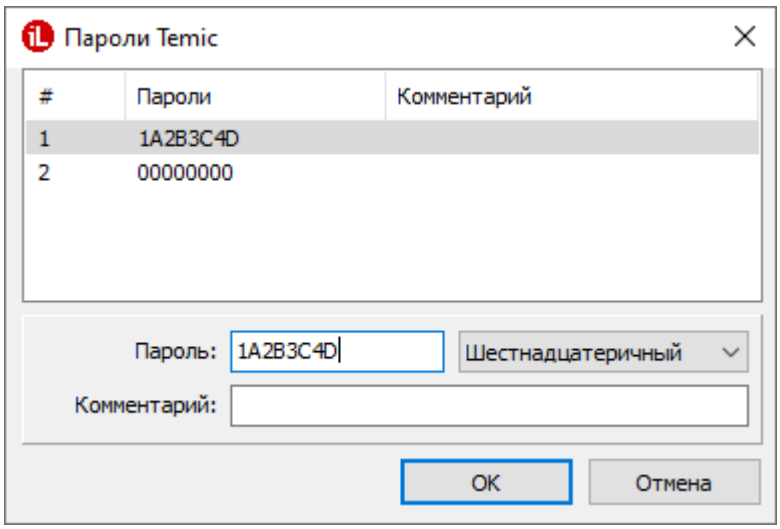


Рисунок 3. Окно «Пароли Temic»

Подключение к Mifare-считывателю

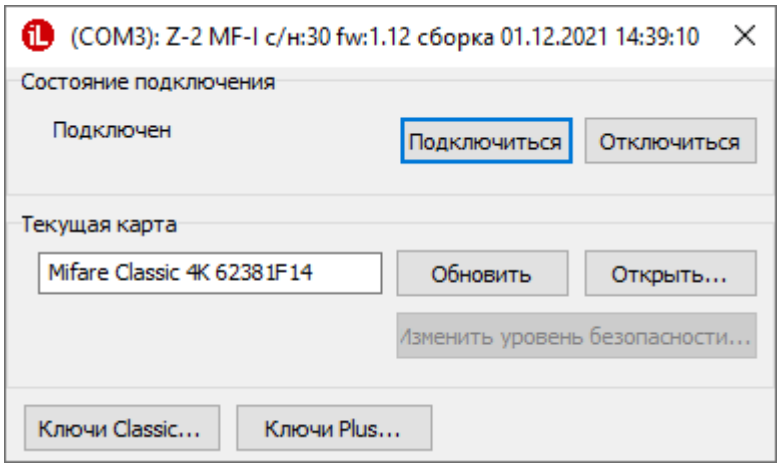


Рисунок 4. Окно считывателя с поддержкой чтения/записи карт Mifare

Демо: Mifare Ultralight

При открытии карты Mifare Ultralight для редактирования появляется окно «Mifare Ultralight»:

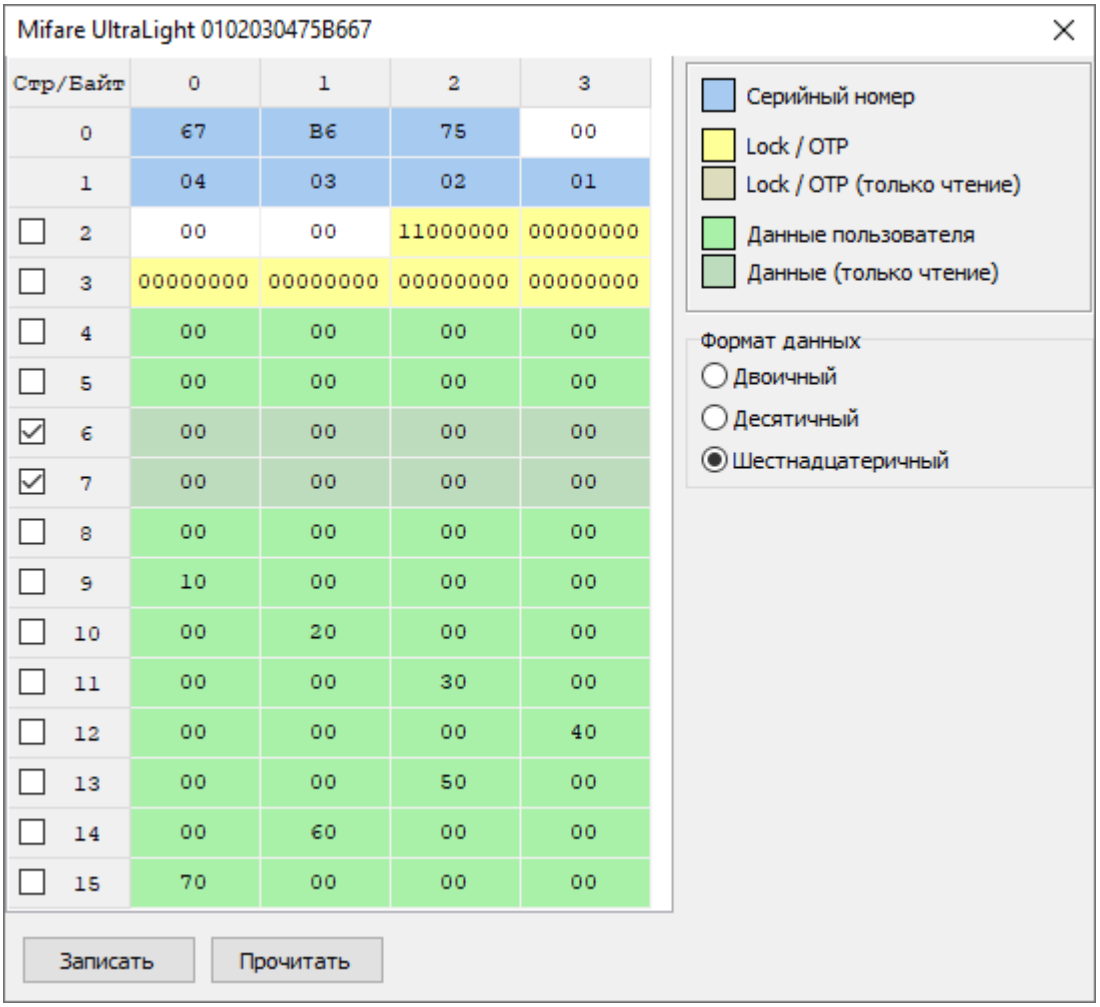


Рисунок 5. Окно Mifare Ultralight

Открыть карту Mifare Ultralight для редактирования позволяют следующие считыватели:

- Z-2 (мод. RD_ALL)/Z-2 USB
- Z-2 (мод. MF)/Z-2 USB MF
- Z-2 (мод. MF-I)
- Z-2 (мод. MF CCID)
- Z-2 (мод. E HTZ RF) / Z-2 EHR
- Matrix-III (мод. RD_All)
- Matrix-III (мод. MF K Net) / Matrix-III Net
- CP-Z-2 (мод. MF-I)

При открытии карты данные считываются автоматически методом IILReader.ReadMfUlralight.

В таблице отображаются байты памяти карты в формате, выбранном в группе «Формат данных». Чтобы редактировать значения байт выделите нужную ячейку (клик ЛКМ по ячейке) и введите новое значение. Флажки слева позволяют навсегда заблокировать страницу памяти от перезаписи. Кнопка «Записать» записывает все страницы в память карты методом IILReader.WriteMfUlralight.

Демо: Mifare Classic

При открытии карты Mifare Classic или Mifare Plus SL1 для редактирования появляется окно «Mifare Classic»:

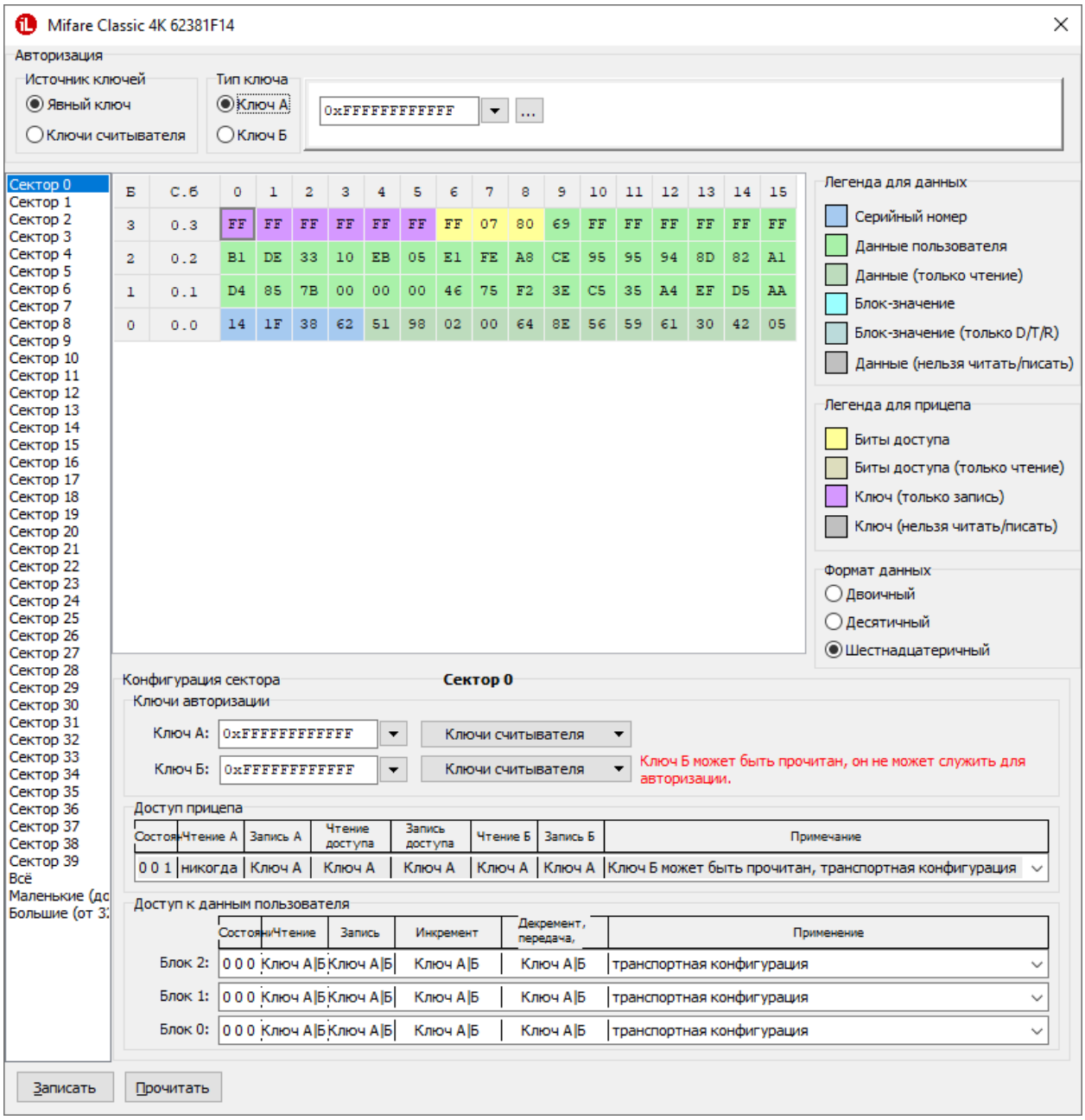


Рисунок 6. Окно Mifare Classic

Открыть карту Mifare Classic для редактирования позволяют следующие считыватели:

- Z-2 (мод. MF)/Z-2 USB MF
- Z-2 (мод. MF-I)
- Z-2 (мод. MF CCID)
- Matrix-III (мод. MF K Net) / Matrix-III Net
- CP-Z-2 (мод. MF-I)

При открытии карты данные считываются автоматически методом IILReader.ReadMfClassic, предварительно авторизовав каждый сектор методом IILReader.AuthMfCard.

В группе «Авторизация» (наверху) нужно выбрать способ авторизации сектора:

- Явный ключ - по ключу, указанному в поле ввода справа;
- Ключи считывателя - по списку ключей, сохранённых в памяти считывателя методом IILReader.WriteMfAuthKeyToReader.
- Ключ А, Ключ Б - типы ключа авторизации, по ключу Б можно авторизоваться только в некоторых конфигурациях.

В таблице отображаются байты памяти карты в формате, выбранном в группе «Формат данных». Чтобы редактировать значения байт выделите нужную ячейку (клик ЛКМ по ячейке) и введите новое значение. Кнопка «Записать» записывает все страницы в память карты методом IILReader.WriteMfClassic.

Демо: Mifare Plus

При открытии карты Mifare Plus SL3 для редактирования появляется окно «Mifare Plus»:

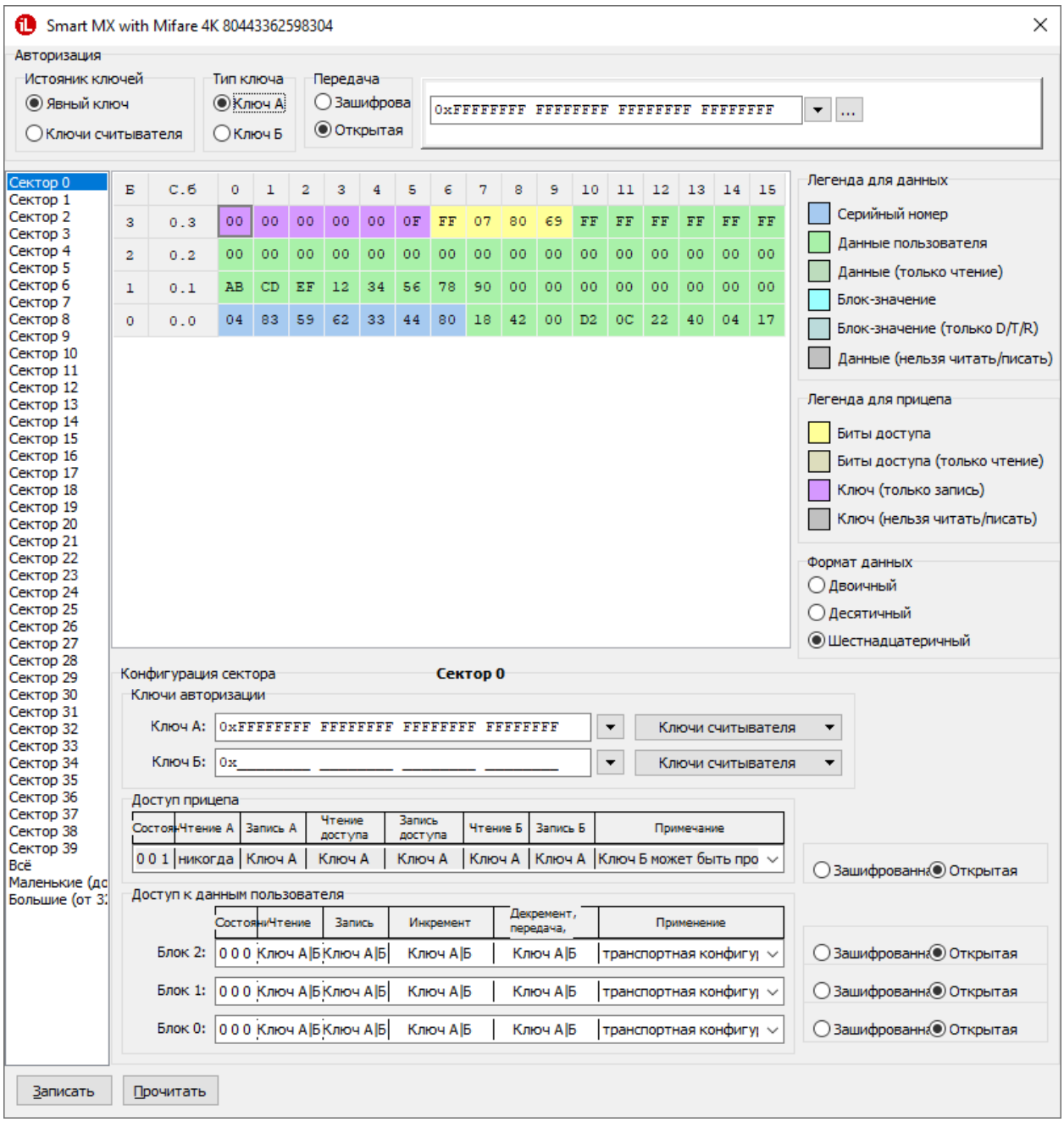


Рисунок 7. Окно Mifare Plus

Открыть карту Mifare Plus для редактирования позволяют следующие считыватели:

- Z-2 (мод. MF-I)

При открытии карты данные считываются автоматически методом IILReader.ReadMfPlus, предварительно авторизовав каждый сектор методом IILReader.AuthMfCard.

В группе «Авторизация» (наверху) нужно выбрать способ авторизации сектора:

- Явный ключ - по ключу, указанному в поле ввода справа;
- Ключи считывателя - по списку ключей, сохранённых в памяти считывателя методом IILReader.WriteMfPlusAuthKeyToReader.
- Ключ А, Ключ Б - типы ключа авторизации, по ключу Б можно авторизоваться только в некоторых конфигурациях.

Зашифрованная, Открытая - способ передачи данных между считывателем и картой, каждая область каждого сектора карты конфигурируется под определённый способ передачи, при чтении данных с неправильно указанным способом передачи данные будут прочитаны не верно.

В таблице отображаются байты памяти карты в формате, выбранном в группе «Формат данных». Чтобы редактировать значения байт выделите нужную ячейку (клик ЛКМ по ячейке) и введите новое значение. Кнопка «Записать» записывает все страницы в память карты методом IILReader.WriteMfPlus.

Демо: Temic

При открытии карты Temic для редактирования появляется окно «Temic»:

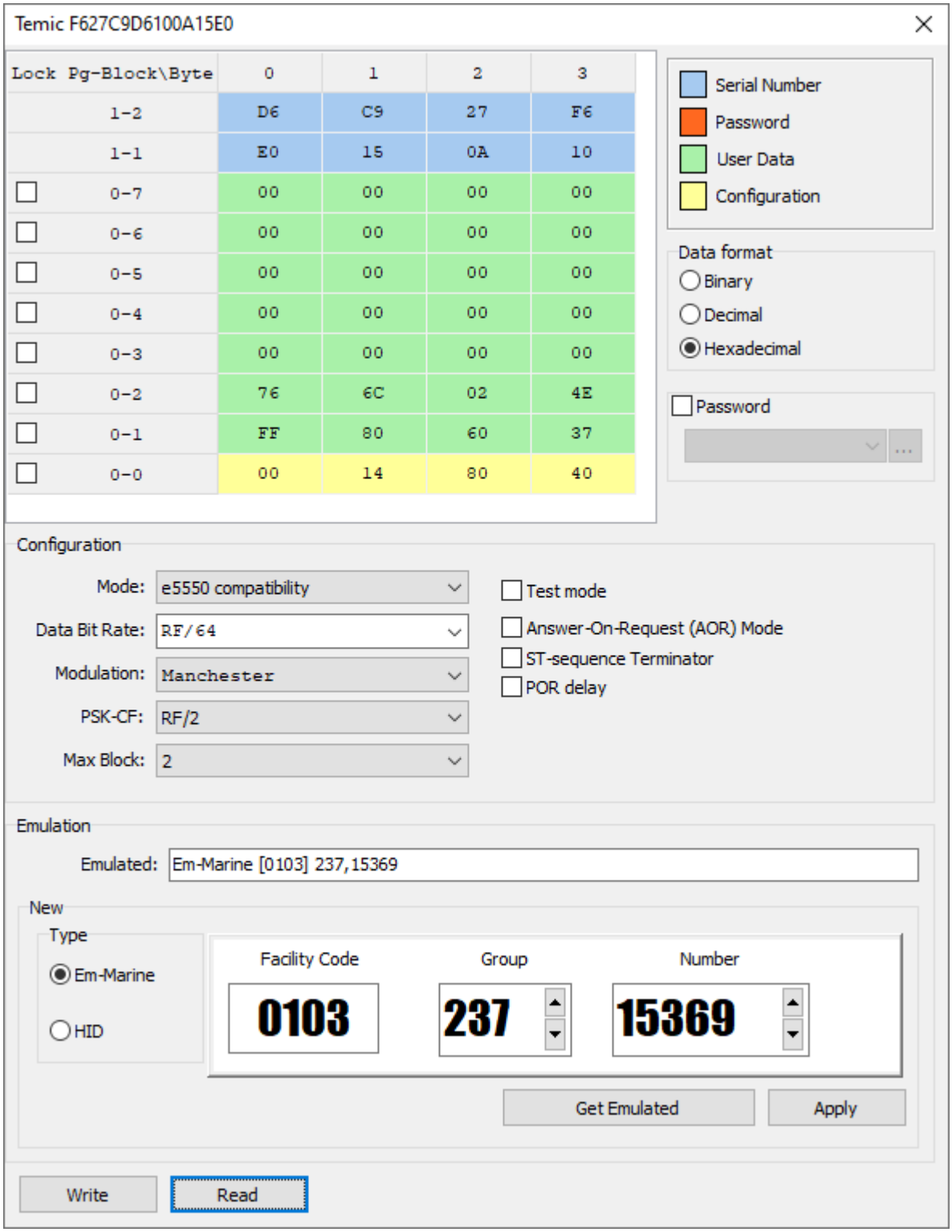


Рисунок 8. Окно Temic

Открыть карту Temic для редактирования позволяют следующие считыватели:

- Z-2 (мод. RD_ALL)/Z-2 USB
- Z-2 (мод. E HTZ RF) / Z-2 EHR

При открытии карты данные считываются автоматически методом IILReader.ReadTemic.

В таблице отображаются байты памяти карты в формате, выбранном в группе «Формат данных». Чтобы редактировать значения байт выделите нужную ячейку (клик ЛКМ по ячейке) и введите новое значение. Флажки слева позволяют навсегда заблокировать страницу памяти от перезаписи. Кнопка «Записать» записывает все страницы в память карты методом IILReader.WriteTemic.

Примеры

Примеры использования SDK:

Пример	**Описание**
EnumReaders	Поиск считывателей.
ReaderDetector	Поиск считывателей, уведомление о нахождении/потери считывателя.
ConnectToReader	Подключение к считывателю, уведомления о потери/восстановлении связи.
CardDetector	Поиск карт, уведомление о поднесении/удалении карты.
MfUltralight	Чтение/запись данных карты Mifare Ultralight.
MfClassic	Чтение/запись данных карты Mifare Classic и Mifare Plus SL1.
MfPlus	Чтение/запись данных карты Mifare Plus SL3.
Temic	Чтение/запись данных карты Temic.