Подсистема сценариев в программе АРМ «Скиф»

Назначение подсистемы сценариев

Подсистема сценариев предназначена для расширения функциональности программы АРМ «Скиф». Приведем возможные задачи, решаемые при помощи сценариев:

- Постановка на охрану разделов в заданное время.
- Включение релейных выходов в заданное время или при определенных событиях.
- Отправка сообщения на почту при изменении состояния раздела (шлейфа, прибора...).
- Отправка сообщения СМС при изменении состояния раздела (шлейфа, прибора...).
- Передача сторонней системе видеонаблюдения информации о тревоге, например для вывода изображения с определенной телекамеры на полный экран или тревожный монитор.
- Передача купольной камере команды перехода в определенную предустановку при тревоге извещателя на периметре охраняемой территории.

Некоторые возможности (например отправка сообщения на почту, отправка СМС сообщения и т.д.) должны быть реализованы внешними сторонними программами, сценарий может лишь вызвать эти программы и передать им необходимые параметры и данные.

Типы запуска сценариев

В клиентском ПО существует два типа запуска сценариев:

- **Автозапуск во время старта программы.** При этом, сценарий может быть загружен и выполняться, но определенные действия сценария будут выполнены в определенный момент (т.е. не в момент загрузки, а в момент наступления каких-то событий). Файл сценария должен быть помещен в папку **«Scripts\ScriptsClientAutoRun»**.
- **Запуск пользователем.** Подобные сценарии отображаются на вкладке [Дежурство] → [Сценарии]. Внутренне такие сценарии ничем не отличаются от сценариев автозапуска, но начинают выполняться по команде пользователя. Файл сценария должен быть помещен в папку **«Scripts\ScriptsClientMenu»**. Если требуется чтобы в клиентском ПО у сценария была иконка, то рядом с файлом сценария должен быть файл с таким же именем, но с расширением **«.png»** (разрешение изображения желательно 24x24 или 32x32 точки). Такие сценарии можно располагать в каталогах произвольной глубины вложенности. Иконка каталога - файл **«image.png»**.

В серверном ПО существует только один типа запуска сценариев: **«Автозапуск во время старта программы»**. Файл сценария должен быть помещен в папку **«Scripts\ScriptsServer»**.

Структура сценариев

Интерпретатор сценариев в программе **АРМ «Скиф»** базируется на стандарте **ECMA-262** (так же известном, как **JavaScript**). Информацию по стандарту **ECMA-262** можно найти в Internet или печатных изданиях. Для написания сценариев, совсем не обязательно изучать стандарт **ECMA-262**, достаточно прочитать до конца данный материал и посмотреть примеры в папке **«Scripts\Example»**. В сценариях, для доступа к функциям программы, используется глобальный объект **«armSkif»**. Приведем примеры сценариев:

Пример 1. Записать строку в лог-файл.

```
// Так обозначается однострочный комментарий.
// Помимо глобального объекта armSkif, доступны еще две глобальные переменные:
// scriptPatch – путь к сценарию,
// scriptFileName – имя сценария.

armSkif.writeLog( "Start script: " + scriptFileName );
armSkif.writeLog( "Script patch: " + scriptPatch );
```

Пример 2. Запуск внешней программы.

```
armSkif.runProgram("calc");
```

Пример 3. Включить реле. Сценарий для запуска пользователем.

```
const numServer=1; // Номер сервера
const numPKU  = 1; // Номер пульта
const numDev   = 1; // Номер прибора
const numRelay= 1; // Номер реле

var serverNumPKU = armSkif.convert2ByteToWord(numServer, numPKU);
armSkif.controlRelay(serverNumPKU, numDev, numRelay, armSkif.RL_ON );
```

Пример 4. Включить реле после старта и с задержкой. Сценарий для серверного ПО.

```
const numPKU = 1; // Номер пульта
const numDev  = 1; // Номер прибора
const numRelay= 1; // Номер реле

// Вызвать функцию controlRelay после опроса доступных приборов
armSkif.signalPollFinished.connect( controlRelay ); // Так применяются сигналы

// Запланировать вызов функции controlRelay через 10 минут
armSkif.setTimeout(controlRelay, 600000); // Так применяются таймеры

function controlRelay() {
  armSkif.controlRelay(numPKU, numDev, numRelay, armSkif.RL_ON );
}
```

В таблицах ниже приведены функции объекта armSkif, константы, состояния агрегатов.

Таблица 1. Функции объекта armSkif.

Функция	Описание
Списки агрегатов	
getListPKU()	Возвращает массив номеров пультов.
getListDevices(numPKU)	Возвращает массив номеров приборов пульта.
getListParts(numPKU)	Возвращает массив номеров разделов пульта.
getListGParts(numPKU)	Возвращает массив номеров групп разделов пульта.
getListPartSh(numPKU, part)	Возвращает массив номеров шлейфов раздела.
getListPartExit(numPKU, part)	Возвращает массив номеров контролируемых цепей раздела.
getListPartDev(numPKU, part)	Возвращает массив номеров приборов раздела.
getListGPartParts (numPKU, gpart)	Возвращает массив номеров разделов, входящих в группу разделов.
getListSh(numPKU, dev)	Возвращает массив номеров шлейфов прибора.
getListExit(numPKU, dev)	Возвращает массив номеров контролируемых цепей прибора.
getListRelay(numPKU, dev)	Возвращает массив номеров релейных выходов прибора.
getListReader(numPKU, dev)	Возвращает массив номеров считывателей прибора.
Пульты	
getPKUVersion(numPKU)	Возвращает версию пульта.
getPKUTypeStr(numPKU)	Возвращает тип пульта: «С2000», «С2000М».
getPKUDescription(numPKU)	Возвращает описание пульта в конфигурации клиента.
Приборы	
getDeviceDescription (numPKU,dev)	Возвращает описание прибора в конфигурации пульта.
getDeviceTypeStr (numPKU, dev)	Возвращает строковый тип прибора.
getDeviceType(numPKU, dev)	Возвращает тип прибора (**Таблица А.1**).
getDeviceVersion (numPKU, dev)	Возвращает версию прибора.
getDevicePart(numPKU, dev)	Возвращает раздел, в который входит прибор.
getDeviceTamperState (numPKU, dev)	Возвращает состояние тампера прибора (**Таблица А.7**).
getDevicePowerState (numPKU, dev)	Возвращает состояние электропитания прибора (**Таблица А.7**).
Шлейфы	
getShState(numPKU, fsh)	Возвращает состояние шлейфа сигнализации (**Таблица А.7**).
getShState(numPKU, dev, sh)	-//-//-
getShPart(numPKU, fsh)	Возвращает номер раздела, в который входит шлейф.
getShPart(numPKU, dev, sh)	-//-//-
getShDescription (numPKU, fsh)	Возвращает текстовое описание шлейфа сигнализации.

getShDescription (numPKU, dev, sh)	-//-//-
requestShADC(numPKU, fsh)	Запросить АЦП шлейфа. Ответ будет в signalUpdateADC.
requestShADC (numPKU, dev, sh)	-//-//-
Релейные выходы	
getRelayState(numPKU, frl)	Возвращает состояние релейного выхода: 0 - не определено; 1 - включено; 2 - выключено; 3 - мигает.
getRelayState (numPKU, dev, rl)	-//-//-
getExitState(numPKU, frl)	Возвращает состояние контролируемой цепи (**Таблица А.7**).
getExitState(numPKU, dev, rl)	-//-//-
getExitPart(numPKU, frl)	Возвращает номер раздела, в который входит контролируемая цепь.
getExitPart(numPKU, dev, rl)	-//-//-
getRelayDescription (numPKU, frl)	Возвращает текстовое описание релейного выхода.
getRelayDescription (numPKU, dev, rl)	-//-//-
Считыватели	
getReaderState(numPKU, frd)	Возвращает состояние считывателя: бит 0 - запрет выхода (по кнопке); бит 1 - запрет входа; бит 2 - свободный проход.
getReaderState (numPKU, dev, rd)	-//-//-
getReaderDescription (numPKU, frd)	Возвращает текстовое описание считывателя.
getReaderDescription (numPKU, dev, rd)	-//-//-
Разделы	
getPartState(numPKU, part)	Возвращает состояние раздела (**Таблица А.7**).
getPartDescription (numPKU, part)	Возвращает текстовое описание раздела.
Управление	
controlSh_Arm(numPKU, fsh)	Постановка шлейфа сигнализации на охрану.
controlSh_Arm (numPKU, dev, sh)	-//-//-
controlSh_DisArm (numPKU, fsh)	Снятие шлейфа сигнализации с охраны.
controlSh_DisArm	-//-//-

(numPKU, dev, sh)	
controlPart_Arm (numPKU, part)	Постановка раздела на охрану.
controlPart_DisArm (numPKU, part)	Снятие раздела с охраны.
controlRelay (numPKU, frl, prog)	Управление релейными выходами. prog - программа управления **(Таблица А.2).**
controlRelay (numPKU, dev, rl, prog)	-//-//-
controlRelay(numPKU, frl, prog, mask, delay, time)	Расширенное управление релейными выходами. prog - программа управления **(Таблица А.2).** mask - маска мигания (**Таблица А.3**). delay - задержка включения (0...65535). time - продолжительность включения (0...65535). Время задается с дискретностью 0,125 сек., т.е. 1000=125сек.
controlRelay(numPKU, dev, rl, prog, mask, delay, time)	-//-//-
controlReader (numPKU, frd, prog)	Управление считывателем. prog - программа управления считывателем **(Таблица А.4).**
controlReader (numPKU, dev, rd, prog)	-//-//-
sendDeviceBeep (numPKU, dev, prog)	Для С2000-К. Воспроизвести звуковой сигнал. prog — тип звукового сигнала (**Таблица А.5**).
sendDeviceText (numPKU, dev, text)	Для С2000-К. Отобразить на экране текстовое сообщение.
runProgram(program)	Запустить внешнюю программу.
Таймеры (вызывают функцию при наступлении определенного времени)	
setTimeout (expression, delay)	Однократно вызвать функцию через определенный интервал времени (мс). Возвращает идентификатор таймера timerId.
clearTimeout(timerId)	Останавливает таймер с идентификатором timerId, запущенный функцией setTimeout.
setInterval(expression, delay)	Многократно вызывать функцию через определенные интервалы времени (в милисекундх). Возвращает идентификатор таймера timerId.
clearInterval(timerId)	Останавливает таймер с идентификатором timerId, запущенный функцией setInterval.
setSheduler (expression, dateTime)	Однократно вызвать функцию в определенный час, минуту, секунду. Возвращает идентификатор таймера timerId.
clearSheduler(timerId)	Останавливает таймер с идентификатором timerId, запущенный функцией setSheduler.
Сигналы (сигналы посылаются при наступлении определенного события)	
signalPollFinished(numPKU)	Окончен опрос работающих приборов. Данный сигнал актуален только для скриптов автозапуска.
signalUpdateDeviceState (numPKU, dev)	Обновилось состояние прибора (тампер, питание). Если dev=0, то изменились состояния всех приборов данного ПКУ.

signalUpdateSh (numPKU, fsh)	Изменилось состояние шлейфа сингализации (ШС). Если fsh = 0, то изменились состояния всех ШС данного ПКУ.
signalUpdateExit (numPKU, frl)	Изменилось состояние контролируемой цепи (КЦ). Если frl = 0, то изменились состояния всех КЦ данного ПКУ.
signalUpdateRelay (numPKU, frl)	Изменилось состояние релейного выхода (Рл). Если frl = 0, то изменились состояния всех Рл данного ПКУ.
signalUpdateReader (numPKU, frd)	Изменилось состояние считывателя (Сч). Если frd = 0, то изменились состояния всех Сч данного ПКУ.
signalUpdatePart (numPKU, part)	Изменилось состояние раздела. Если part = 0, то изменились состояния всех разделов данного ПКУ.
signalUpdateGPart (numPKU, gpart)	Изменилось состояние группы разделов. Если gpart = 0, то изменились состояния всех групп разделов данного ПКУ.
signalNewEvent (numPKU, count)	В журнале событий появилось новое событие (тольо для серверного ПО). count — всего событий в журнале. (count-1) — номер последнего события.
signalUpdateADC (numPKU, fsh, adc, value);	Получено АЦП шлейфа (adc) и значение АЦП (value).
signalAnswerGET(data)	Ответ на команду networkRequestGET.
Вспомогательные функции	
writeLog(str)	Записывает строку в файл лога сценариев. Файл лога для серверного ПО: «log\ServerSkif_script.log». Файл лога для клиентского ПО: «log\ClientSkif_script.log».
writeFile(fileName, str)	Записывает строку в файл. Путь файла задается в переменной fileName относительно каталога log.
fileExists(fileName)	Возвращает истину, если файл существует.
convert2ByteToWord(b1, b2)	Возвращает (b1<<8) | b2. Удобно для получения полного номера пульта в сценариях клиентского ПО: fullNumPKU = convert2ByteToWord(numServer, numPKU);
convertStateToDescript(state)	Получение текстового описания состояния (**Таблица А.7**).
convertStateToPriority(state)	Получение приоритета состояния (**Таблица А.7**).
convertStateToGroup(state)	Получение группы состояния (**Таблицы А.6,А.7**).
isAlarmState(state)	Возвращает истину, если состояние тревожное (**Таблица А.7**).
isTypeRip(type)	Возвращает истину, если прибор РИП.
isTypeSKD(type)	Возвращает истину, если прибор СКД (С2000-2, С2000-4).
isOsLinux()	Возвращает истину, если ОС GNU/Linux.
isOsWindows()	Возвращает истину, если ОС MS Windows.
isOsMac()	Возвращает истину, если ОС Mac.
networkRequestGET(data)	Команда для отправки данных методом GET (протокол HTTP(S)) в сторонние системы, например Trassir. Пример: networkRequestGET("https://127.0.0.1:8080/objects/?password=123")

Диалоговые окна сообщений (только для клиентского ПО)	
messageWarning(title, text)	Диалоговое окно предупреждения.
messageCritical(title, text)	Диалоговое окно ошибки.
messageInformation(title, text)	Диалоговое окно информации.
messageQuestion(title, text)	Диалоговое окно с вопросом и двумя кнопками. В зависимости от ответа, функция возвращает true или false.
Журнал событий (только для серверного ПО)	
getEventCount (numPKU)	Возвращает количество событий в журнале событий.
getEvent (numPKU, num)	Возвращает код события (Таблица А.7).
getEventPart (numPKU, num)	Возвращает номер раздела в событии.
getEventZone (numPKU, num)	Возвращает зону доступа в событии.
getEventDev (numPKU, num)	Возвращает номер прибора в событии.
getEventUnit (numPKU, num)	Возвращает номер аграгата (шлейф/реле/считыватель)
isUnitSh (numPKU, num)	Возвращает истину если агрегат – шлейф.
isUnitRl (numPKU, num)	Возвращает истину если агрегат – реле.
isUnitRd (numPKU, num)	Возвращает истину если агрегат – считыватель.
getEventKey (numPKU, num)	Возвращает код карты (пароли не возвращаются).
getEventAddr (numPKU, num)	Возвращает адрес агрегата в текстовом виде.
getEventDescr (numPKU, num)	Возвращает текстовое описание события.
getEventUser (numPKU, num)	Возвращает имя пользователя в событии.
getEventDate (numPKU, num)	Возвращает дату и время в событии.

Таблица 2. Параметры функций из таблицы 1.

Параметр функции	**Описание**
numPKU	Номер пульта. В конфигурации сервера: 1...255. В конфигурации клиента состоит **из двух байт**: младший байт - номер пульта в конфигурации сервера (1...255), старший байт - номер сервера в конфигурации клиента (1...255). Смотрите так же функцию convert2ByteToWord(b1, b2).
part	Номер раздела в конфигурации пульта: 1...9999. Максимум 511 разделов.
gpart	Номер группы разделов в конфигурации пульта. Номера пересекаются с номерами разделов и входят в ограничение 511 разделов на пульт.
dev	Номер прибора в конфигурации пульта: 1...127.
sh	Номер шлейфа 1...255 (в зависимости от прибора).
fsh	Полный номер шлейфа fsh = convert2ByteToWord(dev, sh).
rl	Номер реле 1...255 (в зависимости от прибора).
frl	Полный номер реле frl = convert2ByteToWord(dev, rl).
rd	Номер считывателя.
frd	Полный номер считывателя frd = convert2ByteToWord(dev, rd).

Контролируемая цепь — релейный выход с контролем состояния сопротивления цепи (норма/обрыв/короткое замыкание). Сам релейный выход, при этом, может иметь состояние вкл./вык. Таким образом, контролируемая цепь имеет два состояния: состояние сопротивления и состояние контактов релейного выхода.

В раздел могут входить шлейфы, контролируемые цепи, приборы. Состояние раздела зависит от приоритета состояний входящих в него агрегатов.

Протокол MQTT

MQTT (Message Queuing Telemetry Transport) - протокол обмена сообщениями, реализующий модель "публикация/подпись". Для обмена по этому протоколу требуется брокер (сервер) MQTT, например mosquitto. Реализация протокола MQTT в подсистеме сценариев АРМ «Скиф» позволяет при помощи сценария устанавливать соединение с брокером MQTT и обмениваться с ним информацией. В дальнейшем, к брокеру MQTT могут подключаться различные программы «умный дом», например OpenHAB, что позволяет отображать состояния агрегатов (шлейфы, разделы...), а так же управлять ими.

Для доступа к функциям данного протокола, в подсистему скриптов АРМ «Скиф» (только для сервера) был добавлен глобальный объект **mqtt**. Перед первым использованием данного объекта, необходимо произвести его инициализацию командой:
mqtt.init(host, port, clietnId, username, password);

После инициализации объекта **mqtt**, можно взаимодействовать с брокером MQTT при помощи функций и сигналов, приведённых в таблице 3.

Таблица 3. Функции объекта mqtt.

Функция	**Описание**
connect()	Устанавливает соединение с брокером MQTT.
disconnect()	Разрывает соединение с брокером MQTT.
subscribe(topic)	Подписывается на сообщения топика.
unsubscribe(topic)	Отписывается от сообщений топика
publish(topic, payload, retain)	Публикует сообщение. topic — топик сообщения, например "test/test/1" payload — сообщение, например "проверка". retain — если true, то брокер будет хранить сообщение данного топика даже при перезагрузке.
Сигналы (сигналы посылаются при наступлении определенного события)	
connected()	Сигнал посылается при установлении TCP соединения с брокером.
connacked()	Сигнал посылается когда брокер подтверждает готовность обмениваться данными.
disconnected()	Сигнал посылается при разрыве соединения.
received(topic, payload)	Сигнал вызывается при получении сообщения от брокера.
error(strErr)	Сигнал посылается при ошибке.
subacked(mid)	Сигнал посылается при подтверждении подписки. mid - количество подписок.
unsubacked(mid)	Сигнал посылается при подтверждении отписки. mid - количество подписок.

Примеры находятся в каталоге «Scripts\Examples»: «25_SimpleMQTT.js», «26_MQTT.js».

Версия документа

Версия	**Изменения**
Версия **1.5** 19.06.2017	Добавлена возможность отправлять запросы GET: функция networkRequestGET, сигнал signalAnswerGET.
Версия **1.4** 29.09.2015	Добавлен протокол MQTT. Добавлены функции запроса АЦП: requestShADC, signalUpdateADC. Таблицы 3...9 перемещены в приложение А (А1..А7).
Версия **1.3** 20.04.2015	Добавлены функции для работы с журналом событий: fileExists, writeFile, getEventCount, getEvent, getEventPart, getEventZone, getEventDev, getEventUnit, isUnitSh, isUnitRl, isUnitRd, getEventKey, getEventAddr, getEventDescr, getEventUser, getEventDate, signalNewEvent
Версия **1.2**, 19.01.2015	Обновлена **Таблица А.7** «Состояния шлейфов, контролируемых цепей, приборов, разделов.»
Версия **1.1**, 10.10.2014	Устранена опечатка в функциях API: signa**l**UpdatePart и **s**ignalUpdateSh.
Версия **1.0**, 25.08.2014	Тестовая версия, опубликована только на форуме.