Руководство по скриптам Lua

Одной из отличительных особенностей всех продуктов комплекса НЕЙРОСС и, в частности, IP-контроллеров БОРЕЙ, ЯРС v.1 (снят с производства!) и других является возможность модификации и расширения алгоритмов работы продукта посредством *скриптов автоматизации*. С помощью скрипта возможно, например, изменить или полностью заменить алгоритм доступа в контроллере БОРЕЙ. Или реализовать логику сопряжения со внешней информационной системой по протоколу HTTP. Или управлять исполнительными устройствами по заданному алгоритму.

В данном документе приведены общие сведения о скриптах автоматизации, их возможностях и руководство программиста по написанию таких скриптов.

> ⚠ **ВНИМАНИЕ**
>
> Скрипт автоматизации может существенно изменить «заводскую» логику работы продукта. Потому разработку скрипта рекомендуется проводить только опытному пользователю. Производитель продукта не несёт ответственности за несоответствие реальных и заявленных свойств продукта в случае, если такое расхождение возникло вследствие внесения изменений в скрипт автоматизации.

Оглавление

Общие сведения

Скрипт автоматизации — текстовый сценарий, выполнение которого осуществляется программными средствами продукта.

Данный сценарий всегда один. Он может быть пустым — в таком случае скрипт не делает ничего. В скрипте могут быть команды управления — они выполняются по мере выполнения скрипта.

Скрипт автоматизации должен быть написан на языке Lua (версия 5.1). Разработчику доступны стандартные средства языка (операторы, структуры данных и др.), часть функций стандартной библиотеки (см. далее), а также наборы констант и функций, определённых в *прикладных Lua-модулях* продукта. Прикладные Lua-модули специально разработаны для упрощения написания скриптов и двустороннего взаимодействия с программными средствами продукта — чтобы можно было из скрипта вызывать какие-либо функции продукта или, наоборот, передавать управление в скрипт в процессе функционирования продукта.

> В разных продуктах могут быть доступны разные прикладные модули. Так, модуль управления доступом pacs.lua недоступен (не работает) в ИГНИС и, наоборот, функции ИГНИС в модуле ignis.lua не работают в контроллере БОРЕЙ.

Скрипт автоматизации выполняется в «песочнице» — защищённом окружении. Если в ходе работы скрипта возникает ошибка, то выполнение скрипта просто прерывается — нарушается работа функций, так или иначе связанных со скриптами (например, алгоритмов доступа), НО общая работоспособность продукта сохраняется, на стабильность работы программных средств поведение скрипта не влияет.

> **ВНИМАНИЕ**
>
> В случае IP-контроллеров (БОРЕЙ, ИГНИС и пр.) скрипт автоматизации является изменяемыми данными также, как любые другие конфигурационные данные продукта (как, например, настройки сетевых интерфейсов или параметры доступа). Потому после внесения изменений в скрипт, успешной отладки и апробации сценария ОБЯЗАТЕЛЬНО создавайте резервную копию. Резервная копия будет содержать скрипт автоматизации, и в случае, если файловая система на SD-карте вдруг будет повреждена, скрипт будет восстановлен вместе с остальными конфигурационными данными из резервной копии.

Как редактировать скрипт

Для удобство редактирования и отладки скрипта автоматизации предназначен текстовый online-редактор, доступный в наших продуктах по адресу http://[ip-address]/lua-editor/ или http://[ip-address]/internal/lua-editor/.

1. В левом блоке — текст скрипта автоматизации. Внесите в него изменения, для сохранения нажмите на кнопку «Сохранить» внизу.
2. В правом блоке можно просматривать lua-код модулей и сниппеты — примеры скриптов автоматизации. Для просмотра сниппета или модуля выберите его в раскрывающемся списке под правым блоком.
   a. Lua-код модулей обычно содержит определение констант, структур, функций, а также обеспечивает удобную для разработчика «обёртку» на доступными функциями основных программных средств.
   ЗАМЕЧАНИЕ: в списке обычно можно посмотреть Lua-код всех модулей

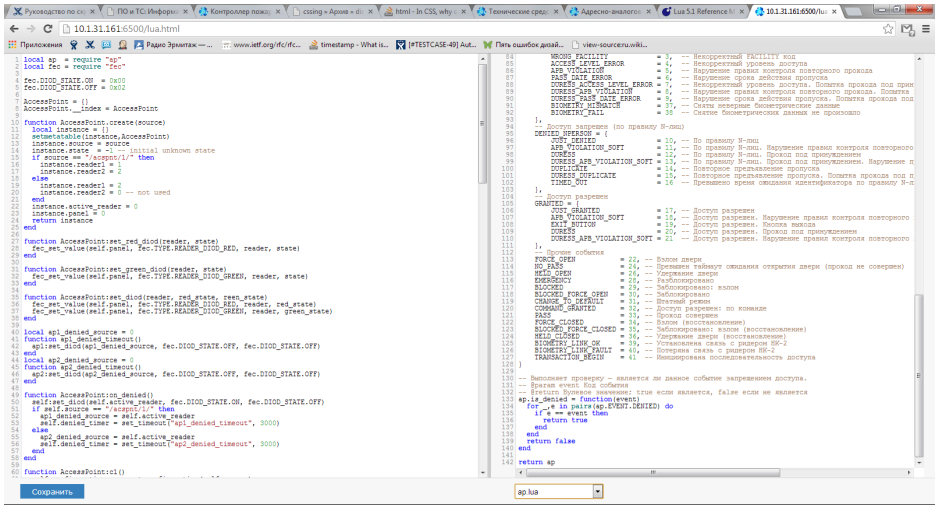
всех продуктов, их можно подключить к скрипту автоматизации, но фактически «отсутствующий» модуль просто не будет работать.

   b. Сниппеты — блоки кода, из которых можно почерпнуть идеи при разработке скрипта. Или можно скопировать «куски» сниппета в свой скрипт. Библиотека сниппетов постоянно пополняется разработчиками с добавлением примеров создания скриптов автоматизации.

3. При сохранении скрипта автоматизации выполнение текущего скрипта будет автоматически прервано. После чего будет запущен новый скрипт.

> **ВНИМАНИЕ**
>
> При сохранении скрипта автоматизации полностью перезапускается вся виртуальная Lua-машина, в следствие чего перезапускаются и системные Lua-скрипты — такие, например, как алгоритмы управления доступом и другие.

Скриншот редактора скриптов приведён на изображении ниже.

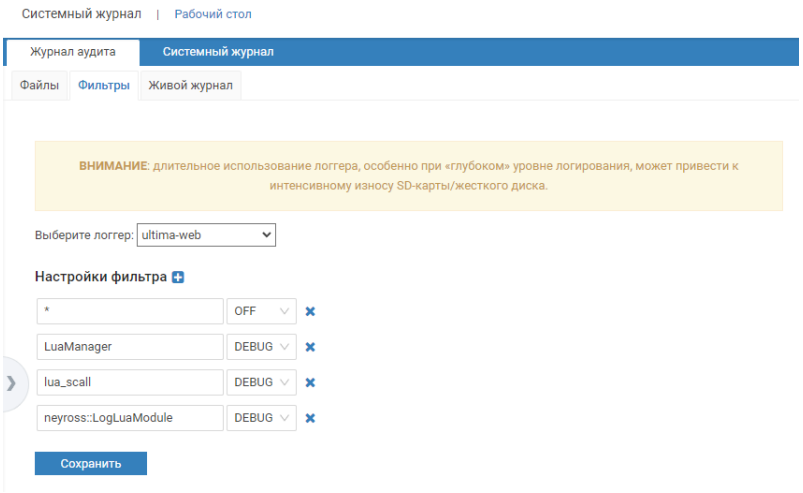


Как отлаживать скрипт

Сообщения об ошибках в Lua-скрипте, а также отладочные сообщения, выводимые из скрипта с помощью модуля log.lua, записываются в журнал аудита. Просмотр этих сообщений в реальном времени возможен при помощи веб-приложения «Журнал аудита», доступного с веб-страницы Рабочий стол.

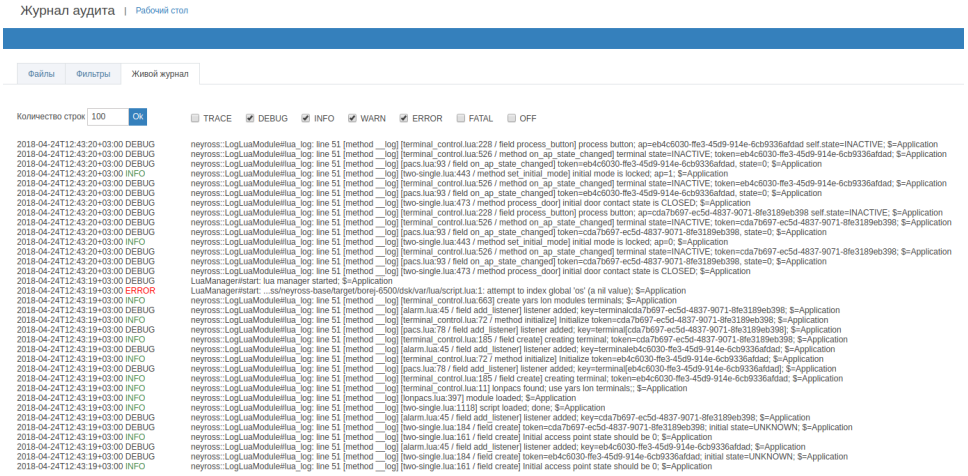
1. Авторизуйтесь в веб-интерфейсе и перейдите на Рабочий стол.
2. Перейдите в веб-приложение «Журнал аудита».
3. На второй вкладке «Фильтры» выберите в списке логгер web.
4. В настройках фильтрации укажите следующие строки (первая строка:
   a. `* — OFF`
   b. `LuaManager — DEBUG`
   c. `lua_scall — DEBUG`
   d. `neyross::LogLuaModule — DEBUG`

5. Пример корректно настроенного логгера:



6. Перейдите на третью вкладку «Живой журнал».
7. Установите флаги DEBUG, INFO, WARN, ERROR для вывода сообщений всех возможностей уровней тревожности.

8. При выполнении Lua-скрипта на странице будут появляться соответствующие сообщения. Более свежие сообщения выводятся сверху.



Руководство разработчика

Общие сведения о Lua

Скрипт автоматизации должен быть написан на языке Lua версии 5.1.

Официальный сайт языка Lua (на английском языке) — http://www.lua.org/

Официальная документация по языку Lua версии 5.1 — http://www.lua.org/manual/5.1/ (английский язык)

Перевод документации по языку Lua версии 5.1 на русский язык: http://www.lua.ru/doc/

Также можно ознакомиться с документацией на русском языке для последней актуальной версии Lua (5.3 на момент написания статьи) по ссылке http://lua.org.ru/contents_ru.html. Большая часть содержимого также актуальна и для версии 5.1, поддерживаемой продуктами НЕЙРОСС.

Ограничения стандартной библиотеки

В скриптах автоматизации доступны следующие разделы стандартной библиотеки:

- Basic Functions
- Coroutine Manipulation
- Modules
- String Manipulation
- Table Manipulation
- Mathematical Functions

Следующие разделы недоступны (не могут быть использованы) в скрипте автоматизации:

- Input and Output Facilities
- Operating System Facilities

Прикладные Lua-модули

В таблице ниже перечислены прикладные модули, их описание и продукты, в которых эти модули могут быть использованы.

Модуль	Продукты	Краткое описание
alarm.lua	БОРЕЙ ЯРС	Модуль интеграции с подсистемой охранно-тревожной сигнализации контроллера.
common.lua	Все продукты	Модуль общесистемных функций. Получение системных даты / времени, генерация UUID-ов и так далее.
deviceio.lua	БОРЕЙ ЯРС	
fec.lua	БОРЕЙ ЯРС	Модуль взаимодействия с периферией на контроллерах БОРЕЙ и ЯРС. Модуль позволяет: 1. Получать состояние входов. 2. Управлять выходами (рекомендуется управлять выходами через deviceio-сервис). 3. Управлять светодиодной и звуковой индикацией на Wiegand-считывателях.
http.lua	Все продукты	Модуль поддержки HTTP-протокола; модуль позволяет: 1. Отправлять HTTP-запросы на указанный адрес. 2. Асинхронно обрабатывать ответы на отправленные ранее HTTP-запросы.
ignis.lua	ИГНИС	

log.lua	Все продукты	Модуль аудита позволяет выводить в журнал аудита текстовые сообщения с заданным уровнем логирования
longate.lua	ЯРС	
loninput.lua	ЯРС	
lonpacs.lua	ЯРС	
pacs.lua	БОРЕЙ ЯРС	Модуль интеграции с подсистемой доступа в контроллерах БОРЕЙ, ЯРС. Модуль позволяет реализовать свой алгоритм доступа вместо предустановленного или внести изменения в стандартные алгоритмы, переопределив отдельные части алгоритма.
storage.lua	Все продукты	Модуль для долговременного сохранения настроек / параметров скрипта автоматизации в форме ключ=значение
tenso.lua		
timer.lua	Все продукты	Асинхронное выполнение отложенных по времени или периодических действий. ⚠ Для данного модуля отсутствует lua-файл, соответствующий код недоступен для просмотра в веб-редакторе скрипта автоматизации.
terminal.lua		

СОВЕТ

Каждый прикладной Lua-модуль обычно сопровождается соответствующим lua-файлом, который можно просматривать в правой части веб-редактора скрипта автоматизации. В таких lua-файлах каждая функция, константа или перечисление обычно сопровождаются подробными комментариями с указанием назначения и способа использования соответствующего кода.

Подключение прикладного Lua-модуля осуществляется функцией require. Рекомендуемая форма использования:

```
local log = require("log")
```

Вызвать функцию Lua-модуля в таком случае можно следующим образом, например:

```
local logger = log.get_logger("MyLogger")
```

alarm.lua

Описание временно недоступно.

common.lua

Данный модуль предоставляет общесистемные функции. Перечень функций приведён в таблице ниже.

Функция	Аргументы	Результат	Описание
`now()`	нет	число с плавающей точкой	Возвращает число с плавающей точкой, соответствующее количеству секунд, прошедшему с 1970 года (с точностью до миллисекунд)
`uuid()`	нет	строка	Генерирует уникальный идентификатор по стандарту ISO. Возвращает строковое представление идентификатора, например: 550e8400-e29b-41d4-a716-446655440000
`localtime(t)`	число	таблица Lua	Для переданного числа возвращает структуру данных с полями для года, месяца, дня и так далее. В аргументе передать число с плавающей точкой, соответствующее количеству секунд, прошедшему с 1970 года (с точностью до миллисекунд). Структура данных соответствует tm (POSIX), см. https://linux.die.net/man/3/localtime_r, к структуре добавлено поле ms с количеством миллисекунд.
`mktime(tm)`	таблица Lua	число	Принимает на вход структуру tm (POSIX; см. https://linux.die.net/man/3/localtime_r) в форме Lua-таблицы и возвращает число с плавающей точкой, равное количеству секунд, прошедшему с 1970 года (с точностью до миллисекунд)
`iso8601_print(t)`	число	строка	Для переданного числа (формат см. common.now()) возвращает строку представления даты и времени в формате ISO8601.

			В аргументе передать число с плавающей точкой, соответствующее количеству секунд, прошедшему с 1970 года (с точностью до миллисекунд).
`iso8601_parse(s)`	строка	число	Принимает на вход строку в формате ISO8601 и возвращает соответствующее данной строке число с плавающей точкой, равное количеству секунд, прошедшему с 1970 года (с точностью до миллисекунд)

deviceio.lua

Описание временно недоступно.

fec.lua

Данный модуль позволяет управлять выходами (на плате контроллера БОРЕЙ, ЯРС, а также на модулях расширения АМ-06) и световой / звуковой индикацией подключенных к контроллеру считывателей.

В перечислении fec.TYPE описаны различные типы периферии. Для управления пригодны типы fec.TYPE.RELE (выход на плате контроллера) и fec.TYPE.SART_RELE (выход на модуле расширения). Для управления выходами следует использовать функцию set_value(тип периферии, ID элемента, состояние), где в первый параметр необходимо передать одно из значений перечисления fec.TYPE, во второй параметр — идентификатор (адрес) элемента для управления, в третий параметр — требуемое состояние (для выходов это 0 = Выключено, 1 = Включено). Например:

```
local fec = require("fec")
fec.set_value(fec.TYPE.RELE, 1, 1)
```

В этом примере выполняется включение первого реле на плате контроллера.

Нумерация выходов начинается с 1. Адрес выхода на модуле расширения определяется его адресом на шине.

К контроллеру БОРЕЙ (и ЯРС) можно подключить два считывателя, на плате есть выходы для управления красным, зелёным светодиодами и звуковой индикацией. Управлять светодиодами и бипером можно также с помощью функции set_value. Возможные режимы индикации светодиода определены в перечислении fec.DIOD. Доступно некоторое количество предустановленных режимов различной частоты и скважности. Звуковую индикацию можно только либо включить, либо выключить (см. перичесление fec.BEEPER).

Поскольку управление светодиодной / звуковой индикацией на считывателях осуществляется в рамках алгоритмов доступа и состояние красного, зелёного светодиодов и звука взаимосвязано (если красный светодиод горит, то зелёный погашен и др.) — для удобства управления всей индикацией реализована функция-обёртка fec.set_reader(ID считывателя, зелёный светодиод, красный светодиод, звук). Пример использования функции приведён ниже:

```
local fec = require("fec")
fec.set_reader(1, fec.DIOD.MODE_OFF, fec.DIOD.MODE_025_025, fec.BEEPER.
MODE_OFF)
```

http.lua

Модуль позволяет отправлять HTTP-запросы прямо из Lua-скрипта, а также асинхронно обрабатывать ответы на отправленные ранее запросы.

Для отправки запроса и обработки ответа необходимо:

1. Определить объект-обработчик ответа (Lua-таблицу) с методом `on_http_response( )`.
2. Создать объект-запрос посредством вызова функции-фабрики `http.with(-)`. Функция возвращает создаёт и возвращает объект-запрос.
3. Вызывать на объекте-запросе метод `transmit( )`. Запрос будет отправлен в соответствии с заданными параметрами запроса. По мере получения ответа (или если запрос не удастся) будет асинхронно вызван метод `on_http_response` у созданного на первом шаге объекта-обработчика.

Ниже приведён пример использования модуля HTTP для отправки GET-запроса на IP-адрес веб-сайта http://example.com.

```
local log = require("log")
local http = require("http")
local logger = log.get_logger("script.lua")

-- обработчик ответа на HTTP-запрос
local example_handler = {}

-- этот метод обработчика будет вызван при получении ответа на запрос
function example_handler:on_http_response(response_data)
  if response_data == nil then
    logger:error("connection has failed; no response received")
  else
    logger:debug("response received; status="..response_data.status..", type(body)="..
type(response_data.body))
  end
end

-- параметры запроса
local http_request = {
  url = "http://93.184.216.34",  -- адрес http://example.com
  method = http.METHOD.GET,
  version = http.VERSION.HTTP10,
  timeout = 5000
}

-- отправка HTTP-запроса
```

```
logger:debug("querying http://example.com")
http.with(example_handler):transmit(http_request)
```

В параметрах запроса можно указать следующие данные:

- `method` — используемый HTTP-метод, значение из перечисления `http.METHOD`; значение по умолчанию `http.METHOD.GET`;
- `version` — используемая версия протокола HTTP, значение из перечисления `http.VERSION`; значение по умолчанию `http.VERSION.HTTP11`;
- `url` — целевой адрес / URL запрашиваемого ресурса, обязательный параметр;
- `timeout` — таймаут ожидания ответа / сетевого соединения, в миллисекундах; по умолчанию 13000 (13 секунд);
- `header` — таблица HTTP-заголовков (см. далее);
- `body` — тело запроса (см. далее).

Заголовки HTTP-запроса

Можно указать произвольные заголовки HTTP-запроса в форме Lua-таблицы со строковыми парами типа «ключ = значение». При этом «ключ» будет выступать в роли имени HTTP-заголовка, а «значение» — в роли значения HTTP-заголовка.

Пример:

```
-- параметры запроса
local http_request = {
  url = "http://93.184.216.34",
  method = http.METHOD.GET,
  version = http.VERSION.HTTP10,
  timeout = 5000
}

http_request.header = {}
http_request.header["X-Extra-Header"] = "my extra header value"
```

Тело HTTP-запроса

Модуль позволяет отправлять в HTTP-запросах (PUT или POST) данные в форме текста или JSON.

Для передачи запроса с телом в форме текста достаточно присвоить параметру запроса body соответствующее строковое значение. Например:

```
local http_request = {
  url = "http://10.1.30.3:8080/myservice",
  method = http.METHOD.POST,
  body = "<a>hello XML</a>"
}
http_request.header = {}
http_request.header["Content-Type"] = "text/xml; charset=UTF-8"
```

Модуль также позволяет передавать запрос с телом в форме JSON-документа, который автоматически формируется из Lua-таблицы. Например:

```
local http_request = {
  url = "http://10.1.30.3:8080/myservice",
  method = http.METHOD.POST
}
http_request.body = {
  myParam = "A",
  wowThisIsSubTable = {}
}
http_request.body["ohNumberHere"] = 42
http_request.body.wowThisIsSubTable["robot"] = "kill all human"
```

При отправке запроса с такими параметрами HTTP-заголовок Content-Type будет автоматически установлен в application/json, а тело запроса будет содержать следующий JSON-документ:

```
{
  "myParam": "A",
  "ohNumberHere": 42,
  "wowThisIsSubTable": { "robot": "kill all human" }
}
```

ВНИМАНИЕ

При преобразовании Lua-таблицы в JSON поддерживаются все типы данных КРОМЕ массивов (таблиц с числовыми ключами).

Обработка ответа

При поступлении ответа или сбое при отправке запроса асинхронно вызывается метод on_http_response(response_data) на объекте-обработчике. Параметр метода — это Lua-таблица, содержащая следующие данные:

- `status` — код HTTP-ответа, число в соответствии со спецификацией [https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP];
- `header` — HTTP-заголовки ответа в форме Lua-таблицы (аналогично HTTP-заголовкам запроса);
- `body` — тело ответа.

Если отправка запроса не удалась и ответ не был получен, то аргумент метода равен nil.

Тело ответа в зависимости от Content-Type может быть либо Lua-таблицей (если Content-Type ответа равен application/json — тело автоматически преобразуется в Lua-таблицу), либо текстом / строкой (в остальных случаях).

ignis.lua

Описание временно недоступно.

log.lua

Модуль позволяет выводить текстовые сообщения в журнал аудита.

Для вывода сообщений разработчику необходимо:

1. Подключить модуль `log.lua` в скрипт автоматизации командой require.
2. Создать объект-логгер с заданным именем, вызвав функцию `log.get_logger(-)`.
3. В требуемом месте скрипта вывести сообщение в журнал аудита с заданным уровнем «тревожности», вызвав один из методов `trace`, `debug`, `info`, `warn` или `error` у объекта-логгера.

Пример приведён ниже:

```
local log = require("log")
local logger = log.get_logger("MyLogger")

local answer = 42
logger:debug("The answer is "..tostring(answer))
```

Обратите внимание:

1. В журнал аудита сообщения выводятся от имени компонента программных средств `neyross::LogLuaModule`, а потому в настройки журнала аудита следует добавить соответствующий фильтр, например: `neyross::LogLuaModule | DEBUG`.
2. Имя объекта-логгера также выводится в журнале аудита перед заданным текстовым сообщением. В скрипте автоматизации можно создать и использовать любое количество объектов-логгеров с разными именами. Это позволяет различить сообщения в журнале аудита от разных логических разделов скрипта автоматизации.
3. В журнале аудита для каждой записи также выводится имя функции (если таковая есть), в которой вызван объект-логгер и номер соответствующей строки в скрипте, что ещё более упрощает понимание — к какой части скрипта относится соответствующее сообщение в журнале аудита.
4. Различные методы объекта-логгера выводят в журнал сообщения с разным уровнем «тревожности». Доступны следующие методы и уровни по приоритетам (от низкого / неважного до высокого / тревожного):
   a. `trace` (только для отладочного режима, в журнал не попадают никогда);
   b. `debug` (отладочные сообщения);
   c. `info` (информационные сообщения);
   d. `warn` (сообщения-предупреждения);
   e. `error` (сообщения об ошибках).

5. Методы объекта-логгера принимают на вход один строковой параметр. Для формирования строки из нескольких параметров используйте оператор конкатенации строк `..` и функцию приведения нестроковых типов к строке `tostring` из стандартной библиотеки.

longate.lua

Описание временно недоступно.

loninput.lua

Модуль предоставляет API для мониторинга состояния входов (зон) на модулях расширения ЯРС.

loninput.lua позволяет «подписаться» на изменения состояния входов. Для управления «подписчиками» в модуле доступны следующие функции:

Функция	Назначение	Комментарии
add_listener (key, listener)	Добавить объект подписчика	Функция принимает следующие параметры: • `key` — уникальный идентификатор подписчика, позволяет впоследствии «удалить» подписчика по указанному ключу, см. `remove_listener`; • `listener` — Lua-таблица (объект) с функцией-методом `on_loninput_input_state_changed`; требования к таблице подписчика приведены ниже.
`remove_listener (key)`	Удалить объект подписчика	Единственный аргумент `key` указывает на добавленного ранее подписчика, которого необходимо «отписать» от получения уведомлений.

Каждый объект-подписчик должен быть представлен Lua-таблицей с функцией on_loninput_input_state_changed(device_address, input_index, raw_state, logical_state).

Данная функция будет вызвана:

1. при изменении состояния входа какого-либо модуля расширения,
2. сразу в процессе добавления подписчика через `add_listener` для всех входов всех модулей (что позволяет получить текущее состояние нужных входов в момент добавления подписчика).

В функцию on_loninput_input_state_changed передаются следующие параметры:

Параметр	Тип данных	Описание
`device_address`	Число	Адрес устройства, к которому относится вход (зона)

`input_index`	Число	Индекс входа (зоны), которому соответствует состояние
`raw_state`	Число	Технический код состояния входа (зоны), полученный от модуля; возможные коды перечислены в таблице `RAW_INPUT_STATE`
`logical_state`	Число	Логическое состояние входа значения из перечисления `LOGICAL_INPUT_STATE`

Логическое состояние входа может быть следующим:

- 0 (`LOGICAL_INPUT_STATE.UNKNOWN`) — значение не известно;
- 1 (`LOGICAL_INPUT_STATE.OFF`) — вход выключен, нормальное состояние (логический 0);
- 2 (`LOGICAL_INPUT_STATE.ON`) — вход включен, тревожное состояние (логическая 1).

lonpacs.lua

Модуль предоставляет API для интеграции с подсистемой доступа модулей расширения ЯРС, таких как МДС и М2. Модуль позволяет модифицировать стандартные алгоритмы доступа.

> Подсистема доступа в модулях расширения ЯРС по своей структуре и поведению отвечает требованиям спецификаций Onvif (Profile C, Access Control Service, Door Control Service). В частности, состояние точек доступа, команды управления ими и другие аспекты отвечают указанным спецификациям. Рекомендуется ознакомиться с данными спецификациями перед использованием модуля lonpacs.lua.

Получить список точек доступа с их идентификаторами можно через поле lonpacs.ap_instance, которое заполняется при инициализации модуля и виртуальной Lua-машины (до того, как будет выполняться пользовательский Lua-скрипт). Поле lonpacs.ap_instance содержит Lua-таблицу, в которой ключами являются токены точек доступа, а значениями — Lua-таблицы с системными параметрами точек доступа. Структура таблицы представлена в коде ниже:

```
lonpacs.ap = {
  "токен первой точки доступа" = {
    ...
  },
  "токен второй точки доступа" = {
    ...
  },
  ... -- и так далее
}
```

Каждая точка доступа в списке — это либо точка доступа модуля расширения МДС, либо одна из точек доступа М2.

Управление точкой доступа

Для управления точками доступа на модулях расширения доступна функция transmit_status(token, status), которая позволяет отправлять модулям команды управления. Функция принимает следующие параметры:

Параметр	Тип	Описание
`token`	Строка	Идентификатор (токен) точки доступа, которой необходимо управлять
`status`	Число	Команда управления (требуемый статус точки доступа); значение из перечисления `lonpacs.AP_COMMAND` (см. ниже)

Для управления точкам доступа доступны следующие команды (значения параметра status):

Команда	Значение	Описание
`STATUS_QUERY`	-1	Запрос состояния точки доступа
`RESTORE`	12	Переключиться в дежурный режим работы
`OPEN_BLOCKED`	21	Разблокировать точку доступа
`CLOSE_BLOCKED`	22	Заблокировать точку доступа
`DISABLE`	13	
`INITIATE_PASS`	18	Инициировать проход

Состояние точки доступа

Модуль lonpacs.lua позволяет «подписаться» на события изменения состояния точек доступа, мониторинга связи с точками доступа, изменения состояния прохода (фазы алгоритма прохода). Для управления «подписчиками» в модуле доступны следующие функции:

Функция	Назначение	Комментарии
add_listener (key, listener)	Добавить объект подписчика	Функция принимает следующие параметры: • `key` — уникальный идентификатор подписчика, позволяет впоследствии «удалить» подписчика по указанному ключу, см. `remove_listener`; • `listener` — Lua-таблица (объект) с функциями-обработчиками изменений; требования к таблице подписчика приведены ниже.
`remove_listener (key)`	Удалить объект	Единственный аргумент `key` указывает на добавленного ранее подписчика, которого

	подписчика	необходимо «отписать» от получения уведомлений.

Каждый объект-подписчик должен быть представлен Lua-таблицей со следующими функциями:

Функция	Назначение	Комментарии
`lonpacs_on_ap_status_changed (token, ap_status)`	Получение и обработка изменений состояния точки доступа	В функцию передаются следующие параметры: • `token` (строка) — уникальный идентификатор (токен) точки доступа; • `ap_status` (число) — код состояния точки доступа; значение из перечисления `lonpacs.AP_STATUS` (см. ниже);
`lonpacs_on_pass_status_changed (token, status, notify_key)`	Получение и обработка изменений состояния алгоритма прохода	В функцию передаются следующие параметры: • `token` (строка) — уникальный идентификатор (токен) точки доступа; • `status` (число) — код состояния алгоритма прохода (см. ниже); • `notify_key` (строка) — код уведомления для отправки соответствующего извещения (возможного вызова `lonpacs.notify`).
`lonpacs_on_connection_changed (token, connection)`	Получение и обработка изменений состояния связи с точкой доступа	В функцию передаются следующие параметры: • `token` (строка) — уникальный идентификатор (токен) точки доступа; • `connection` (число) — состояние связи, возможные значения определены в перечислении

		lonpacs.CONNECTION: • 0 (ONLINE) — точка доступа на связи; • 1 (OFFLINE) — не связи с точкой доступа; • 2 (UNKNOWN) — состояние связи неизвестно.

Параметр ap_status может принимать одно из следующих значений:

Состояние точки доступа	Значение	Описание
USER_QUERY	0	Точка доступа в состоянии ожидания транзакции
FORCE_OPEN	1	Взлом двери
EPB	2	Старт транзакции прохода по кнопке выхода
HOLD_OPEN	5	Дверь удержана открытой (удержание двери)
DENIAL	6	Завершение транзакции прохода: • истек таймаут ожидания ответа от базы данных • отказ от базы данных • невалидный уровень доступа пользователя в ответе от базы данных • невалидная временная зона в уровне доступа пользователя
OPEN	10	Дверь открыта при совершении прохода
DISABLE	13	Точка доступа в состоянии "заблокирована". • дверной контакт не работает • кнопка выхода не работает • входные сообщения от считывателя игнорируются.

DOOR_CLOSED	15	• Снята тревога "взлом двери" (дверь закрыта) • Дверь закрыта при при завершении транзакции прохода через точку доступа
INITIATE_PASS	18	Старт транзакции прохода пользователя (при получении разрешения на доступ). Точка доступа переходит в состояние ожидания открытия двери.
OPEN_BLOCKED	21	Точка доступа в состоянии " открыта и заблокирована": • замок двери открыт • кнопка выхода не работает • дверной контакт не работает • входные сообщения от считывателя игнорируются Текущая транзакция пользователя обрывается.
CLOSE_BLOCKED	22	Точка доступа в состоянии " закрыта и заблокирована": • кнопка выхода не работает • дверной контакт работает • входные сообщения от считывателя игнорируются Текущая транзакция пользователя обрывается.
LOCK_ACTUATOR_ERROR	32	Неисправное состояние замка (не является состоянием точки доступа)
BLOCKED_HOLD_OPEN	53	Удержание двери точки доступа в состоянии "закрыта и заблокирована"

BLOCKED_FORCE_OPEN	54	Взлом двери точки доступа в состоянии "закрыта и заблокирована"
ARMED_CLOSE_BLOCKED	105	Точка доступа в состоянии "закрыта и заблокирована постановкой под охрану связанной группы зон": • кнопка выхода не работает • дверной контакт работает • считыватель работает Текущая транзакция пользователя обрывается.
ARMED_CLOSE_BLOCKED_FORCE_OPEN	106	Взлом двери точки доступа в состоянии "закрыта и заблокирована постановкой под охрану связанной группы зон"

ap_status определяет интегральное состояние составляющих точку доступа элементов (замок, дверной контакт). Для получения состояния этих элементов нужно использовать функцию lonpacs.parse_ap_status(ap_status).

Функция parse_ap_status возвращает Lua-таблицу следующего вида:

```
{
  door_state = ..
  lock_state = ..
  mode = ..
  alarm = ..
}
```

где значение полей таблицы следующее:

Поле	Описание	Возможные значения
door_state	Состояние двери (открыта, закрыта, ...)	Возможные значения определены в lonpacs.AP_DOOR_PHYSICAL_STATE: • UNKNOWN — состояние неизвестно; • OPEN — дверь открыта; • CLOSED — дверь закрыта; • FAULT — неисправность двери.
lock_state	Состояние замка	Возможные значения определены в lonpacs.AP_LOCK_STATE:

		• UNKNOWN — состояние неизвестно; • LOCKED — закрыт; • UNLOCKED — открыт; • FAULT — неисправность замка.
mode	Режим точки доступа (разблокирована, заблокирована, ...)	Возможные значения определены в lonpacs.AP_MODE: • UNKNOWN — состояние неизвестно; • LOCKED — дежурный режим; • UNLOCKED — разблокирована; • ACCESSED — проход; • BLOCKED — заблокирована.
alarm	Состояние тревоги точки доступа (удержана дверь и др.)	Возможные значения определены в lonpacs.AP_ALARM_STATE: • UNKNOWN — неизвестно; • NORMAL — норма; • DOOR_OPEN_TOO_LONG — удержание двери; • DOOR_FORCED_OPEN — взлом двери.

Значение кода состояния алгоритма прохода (фазы прохода) status, полученного в функции lonpacs_on_pass_status_changed(token, status, notify_key), может принимать одно из следующих значений:

Код (условный)	Принимаемое значение	Описание
ACS_DENIED	42	Доступ запрещен
NO_PASS	3	Проход не совершен
PASS	4	Проход совершен
ACS_ID_PRESENT	95	Предъявлен идентификатор СКУД (запрос пользователя СКУД в базе ЯРС)
ARM_ID_PRESENT	107	Предъявлен идентификатор для управления постановкой на охрану связанной группы зон
DISARM_ID_PRESENT	108	Предъявлен идентификатор для управления снятием с охраны связанной группы зон

REPEATED_PRESENT	59	Повторное предъявление идентификатора в ходе транзакции СКУД

pacs.lua

Модуль предоставляет API для интеграции с подсистемой доступа (только для контроллеров БОРЕЙ и ЯРС). Модуль позволяет либо модифицировать стандартные алгоритмы доступа, либо реализовать собственный алгоритм.

> Подсистема доступа в контроллерах БОРЕЙ, ЯРС по своей структуре и поведению отвечает требованиям спецификаций Onvif (Profile C, Access Control Service, Door Control Service). В частности, состояние точек доступа, команды управления ими и другие аспекты отвечают указанным спецификациям. Рекомендуется ознакомиться с данными спецификациями перед использованием модуля pacs.lua.

Общие сведения

Подсистема доступа в контроллерах БОРЕЙ, ЯРС поддерживает две точки доступа (поддержка дополнительных точек доступа на модулях расширения ЯРС осуществляется в Lua-модуле lonpacs.lua). Соответственно, первый считыватель относится к первой точке доступа, а второй — ко второй. У каждой точки доступа есть строковой уникальный идентификатор, называемый *токеном*.

Получить список точек доступа с их идентификаторами можно через поле pacs.ap, которое заполняется при инициализации модуля и виртуальной Lua-машины (до того, как будет выполняться пользовательский Lua-скрипт). Поле pacs.ap содержит Lua-таблицу, в которой ключами являются токены точек доступа, а значениями — Lua-таблицы с системными параметрами точек доступа. Структура таблицы представлена в коде ниже:

```
pacs.ap = {
  "токен первой точки доступа" = {
    "#" = 0,
    ...
  },
  "токен второй точки доступа" = {
    "#" = 1,
    ...
  }
}
```

Соответственно, для перебора точек доступа (например, чтобы для каждой точки доступа проинциализировать свой алгоритм) удобно использовать следующий код:

```
for key,item in pairs(pacs.ap) do
  if item["#"] == 0 or item["#"] == 1 then
    -- код обработки точки доступа с токеном = key
  else
```

```
    logger:error("unexpected access point number; #="..item["#"].."; token="..key)
  end
end
```

В контроллере БОРЕЙ точек доступа всегда ровно две, с номерами 0 и 1. Точка доступа с номером 0 (у которой значение свойства pacs.ap["токен"]["#"] равно 0 — это первая точка доступа), точка доступа с номером 1 — вторая.

В настройках доступа у контроллера можно выбрать «Режим работы» — «Две односторонние», «Одна двусторонняя», «Пользовательский». В стандартных режимах работы «Две односторонние» и «Одна двусторонняя» программные средства самостоятельно реализуют алгоритм доступа, таблицы параметров (pacs.ap["токен"]) содержат параметры, которые пользователь может изменить через веб-приложение. В режиме «Пользовательский» таблицы параметров по умолчанию обычно пусты.

> ⚠ **ВНИМАНИЕ**
>
> Для реализации собственной логики доступа с помощью Lua-скриптов пользователю следует выбрать в настройках контроллера режим = «Пользовательский». В этом режиме *системные параметры* алгоритма доступа (которые обрабатываются не в Lua-скрипте, а в основных программных средствах) не задаются через веб-интерфейс, и должны быть определены пользователем прямо в Lua-скрипте. Инструкция по установке системных параметров приведена в разделе «Изменение системных параметров» далее.

Управление состоянием точки доступа, а также режимом её работы (начало транзакции доступа, завершение транзакции и так далее) осуществляется из Lua-скрипта. Соответствующие функции описаны в разделе «Управление точкой доступа».

В каждый момент времени состояние точки доступа описывается N-мерным вектором:

Режим работы точки доступа	1. Дежурный / Ожидание прохода (Locked) 2. Выполняется проход (Accessed) 3. Заблокировано (Blocked) 4. Разблокировано (Unlocked)	1. Режим Locked устанавливается в программных средствах при сбросе транзакции доступа (см. `pacs_reset`). 2. Режимы Blocked / Unlocked устанавливаются по командам блокировки / разблокировки точки доступа (см. `pacs_block`, `pacs_unlock`). 3. Режим Accessed должен быть установлен из Lua-скрипта в момент начала доступа (открытия двери).
Состояние двери	Неизвестно (Unknown) Открыто (Open) Закрыто (Closed)	Определяется из Lua-скрипта.

	Неисправность (Fault)	
Состояние замка	Неизвестно (Unknown) Открыт (Unlocked) Закрыт (Locked) Неисправность (Fault)	Определяется из Lua-скрипта.
Тревожный статус	Норма (Normal) Взлом двери (DoorForcedOpen) Удержание двери (DoorOpenTooLong)	Определяется из Lua-скрипта.
Статус тампера	Неизвестно (Unknown) Корпус закрыт (NotInTamper) Вскрытие корпуса (TamperDetected)	Автоматически определяется программными средствами по факту вскрытия / закрытия корпуса контроллера. Не следует управлять этой осью состояния из Lua-скрипта.

Точки доступа поддерживают внешние команды управления блокировки, разблокировки, восстановления режима, включения / отключения, разрешения и подтверждения доступа. При обработке таких команд программные средства передают управление в Lua-скрипт. Также при предъявлении идентификатора, изменении состояния периферии (входов), выполенении внешних команд над точками доступа и т.д. подсистема доступа передаёт управление в Lua-скрипт. Пользователь может запрограммировать реакцию на соответствующие события с помощью функций-обработчиков (см. раздел «Функции-обработчики»).

Через каждую точку доступа в один момент времени может осуществляться одна транзакция доступа. Транзакция доступа — это процесс, который начинается с момента идентификации доступа, и завершается при возврате точки доступа в начальное состояние. В случае удержания двери транзакция доступа продолжается .

Транзакция доступа начинается либо вызовом pacs.authenticate в случае разрешения доступа, либо по вызову pacs_grant (для случая анонимного доступа). Завершается транзакция по вызову pacs_reset. Соответствующие функции описаны в разделе «Управление точкой доступа».

В процессе транзакции сохраняются идентификационные признаки посетителя и прочая информация, эти данные подставляются во все извещения, формируемые контроллером.

Часть извещений (в основном о запрещении доступа) формируются программными средствами. Пользователь также может в любой момент времени отправить извещение из Lua-скрипта с помощью функции pacs_notify, с указанием кода извещения. Для отправки доступно более полусотни извещений. Полный список доступных для отправки извещений приведён в документации на HTTP-API контроллера в части СКУД.

Изменение системных параметров

В режиме доступа «Пользовательский» для реализации алгоритма может потребоваться задать из Lua-скрипта для каждой точки доступа системные параметры. Для указания параметров из Lua-скрипта предназначена функция pacs.set_configuration(token, {...}), в первом аргументе необходимо передать токен точки доступа, во втором аргументе — Lua-таблицу с параметрами. Например:

```
for key,item in pairs(pacs.ap) do
  local configuration = {}
  configuration["input.mode"] = "card"
  pacs.set_configuration(key, configuration)
end
```

Ниже приведены основные системные параметры, которые можно / нужно настраивать:

Параметр	Ключ	Значение	Описание
Режим идентификации	`input.mode`	Одно значение из: card, pincode, card_and_pincode, card_or_pincode	Позволяет определить требуемый режим идентификации — какие идентификаторы посетителя ожидаются на вход + как они будут проверяться. **ВНИМАНИЕ**: этот параметр необходимо задать, если из Lua-скрипта будет вызвана функция pacs_authenticate с передачей идентификаторов посетителя; в противном случае идентификаторы не будут верифицированы по базе данных контроллера, доступ безусловно будет запрещён.
Число попыток ввода данных	`input.retry_limit`	Целое число больше 0	В случае, если input.mode равен `card_and_pincode` — определяет количество попыток ввода пинкода после предъявления карты.
Время ожидания ввода данных	`input.retry_timeout`	Целое число больше 0, в секундах	В случае, если input.mode равен `card_and_pincode` — определяет время ожидания ввода пинкода после предъявления карты.
Формат карты	`other.card_format`	Одно значение из: `auto`, `Raw-64`, `Raw-VL`	auto — автоматический выбор формата; для Wiegand формат выбирается исходя из количества переданных бит (26 — Wiegand-26, 37 — Wiegand-37), для 1-Wire всегда Wiegand-26

			Raw-64 — используются все 64 бита карты (отсутствующие биты дополняются 0) в форме одного 64-битного числа; код предприятия при этом равен 0. Raw-VL — используются только переданные от считывателя биты, в форме одного 64-битного числа; код предприятия при этом равен 0.
Маска (номера карты)	`other.card_mask`	`FFFFFFFFFFFFFFFF`	64 бита маски в формате HEX-строки из 16 символов
Набор параметров блока «Контроль повторного прохода»			Описание временно недоступно.
Набор параметров блока «Проход под принуждением»			Только в режиме идентификации «Карта и пинкод» Описание временно недоступно.
Набор параметров блока «Проход с подтвержденеим»			Описание временно недоступно.
Набор параметров блока «Доступ по правилу N-лиц»			Описание временно недоступно.

Функции-обработчики

Модуль позволяет разработчику определить глобальные Lua-функции с заданными именами. Такие функции-обработчики будут вызваны программными средствами контроллера по мере выполнения функций доступа.

ВНИМАНИЕ

В данном случае это должны быть именно «глобальные функции», определённые, например, следующим образом:

```
function pacs_reader_input(token, data)
  -- тело функции
end
```

Локальные функции с таким же именем не будут вызваны:

```
local pacs_reader_input = function(token, data)
  -- функция не будет вызвана
end
```

Ниже приведён список доступных для определения функций-обработчиков:

Функция	Назначение	Комментарии
`pacs_reader_input(token, data)`	К считывателю предъявлен идентификатор	Вызывается тогда, когда к считывателю контроллера предъявлен идентификатор **ВАЖНО**: проверка идентификатора ещё не осуществлялась. В случае, если данные предъявлены считывателю Wiegand / 1-W выполняется разбор считан битов и определение номер карты / кода предприятия в соответствии с системными параметрами other.card_for other.card_mask. В первом аргументе token передаётся токен точки доступа, на которой предъявлен идентификатор втором аргументе передаёт Lua-таблица с идентификаторами: `{ "facility_code" = "123", "card_number" = "12345", "pincode" = "12345" }` Если идентификатор отсутствует (например, не вводился пинкод), то соответствующее поле таблицы также отсутствует Обычно в коде данной функции выполняется вызо pacs.authenticate для начал транзакции доступа.
`pacs_accessed(token, args)`	Получена внешняя	Первый аргумент token — токен точки доступа.

	команда на инициацию транзакции доступа.	Второй аргумент args — параметры разрешения доступа:
		
		{
		"use_extended_time" = tru
		"access_time_duration" = число (unix-timestamp),
		
		"open_too_long_time_durati = число (unix-timestamp),
		"pre_alarm_time_duration" число (unix-timestamp)
		}
		
		Подробную спецификацию команды инициации транзакции доступа смотри спецификациях Onvif.

`pacs_granted(token)`	Доступ разрешён (в результате проверки идентификатора)	Данная функция-обработчи вызывается тогда, когда вы pacs.authenticate завершил успешно (если идентификаторы были переданы в pacs.authentica то они были верифицирова по базе данных, у пропуска есть все необходимые прав разрешения на доступ). Первый аргумент token — токен точки доступа.
`pacs_denied(token)`	Доступ запрещён (в результате проверки идентификатора)	Данная функция-обработчи вызывается тогда, когда в результате pacs.authenticat принято решение о запрещении доступа. Первый аргумент token — токен точки доступа.
`pacs_enabled(token)`	Точка доступа включена (по внешней команде)	Первый аргумент `token` — токен точки доступа.
`pacs_disabled(token)`	Точка доступа отключена (по внешней команде)	Первый аргумент `token` — токен точки доступа.

`pacs_identification(token, event)`	Изменилось состояние ожидания ввода данных в режиме «Карта и пинкод»	Первый аргумент token — токен точки доступа. Второй аргумент event — о из retry (требуется повторн ввод пинкода), card (считан карта, требуется ввод пинкода), timeout (истекло время ожидания ввода пинкода).
`pacs_external_locked(token) boolean`	Должен быть восстановлен режим работы точки доступа (обычный режим работы)	Режим работы точки достуг должен быть восстановлен по внешней команде (из ве интерфейса или по Onvif; с Onvif Profile C). Это функци обработчик для внешней команды, обычно в коде данной функции выполняет вызов pacs_reset. Функция должна возвращат булевое значение — true, е точка доступа перешла в обычный режим работы, и f , если команда не может бы выполнена. Первый аргумент token — токен точки доступа.
`pacs_external_unlocked (token) boolean`	Точка доступа должна быть разблокирована	Режим работы точки достуг должен быть изменён на «Разблокировано» — по внешней команде (из веб-интерфейса или по Onvif; с Onvif Profile C). Это функци обработчик для внешней команды, обычно в коде данной функции выполняет вызов pacs_unlock. Функция должна возвращат булевое значение — true, е точка доступа разблокиров и false, если команда не мо быть выполнена. Первый аргумент token — токен точки доступа.
`pacs_external_blocked (token) boolean`	Точка доступа должна быть заблокирована	Режим работы точки достуг должен быть изменён на «Заблокировано» — по

		внешней команде (из веб-интерфейса или по Onvif; с Onvif Profile C). Это функци обработчик для внешней команды, обычно в коде данной функции выполняет вызов pacs_block. Функция должна возвращат булевое значение — true, е точка доступа заблокирова и false, если команда не мо быть выполнена. Первый аргумент token — токен точки доступа.
`pacs_external_authorization (token, decision)`	Разрешение доступа по внешней команде	Первый аргумент `token` — токен точки доступа.
`pacs_input(port, device, type_, id, raw)`	Изменение состояния элемента периферии	Параметры port и device вс константны, значения не имеют. Параметр type_ определяе тип элемента периферии, ч состояние изменилось. Значение из перечисления TYPE (модуль fec.lua). Параметр id определяет идентификатор элемента периферии, чьё состояние изменилось. Например, для входов на плате БОРЕЙ эт значения от 1 до 8. Параметр raw содержит «сырое» значение состоян элемента периферии. Спос приведения «сырого» значе к логическому описан ниже разделе «Изменение состояние периферии).
`pacs_cancel_transaction (token)`	Получена внешняя команда на прерывание транзакции доступа	Первый аргумент `token` — токен точки доступа.

СОВЕТ

При использовании стандартных алгоритмов «Две односторонних точки доступа», «Одна двусторонняя точка доступа» перечисленные в таблице функции-обработчики *уже определены*, в них выполняется логика стандартных алгоритмов доступа. Пользователь *может переопределить* эти обработчики в своём скрипте. Таким образом, переопределив ВСЕ функции пользователь реализует *полностью свою* логику доступа, НО при этом получит доступ ко всем параметрам, задаваемым для стандартных алгоритмов через веб-интерфейс — через поле pacs.ap (см. раздел «Модель данных» выше).

Изменение состояния периферии

При изменении состояния какого-либо элемента периферии (обычно, это различные входы) вызывается функция-обработчик pacs_input. Ниже приведён Lua-код, который позволит в зависимости от типа элемента периферии получить его логическое состояние.

```
local fec = require("fec")

-- Логическое состояние элемента
local VALUE = {
        OFF     = 0,
        ON      = 1,
        SHORT   = 2,
        BROKEN  = 3
}

-- Получение логического состояния для элемента типа «Сухой контакт»
function get_dry_contact_value(raw, invert)
    if raw == 1 then
        if invert then
            return VALUE.OFF
        else
            return VALUE.ON
        end
    else
        if invert then
            return VALUE.ON
        else
            return VALUE.OFF
        end
    end
end

-- Получение логического состояния для элемента типа «Резистивный шлейф»
function get_rin_value(raw,invert)
```

```
        if raw <= 17 then -- эмпирический порог резистивного шлейфа
            if invert then
                return VALUE.OFF
            else
                return VALUE.ON
            end
        else
            if invert then
                return VALUE.ON
            else
                return VALUE.OFF
            end
        end
end

-- Получение логического состояния входов разного типа
-- Дополнительный аргумент invert позволяет обрабатывать настройку вида
«нормально-открыт»
function get_input_value(raw, invert, type_)
    if (type_ == fec.TYPE.DRY_CONTACT) then
        return get_dry_contact_value(raw, invert)
    elseif (type_ == fec.TYPE.RADIAL_INPUT) then
        return get_rin_value(raw, invert)
    else
        log_error("invalid type: "..type_)
        return 0
    end
end
```

Управление точкой доступа

Для управления точкой доступа / транзакцией доступа по выбранной точке доступа предназначены следующие функции:

Функция	Назначение	Комментарии

`pacs.authenticate (token, {...})`	Выполнить проверку переданных идентификаторов и начать транзакцию доступа	Первый аргумент — токен точки доступа. Второй аргумент — Lua-таблица с идентификаторами (такая же, какая получена в функции-обработчике pacs_reader_input):
		`{`
		`"facility_code" = "123",`
		`"card_number" = "12345",`

		"pincode" = "12345", "transaction_uuid" = "..." } В результате проверки будет вызвана функция обработчик pacs_granted (если доступ разрешён; при этом будет начата транзакция доступа, см. pacs_begin) или pacs_denied (если доступ запрещён). Если передан transaction_uuid, то соответствующее значение будет использоваться в роли идентификатора транзакции и будет подставлено во все отправляемые извещения. Если значение не будет указано, то будет сгенерирован уникальный UUID транзакции. **ВАЖНО**: если таблица во втором аргументе будет пуста, то проверка идентификаторов по базе данных проводиться не будет, автоматически будет выдано разрешение на доступ, будет вызвана функция-обработчик pacs_granted.
`pacs_reset(token)`	Восстановить обычный режим работы точки доступа	Первый аргумент — токен точки доступа. **ВАЖНО**: текущая транзакция прекращается, все связанные с транзакцией данные (пропуска и пр.) очищаются. Данную функцию обязательно надо вызвать при завершении транзакции доступа.
`pacs_block(token)`	Перевести точку доступа в состояние «Заблокировано»	Первый аргумент — токен точки доступа.
`pacs_unlock(token)`	Перевести точку доступа в	Первый аргумент — токен точки доступа.

	состояние «Разблокировано»	
`pacs_pause(token)`	Прекратить приём идентификаторов на точке доступа	Первый аргумент — токен точки доступа. Транзакция доступа не прерывается. Обычно используется при переходе в состояния типа «Взлом двери» или «Удержание двери».
`pacs_grant(token)`	Разрешить анонимный доступ и начать анонимную транзакцию доступа	Обычно применяется для начала транзакции по кнопке выхода. Поведение отличается от pacs.authenticate + pacs_begin тем, что извещения о событиях доступа будут содержать тему «анонимного доступа».
`pacs_notify(token)`	Отправить извещение с заданным кодом	Доступный список кодов см. в документации на HTTP-API контроллера в части СКУД.

`pacs_update_door_state (token, state)`	Изменить состояние точки доступа	Первый аргумент — токен точки доступа.
		
		Второй аргумент — Lua-таблица со значением состояния по различным «осям»:
		
		`{`
		`  "lock" = Unlocked,`
		`  "door" = Open,`
		`  "mode" = Accessed,`
		`  "alarm" = Normal,`
		`}`
		
		Во втором аргументе можно отправлять только те «оси» состояния, которые необходимы. Доступные значения состояния см. в разделе «Общие сведения».

Поиск в базе данных

Описание временно недоступно.

storage.lua

Модуль предоставляет функции для перманентного сохранения и последующего чтения конфигурационных параметров или параметров скрипта автоматизации.

В некоторых случаях может потребоваться сохранять состояние или параметры работы скрипта автоматизации при перезапуске продукта. Или, например, может возникнуть потребность, чтобы определённые параметры были сохранены в резервной копии и восстановлены в случае сбоя файловой системы на IP-контроллере (БОРЕЙ, ЯРС и пр.). Для работы с такими данными и предназначен модуль storage.lua.

Модуль позволяет сохранять в персистентное хранилище Lua-таблицы с данными под заданным именем, а впоследствии извлекать их из хранилища по имени. Таблица с данными может содержать только пары «ключ = значение», где «ключ» это всегда строка, а «значение» может быть строкового, числового или булевого типа.

Для помещения данных в хранилище предназначена функция storage.set(ключ, данные), для сохранения — storage.save(ключ), для извлечения — storage.get(ключ) .

> ⚠ Вызов функции set(...) не приводит к сохранению данных, данные только помещаются в хранилище! Данные будут сохранены только после вызова функции save(...).

Пример использования модуля приведён ниже:

```
local storage = require("storage")

local my_options = storage.get("my_key")
my_options.param1 = "A"
my_options.param2 = 42
storage.save("my_key")

local another_options = {
  param1 = "B",
  param2 = 43
}
storage.set("my_key", another_options) -- my_options более не ссылается на объект в хранилище!
storage.save("my_key")
```

tenso.lua

Описание временно недоступно.

timer.lua

Данный модуль предоставляет для скрипта автоматизации две функции для выполнения отложенных во времени вычислений. Первая функция — это set_timeout. Она предназначена для разового выполнения некоторой Lua-функции спустя заданное количество времени. Вторая функция — set_interval. Она позволяет запланировать выполнение некоторой Lua-функции каждые N миллисекунд. Функция будет выполняться периодически до тех пор, пока задание на выполнение не будет явно отменено (см. clear_interval далее) или Lua-машина не будет перезапущена.

Также модуль предоставляет две функции для отмены запланированных ранее заданий: clear_timeout и clear_interval.

> ⚠ **ЗАМЕЧАНИЕ**
>
> Для использования функций модуля не требуется ничего подключать с помощью require. Функции set_timeout, set_interval, clear_timeout, clear_interval доступны глобально в любом месте скрипта автоматизации.

set_timeout

Запланировать вызов указанной Lua-функции спустя заданное количество миллисекунд.

Полная сигнатура функции: set_timeout(имя-функции, длительность, ключ), где

- имя функции — строка, имя Lua-функции для вызова;
- длительность — число, задержка выполнения функции, в миллисекундах;
- ключ — строка, ключ, по которому можно будет понять в Lua-функции, кто и когда запланировал её отложенный вызов.

При вызове set_timeout возвращает уникальный идентификатор созданного задания на отложенное выполнение. Это значение можно использовать в последующем вызове clear_timeout для отмены запланированного ранее задания.

Пример:

```
local log = require("log")
local logger = log.get_logger("script.lua")

local key1_id = set_timeout("my_timeout", 10000, "key1") -- выполнить my_timeout
через 20 секунд с key1
local key2_id = set_timeout("my_timeout", 20000, "key2") -- выполнить my_timeout
через 20 секунд с key2

function my_timeout(key)
  if key == "key1" then
    logger:debug("stopping timeout on key2")
    clear_timeout(key2_id) -- отменяем задание с key2
  elif key == "key2" then
    logger:error("this should be never invoked")
  else
```

```
    logger:error("unexpected key="..key)
  end
end
```

clear_timeout

Функция позволяет отменить ранее запланированное задание: clear_timeout(ID задания), где ID задания — результат выполнения функции set_timeout.

set_interval

Сигнатура налогична set_timeout: set_interval(имя-функции, длительность, ключ), где

- имя функции — строка, имя Lua-функции для вызова;
- интервал — число, период выполнения функции, в миллисекундах;
- ключ — строка, ключ, по которому можно будет понять в Lua-функции, кто и когда запланировал её отложенный вызов.

> **ВНИМАНИЕ**
>
> Также как и в JavaScript, рекомендуется избегать использования set_interval, сделав выбор в пользу рекурсивного set_timeout.

clear_interval

Аналогично clear_timeout функция позволяет отменить запланированное ранее периодическое задание: clear_interval(ID задания), где ID задания — результат выполнения функции set_interval.

terminal.lua

Описание временно недоступно.