LogicMachine

Руководство программиста

1.1.1.3 2013.07.22.

Назначение руководства.

В данном документе описывается ...

Рруководство предназначено для:

- Инженеров проектировщиков;
- Инженеров инсталляторов.

СОДЕРЖАНИЕ

▲ ... ►

Авторские права

Товарные знаки

Уведомление

EVIKA сохраняет за собой право вносить изменения в данный документ без оповещений.

EVIKA не несет ответственности за любые ошибки, которые могут быть допущены в данном документе.

Техническая поддержка

Ремонт устройств реализованных на территории РФ и СНГ осуществляется EVIKA.

Ремонт устройств реализованных на территории стран ЕвроСоюза осуществляется Embedded Systems SIA.

Служба технической поддержки:

Время работы:	по рабочим дням Понедельник, .., Пятница 09:00 .. 18:00 (Москва: GMT + 04:00).
Телефон:	8-800-775-06-34 (звонки из любых регионов России - бесплатны).
E-Mail:	Support@Evika.Ru
Site:	www.Evika.Ru

Терминология

ПК, Персональный Компьютер.

Инсталлятор

Специалист создающий систему, в том числе подключающий и настраивающий устройство для работы в этой системе.

KNX, KNX/EIB

Один из современных стандартов распредёлённого управления инженерным оборудованием, широко применяющийся для целей диспетчеризации и автоматизации зданий.

ETS

Программа на ПК Инсталлятора предназначенная для обслуживания и настройки сетей KNX.
http://www.konnex-russia.ru/knx-standard/knx-tools/ets/

▲ ... ◄

LM Remote services (управление LogicMachine через HTTP запросы)

LM Remote services – это сервис, позволяющий управлять **Logic Machine** используя HTTP протокол.
Через **LM Remote services** можно:

- получать информацию об объектах LM и изменять их свойства,
- получать сообщения о тревожных событиях и ошибках в системе.

LM Remote services – это сервис, позволяющий управлять Logic Machine используя GET HTTP запросы. Задавая параметры и значения в URL, можно получать информацию об объектах, изменять их свойства, получать сообщения о тревожных событиях и ошибках в системе в удобном формате

LM Remote services – это сервис, позволяющий управлять Logic Machine используя GET HTTP запросы. Задавая параметры и значения в URL, можно получать информацию об объектах, изменять их свойства, получать сообщения о тревожных событиях и ошибках в системе в удобном формате

URL

Для использования примера измените IP адрес и пароль в соответствии с вашими настройками LM

```
http://remote:remote@192.168.0.10/cgi-bin/scada-remote/request.cgi?m=rss&r=alerts
```

Параметры

`m` формат возвращаемого значения

- `json`
- `xml`
- `rss` только для сообщений о тревожных событиях и ошибках

`r` типы функций

- `alerts` возвращает последние 50 сообщений о тревожных событиях

  Формат возвращаемых значений:
  - `alert` текст сообщения
  - `time` время события (формат UNIX timestamp)
  - `date` дата и время события (формат RFC date)
- `errors` возвращает последние 50 сообщений об ошибках

  Формат возвращаемых значений:
  - `error` текст сообщения об ошибке
  - `script` имя скрипта в котором произошла ошибка
  - `time` время записи сообщения об ошибке (формат UNIX timestamp)
  - `date` дата и время записи сообщения об ошибке (формат RFC date)
- `objects` возвращает список объектов, имеющих признак «export», отсортированных по времени обновления текущего значения

  Формат возвращаемых значений:
  - `address` адрес объекта (например «1/1/1»)

- `name` имя объекта (например «My object»)
- `data` значение объекта (например 42 или «01.01.2012»)
- `datatype` тип данных объекта (например 1 или 5.001)
- `time` время обновления значения объекта в формате UNIX timestamp
- `date` дата и время обновления значения объекта в формате RFC date
- `comment` комментарий (например «Светильник на втором этаже»)
- `tags` необязательный список меток объекта (например «Свет», «Второй этаж»)

- `grp` вызов функций по управления объектами

  Параметры:
  - `fn` имя функции, обязательно
    - `getvalue` возвращает текущее значение объекта
    - `find` возвращает информацию об объекте
    - `write` установка нового значения объекта (посылка соответствующей телеграммы на шину KNX)
    - `response` посылка на шину KNX телеграммы response к объекту
    - `read` посылка на шину KNX телеграммы read к объекту
    - `update` установка нового значения объекта только в Logic Machine без посылки телеграммы на шину KNX
  - `alias` имя или адрес объекта, обязательно
  - `value` новое значение объекта, обязательно для функций write / response / update, кроме типов данных time и date

    параметры для типа данных `time`
    - `day` число (0-7), день недели, необязательно
    - `hour` число (0-23) час
    - `minute` число (0-59) минуты
    - `second` число (0-59) секунды

    Параметры для типа данных `date`
    - `day` число (1-31) день месяца
    - `month` число (1-12) номер месяца
    - `year` число (1990-2089) год
  - `datatype` в случае если объект не определен в базе данных Logic Machine, то для функций write / response / update можно задать тип данных объекта, необязательный

    Возможные значения:

    `bool` `bit2` `bit4` `char` `uint8` `int8` `uint16` `int16` `float16`
    `time` `date` `uint32` `int32` `float32` `access` `string`

Примеры

Записать значение 50 в объект `1/1/1`

```
http://remote:remote@192.168.0.10/cgi-bin/scada-remote/request.cgi?m=json&r=grp&fn=write&alias=1/1/1&value=50
```

Записать значение типа boolean в объект 1/1/2. Вы можете использовать как true или false, так 1 или 0

```
http://remote:remote@192.168.0.10/cgi-bin/scada-remote/request.cgi?m=json&r=grp&fn=write&alias=1/1/2&value=true
```

Можно задать тип данных который преобразуется значение в тип данных объекта. Например если мы зададим тип данных scale, записывая 50 в объект 1/1/1 имеющий тип byte, то получим значение объекта равное 127

```
http://remote:remote@192.168.0.10/cgi-bin/scada-remote/request.cgi?m=json&r=grp&fn=write&alias=1/1/1&value=50&datatype=scale
```

LM trend log – библиотека функций по получению статистики из модуля Logic Machine Trend Log

```
require('genohm-scada.trends')
```

Подключение библиотеки

```
trends.fetch(name, mode, period)
trends.fetchone(name, mode, period)
```

Получение одного или нескольких значений за период

Параметры:

- `name` - имя лога трендов, обязательно
- `mode` – режим 'day' (день), 'month' (месяц) или 'year' (год), обязательно
- `period` *optional*, will use current date if not specified – опционально, если не определен, будет использоваться текущая дата. Для задания периода должна использоваться переменная типа Lua table со следующими полями:
  - `day` - обязательно только для режима 'day'
  - `month` - обязательно только для режима 'day' и 'month'
  - `year` - обязательно для всех режимов

Возвращаемые значения:

`fetch` Функция возвращает переменную типа Lua table которая содержит данные за этот период или nil если ошибка. Число значений зависит от периода и настроек trend log. Например, в режиме 'month' функция может вернуть как данные за каждый день, так и одно значение в целом за весь месяц.

fetchone Функция возвращает одиночное значение для данного периода или nil в случае ошибки

Example:

```
require('genohm-scada.trends')
-- возвращаем текущее значение
today = trends.fetchone('Gas', 'day')

-- Получаем текущую дату как переменную типа Lua table
-- и устанавливаем дату на вчерашний день
date = os.date('*t')
date.day = date.day - 1
-- получаем значение на вчерашний день
yesterday = trends.fetchone('Gas', 'day', date)
```

`trends.NaN` Значение используемое для измерений, содержащих некорректные значения или которые не могут быть получены. По умолчанию значение равно 0, но можно установить как 0/0 (NaN – not a number – не число)

Example:

```
require('genohm-scada.trends')
-- используем NaN как ошибочное значение
trends.NaN = 0 / 0

-- получаем данные по потреблению горячей воды за 2011 год
value = trends.fetchone('Hot Water', 'year', { year = 2011 })

-- пользуемся свойством что ошибочное значение не равно само себе, NaN ~= NaN
if value ~= value then
        return
end
```

Semaphore library (Библиотека семафоров)

Библиотека семафоров используется для контроля выполнения параллельно запущенных скриптов с целью регулировать доступ к системным ресурсам контролера (последовательные порты, файлы и так далее)

```
require('sem')
```

Подключаем библиотеку

```
semaphore = sem.open(name, [value = 1])
```

Открываем семафор, функция возвращает handle (объект) семафора или ошибку.

Параметры:

- `name` - имя семафора, обязательно
- `value` - инициализирующее значение, опционально, по умолчанию 1 (семафор свободен)

```
semaphore:getvalue()
```

Функция возвращает текущее значение семафора или nil в случае ошибки. Значение «0» означает что семафор заблокирован

```
semaphore:post()
```

Функция увеличивает значение семафора на единицу, и этим разблокирует (освобождает) семафор.

```
semaphore:trywait()
```

Функция проверяет, если семафор заблокирован, возвращает значение true и уменьшает значение семафора на единицу, если значение не равно нулю. Если семафор свободен возвращает false

---

```
semaphore:wait()
```

Функция уменьшает значение семафора на единицу. ВАЖНО: функция будет блокировать выполнение скрипта до тех пор пока значение семафора не увеличится. В большинстве случаев лучше использовать функцию `trywait`

`semaphore:close()`

Функция закрывает объект семафора. Автоматически вызывается когда выполнение скрипта завершается

Пример (экземпляры событийного скрипта которые не могут выполняться параллельно, с возможностью выхода по timeout)

```
require('sem')
semaphore = sem.open('eventlock')
timeout = 5 * 10
-- ждем когда выполнится параллельно запущенная копия скрипта или 5 секунд
while not semaphore:trywait() and timeout > 0 do
      sleep(0.1)
      timeout = timeout - 1
end
 -- выполняем код скрипта
-- разблокируем семафор
semaphore:post()
```

LogicMachine serial library (Библиотека последовательных портов)

Для использования функций последовательных портов необходимо подключить соответствующую библиотеку:

```
require('serial')
```

serial.open()

```
res, err = serial.open(device[, params])
```

Открывает порт.
Возвращает

res **port handle,** или **nil** если возникла проблема.

err сообщение о ошибке.

Входные параметры:

device Имя устройства порта.

params таблица параметров
необязательный:

baudrate Скорость:
300, 600, 1200, 2400, 4800, 9600,
19200, 38400, 57600, 115200, 230400.
по умолчанию: 115200.

parity Четность:
'none', 'even', 'odd'
по умолчанию: 'none'

databits Длина:
5, 6, 7, 8.
по умолчанию: 8

stopbits Stop биты:
1, 2.
по умолчанию: 1

duplex Duplex режим:
'full', 'half'.
по умолчанию: 'full'
Для RS-485 используется 'half'.

port:read()

```
res, err = port:read(bytes)
```

Считываете указанное количество байтов из порта.
Возвращает

res бинарную строку.

err сообщение о ошибке, если возникла проблема.

Входные параметры:

bytes Количество байт для считывания.

Внимание! До завершения чтения указанного количества байт выполнение программы приостанавливается.

port:read()

```
res, err = port:read(bytes, timeout)
```

Считывает из порта либо указанное количество байтов, либо всё поступившее за указанное время.
Возвращает

res	бинарную строку или **nil** при проблемах.
err	сообщение о ошибке, если возникла проблема.

Входные параметры:

bytes	Количество байт для считывания. Если считано всё но время ожидания не окончилось, функция завершается.
timeout	Длительность приёма информации для считывания. Если время вышло, функция завершается. Секунды, квант: 0.1 s.

Выходные параметры:

bytes	Количество считанных байт.

port:flush()

port:flush()
Освобождает буферы порта.

- Отсылается непреданная информация из буфера.
- Освобождается память используемая буферами.

port:close()

port:close()
Освобождает объект. Для повторных обращений его следует открыть.

Пример использования LM serial Библиотеки

(resident script, RS-485 echo test):

```
-- open port on first call
if not port then
  require('serial')
  port = serial.open('/dev/ttyS2', { baudrate = 9600, parity = 'even', duplex = 'half'})
  port:flush()
end

-- port ready
if port then
  -- read one byte
  char = port:read(1, 1)
  -- send back if read succeeded
  if char then
    port:write(char)
  end
end
```

◄