



**Набор средств разработки (SDK)
системы распознавания
по радужной оболочке глаза
(«Циркон»)**



Руководство разработчика

© «Системы Папилон», v.1.1.0, декабрь 2008 г.,

Информация, содержащаяся в настоящем документе, может быть
изменена без дополнительного уведомления.

Назначение

Рисунок радужной оболочки глаза (РОГ) является индивидуальным для каждого человека. Он достаточно сложен, и его практически невозможно подделать, поэтому верификация и идентификация по радужной оболочке глаза является одним из самых надежных методов установления личности.

Представленные SDK и комплект оборудования предназначены для построения автоматизированных систем контроля и управления доступом (АСКУД) с использованием биометрии радужной оболочки глаза, а также для интеграции в эксплуатируемые АСКУД в виде дополнительных модулей верификации/идентификации личности, повышающих совокупную надежность системы.

В АСКУД может быть интегрировано произвольное количество унифицированных блоков, размещаемых на пунктах доступа в охраняемый периметр объекта или группы объектов.

Блок доступа включает два взаимосвязанных элемента – иридосканер «Папилон» и специальный вычислитель.

Иридосканер «Папилон» предназначен для получения изображения радужной оболочки глаза и передачи его в вычислитель.

Вычислитель используется для обработки и хранения биометрических данных, поступающих с иридосканера или из внешней базы данных, а также для обработки запросов на верификацию или идентификацию.

В состав поставки блока доступа также входит библиотека SDK на компакт-диске и документация.

При помощи набора функций SDK осуществляется получение и добавление в БД вычислителя биометрических данных конечных пользователей, а также управление процессами верификации и идентификации.

Подключение оборудования к уже существующей системе контроля доступа осуществляется по сети Ethernet и через последовательный порт.

Команды, передаваемые по сети Ethernet, используются для управления процессами получения и добавления в БД вычислителя биометрических данных, а команды, передаваемые через последовательный порт, – для управления процессами верификации или идентификации.

Количество добавляемых модулей верификации/идентификации личности по радужной оболочке глаза («Циркон») в уже существующую систему АСКУД ограничивается только возможностью масштабирования последней. Каждый модуль верификации/идентификации личности содержит свою базу биометрических данных, что обеспечивает необходимую гибкость настройки системы доступа для различных защищаемых периметров.

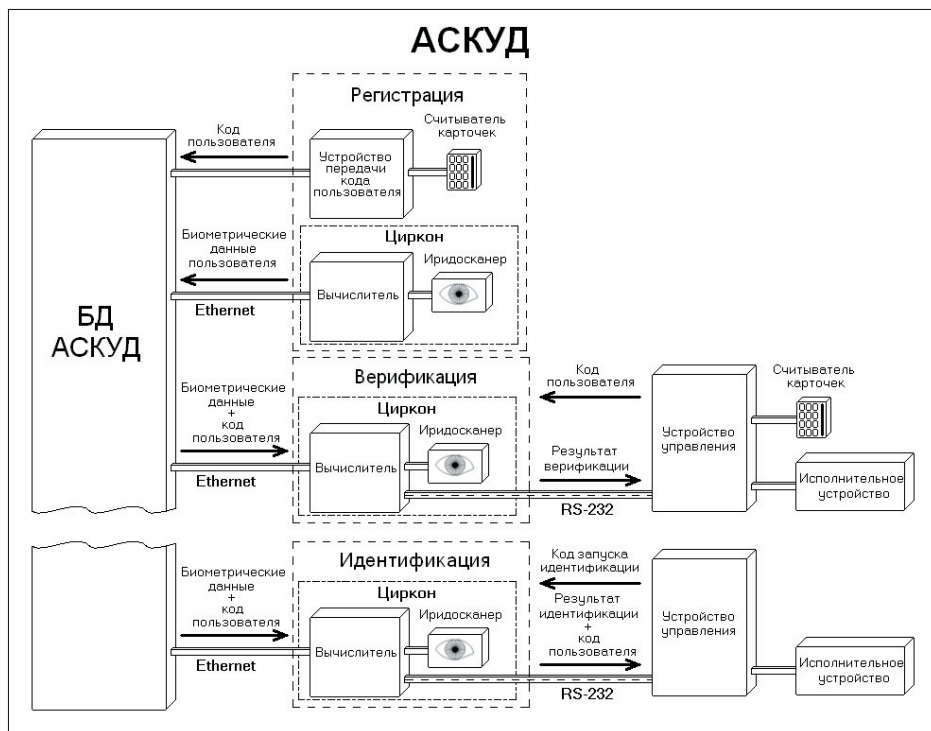


Рис. 1. Типовая схема интеграции в АСКУД



Библиотека функций SDK

Библиотека функций SDK именуется **libscaniris.dll** и обеспечивает получение блока биометрических данных (ББД), предназначенного для хранения в базе данных вычислителя и используемого для сравнения с вновь вводимыми биометрическими данными.

Библиотека содержит следующий перечень функций:

create	Выделяет необходимую память, используемую при работе с библиотекой — создает “объект” и возвращает его дескриптор, который передается в остальные функции библиотеки.
destroy	Удаляет объект, созданный ранее функцией create .
scaniris	Запускает интерактивную процедуру получения изображения РОГ, сканирует изображение и возвращает блок биометрических данных, полученный при сканировании.
makeTransferBlock	Объединяет два блока биометрических данных (левого и правого глаза) в блок, предназначенный для хранения в БД вычислителя в виде шаблона, предназначенного для верификации или идентификации.
get_last_error	Возвращает строку, содержащую описание последней ошибки.
get_lib_version	Возвращает строку, содержащую номер версии SDK.

Создание объекта и получение его дескриптора

Для создания объекта и получения дескриптора используется функция **create**.

```
libscaniris_handle_t libscaniris_create()
```

Входные параметры: отсутствуют.

Возвращает:

`handle` — дескриптор, указывающий на объект, в виде значения с типом `libscaniris_handle_t`.

0 — в случае ошибки (например, отсутствие свободной памяти)

Удаление объекта

Функция **destroy** удаляет объект, созданный ранее функцией **create**.

```
int libscaniris_destroy(libscaniris_handle_t handle)
```

Входные параметры:

`handle` — полученный при помощи функции **create** дескриптор, указывающий на объект.

Возвращает:

0 — в случае успешного удаления объекта.

-1 — если объект, согласно указанному дескриптору, отсутствует.

Получение блока данных одного глаза

Для сканирования изображения радужной оболочки глаза, получения блока биометрических данных и сохранения его в буфер используется функция **scaniris**.

Перед вызовом функции `scaniris` необходимо инициализировать структуру `scaniris_data` с типом `libscaniris_scaniris_data_t`.

```
typedef struct scaniris_data
{
    unsigned char eye;
    unsigned char* buf;
    unsigned int* buf_len;
    int timeout_for_scan;
}
libscaniris_scaniris_data_t;
```

В структуре устанавливаются следующие значения полей:

eye с типом `unsigned int` — значение подтипа радужной оболочки глаза (`SCAN_LEFT_EYE` — левый глаз или `SCAN_RIGHT_EYE` — правый глаз).

buf с типом `unsigned int` — указатель на буфер хранения получаемого блока данных. Буфер для хранения принимаемого блока данных (размером не менее 768 байт) должен быть выделен заранее.

buf_len с типом `unsigned char` — указатель на поле, содержащее размер полученного блока данных. Заполнение поля происходит после получения блока биометрических данных и содержит размер этого блока.

timeout_for_scan с типом `in` — период времени, отводимый для получения одного изображения РОГ. Значение периода может быть установлено от 1 до 600 секунд, если установлено значение **-1**, то период сканирования не ограничен временным интервалом.

При обращении клиентского приложения к функции `scaniris` на рабочей станции пользователя открывается интерактивное окно, используемое для отображения процесса захвата изображения РОГ (захват изображения радужной оболочки одного глаза производится три раза).

```
int libscaniris_scaniris(libscaniris_handle_t      handle,
                        libscaniris_scaniris_data_t* data,
                        const char*                 pszHostName,
                        int                          port)
```

Входные параметры:

handle — полученный при помощи функции `create` дескриптор, указывающий на объект.

data — указатель на инициализированную структуру `scaniris_data` с типом `libscaniris_scaniris_data_t`, в которой сохраняются получаемые данные.

pszHostName — IP адрес вычислителя, в виде значения с типом `const char` (по умолчанию IP-адрес вычислителя установлен — 192.168.1.2).

port — номер TCP порта в виде значения с типом `int` (по умолчанию номер TCP порта — 6120).

Возвращает:

0 — в случае успешного получения изображения, его обработки и сохранения блока биометрических данных в указанный буфер.

-1 — если при получении, обработке или сохранении биометрических данных произошла ошибка.

Объединение двух блоков данных

Для получения блока данных, содержащего биометрические данные обоих глаз и предназначенного для хранения в БД вычислителя, используется функция **makeTransferBlock**.

```
int libscaniris_makeTransferBlock (libscaniris_handle_t      handle,
                                  libscaniris_scaniris_data_t* data,
                                  libscaniris_scaniris_data_t* data1,
                                  libscaniris_scaniris_eyes_block_t* block)
```

Перед вызовом функции **makeTransferBlock** необходимо инициализировать структуру **scaniris_eyes_block**, тип которой **libscaniris_scaniris_eyes_block_t**.

```
typedef struct scaniris_eyes_block
{
    unsigned char* buf;
    unsigned int* buf_len;
}
libscaniris_scaniris_eyes_block_t;
```

В структуре устанавливаются следующие параметры полей:

- **buf** с типом **unsigned char** — указатель на буфер хранения объединенного блока данных. Буфер, предназначенный для хранения объединенного блока биометрических данных левого и правого глаза, должен быть выделен заранее. Размер буфера вычисляется по формуле:

$$n * (\text{libscaniris_scaniris_data_t.buf_len}) * 2.7$$

где:

n — количество объединяемых ББД (принимает значение 1 или 2)

libscaniris_scaniris_data_t.buf_len — размер ББД

- **buf_len** с типом **unsigned int** — указатель на поле, содержащее размер полученного объединенного блока данных. Заполнение поля происходит после получения объединенного блока биометрических данных.

Входные параметры:

handle — полученный при помощи функции **create** дескриптор, указывающий на объект.

`data`, `data1` — указатели на инициализированную структуру `scaniris_data`, из которой будут извлекаться биометрические данные, тип которых — `libscaniris_scaniris_data_t`. В случае, если данные отсутствуют, то в качестве входного параметра указывается значение `NULL`.

`block` — указатель на инициализированную структуру данных `scaniris_eyes_block`, тип которых — `libscaniris_scaniris_eyes_block_t`, в которую будут сохраняться данные, получаемые при объединении двух блоков биометрических данных.

Примечание | В параметрах `data` и `data1` должны быть указаны блоки биометрических данных разных подтипов `POG`.

Возвращает:

0 — в случае успешного объединения блоков в один блок биометрических данных, предназначенных для хранения в БД вычислителя и записи его в указанный буфер.

-1 — если при объединении или записи биометрических данных произошла ошибка.

После выполнения функции `makeTransferBlock` полученный блок биометрических данных будет записан в указанный буфер. Блок данных предназначен для помещения его в БД вычислителя и не подлежит какому-либо изменению или дополнению. Кроме того, один и тот же блок данных может быть использован для верификации или идентификации в нескольких независимых БД вычислителей или храниться в произвольной БД.

Получение сообщения о последней ошибке

Функция `get_last_error` возвращает сообщение о последней ошибке, возникшей при выполнении функций SDK.

```
const char* libscaniris_get_last_error(libscaniris_handle_t handle)
```

Входные параметры:

`handle` — полученный при помощи функции `create` дескриптор, указывающий на объект.

Возвращает:

`string` — строка, содержащая описание последней ошибки, в виде значения с типом `const char`.

пустая строка, т.е. `strlen(string)==0` — если произошла ошибка выполнения функции.

Возможные сообщения об ошибках, возвращаемые функцией:

Сообщение	Пояснение
Wrong libscaniris_scandata structure!	Вместо указателя на структуру libscaniris_scaniris_data_t было передано значение NULL
Wrong pointer on buffer!	В поле buf структуры libscaniris_scaniris_data_t содержится нулевой указатель
Wrong hostname!	Отсутствует значение параметра pszHostName
Wrong port!	Значение параметра port установлено равным 0
Two data block with left eye info	Функции makeTransferBlock были переданы два ББД, содержащие биометрические данные только левых глаз
Two data block with right eye info	Функции makeTransferBlock были переданы два ББД, содержащие биометрические данные только правых глаз
Empty call of function	Неверный формат переданных структур libscaniris_scaniris_data_t
Not enough memory to store received data!\n	Не хватает памяти для хранения внутренних данных библиотеки
Wrong block of iris data!\n	При передаче данных между вычислителем и клиентским приложением возникли ошибки.
Wrong structure of iris data!\n	
Error in protocol!	
Button 'Cancel' clicked!	Нажата кнопка Отмена в интерактивном диалоге сканирования РОГ

Получение информации о версии SDK

Функция **get_lib_version** возвращает информацию о версии SDK.

```
int* libscaniris_get_lib_version()
```

Входные параметры: отсутствуют.

Возвращает:

0x00XXYYZZ – шестнадцатеричное число, содержащее номер версии,

где:

- XX – major;
- YY – minor;
- ZZ – patch.

-1 – если произошла ошибка выполнения функции.



Протокол обращения к БД вычислителя

База данных вычислителя представляет собой набор записей, каждая из которых содержит блок биометрических данных, полученный в результате выполнения функции **makeTransferBlock** и код записи **ID_KEY**. Код записи представляет собой 8-байтное беззнаковое целое число и однозначно идентифицирует принадлежность биометрических данных конкретному лицу (конечному пользователю).

Порядок генерации или получения **ID_KEY** может быть произвольным и зависит только от того, в какую систему контроля и управления доступом интегрируется представленный комплект оборудования.

Работа с БД вычислителя производится по сети через интерфейс Ethernet по протоколу **tcp/ip** (сервис запущен на порту **6120**). После установления соединения с вычислителем выдается информационная строка:

```
0 Zircon ethernet configuration protocol\n0 Version 1.0.2\n0
```

```
Ok wait for request\n
```

где: **Version** — версия конфигурационного протокола для Ethernet

Для работы с БД вычислителя используются следующие команды:

db get count	Получить значение общего количества записей в БД вычислителя
db add user	Добавить биометрические данные конечного пользователя в БД вычислителя;
db delete user	Удалить из БД биометрические данные указанного пользователя
db delete all	Очистить БД вычислителя

Общий формат команд, передаваемых вычислителю при использовании протокола Ethernet, имеет вид:

```
<Len>< ><Command Subcommand1....SubcommandN>\n  
[Data]
```

где:

- <Len> — десятичное число в текстовом представлении, содержащее длину массива бинарных данных, передаваемых в поле [Data]. Если данных нет (поле [Data] пустое), то в поле Len устанавливается значение 0.
- < > — символ пробела.
- <Command Subcommand> — команда работы с БД вычислителя. Команда может содержать дополнительный параметр в виде шестнадцатеричного числа в текстовом представлении.
- \n — перенос строки.
- [Data] — БДД либо иная информация в зависимости от передаваемой команды.

Внимание! При перезагрузке вычислителя база записей биометрических данных очищается, и ее требуется загружать заново.

Подсчет записей, содержащихся в БД вычислителя

Для получения общего количества записей, содержащихся в БД вычислителя, используется команда **db get count**:

```
0 db get count\n
```

После выполнения команды вычислитель произведет подсчет количества записей, хранящихся в БД, и выдаст следующую информацию:

1. 0 ok get count <DbSize>\n — в БД содержится определенное количество записей.

где: <DbSize> — количество записей, содержащееся в БД, в виде 4-байтового беззнакового целого числа. Число DbSize представлено в шестнадцатеричном виде в текстовом представлении и включает ведущие нули, т.е. состоит из восьми символов, независимо от своего значения.

2. 0 error get count\n — произошла ошибка при подсчете количества записей.

Например.

```
0 db get count\n
```

где: db get count — команда работы с БД вычислителя (подсчет записей, содержащихся в БД вычислителя).

После выполнения команды будет выдана следующая информация:

1. 0 ok get count 0000020D\n

где:

- ok get count — информация, подтверждающая, что подсчет записей прошел успешно.
- 0000020D — 4-байтовое беззнаковое целое число, говорящее о том, что в БД вычислителя содержится 525 записей.

2. 0 error get count\n

где: error get count — информация, говорящая о том, что количество записей не подсчитано.

Добавление нового пользователя

Добавление нового пользователя БД вычислителя производится при помощи команды **db add user**:

```
<Len> db add user <ID_KEY>\n
```

```
<KEY_DATA>
```

где:

- <Len> — количество байт блока данных <KEY_DATA>, представленное в виде десятичного числа в текстовом представлении.
- <ID_KEY> — 8-байтный код конечного пользователя, соответствующий передаваемому в БД вычислителя массиву данных <KEY_DATA>. Каждый байт ID_KEY должен быть представлен в виде шестнадцатеричного числа в текстовом представлении и включать, если требуется, ведущие нули. Порядок следования байтов: старший байт — первый.
- <KEY_DATA> — массив данных, содержащий блок биометрических данных пользователя, соответствующий передаваемому 8-байтному коду <ID_KEY>.

После выполнения команды вычислитель произведет добавление записи пользователя, содержащей его 8-байтный код и биометрические данные, в БД вычислителя и выдаст следующую информацию:

1. 0 ok add user <ID_KEY>\n — запись успешно добавлена.

где: <ID_KEY> — 8-байтный код добавленного в БД пользователя в виде шестнадцатеричного числа в текстовом представлении.

2. 0 error add user <ID_KEY>\n — ошибка добавления записи.

где: <ID_KEY> — 8-байтный код записи пользователя в виде шестнадцатеричного числа в текстовом представлении, биометрические данные которого не добавлены в БД.

Например:

```
1230 db add user 00000000AF23BC03\nBC0DAA54FA527102D42A059641A853...
```

где:

- 1230 — <Len> в виде десятичного числа в текстовом представлении.
- db add user — команда работы с БД вычислителя (добавление нового пользователя в БД вычислителя).
- 00000000AF23BC03 — 8 байт кода пользователя <ID_KEY> в виде шестнадцатеричного числа в текстовом представлении.
- BC0DAA54FA527102D42A059641A853... — массив данных <KEY_DATA>.

После выполнения команды будет выдана следующая информация:

```
1. 0 ok add user 00000000AF23BC03\n
```

где:

- ok add user — информация, подтверждающая успешное добавление записи в БД вычислителя.
- 00000000AF23BC03 — 8 байт кода пользователя <ID_KEY> в виде шестнадцатеричного числа в текстовом представлении, биометрические данные которого добавлены в БД вычислителя.

```
2. 0 error add user 00000000AF23BC03\n
```

где:

- error add user — информация, говорящая о том, что запись не добавлена в БД вычислителя.
- 00000000AF23BC03 — 8 байт кода пользователя <ID_KEY> в виде шестнадцатеричного числа в текстовом представлении, биометрические данные которого не добавлены в БД вычислителя.

Удаление пользователя

Удаление из БД вычислителя биометрических данных пользователя, имеющего определенный <ID_KEY>, производится при помощи команды **db delete user**:

```
0 db delete user <ID_KEY>\n
```

где: <ID_KEY> — 8-байтный код пользователя, биометрические данные которого необходимо удалить. Каждый байт ID_KEY должен быть представлен в виде шестнадцатеричного числа в текстовом представлении и включать, если требуется, ведущие нули.

После выполнения команды будет произведено удаление записи, содержащей биометрические данные пользователя, имеющего соответствующий <ID_KEY>, и будет передана следующая информация:

1. 0 ok delete user <ID_KEY>\n — запись успешно удалена.

где: <ID_KEY> — 8-байтный код удаленного из БД пользователя в виде шестнадцатеричного числа в текстовом представлении.

2. 0 error delete user <ID_KEY>\n — запись с указанным <ID_KEY> в БД вычислителя отсутствует или произошла ошибка удаления записи.

где: <ID_KEY> — 8-байтный код записи пользователя в виде шестнадцатеричного числа в текстовом представлении, данные которой не удалены или не найдены в БД.

Например:

```
0 db delete user 00000000AF23BC03\n
```

где:

- **db delete user** — команда работы с БД вычислителя (удаление пользователя из БД вычислителя).
- **00000000AF23BC03** — 8 байт кода пользователя <ID_KEY> в виде шестнадцатеричного числа в текстовом представлении, биометрические данные которого необходимо удалить из БД вычислителя.

После выполнения команды будет выдана следующая информация:

```
1. 0 ok delete user 00000000AF23BC03\n
```

где:

- **ok delete user** — информация, подтверждающая успешное удаление записи из БД вычислителя.
- **00000000AF23BC03** — 8 байт кода пользователя <ID_KEY> в виде шестнадцатеричного числа в текстовом представлении, биометрические данные которого удалены из БД вычислителя.

2. 0 error delete user 00000000AF23BC03\n

где:

- error delete user — информация, говорящая о том, что запись пользователя не удалена из БД вычислителя.
- 00000000AF23BC03 — 8 байт кода пользователя <ID_KEY> в виде шестнадцатеричного числа в текстовом представлении, биометрические данные которого не удалены из БД вычислителя.

Очистка БД вычислителя

Для удаления всех записей БД вычислителя используется команда **db delete all**.

0 db delete all\n

После выполнения команды все данные из БД вычислителя будут удалены, и будет передана следующая информация:

1. 0 ok delete all\n — БД вычислителя очищена.
2. 0 error delete all\n — при очистке БД вычислителя произошла ошибка.

Протокол обмена данными при верификации/идентификации

Управление процессом верификации или идентификации конечного пользователя, находящегося непосредственно перед иридосканером, осуществляется командами, передаваемыми с использованием последовательного порта, следующим образом:

1. Управляющее устройство:

- a. Иницирует процесс верификации или идентификации на вычислителе путем передачи на него соответствующего пакета данных:
 - для инициализации процесса верификации на вычислитель передается код пользователя
 - для инициализации процесса идентификации — 8-байтное число, каждый байт которого имеет значение 0xFF
- b. С указанной периодичностью запрашивает статус выполняемого процесса (верификации или идентификации);

2. Вычислитель:

- a. Обрабатывает полученный пакет данных, и либо:
 - выбирает из БД биометрические данные лица, согласно полученному коду, и сравнивает их с биометрическими данными, полученными с иридосканера;
 - поочередно сравнивает биометрические данные лиц из БД с биометрическими данными, полученными с иридосканера;
- b. После завершения процесса обработки данных и, получив очередную команду запроса статуса, выдает результат.

Задачей устройства, управляющего процессом верификации/идентификации и подключенного к вычислителю через последовательный порт, является:

1. Формирование пакета данных, содержащего код пользователя либо 8-байтное число, каждый байт которого имеет значение 0xFF, и передача его на вычислитель;

2. Формирование запросов для проверки статуса выполнения процесса верификации/идентификации;
3. Обработка результатов верификации/идентификации, полученных в виде статуса процесса .

Обмен информацией между системой, управляющей процессом верификации/идентификации, и вычислителем осуществляется только в режиме запрос – ответ. Т.е. информация о статусе выполнения верификации/идентификации на вычислителе будет получена системой только после отправки ему соответствующего запроса.

Формирование пакета данных

Общий формат пакета данных, передаваемых на вычислитель, имеет следующий вид:

START	ADDR	KOP	LEN	DATA	CRC	STOP
-------	------	-----	-----	------	-----	------

где:

- **START** — маркер начала пакета 0x80 (размер поля 1 байт)
- **ADDR** — адрес устройства на шине RS-232, которому предназначается пакет — 0÷255 (размер поля 1 байт)
- **KOP** — код операции (команда) — 0÷255 (размер поля 1 байт)
- **LEN** — суммарный размер блока данных, передаваемых в поле **DATA**, и один байт, соответствующий размеру данных, передаваемых в поле **CRC** (размер поля 1 байт)
- **DATA** — необязательный блок данных произвольного размера
- **CRC** — контрольная сумма пакета (размер поля 1 байт)
- **STOP** — маркер конца пакета 0x81 (размер поля 1 байт)

Пакет, передаваемый на вычислитель, формируется в следующем порядке:

1. В поле **ADDR** указывается адрес вычислителя, для которого предназначен пакет. Вычислитель может быть интегрирован в сеть, построенную на базе интерфейса RS-485 через преобразователь RS-232/RS-485, при этом в поле **ADDR** должен быть указан адрес именно того вычислителя, перед иридосканером которого находится проверяемое лицо.
2. В поле **KOP** указывается код команды, которую должен выполнить вычислитель:
 - 0x01 — вернуть статус выполнения верификации/идентификации;
 - 0x02 — произвести верификацию/идентификацию;
 - 0x03 — остановить процесс верификации/идентификации.

3. В поле DATA заносится блок передаваемых данных. Если формируется пакет для запроса статуса вычислителя или пакет для остановки процесса верификации/идентификации, то данное поле не заполняется.
4. В поле LEN заносится значение, равное количеству байтов, содержащихся в блоке данных, если они присутствуют в запросе. В случае отсутствия данных в поле DATA значение, устанавливаемое в поле LEN, равно 0x01.
5. В поле CRC заносится контрольная сумма пакета, вычисляемая по формуле:

$$\text{CRC} = 0x00 - (\text{ADDR} + \text{KOP} + \text{LEN} + \text{SUM}(\text{DATA})).$$

где:

- ADDR — адрес устройства, которому предназначается пакет.
 - KOP — код операции.
 - LEN — длина блока данных.
 - SUM(DATA) — сумма байт кода верифицируемого/идентифицируемого лица (может отсутствовать)
6. Значение, содержащееся в поле LEN, увеличивается на один байт (добавлен размер поля CRC). В случае отсутствия данных в поле DATA значение, устанавливаемое в поле LEN, всегда устанавливается равным 0x01.
 7. Полученный блок данных проверяется на наличие байтов, идентичных служебным маркерам (0x80, 0x81, 0x82). Если в блоке присутствуют байты, идентичные служебным маркерам, то они заменяются на следующие последовательности байтов:
 - 0x80 заменяется на 0x82 0x00
 - 0x81 заменяется на 0x82 0x01
 - 0x82 заменяется на 0x82 0x02

Таким образом, формируется блок данных, не содержащий служебные маркеры начала и конца пакета. На принимающей стороне происходит обратное преобразование таких последовательностей (т.е. 0x82 0x00 в 0x80, 0x82 0x01 в 0x81, 0x82 0x02 в 0x82), а затем следует разбор пакета.

8. В поле START заносится служебный маркер начала пакета — 0x80.
9. В поле STOP заносится служебный маркер конца пакета — 0x81.

Пакет данных, отправляемый вычислителем, преобразовывается аналогичным образом, поэтому, перед анализом полученного пакета, его необходимо преобразовать в исходный вид и проверить контрольную сумму.

Запрос статуса процесса выполнения верификации/идентификации

Запрос статуса формируется командой 0x01, указанной в поле КОР (поле DATA не заполняется).

После получения запроса на возвращение статуса, вычислитель формирует пакет, где в поле DATA присутствует байт, содержащий информацию о статусе выполнения верификации/идентификации и при успешном выполнении идентификации – 8-битный код пользователя .

Байт статуса содержит следующие биты:

Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
				STAT 3	STAT 2	STAT 1	STAT 0

где:

STAT 3, STAT 2, STAT 1 и STAT 0 — биты статуса

Байт статуса может принимать следующие значения:

- 0000 — состояние ожидания команд
- 0001 — идет поиск объекта
- 0010 — верификация успешно завершена
- 0011 — ошибка верификации/идентификации (объект не совпал)
- 0100 — ошибка (ID_KEY нет в БД)
- 0101 — истечение таймаута
- 0110 — процесс принудительно остановлен
- 0111 — идентификация успешно завершена

После получения статуса выполнения верификации или идентификации, отличного от 0000, 0001 или 0110 — в следующем пакете запроса статуса данных необходимо выслать байт подтверждения приема. Байт подтверждения приема записывается в поле DATA и имеет значение 0x00. После передачи подтверждения статус сбрасывается на 0000 (состояние ожидания команд).

При положительном завершении процедуры идентификации вместе с байтом статуса в поле DATA передается 8-байтный код найденного пользователя (ID_KEY). Первым передается старший байт.

Пример:

Отправлен запрос статуса	0x80,0x00,0x01,0x01,CRC,0x81
Получен статус “ошибка сравнения”	0x80 0x00,0x01,0x02,0x03,CRC,0x81
Отправлен пакет с байтом подтверждения	0x80,0x00,0x01,0x02,0x00,CRC,0x81
Получен статус “ожидание команд”	0x80,0x00,0x01,0x02,0x00,CRC,0x81

Запуск процесса верификации/идентификации

Запуск процесса верификации и идентификации осуществляется командой 0x02, указанной в поле KOP пакета данных, передаваемого на вычислитель.

Информация в поле DATA может быть представлена двумя способами. В первом случае явно указывается значение периода времени, отводимого для получения одного изображения РОГ, а во втором случае период времени не указывается (сканирование будет производиться в течение 60 секунд) и, соответственно, длина поля DATA будет меньше на два байта.

Кроме того, для запуска процесса верификации в поле DATA должен содержаться 8-байтный код пользователя, а для запуска процесса идентификации – 8-байтное число, каждый байт которого имеет значение 0xFF. При передаче кода пользователя первым передается старший байт.

1. Вариант с явным указанием периода сканирования

START	ADDR	KOP	LEN	DATA		CRC	STOP
				TIME_OUT	ID_KEY		

В приведенных примерах в поле DATA содержатся следующие значения:

- TIME_OUT – значение периода времени (от 1 до 600 секунд), отводимого для получения одного изображения РОГ (размер поля 2 байта)
- ID_KEY – код конкретного пользователя, находящегося непосредственно перед иридосканером (**пример 1**), либо 8-байтное число, каждый байт которого имеет значение 0xFF (**пример 2**). Размер поля 8 байт.

Пример 1 – запуск процесса верификации:

В поле DATA занесено 2-байтное значение периода времени – 0x00, 0x78 (период сканирования равен 120 секунд) и 8-байтный код верифицируемого лица – 0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8.

Отправлена команда на запуск верификации	0x80,0x00,0x02,0x0B,0x00,0x78,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,CRC,0x81
Получен статус “идет поиск объекта”	0x80,0x00,0x02,0x02,0x01,CRC,0x81

Пример 2 – запуск процесса идентификации

В поле DATA занесено 2-байтное значение периода времени – 0x00, 0xB4 (период сканирования равен 180 секунд) и 8-байтное число – 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF.

Отправлена команда на запуск идентификации	0x80,0x00,0x02,0x0B,0x00,0xB4,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,CRC,0x81
Получен статус “идет поиск объекта”	0x80,0x00,0x02,0x02,0x01,CRC,0x81

2. Вариант без указания периода сканирования

START	ADDR	KOP	LEN	DATA	CRC	STOP
				ID_KEY		

В приведенных примерах в поле DATA содержится только ID_KEY – код конкретного пользователя, находящегося непосредственно перед иридо-сканером (пример 1), либо 8-байтное число, каждый байт которого имеет значение 0xFF (пример 2). Размер поля 8 байт. Сканирование будет производиться в течение 60 секунд.

Пример 1 – запуск процесса верификации:

В поле DATA занесен только 8-байтный код верифицируемого лица – 0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8.

Отправлена команда на запуск верификации	0x80,0x00,0x02,0x09,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,CRC,0x81
Получен статус “идет поиск объекта”	0x80,0x00,0x02,0x02,0x01,CRC,0x81

Пример 2 – запуск процесса идентификации

В поле DATA занесено только 8-байтное число – 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF.

Отправлена команда на запуск идентификации	0x80,0x00,0x02,0x09,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,CRC,0x81
Получен статус “идет поиск объекта”	0x80,0x00,0x02,0x02,0x01,CRC,0x81

Остановка процесса верификации/идентификации

Остановка процесса верификации/идентификации осуществляется командой 0x03, указанной в поле KOP пакета данных, передаваемого на вычислитель.

Пример:

Отправлена команда на остановку процесса верификации/идентификации	0x80,0x00,0x03,0x01,CRC,0x81
Получен статус “процесс принудительно остановлен”	0x80,0x00,0x03,0x02,0x06,CRC,0x81



Настройки системы верификации/ идентификации по радужной оболочке глаз

Вычислитель — электронное вычислительное устройство, предназначенное для хранения и обработки биометрической информации, поступающей с иридосканера или внешней БД, а также для обработки запросов на верификацию или идентификацию личности по локальной БД вычислителя.

Настройка параметров сетевого доступа к вычислителю по сети Ethernet

По умолчанию на вычислителе установлены следующие сетевые параметры:

IP-адрес	192.168.1.2
Маска подсети	255.255.255.0
Основной шлюз	192.168.1.1

Изменение параметров сети (IP-адреса, маски подсети, адреса основного шлюза и т.д.), установленных по умолчанию, может быть произведено как локально, непосредственно на вычислителе, так и удаленно, используя протокол SSH следующим образом:

1. Откройте терминал и зарегистрируйтесь пользователем **root**.
2. Подмонтируйте раздел **sda5** с файлами конфигурации в режим чтения-записи, для этого введите команду:
`mount /dev/sda5 -o rw /mnt/config`
3. Перейдите в каталог `/mnt/config/` при помощи команды:
`cd /mnt/config`

4. В файл `/etc/network/interfaces` внесите требуемые изменения параметров сети и сохраните его, например:

```
iface eth0 inet static
address 192.168.21.16
netmask 255.255.255.0
gateway 192.168.21.1
dns-search papillon.ru
dns-nameservers 192.168.21.2
```

5. Запустите скрипт для перерасчета контрольных сумм командой:

```
./gen_md5.sh
```

По завершении генерации сумм на экран будет выведено сообщение “Generation MD5 complete”

6. Размонтируйте раздел командой:

```
umount /dev/sda5
```

7. Произведите перезагрузку вычислителя командой:

```
/sbin/reboot
```

Настройка адреса вычислителя на шине RS 232/485

Установка (изменение) адреса вычислителя на шине RS 232/485 может быть произведено как локально, непосредственно на вычислителе, так и удаленно, используя протокол SSH следующим образом:

1. Откройте терминал и зарегистрируйтесь пользователем **root**.
2. Подмонтируйте раздел **sda5** с файлами конфигурации в режим чтения-записи, для этого введите команду:

```
mount /dev/sda5 -o rw /initrd
```
3. Откройте файл `/initrd/cfg/zircon.cfg` в режиме редактирования.
4. Измените значение параметра `SerialAddress=N`, где N — адрес вычислителя на шине RS 232/485 (значение от 0 до 255).
5. Сохраните изменения.
6. Размонтируйте раздел командой:

```
umount /dev/sda5
```
7. Произведите перезагрузку вычислителя командой:

```
/sbin/reboot
```

Обновление программного обеспечения вычислителя

Обновление программного обеспечения поставляется в виде следующих файлов:

1. `ramdisk.img.upd` — файл обновлений ПО.
2. `ramdisk.md5` — файл контрольной суммы.

Для проведения процедуры обновления используются скрипты `mount_fs.sh`, `umount_fs.sh` и `update.sh`, хранящиеся в каталоге `/etc/update/` вычислителя.

Обновление программного обеспечения может быть произведено как локально, непосредственно на вычислителе, так и удаленно, используя протокол **SSH** следующим образом:

1. Откройте терминал и зарегистрируйтесь пользователем **root**.
2. Из каталога `/etc/update/` запустите скрипт `mount_fs.sh` командой:

```
/etc/update/mount_fs.sh
```

Будет подмонтирован требуемый раздел файловой системы.

В случае ошибки монтирования будет выведено соответствующее сообщение (возможные ошибки — отсутствуют требуемые для работы скрипта программы или каталоги, файловая система уже смонтирована, ошибка монтирования и т.д.).

3. Скопируйте файл обновлений ПО и файл контрольной суммы в каталог `/mnt/config/boot/`.
4. Запустите скрипт `update.sh` командой:

```
/etc/update/update.sh ramdisk.img.upd ramdisk.md5
```

где:

- `ramdisk.img.upd` — имя файла обновлений (путь к файлу не указывается).
- `ramdisk.md5` — имя файла контрольной суммы (путь к файлу не указывается).

При удачном обновлении программного обеспечения удаляются файлы обновлений и контрольной суммы из каталога `/mnt/config/boot/`, а подмонтированный раздел размонтируется.

5. Произведите перезагрузку вычислителя командой:

```
/sbin/reboot
```

Внимание!

Если по какой-либо причине обновление не было произведено, подмонтированный раздел должен быть размонтирован, для этого используется скрипт `umount_fs.sh`. Для запуска скрипта выполните команду:

```
/etc/update/umount_fs.sh
```

В случае ошибки будет выведено соответствующее сообщение (возможные ошибки – отсутствуют требуемые для работы скрипта программы или каталоги, файловая система не была смонтирована, ошибка размонтирования и т.д.).

Содержание

Назначение	3
------------------	---

ФУНКЦИИ SDK

Библиотека функций SDK	5
Создание объекта и получение его дескриптора	6
Удаление объекта	6
Получение блока данных одного глаза	6
Объединение двух блоков данных	8
Получение сообщения о последней ошибке	9
Получение информации о версии SDK	11

ПРОТОКОЛЫ ВЗАИМОДЕЙСТВИЯ С ВЫЧИСЛИТЕЛЕМ

Протокол обращения к БД вычислителя	13
Подсчет записей, содержащихся в БД вычислителя	14
Добавление нового пользователя	15
Удаление пользователя	17
Очистка БД вычислителя	18
Протокол обмена данными при верификации/идентификации	19
Формирование пакета данных	20
Запрос статуса процесса выполнения верификации/идентификации	22
Запуск процесса верификации/идентификации	23
Вариант с явным указанием периода сканирования	23
Вариант без указания периода сканирования	24
Остановка процесса верификации/идентификации	25

ПРИЛОЖЕНИЕ

Настройки системы верификации/идентификации по радужной оболочке глаз	26
Настройка параметров сетевого доступа к вычислителю по сети Ethernet ...	26
Настройка адреса вычислителя на шине RS 232/485	27
Обновление программного обеспечения вычислителя	28