SystemeHD Works

Библиотека SeHVAC.lua

Описание компонентов

Редакция 1.0

Systême soft

Содержание

1. ОБЩАЯ ИНФОРМАЦИЯ

1.1. О продукте

SeHVAC.lua – набор объектов и функций, необходимых для разработки проектов автоматизации в сегменте HVAC.

Библиотека содержит в себе компоненты, позволяющие ускорить разработку алгоритмов управления системами отопления, вентиляции, освещения и др. Встроенные возможности языка LUA делают реализацию работы с системами реального времени достаточно трудоемкой задачей, которую можно значительно упростить при использовании данной библиотеки.

1.2. Подготовка к работе

1.2.1. Требования к окружению

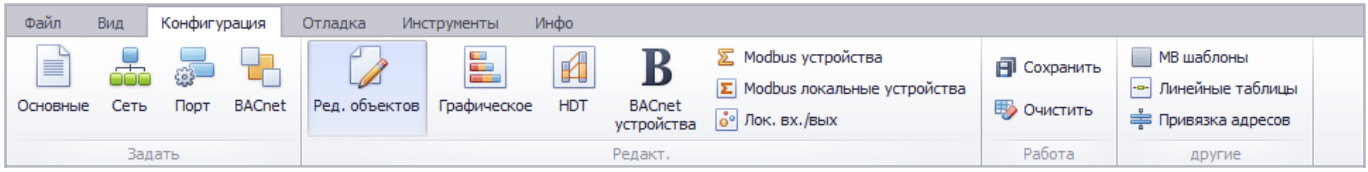
Для применения библиотеки SeHVAC.lua необходима установленная среда разработки **SystemeHD Works** не младше версии XX.XX.XX

1.2.2. Установка, удаление или восстановление

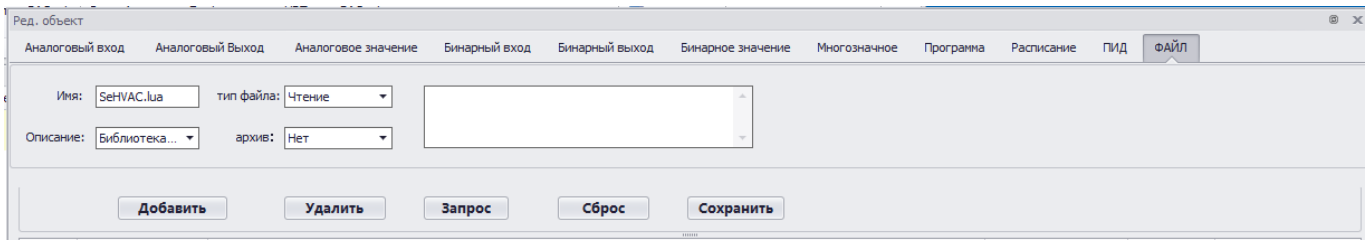
Библиотека SeHVAC.lua представляет собой текстовый файл с расширением lua, не требующий установки.

1.3. Встраивание SeHVAC.lua в проекты SystemeHD Works

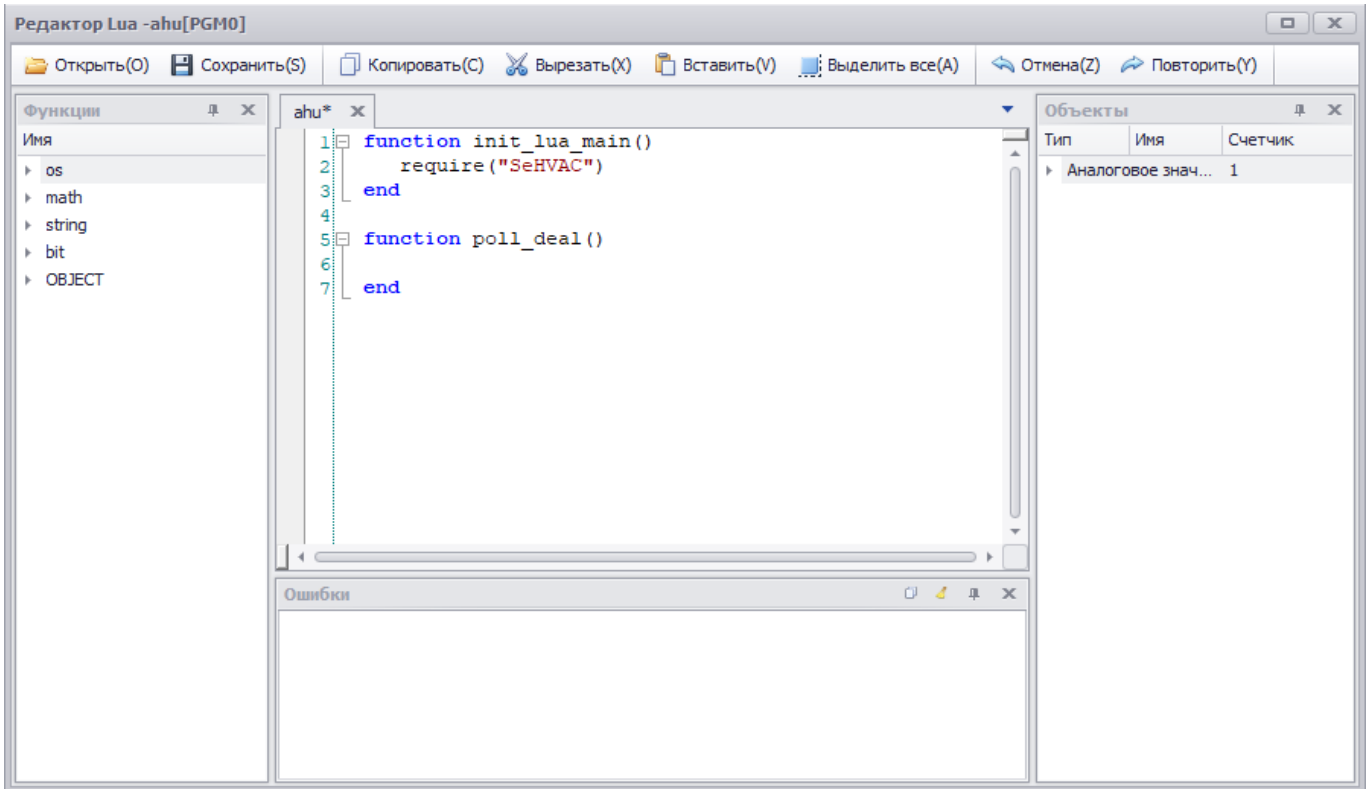
Чтобы начать пользоваться библиотекой необходимо ее подключить в конфигурации проекта. Для этого в разделе **Конфигурация** необходимо открыть редактор объектов: Объектов-ФАЙЛ нужно добавить новый файл с названием SeHVAC.lua.



В редакторе объектов необходимо перейти на вкладку **ФАЙЛ** и добавить файл с названием **SeHVAC.lua**.



В той программе, где требуются функции или объекты этой библиотеки необходимо подключить ее, используя служебную команду **require(“SeHVAC”).** После этого программа получит доступ ко всему функционалу библиотеки.



ПРИМЕЧАНИЕ

После первой загрузки проекта в контроллер необходимо загрузить файл SeHVAC.lua в контроллер. Для этого необходимо функцией загрузки файлов в режиме отладки. Более подробно см. Руководство пользователя SystemeHD.

2. РАБОТА С БИБЛИОТЕКОЙ

2.1. Состав библиотеки

Библиотека подключается к программе с помощью служебной команды **require("SeHVAC")**. При наличии загруженного в контроллер файла библиотеки программа получает доступ ко всем ее объектам.

Библиотека содержит в себе следующие элементы:

- Функции работы с битами (упаковка 16 бит в «слово», распаковка «слова» в массив из 16 бит)
- Функции работы с объектами контроллера
- Функциональные блоки работы со временем, устройствами
- Функциональные блоки регулирования

ПРИМЕЧАНИЕ

Библиотека SeHVAC написана с использованием языка программирования LUA 5.1. Для успешного применения библиотеки в своих проектах необходимо предварительно изучить документацию по этому языку.

2.2. Работа с функциями

Функции – это элементы, не требующие инициализации, применяются непосредственно в том месте программы, где необходимы и выдают немедленно результат. Они не имеют возможности хранить в себе какие-то промежуточные результаты и фактически просто позволяют повторно использовать некий алгоритм, используя короткую запись. У функции может быть любое количество входных параметров (аргументов) и выходов (возвращаемых функцией значений).

Например, функция преобразования числа любого типа в значение типа **bool**:

```
ToBool = function(value)
    if type(value) == "boolean" then
        return value
    else
        if value > 0 then
            return true
        else
            return false
        end
    end
end
```

Эта функция позволяет преобразовать любое число в тип **bool**, принимающий значения **true** или **false**. На вход функции подается число любого типа и если его значение больше **0**, то функция возвращает значение **true**, в противном случае возвращается значение **false**. Эта функция описана в библиотеке и для применения этого преобразования в вашей программе необходимо прописать следующую команду:

```
IntVal = 1
BoolVal = ToBool(IntVal)
```

Результатом работы этой функции будет присваивание переменной **BoolVal** значения **true/false** в зависимости от значения **IntVal**.

Список доступных функций приведен в разделе 3.1.

2.3. Работа с функциональными блоками

Функциональные блоки отличаются от функций тем, что функциональный блок имеет возможность хранить внутри себя значения промежуточных вычислений, параметров работы блока и т.д. Для реализации этих функций, а также для повышения удобства работы с библиотекой функциональные блоки библиотеки строятся на базе объектно-ориентированного подхода. Каждый функциональный блок представляет собой объект, содержащий в себе следующие элементы:

- Параметры – элементы, хранящие в себе некоторые значения, задаваемые пользователем, или вычисляемые в процессе работы блока.
- Методы – это функции, которые содержит в себе объект, определяющие его возможности.

2.3.1. Методы

Методы по внешнему виду очень похожи на функции (см. 2.2) с тем лишь отличием, что они являются неотъемлемой частью блока и работают не только с теми параметрами, которые указываются при их вызове, но и с внутренними параметрами конкретного экземпляра функционального блока.

Методы вызываются через специальный символ «:». Например:

```
PuRtn = rt1:run(PuWrkSta, PuRtnRst)
```

В этом примере rt1 – это функциональный блок, а run(PuWrkSta, PuRtnRst) – это метод run с параметрами PuWrkSta, PuRtnRst. Входные параметры метода позволяют изменить его работу, но кроме этих явно заданных параметров метод использует ряд внутренних параметров со значениями, индивидуальными для каждого экземпляра блока.

2.3.2. Параметры

Параметры хранят значения, необходимые для работы блока и значения их задаются для каждого экземпляра функционального блока независимо.

Для доступа к значениям внутренних параметров необходимо использовать после имени блока точку, например:

```
rt1:run(PuWrkSta, PuRtnRst)

PuRtn = rt1.out
```

Здесь out – это внутренний параметр блока, который хранит в себе результат работы метода run. Если в программе необходимо использовать результат работы блока несколько раз, то метод необходимо вызывать один раз, а в тех местах, где необходимо использовать результат его работы можно использовать уже выходной параметр функционального блока.

Если для задания внутренних параметров у блока есть отвечающий за это метод, то правильным стилем программирования будет использование метода, а не прямое задание значений параметрам. Дело в том, что методы не только задают значения параметрам, но и проверяют правильность вводимых данных. При прямом присвоении значений обязанность по этой проверке ложится уже на плечи программистов. В связи с тем, что язык LUA не имеет возможности ограничить доступ к параметрам, решение о том, как использовать блок, принимает программист.

2.3.3. Наследование

Базовым объектом для всех блоков является объект **Class**. В нем определены все возможности объектов по инициализации и наследованию свойств объектов-родителей. На базе этого объекта строятся все остальные функциональные блоки работы со временем, оборудованием, с процессами регулирования и т.д.

Функциональные блоки могут создаваться как на основе базового объекта **Class**, так и на основе других объектов. Например:

```
NewObject = Class()             -- создание типа на основе базового Class
ChildObject = Class(NewObject)  -- создание типа на основе типа NewObject.
```

При создании типа на базе родительского NewObject происходит наследование всех свойств и возможностей родительского типа. Таким образом блок управления преобразователем частоты FCtrl строится на базе родительского типа DeviceObj, в котором определены свойства и функции, свойственные всем исполнительным механизмам. Тип FCtrl повторно не описывает эти функции, а пользуется возможностями родительского типа и при этом добавляет функционал, характерный именно для преобразователей частоты. При корректировке функционала DeviceObj произойдет корректировка поведения и объектов-наследников.

В связи с тем, что каждый экземпляр функционального блока в программе должен обладать собственным набором параметров, при работе с ними необходимо совершать дополнительные действия – инициализацию каждого экземпляра функционального блока. При этом каждый экземпляр должен иметь уникальное в рамках программы имя. При инициализации создается объект, обладающий всеми возможностями того типа, на базе которого он создан, но при этом имеющий свой собственный, независимые от всех остальных экземпляров, набор параметров.

Создание и инициализация функциональных блоков должна производиться в разделе **init_lua_main** вашей программы, так как этот раздел выполняется только при загрузке контроллера. Повторная инициализация объекта сбросит его текущее состояние в исходное, поэтому не стоит инициализировать объекты в теле главного цикла программы.

2.3.4. Пример работы с блоком

Рассмотрим пример работы с функциональным блоком таймера **RT**.

Для начала работы с блоком создаем и инициализируем экземпляр таймера:

```
rt1 = RT(0, getIO("PuRtn"))
```

Здесь:

- rt1 – имя создаваемого экземпляра таймера;
- RT(0, getIO("PuRtn")) – инициализация экземпляра блока типа RT, аргументами которого являются параметры конфигурации таймера: 0 – таймер считает часы, getIO("PuRtn") – начальное значение таймера, получаемое из текущей сохраненной наработки насоса.

Результатом выполнения этой команды будет создание таймера **rt1**, настроенного считать часы и на старте имеющего значение наработки насоса, посчитанное до перезагрузки контроллера. решен

Пример использования таймера, считающего наработку насоса приведен ниже:

```
PuWrkSta    = getIO(“PuWrkSta”)     -- получение статуса работы насоса
PuRtnRst    = getIO(“PuRtnRst”)     -- значение сигнала сброса наработки насоса
PuRtn = rt1:run(PuWrkSta, PuRtnRst) -- подсчет наработки и ее сброс
setIO(“PuRtnRst”, 0)                -- возвращаем сигналу сброса значение 0
setIO(“PuRtn”, PuRtn)               -- запись значения наработки в сигнал BACnet
```

Рассмотрим подробно приведенный выше скрипт:

- PuWrkSta = **getIO**(“PuWrkSta”) – этой командой мы запрашиваем значение статуса работы насоса, который создан в конфигурации контроллера под именем «**PuWrkSta**». Функция **getIO** находит объект с указанным в аргументе именем и возвращает его значение.
- PuRntRst = **getIO**(“PuRtnRst”) – получаем текущее значение сигнала сброса наработки.
- PuRtn = **rt1:run**(PuWrkSta, PuRtnRst) – непосредственная работа блока таймера. В качестве аргументов передается значение статуса работы насоса и значение сигнала сброса таймера. Если PuWrkSta равен 1, то таймер осуществляет подсчет отработанного насосом времени, в противном случае счет останавливается и таймер ожидает прихода значения 1. Сигнал PuRtnRst осуществляет обнуление таймера при значении этого сигнала сброса равном 1. Текущее посчитанное время помещается таймером в переменную PuRtn.
- **setIO**(“PuRtnRst”, 0) – эта команда присваивает сигналу сброса наработки значение 0, что снимает с таймера блокировку работы.
- **setIO**(“PuRtn”, PuRtn) – команда записывает текущую посчитанную наработку в объект контроллера с именем PuRtn, что позволяет передать это значение на верхний уровень.

Полный список доступных в библиотеке блоков приведен в разделе 3.2.

3. ОПИСАНИЕ КОМПОНЕНТОВ БИБЛИОТЕКИ

3.1. Функции

Название	Аргументы	Описание.
ToBool(value)	value – число любого типа	Преобразование числа любого типа к типу boolean (false/true)
XOR(bit1, bit2)	bit1 – boolean bit2 - boolean	Выполнение операции XOR с двумя переменными типа boolean
ToInt(value)	value - boolean	Преобразование значение типа boolean в значение типа int: 0/1
setIO(objname,val)	objname – string val – любой тип	Универсальная функция записи значения в объект BACnet контроллера. Поиск объекта осуществляется по имени objname, в который записывается значение val.
getIO(objname)	objname – string	Универсальная функция получения значения объекта BACnet контроллера. Поиск объекта осуществляется по имени objname.
getTmpSns (name, hiLimit, loLimit)	name – string hiLimit – number loLimit - number	Функция возвращает значение объекта BACnet контроллера с именем name и в случае выхода значения за пределы hiLimit и loLimit возвращается статус аварии.
Bits2Word(bitarr)	bitarr – массив из 16 элементов типа numeric со значениям 0/1	Функция возвращает 16-ти битное слово, скомпонованное из элементов массива.
Word2Bits (word)	word – 16-ти битное слово	Функция возвращает массив из 16-ти элементов типа numeric со значениями 0/1, соответствующими значеням битов слова word.
getComAlm(ALM)	ALM – таблица с элементами типа number или boolean	Функция возвращает true, если хотя бы один из элементов таблицы имеет значение true. В противном случае возвращается false.
CurveXY(X, Y, value, limit)	X – массив точек по оси x Y – массив точек по оси y. value – текущее значение по оси x limit – деактивация экстраполяции значений при выходе value за границы значений массива X.	Вычисление значения величины методом линейной интерполяции функции, заданной набором точек с координатами x,y.

3.1.1. ToBool

Функция преобразования числа любого типа в тип boolean**,** имеющий значения true или false**.** В связи с тем, что в языке LUA тип boolean нельзя использовать в арифметических действиях совместно с типом numeric, эта функция позволяет гарантировать то, что используемая вами переменная содержит значение типа boolean.

Синтаксис:

BoolVal = ToBool(Value)

Входные значения:

- Value – значение любого численного типа

Возвращаемые значения:

- BoolVal - число типа boolean.

Пример использования:

```
local BoolVal           -- объявление переменной типа boolean
local IntVal = 1        -- объявление переменной типа number
BoolVal = ToBool(IntVal)
```

В результате работы программы в переменную BoolVar будет помещено значение true.

3.1.2. XOR

Операция исключающего ИЛИ с двумя переменными любого числового типа или boolean. Выдает true только в том случае, когда только один из входов имеет значение true/1. Если на входе у обоих входов значение true/1, то на выходе будет false.

Таблица истинности XOR

Вход 1	Вход 2	Выход
0	0	0
1	0	1
0	1	1
1	1	0

Синтаксис:

BoolVal = XOR(Val1, Val2)

Входные значения:

- Val1 – значения числового типа или boolean
- Val2 – значения числового типа или boolean

Возвращаемые значения:

- BoolVal - число типа boolean.

Пример использования:

```
local PuWrkSta_DI = getIO("PuWrkSta")--получение значения дискретного входа контроллера
local PuWrkSta_nc = true              --настройка полярности входа: true-НЗ, false-НО
local PuWrkSta = XOR(PuWrkSta_DI, PuWrkSta_nc)  --обработанный вход с учетом полярности
```

Программа считывает текущее значение с дискретного входа и записывает во внутреннюю переменную обработанное с учетом полярности значение.

3.1.3. ToInt

Функция преобразования значения типа boolean, в значение типа number. В связи с тем, что в языке LUA тип boolean нельзя использовать в арифметических действиях совместно с типом numeric, эта функция позволяет гарантировать то, что используемая вами переменная содержит значение типа number.

Синтаксис:

IntVal = ToInt(value)

Входные значения:

- value – значение типа boolean

Возвращаемые значения:

- IntVal - число типа number.

Пример использования:

```
Enable = true                       -- переменная типа boolean
PidOut = getIO("HtgPidOut")         -- выход ПИД-регулятора
VlvVal = ToInt(Enable) * PidOut     -- операция возможная только с типом number
setIO("HtgVlvVal", VlvVal)          -- запись значения в аналоговый выход
```

В зависимости от значения переменной Enable на клапан подается либо значение переменной PidOut, либо 0

3.1.4. setIO

Универсальная функция записи значения в объект BACnet контроллера.

Синтаксис:

WriteResult = setIO(Name, Value)

Входные значения:

- Name – переменная типа string, содержащая имя BACnet объекта контроллера.
- Value – значение, которое нужно записать в объект с именем name.

Возвращаемые значения:

- WriteResult - true, если запись прошла успешно, false, если запись прошла неудачно,

Пример использования:

```
AlmSta = true                           --аварийное значение
```

```
WriteResult = setIO("HtgPuAlm", AlmSta)     --запись в объект с именем "HtgPuAlm"
if WriteResult == false then
    print("Сигнала не существует")
end
```

3.1.5. getIO

Универсальная функция чтения значения из объекта BACnet контроллера.

Синтаксис:

Value = getIO(Name)

Входные значения:

- Name – переменная типа string, содержащая имя BACnet объекта контроллера.

Возвращаемые значения:

- Value - значение объекта BACnet при успешной операции, nil-запись прошла неудачно,

Пример использования:

```
PuWrkSta = getIO("HtgPuWrkSta") --получение статуса работы насоса
if PuWrkSta == nil then --если такого объекта нет, то выдается аварийное сообщение
    print("Несуществующий сигнал")
    PuWrkSta = -1
end
```

3.1.6. getTmpSns

Функция получения значения объекта BACnet контроллера с контролем выхода за допустимые границы. Основана на функции getIO.

Синтаксис:

TmpVal, AlmLo, AlmHi = getTmpSns(Name, HiLimit, LoLimit)

Входные значения:

- Name – переменная типа string, содержащая имя BACnet объекта контроллера.
- HiLimit – верхняя допустимая граница измерения
- LoLimit – нижняя допустимая граница измерения

Возвращаемые значения:

- TmpVal - значение объекта BACnet при успешной операции, nil-запись прошла неудачно,
- AlmLo - статус выхода за нижнюю границу
- AlmHi - статус выхода за верхнюю границу

Пример использования:

```
SuTmpSpt = getIO("SuTmpSpt")   --получение уставки температуры притока
SuTmpSptHyst = 2.0             --задание максимального отклонения температуры от уставки
SuTmp, almLo, almHi = getTmpSns("SuTmp", SuTmpSpt+SuTmpSptHyst, SuTmpSpt - SuTmpSptHyst)
if almLo then
    print("Низкая температура притока")
end
if almHi then
    print("Высокая температура притока")
end
```

3.1.7. Bits2Word

Функция формирования битового слова из массива битов

Синтаксис:

Word = Bits2Word(BitArray)

Входные значения:

- BitArray – массив из 16 элементов, содержащий значения 0/1 (false/true).

Возвращаемые значения:

- Word – битовое слово.

Пример использования:

```
AlmPu = false   --статус аварии насоса
AlmFan = false  --статус аварии вентилятора
AlmFire = true  --статус пожара
AlmTs = false   --статус сработки термостата
AlmFlt = false  --статус засора фильтра
AlmArray = {}   --создание массива аварий
AlmArray[0] = AlmPu --задание значений массива
AlmArray[1] = AlmFan
AlmArray[2] = AlmFire
AlmArray[3] = AlmTs
AlmArray[4] = AlmFlt
AlmWord = Bits2Word(AlmArray) --компоновка слова аварий
setIO("AlmWord1", AlmWord)    --запись значения слова в объект
```

3.1.8. Word2Bits

Функция разбора битового слова по битам. Полученные биты помещаются в массив из 16 элементов.

Синтаксис:

BitArray = Word2Bits(Word)

Входные значения:

- Word – битовое слово

Возвращаемые значения:

- BitArray – массив из 16 элементов, содержащий значения 0/1

Пример использования:

```
FcStaWord = getIO("FcStaWord1") --получение слово состояния ПЧ
FcStaArr = Word2Bits(FcStaWord) --разбор слова на биты
AlmSta = FcStaArr[0]           --статус неисправности ПЧ
RdySta = FcStaArr[1]           --статус готовности к запуску
LocSta = FcStaArr[9]           --статус местного режима
RunSta = FcStaArr[10]          --статус работы
```

3.1.9. getComAlm

Функция получения обобщенного статуса аварии. Функция принимает на вход таблицу объектов числовых типов либо boolean и если хотя бы один элемент будет иметь значение 1 или true, то на выходе будет значение true.

Синтаксис:

ComAlm = getComAlm(ALM)

Входные значения:

- ALM – таблица любого размера со значениями числовых типов либо типа boolean

Возвращаемые значения:

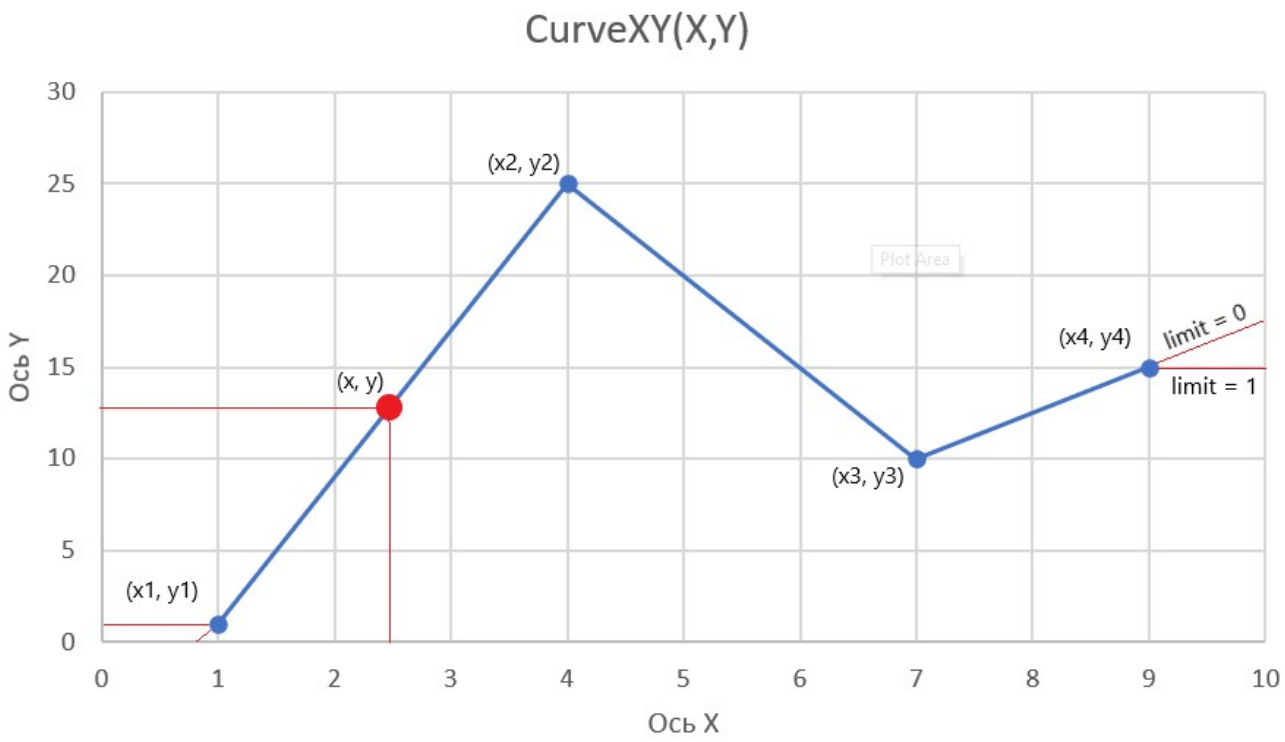
- ComAlm – значение типа boolean

Пример использования:

```
ALM = {}            --создание таблицы
ALM.AlmFire = true  --заполнение таблицы
ALM.AlmFlt = false
ALM.AlmPu = false
ALM.AlmFa = false
ComAlm = getComAlm(ALM) --получение значения обобщенной аварии
setIO("AlmLamp", ComAlm) --управление лампой аварии
```

3.1.10. CurveXY

Функция вычисления значения по графику, заданному набором точек с координатами x и y. Значение функции между заданными точками по оси x определяется методом линейной интерполяции по соседним точкам. Значения вне заданного диапазона точек в зависимости от настроек функции определяются методом линейной экстраполяции, либо ограничиваются граничными значениями функции.



Синтаксис:

```
y = CurveXY(X, Y, x, Limit)
```

Входные значения:

- X – массив любого размера числовых значений. Отвечает за координаты по оси X
- Y – массив числовых значений с размерность, равной размерности массива X. Отвечает за координаты по оси Y.
- x – координата x искомой точки.
- Limit – настройка работы функции: при Limit = 1 значение функции за пределами заданного диапазона ограничивается граничными значениями, при Limit = 0 значение за границами диапазона рассчитывается методом линейной экстраполяции (см. рисунок).

ВАЖНО

Точки графика должны заноситься в порядке возрастания значений по оси X.

Возвращаемые значения:

- y – рассчитанное значение функции

Пример использования:

```
X = {}  -- создание массива X
Y = {}  -- создание массива Y
X[0], Y[0] = -32, 89 -- задание точек графика
X[1], Y[1] = -22, 93
```

```
x[2], y[2] = -5, 79
x[3], y[3] = 5, 61
Limit = 1   -- активация ограничения функции
Oat = getIO("Oat")  -- получение значения наружной температуры
HtgSuSpt = CurveXY(X, Y, Oat, Limit) -- расчет уставки подачи отопления
```

3.2. Функциональные блоки

Название	Описание
Class	Базовый объект для всех функциональных блоков
TICK	Выдача импульса в течение одного цикла с заданным периодом времени (в секундах)
RT	Счетчик времени. Считает время активного состояния входа. Настраивается на подсчет времени в секундах, минутах или часах.
TRIG	Триггер. Выдает импульс в течение одного цикла при изменении состояния бинарного значения на входе. Настраивается на отслеживание любого изменения сигнала, либо на отслеживание перехода значения из 0 в 1 (из false в true), либо из 1 в 0 (из true в false).
OSC	Блок выдачи импульсов с заданным периодом и заданной продолжительностью.
PID	ПИД-регулятор
PWM	Блок работает в связке с ПИД-регулятором и позволяет при работе в нелинейной области работы клапана активировать ШИМ регулирование с заданной амплитудой и периодом.
HYST	Блок гистерезиса для предотвращения дребезга при активации дискретного сигнала. Например, активация летнего режима работы вентиляционной установки при определенном диапазоне наружной температуры.
DELAY	Задержка дискретного сигнала. Возможна задержка как на активацию, так и на деактивацию сигнала
RAMP	Замедление изменения аналогового сигнала путем задания максимального шага изменения сигнала за цикл.
ALARM	Блок фиксации аварийного состояния
DeviceObj	Базовый блок для исполнительных механизмов. Осуществляет контроль обратной связи и фиксацию аварийных состояний.
FCtrl	Расширение блока DeviceObj для управления преобразователем частоты
AHU_DmpDig	Блок управления воздушной заслонкой с дискретным управлением.
AHU_FltDig	Блок контроля состояния фильтра
AHU_Summer	Блок определения режима ЗИМА/ЛЕТО по температуре наружного воздуха, либо вручную оператором.
AHU_Start	Блок запуска установки. Предусмотрен ручной режим и автоматический по внешнему сигналу или расписанию.
AHU_Preheat	Блок предварительного прогрева теплообменника вентиляционной установки перед запуском в зимнем режиме.
AHU_WatHtg	Блок водяного нагрева приточной установки: клапан регулировани, управление насосом и защита от замерзания.

3.2.1. Class

Базовый функциональный блок для всех остальных блоков. Он содержит в себе методы, реализующие функционал объектно-ориентированного подхода: наследование, инициализация экземпляров. Напрямую создавать экземпляры этого блока не требуется, так как он несет исключительно служебную функцию и используется исключительно при создании других объектов.

Синтаксис:

NewClass = Class()

Результатом этой операции будет создание нового функционального блока. Этот блок будет полностью повторять функционал базового функционального блока Class. Для расширения функционала объекту NewClass необходимо добавить параметры и методы (см. руководство по языку LUA).

3.2.2. TICK

Функциональный блок выдачи импульсов продолжительностью в один программный цикл. Импульс выдается в виде значения типа boolean.

Инициализация экземпляра:

```
tick1 = TICK()
```

Методы блока:

TICK:run(sec) - выдается значение true продолжительностью один программный цикл через каждые **sec** секунд (основное действие блока)		
Входные параметры	sec	Период в секундах, с которым выдаются импульсы
Выходные параметры	Out	Импульс длительностью в один программный цикл
Описание Наряду с выдачей значения непосредственно при вызове метода run блок сохраняет это значение во внутреннем параметре Out. Поэтому, после того как вы вызвали этот метод в программном цикле, для доступа к этому значению в других местах программы, можно воспользоваться сохраненным значением.		

Пример использования блока TICK:

```
function init_lua_main()
    require("SeHVAC")
    tick1 = TICK()  -- инициализация блока
    counter = 0     -- инициализация счетчика
 end

 function poll_deal()
    if tick1:run(10) == true then     -- вызов основного метода блока (генерация
                                      импульса каждые 10 секунд)
        counter = counter + 1   -- увеличение счетчика на 1 при появлении импульса
```

```
        end
        WrnLmp = tick1.Out          -- зажигание лампы при импульсе.
    end
```

3.2.3. RT

Функциональный блок выдачи подсчета времени активности сигнала. В зависимости от настройки блока ведется подсчет времени в секундах, в минутах или в часах. Самое распространенное применение блока – подсчет времени наработки оборудования.

Инициализация экземпляра:

```
rt1 = RT(Mode, InitVal)
```

Параметры инициализации блока		
Mode	0 … 2	Параметр задает режим подсчета времени: 0 – подсчет в часах 1 – подсчет в минутах 2 – подсчет в секундах
InitVal	Числовое значение	Начальное значение счетчика времени

Методы блока:

RT:run(InValue, Reset)		
Входные параметры	InValue	Сигнал типа boolean, либо число со значениями 0 или 1. При активном значении параметра (1 или true) блок начинает подсчет времени активности этого значения.
	Reset	Сигнал типа boolean, либо число со значениями 0 или 1. При активном состоянии накопленное значение сбрасывается до 0.
Выходные параметры	out (внутренний параметр)	Посчитанное значение. В зависимости от настройки блока это либо секунды, либо минуты, либо часы.
Описание Если было задано начальное значение при создании блока, то подсчет продолжается с этого значения. При останове подсчета накопленное значение сохраняется и при возобновлении подсчета он продолжается с места остановки. При любой настройке единиц времени счетчика внутрений счет происходит в секундах, поэтому даже, если выбран режим счета в часах и блок работал несколько секунд, продолжение счета начнется не с начала часа, а с тех секунд, которые блок проработал до этого.		

RT:set(InValue)		
Входные параметры	Value	Сигнал числового типа.
Описание При вызове метода произойдет замена накопленного значения счетчика на значение параметра Value. Дальнейший подсчет продолжится уже с этого значения.		

Пример использования блока RT:

```
function init_lua_main()
    require("SeHVAC")
```

```
    rt1 = RT(0,getIO("PuRtn"))  -- инициализация блока с начальным значением, взятым из
BACnet объекта
 end

 function poll_deal()
    PuWrkSta = getIO("PuWrkSta") -- получение статуса работы насоса
    PuRtnRst = getIO("PuRtnRst") -- получение значения сигнала сброса наработки
    SetVal = getIO("SetPuRtn")  -- получение сигнала установки наработки
-- при получении сигнала установки наработки производится присваивание накопленному
значению счетчика нового значения
    if SetVal == 1 then
        rt1:set(getIO("PuRtn"))
        setIO("SetPuRtn",0)
    end
    rt1:run(PuWrkSta, PuRtnRst) -- выполнение подсчета наработки насоса
    setIO("PuRtn", rt1.out)     -- сохранение нового значения наработки насоса в
контроллере
 end
```

3.2.4. TRIG

Функциональный блок выдачи импульса в течение одного программного цикла при изменении дискретного сигнала на отслеживаемом входе. Тип отслеживания изменения сигнала настраивается при инициализации блока и может быть следующим:

- отслеживание переднего фронта изменения сигнала (из false в true)
- отслеживание заднего фронта изменения сигнала (из true в false)
- отслеживание любого изменения сигнала

Инициализация экземпляра:

```
trig1 = TRIG(Mode)
```

Параметры инициализации блока		
Mode	0 … 2	Параметр задает режим отслеживания изменения сигнала: 0 – передний фронт 1 – задний фронт 2 – любое изменение

Методы блока:

TRIG:run(InValue)		
Входные параметры	InValue	Сигнал типа boolean, либо число со значениями 0 или 1. Сигнал, изменение которого отслеживается блоком.
Выходные параметры	out (внутренний параметр)	Сигнал типа boolean, сигнализирующий об изменении сигнала согласно настройке блока.
Описание При изменении сигнала InValue (в зависимости от настроек при инициализации блока) метод возвращает значение true, которое остается активным в течение программного цикла. Выдача значения производится как непосредственно при выполнении метода, так и через параметр out.		

Пример использования блока TRIG:

```
function init_lua_main()
    require("SeHVAC")
    trigAlm = TRIG(0)    -- инициализация блока с контролем переднего фронта
    AlmCnt = 0           -- счетчик аварий
 end

 function poll_deal()
    PuAlmSta = getIO("PuAlmSta") -- получение статуса аварии насоса
    trigAlm:run(PuAlmSta)  -- отслеживание события возникновения аварии
-- при возникновении аварии значение true преобразовывается в 1 и прибавляется к
значению счетчика
    AlmCnt = AlmCnt + ToInt(trigAlm.out)
 end
```

3.2.5. OSC

Функциональный блок выдачи импульса заданной длины с заданным периодом.

Инициализация экземпляра:

```
osc1 = OSC()
```

Методы блока:

OSC:run(InValue, PulseTime, Period)		
Входные параметры	InValue	Сигнал типа boolean, либо число со значениями 0 или 1. Сигнал активации выдачи импульсов.
	PulseTime	Числовое значение. Задание длительности импульса в секундах
	Period	Числовое значение. Задание периода импульсов в секундах
Выходные параметры	out (внутренний параметр)	Сигнал типа boolean. При импульсе выдается значение true
Описание При получении активного сигнала InValue блок начинает генерировать импульсы согласно параметрам продолжительности импульса и его периода. Выдача значения производится как непосредственно при выполнении метода, так и через параметр out.		

Пример использования блока OSC:

```
function init_lua_main()
    require("SeHVAC")
    osc1 = OSC()   -- инициализация блока
    AlmSta = 0 -- Общий статус аварии
    WrnSta = 0 -- Общий статус предупредительной аварии
 end

 function poll_deal()
    AlmSta = getIO("PuAlmSta") -- получение статуса аварии насоса
    WrnSta = getIO("PuWrnSta") -- получение статуса предупредительной аварии
```

```
    AlmLamp = false
-- Если есть авария, то лама зажигется,
-- Если предупреждение, то лампа мигает каждые 2 секунды и горит секунду
-- В нормальном режиме лама не горит
    if AlmSta == 1 then
        AlmLamp = true
    else
        AlmLamp = osc1:run(WrnSta, 1, 2)
    end
    setIO("AlmLamp", AlmLamp)
end
```

3.2.6. PID

Функциональный блок ПИД-регулятора. Блок содержит пропорциональную, интегральную и дифференциальную части. Ключевая особенность блока в том, что он содержит в явном виде входной параметр со значением регулятора в предыдущий цикл. Это позволяет организовывать схемы управления из нескольких параллельных ПИД с синхронизированными выходами – например, управление клапаном водяного нагревателя вентиляционной установки, где требуется поддержание как температуры воздуха, так и контроль температуры обратной воды (смотри пример).

Инициализация экземпляра:

```
pid1 = PID()
```

Методы блока:

PID:updatePidParam(G, Ti, Td, Umax, Umin, StrokeTime, CtrlInt)		
Входные параметры	G	Числовое значение. Коэффициент пропорциональности
	Ti	Числовое значение. Время интегрирования. Может быть нулем. Секунды
	Td	Числовое значение. Время дифференцирования. Может быть нулем. Секунды
	Umax	Числовое значение. Максимальный выход регулятора.
	Umin	Числовое значение. Минимальный выход регулятора.
	StrokeTime	Числовое значение. Время полного хода штока клапана. В секундах.
	CtrlInt	Числовое значение. Период вычисления регулятора. В секундах. Фактический период определяется не только этим параметром, но и заданным временем цикла программы.
Описание Метод предназначен для задания параметров блока, отвечающих за настройку регулятора. При вызове этого метода происходит вычисление всех внутренних параметров регулятора		

PID:run(MV, SP, Mod, DZ, TSg)		
Входные параметры	MV	Числовое значение. Действительное значение регулируемого параметра.
	SP	Числовое значение. Уставка регулируемого параметра.

PID:run(MV, SP, Mod, DZ, TSg)		
	Mod	Числовое значение. Режим работы регулятора. Может принимать следующие значения: 0: регулятор остановлен 1: нормальное регулирование 2: регулятор выдает максимальное значение Umax 3: регулятор выдает минимальное значение Umin
	DZ	Числовое значение. Мертвая зона регулятора. Задается в тех же единицах, что и параметры MV и SP. При изменении сигнала на величину по модулю меньшую параметра DZ. Выход регулятора не изменяется по отношению к предыдущему циклу.
	TSg	Числовое значение. Предыдущее вычисленное значение регулятора.
Выходные параметры	out (внутренний параметр)	Числовое значение. Выход регулятора

Описание

Метод вычисления регулирующего воздействия. Выход определяется по следующей формуле:

Out = TSg + dU

Здесь TSg – это параметр метода, а dU – вычисленное приращение в текущем цикле. Величина dU ограничена в зависимости от диапазона регулирования (параметры Umax и Umin), а также от времени штока (StrokeTime).

При переводе управления агрегатом в ручной режим рекомендуется передавать ручное значение на вход TSg, чтобы при переводе системы в автоматический режим регулятор был в актуальном положении.

Пример использования блока PID:

```
function init_lua_main()
    require("SeHVAC")
    pidAir = PID()   -- инициализация регулятора по воздуху
    pidWater = PID() -- инициализация регулятора по воде
    G = 2
    Ti = 90
    Td = 0
    Stroke = 120
    Dz = 0.05
    pidAir:updatePidParam(G, Ti, Td, 100, 0, Stroke, 1) -- задание параметров ПИД
    pidWater:updatePidParam(G, Ti, Td, 100, 0, Stroke, 1)-- задание параметров ПИД
    valveOut = 0    -- начальное положение клапана
end

function poll_deal()
    Dat = getIO("Dat")          -- температура притока
    SptDat = getIO("SptDat")    -- уставка температуры притока
    Rwt = getIO("Rwt")          -- температура обратной воды
    SptRwt = getIO("SptRwt")    -- уставка температуры обратной воды
    ManVlvMd = getIO("ManVlvMd")-- ручной режим управления клапаном
    ManVlvVal = getIO("ManVlvVal") -- значение ручного регулирования
    pidaOut = pidAir:run(Dat, SptDat, 1, DZ, valveOut)  -- вычисление ПИД по воздуху
    pidwOut = pidWater:run(Rwt, SptRwt, 1, DZ, valveOut)-- вычисление ПИД по воде
```

```
 -- если активный ручной режим, то клапану задается ручное положение
 -- если активен автоматический режим, то на клапан подается большее из двух ПИД
значение
 -- для следующих вычислений ПИД используется значение valveOut
    if ManVlvMd == 1 then
        valveOut = ManVlvVal
    else
        valveOut = math.max(pidaOut, pidwOut)
    end
-- выдача на клапан итогового значения
    setIO("VlvOut", valveOut)
 end
```

3.2.7. PWM

Функциональный блок управления клапаном регулирования в режиме ШИМ. Блок необходим для точного поддержания температуры в случае, когда клапан находится в нелинейном режиме работы (примерно менее 8% открытия). В этом режиме происходит отрывание клапана в режиме ШИМ на фиксированную величину, регулирование достигается изменением скважности импульсов на открытие клапана.

Инициализация экземпляра:

```
pwm1 = PWM()
```

Методы блока:

PWM:run(InValue, Period, MinVlvVal)		
Входные параметры	InValue	Числовое значение. Текущее значение аналогового выхода регулятора
	Period	Числовое значение. Период ШИМ.

PWM:run(InValue, Period, MinVlvVal)		
	MinVlvVal	Числовое значение. Задание значения InValue, ниже которого начинается ШИМ-регулирование.
Выходные параметры	out (внутренний параметр)	Выход на устройство регулирования (например, клапан).
Описание При InValue > MinVlvVal на выход подается значение InValue. При InValue <= MinVlvVal на выход подается ШИМ-сигнал с амплитудой InValue. Скважность импульса зависит от величины InValue.		

Пример использования блока PWM:

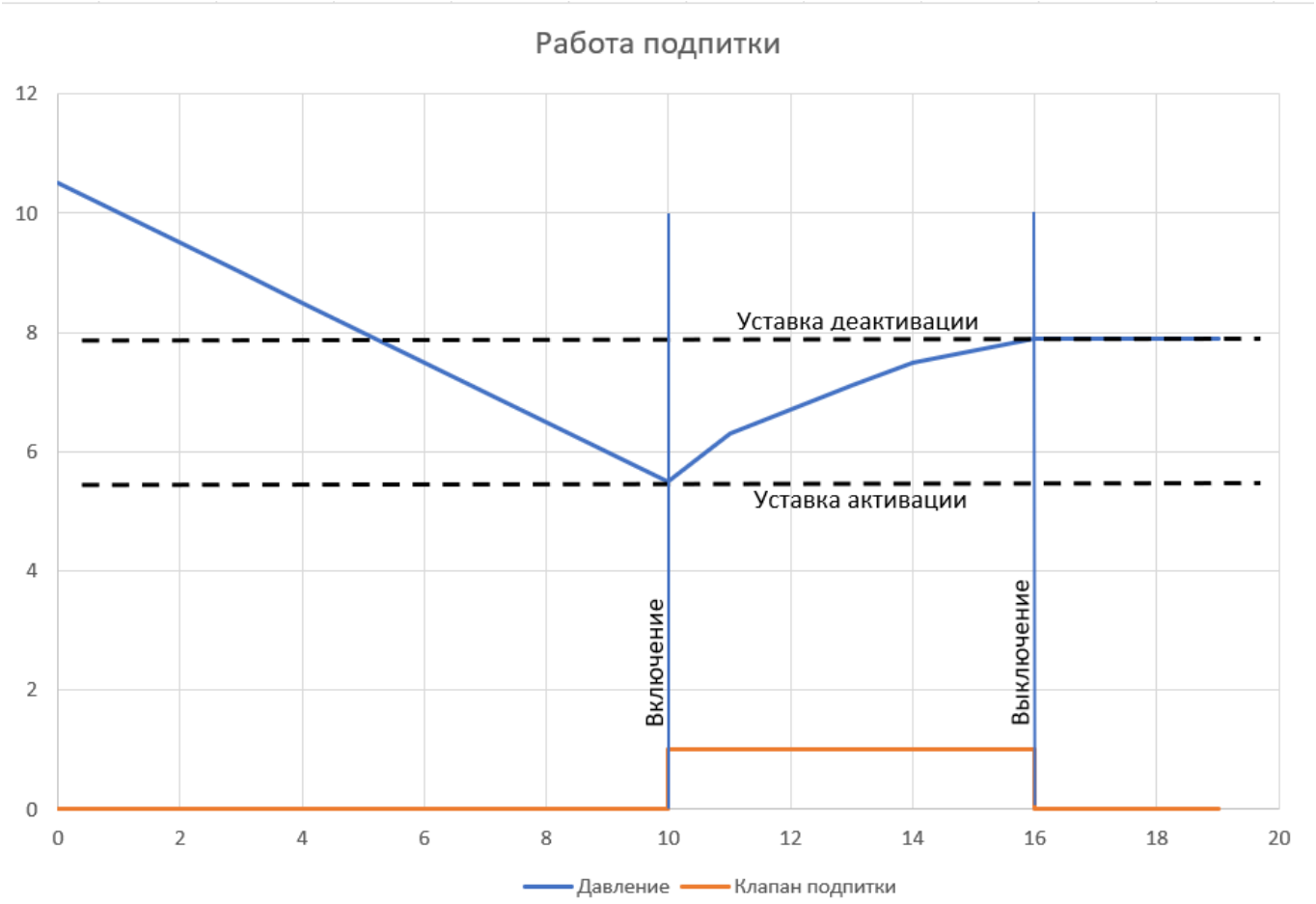
```
function init_lua_main()
    require("SeHVAC")
    pid1 = PID() -- инициализация блока ПИД
    pwm1 = PWM() -- инициализация блока ШИМ
end

function poll_deal()
    Dat = getIO("Dat") -- получение значения температуры притока
    SpTmpSpt = getIO("SuTmpSpt") -- получение уставки притока
    Start = getIO("Start")  -- получение команды пуска
    pid1:updatePidParam(2,120,0,120,1) -- конфигурирование ПИД
    local Mod
    if Start == 1 then -- определение режима ПИД
        Mod = 1
    else
        Mod = 3
    end
    pid1:run(Dat, SpTmpSpt, Mod, 0.05) -- работа ПИД
    local Valve = pwm1:run(pid1.out, 90, 8) -- работа ШИМ (при значении клапана больше
8% на выход подается значение ПИД, при значении ниже 8% на выход подается ШИМ с
амплитудой 8%)
    setIO("ClgVlvRef", Valve)
end
```

3.2.8. HYST

Блок гистерезиса с дискретным выходом со значениями 0 или 1. Задачей блока является формирование дискретного сигнала при пересечении сигнала заданного порога активации и снятие этого сигнала при переходе сигналом порога деактивации. В результате на выходе получается сигнал, устойчивый к малым флуктуациям измеряемой физической величины.

Одно из распространенных применений блока – это формирование режима Зима/Лето по заданным порогам перехода между ними. Например, для перехода в режим Лето задается порог 12°C, а для перехода в режим Зима задается порог в 8°C. Работа блока HYST обеспечивает активацию режима Зима при температуре наружного воздуха ниже 8°C, при превышении этой температуры режим не меняется и переключается в режим Лето только при температуре наружного воздуха выше 12°C. Такой алгоритм позволяет избежать ложных переключений режимов при измеряемой величине вблизи порога.



Инициализация экземпляра:

```
hyst1 = HYST()
```

Методы блока:

HYST:run(InValue, Act, Deact)		
Входные параметры	InValue	Сигнал числового типа.
	Act	Уставка активации выхода блока
	Deact	Уставка деактивации выхода блока
Выходные параметры	out (внутренний параметр)	Сигнал числового типа, значения 0 и 1.
Описание Работа блока зависит от соотношения уставок активации и деактивации: Если Act > Deact, то выход блока активируется при InValue > Act и деактивируется при InValue < Deact Если Act < Deact, то выход блока активируется при InValue < Act и деактивируется при InValue > Deact		

Пример использования блока HYST:

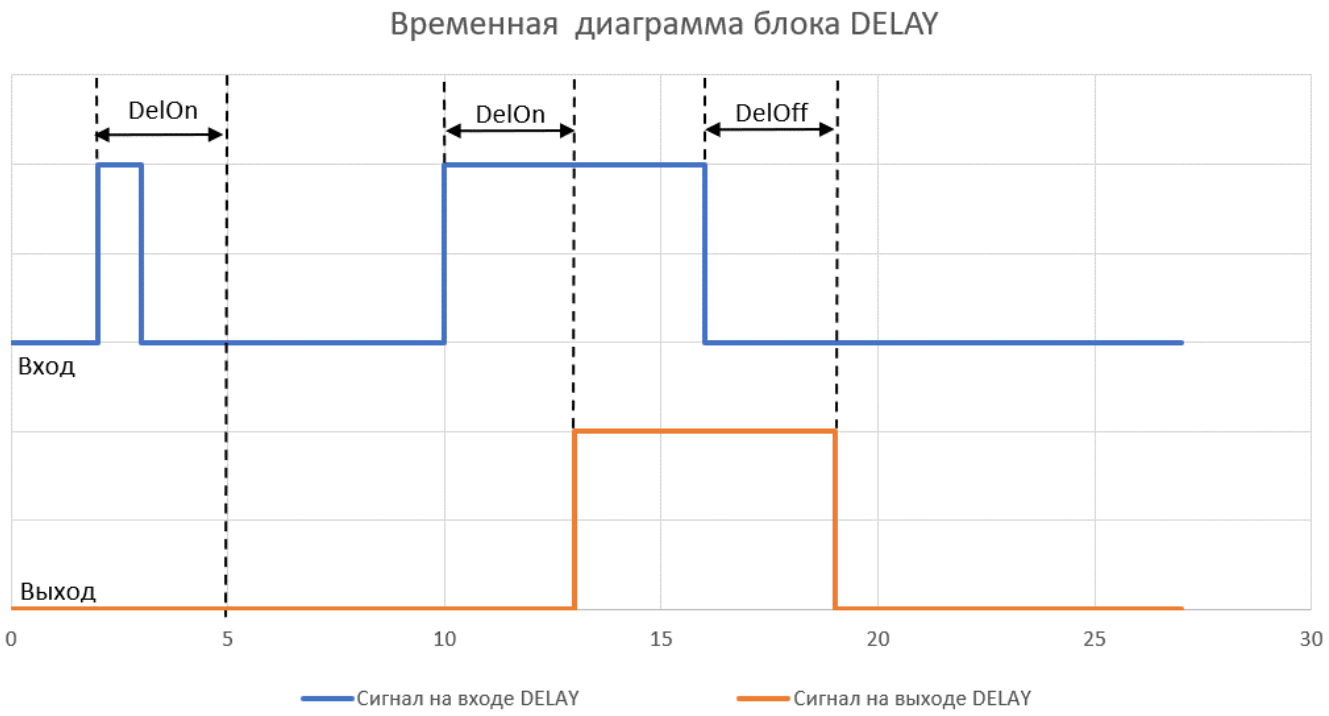
```
function init_lua_main()
    require("SeHVAC")
    hyst1 = HYST()
end

function poll_deal()
```

```
    PE = getIO("Htg1RetPr")                 -- значение давления в контуре
    PeSpt = getIO("Htg1RetPrSpt")           -- уставка давления
    Vlv = hyst1:run(PE, PeSpt - 0.5, PeSpt) -- запуск подпитки при давлении ниже уставки
олее, чем на 0.5 бар
    setIO("Htg1SubsVlvCmd", Vlv)
end
```

3.2.9. DELAY

Блок задержки дискретного сигнала. Доступны как задержка на активацию сигнала, так и задержка на деактивацию. Также доступен режим, когда доступны сразу оба типа задержек.



Инициализация экземпляра:

```
delay1 = DELAY()
```

Методы блока:

DELAY:run(InValue, DelOn, DelOff)		
Входные параметры	InValue	Сигнал числового типа со значениями 0/1 или типа boolean.
	DelOn	Задержка в секундах появления активного сигнала на выходе после поступления активного сигнала на вход.
	DelOff	Задержка в секундах деактивации сигнала на выходе после деактивации сигнала на входе.
Выходные параметры	out (внутренний параметр)	Сигнал числового типа, значения 0 и 1.
Описание Если в параметрт DelOn задать 0, то, получив активный сигнал на входе, блок выдаст активный сигнал на выходе без каких-либо задержек. Аналогично для параметра DelOff – при снятии активного сигнала на входе блока произойдет немедленное снятие активного сигнала на выходе.		

Пример использования блока DELAY:

```
function init_lua_main()
    require("SeHVAC")
    delay1 = DELAY()
    DelFaWrkSta = 10    -- задание задержки на сигнал запуска
    ALM.AL_Fa = 0       -- авария вентилятора
end

function poll_deal()
    FaWrkSta = getIO("FaWrkSta") -- получение статуса работы вентилятора
    if ALM.AL_Fa == 0 then
        Cmd = getIO("Start")    -- получение команды на запуск, если нет аварии
    else
        Cmd = 0                 -- блокировка работы при аварии
    end
    delay1:run(Cmd, DelFaWrkSta , 0) -- работа блока задержки сигнала
    if delay1.out == 1 and FaWrkSta == 0 then -- если через 10 секунд после запуска нет статуса работы вентилятора, то блокируем работу вентилятора и выставляем статусу аварии значение 1
        ALM.AL_Fa = 1
    end
end
```

3.2.10. RAMP

Блок замедления изменения сигнала. На вход блока подается изменяющийся во времени аналоговый сигнал, значение сигнала на выходе стремится к значению на входе, но скорость изменения сигнала на выходе ограничена параметром, задаваемым разработчиком.

Инициализация экземпляра:

```
ramp1 = RAMP(initVal)
или
ramp1 = RAMP()
```

Параметры инициализации блока		
initVal	Числовое значение	Начальное значение на выходе блока. Это значение устанавливает при инициализации и уже относительно этого значения начинает отслеживание значения на входе. Если при инициализации не задавать параметр initVal, то начальным значением на выходе будет 0.

Методы блока:

RAMP:run(InValue, Delta)		
Входные параметры	InValue	Сигнал числового типа.
	Delta	Максимальное приращение к выходному значению.
Выходные параметры	out (внутренний параметр)	Сглаженное значение входного сигнала

RAMP:run(InValue, Delta)
Описание Если изменение сигнала на входе больше, чем Delta, то тогда выходное значение блока изменяется на величину Delta: out = out_old + Delta, где out_old – значение выхода в предыдущем цикле. Каждый последующий цикл к значению на выходе будет добавляться Delta до тех пор, пока значение не выполнится следующее условие: InValue – out < Delta При выполнении этого условия на выход подается значение со входа (InValue).

RAMP:set(Value)		
Входные параметры	Value	Сигнал числового типа.
Описание При вызове этого метода происходит немедленное задание внутреннему параметру out значения Value. Таким образом, метод RAMP:run(InValue, Delta) вернет значение InValue вне зависимости от того, на сколько InValue отличается от out.		

Пример использования блока RAMP:

```
function init_lua_main()
    require("SeHVAC")
    ramp1 = RAMP()
    ALM.AL_Fa = 0       -- авария вентилятора
end

function poll_deal()
    ALM.AL_Fa = getIO("AL_Fa") -- получение статуса аварии
    if ALM.AL_Fa == 0 then
        Cmd = getIO("Start")   -- получение команды на запуск, если нет аварии
        if Cmd == 1 then
            SpdSpt = getIO("SuFaSpdSpt") --получение уставки скорости вентилятора
        else
            SpdSpt = 0
        end
        SpdRef = ramp1:run(SpdSpt, 2) -- ограничение изменения скорости вентилятора
значением 2% в секунду.
    else
        Cmd = 0                       -- блокировка работы при аварии
        SpdRef = ramp1:set(0)         -- быстрый останов вентилятора при аварии
    end
    setIO("FaSpdRef", SpdRef)   -- задание значения выходу контроллера
end
```

3.2.11. ALARM

Базовый объект для дискретных аварий. Блок позволяет фиксировать аварийное состояние по дискретному состоянию, а также устанавливать задержку на фиксацию аварийного состояния в случае «дребезга» входа.

Инициализация экземпляра:

```
alm1 = ALARM()
```

Методы блока:

ALARM:setParam(AlmVal, DelOn, Ack)		
Входные параметры	AlmVal	Значение типа boolean. Значение дискретного сигнала, принятое за аварийное.
	DelOn	Значение числового типа. Задержка в секундах на фиксацию аварийного состояния после прихода аварийного сигнала. Позволяет избежать ложной аварии при «дребезге» контактов.
	Ack	Значение типа boolean. Параметр, задающий необходимость сброса аварийной блокировки. True – сброс требуется False – сброс блокировки происходит автоматически как только аварийное условие перестает выполняться.
Описание Метод задает основные параметры отслеживания и фиксации аварийного состояния.		

ALARM:run(Sta)		
Входные параметры	Sta	Значение типа boolean. Отслеживаемый сигнал.
Выходные параметры	AlmSta (внутренний параметр)	Значение типа boolean. Состояние аварии
Описание Основной метод блока, отслеживающий значение сигнала Sta. Если сигнал Sta принимает значение AlmVal и удерживает это значение более DelOn секунд происходит активация аварийного состояния и параметр AlmSta принимает значение True. Если после фиксации аварийного состояния происходит деактивация сигнала Sta, значение внутреннего параметра AlmSta зависит от параметра Ack. Если Ack = True, то AlmSta сохраняет значение True до момента принудительного сброса, в противном случае AlmSta принимает значение False одновременно с деактивацией Sta.		

ALARM:rst()
Без параметров
Описание Метод осуществляет сброс аварийной блокировки аварийного статуса AlmSta при параметре Ack = True. После применения этого метода при неактивном сигнале Sta происходит присвоение параметру AlmSta значения False.

Пример использования блока ALARM:

```
function init_lua_main()
    require("SeHVAC")
    AL_Fire = ALARM()       -- пожарная тревога
    AL_Fire:setParam(true, 0, true) -- настройки блока
end

function poll_deal()
    AL_Fire:run(getIO("AL_Fire")) -- получение статуса аварии
    FrDmpCmd = AL_Fire.AlmSta -- при пожаре подать сигнал на закрытие ОЗК
    setIO("FrDmpCmd")   -- задание значения выходу управления ОЗК
    if getIO("AlmRst") == 1 then
        AL_Fire:rst()   -- при получении сигнала сброса аварии сбросить пожарную
блокировку
    end
end
```

3.2.12. DeviceObj

Базовый объект исполнительного механизма (ИМ). Задачей объекта является реализация функционала, характерного для всех ИМ для последующего расширения в объектах-потомках для более узких применений.

Объект обладает следующим функционалом:

- Управление ИМ в ручном или автоматическом режиме
- Отслеживание положения переключателя режимов Ручн/Авто/Откл
- Отслеживание обратной связи, для контроля исправности ИМ
- Отслеживание аварийного статуса, предоставляемого самим ИМ
- Фиксация аварийных блокировок и возможностью ручного сброса

Инициализация экземпляра:

```
device1 = DeviceObj()
```

Методы блока:

DeviceObj:setParam(FbkDelTm)		
Входные параметры	FbkDelTm	Значение числового типа. Задержка в секундах на ожидание обратной связи ИМ после подачи команды на включение. При непоступлении сигнала о работе в течение времени FbkDelTm фиксируется аварийное состояние, сигнал на запуск снимается.
Описание Метод задает параметры работы ИМ.		

DeviceObj:setAutH(AutSta, HSta)		
Входные параметры	AutSta	Значение числового типа. Статус положения Авто переключателя. Если переключатель в положении Авто, то сигнал должен быть равен True (или 1).

DeviceObj:setAutH(AutSta, HSta)		
	HSta	Значение числового типа. Статус положения Ручн переключателя. Если переключатель в положении Ручн, то сигнал должен быть равен True (или 1).
Описание Если у ИМ есть на шкафу управления переключатель Авто/Ручн/Откл, и положение этого переключателя отслеживается, то необходимо вызывать этот метод для передачи объекту положения этого переключателя.		

DeviceObj:run(Md, AutCmd, Fbk, AlmSta, EmrgStop, Rst)		
Входные параметры	Md	Значение числового типа. Режим работы ИМ, определяемый программно. 0 – принудительный останов 1 – принудительный запуск при отсутствии аварий 2 – автоматический запуск алгоритмом.
	AutCmd	Значение типа boolean или числового типа со значениями 0/1. Команда, включения ИМ в автоматическом режиме. Действует при Md = 2.
	Fbk	Значение типа boolean или числового типа со значениями 0/1. Сигнал обратной связи – работа. В случае отсутствия в системе такого сигнала, необходимо передать в качестве значения True.
	AlmSta	Значение типа boolean или числового типа со значениями 0/1. Статус аварийного состояния (обычно, аварийный контакт ИМ). При активном состоянии работа ИМ блокируется.
	EmrgStop	Значение типа boolean или числового типа со значениями 0/1. Сигнал экстренного останова. Может быть использован в качестве сигнала отключения без предварительных мероприятий.
	Rst	Сброс аварийных блокировок
Выходные параметры	DO (внутренний параметр)	Значение типа boolean. Команда запуска ИМ.
	Sta (внутренний параметр)	Общее состояние ИМ: 0 – отключен 1 – в работе 2 – аварийная блокировка

DeviceObj:run(Md, AutCmd, Fbk, AlmSta, EmrgStop, Rst)
Описание Основной метод блока. Обеспечивает непосредственную работу ИМ. Параметр Md задает режим работы ИМ: значение 0 принудительно останавливает ИМ, 1 – принудительно запускает ИМ при условии отсутствия аварийных блокировок, 2 – ИМ переводится в режим запуска от внешнего сигнала. Режим Md = 2 является основным режимом работы ИМ, остальные преимущественно используются в целях отладки. При Md = 2 команда на запуск (сигнал DO) подается при следующих условиях: • Сигнал AutCmd = true (или 1) – внешний сигнал запуска. • AlmSta = false (или 0) – отсутствуют аварийные сигналы от ИМ. • EmrgStop = false (или 0) – отсутствует внешняя экстренная блокировка. • Отсутствует внутренняя блокировка из-за отсутствия обратной связи во время предыдущего запуска: внутренний параметр DeviceObj.AL_Fbk.AlmSta = False. При выполнении всех условий, и при получении сигнала на запуск (AutCmd) выдается сигнал на запуск ИМ. В течение FbkDelTm секунд ожидается получение сигнала обратной связи Fbk. Если сигнал не получен, то внутренний объект аварии AL_Fbk фиксирует аварийное состояние ИМ и дальнейший запуск ИМ блокируется. Для повторного пуска ИМ требуется осуществить сброс аварийной блокировки.

DeviceObj:AlmRst()
Без параметров
Описание Метод осуществляет сброс аварийных блокировок.

Внутренние параметры и аварийные блокировки		
Внутренние параметры	DO	Выход управления ИМ
	Sta	Общий статус ИМ (0-остановлен, 1-в работе, 2-авария)
Аварийные блокировки	AL_Aut	Переключатель режимов ИМ находится не в положении Авто.
	AL_HStart	ИМ запущен переключателем (положение Ручн)
	AL_Fbk	Отсутствует обратная связь в течение времени, большего, чем FbkDelTm.
	AL_AlmSta	Получение аварийного сигнала от ИМ.

Пример использования блока DeviceObj:

```
function init_lua_main()
    require("SeHVAC")
    Pu = DeviceObj() -- инициализация насоса
end

function poll_deal()
    Pu:setParam(getIO("PuFbkDelTm")) -- задание времени обратной связи
    Pu:setAutH(getIO("PuAutSta"), getIO("PuHSta")) -- задание положения переключателя
    PuCmd, PuSta = Pu:run(      -- работа насоса
                        2,      -- режим запуска по внешнему сигналу
                        getIO("Ps01Sta"), -- статус реле давления
                        getIO("PuWrkSta"),-- статус работы насоса
                        getIO("StopBtn"), -- кнопка экстренного останова
                        getIO("AlmRst")   -- сигнал сброса аварий
```

```
    )
    setIO("PuCmd", PuCmd)    -- управление релейным выходом
    setIO("PuSta", PuSta)    -- отображение текущего статуса
    setIO("AL_PuFbk", Pu.AL_Fbk.AlmSta) -- индикация аварии обратной связи
    setIO("AL_PuAlmSta", Pu.AL_AlmSta.AlmSta) -- индикация неисправности ИМ
end
```

3.2.13. FCtrl

Объект управления преобразователем частоты. Унаследован от объекта DeviceObj, поэтому обладает всеми методами и параметрами родительского объекта. В описание будут отражены только отличные от родительского методы.

Объект обладает всем функционалом DeviceObj, а также :

- Отслеживание статуса перепада давления
- Плавный разгон и останов двигателя
- Быстрый останов двигателя при экстренной блокировке

Инициализация экземпляра:

```
fc1 = FCtrl()
```

Методы блока:

FCtrl:setParam(FbkDelTm, PdsDelTm, Delta)		
Входные параметры	FbkDelTm	Значение числового типа. Задержка в секундах на ожидание обратной связи ПЧ после подачи команды на включение. При непоступлении сигнала о работе в течение времени FbkDelTm фиксируется аварийное состояние, сигнал на запуск снимается.
	PdsDelTm	Значение числового типа. Задержка в секундах на ожидание сигнала сработки датчика перепада давления на насосе/вентиляторе. При непоступлении сигнала в течение времени PdsDelTm фиксируется аварийное состояние, сигнал на запуск снимается.
	Delta	Значение числового типа. Максимальное изменение производительности ПЧ в % в секунду.
Описание Метод задает параметры работы ПЧ.		

FCtrl:run(Md, AutCmd, Ref, Fbk, AlmSta, Pds, EmrgStop, Rst)		
Входные параметры	Md	Значение числового типа. Режим работы ПЧ, определяемый программно. 0 – принудительный останов 1 – принудительный запуск при отсутствии аварий 2 – автоматический запуск алгоритмом.
	AutCmd	Значение типа boolean или числового типа со значениями 0/1. Команда, включения ПЧ в автоматическом режиме. Действует при Md = 2.
	Ref	Значение числового типа. Уставка производительности ПЧ, %

FCtrl:run(Md, AutCmd, Ref, Fbk, AlmSta, Pds, EmrgStop, Rst)		
	Fbk	Значение типа boolean или числового типа со значениями 0/1. Сигнал обратной связи – работа. В случае отсутствия в системе такого сигнала, необходимо передать в качестве значения True.
	AlmSta	Значение типа boolean или числового типа со значениями 0/1. Статус аварийного состояния (обычно, аварийный контакт ПЧ). При активном состоянии работа ПЧ блокируется.
	Pds	Значение типа boolean или числового типа со значениями 0/1. Статус реле перепада давления.
	EmrgStop	Значение типа boolean или числового типа со значениями 0/1. Сигнал экстренного останова. Может быть использован в качестве сигнала отключения с быстрым остановом двигателя.
	Rst	Сброс аварийных блокировок
Выходные параметры	DO (внутренний параметр)	Значение типа boolean. Команда запуска ПЧ.
	AO (внутренний параметр)	Значение числового типа. Задание производительности ПЧ.
	Sta (внутренний параметр)	Общее состояние ПЧ: 0 – отключен 1 – в работе 2 – аварийная блокировка

Описание

Основной метод блока. Обеспечивает непосредственную работу ИМ.

Параметр Md задает режим работы ПЧ: значение 0 принудительно останавливает ПЧ, 1 – принудительно запускает ПЧ при условии отсутствия аварийных блокировок, 2 – ПЧ переводится в режим запуска от внешнего сигнала. Режим Md = 2 является основным режимом работы ПЧ, остальные преимущественно используются в целях отладки.

При Md = 2 команда на запуск (сигналы DO и AO) подается при следующих условиях:

- Сигнал AutCmd = true (или 1) – внешний сигнал запуска.
- AlmSta = false (или 0) – отсутствуют аварийные сигналы от ИМ.
- EmrgStop = false (или 0) – отсутствует внешняя экстренная блокировка.
- Отсутствует внутренняя блокировка из-за отсутствия обратной связи или отсутствия перепада давления во время предыдущего запуска: внутренние параметры FCtrl.AL_Fbk.AlmSta = False и FCtrl.AL_Pds.AlmSta = False.

При выполнении всех условий, и при получении сигнала на запуск (AutCmd) выдается сигнал на запуск ПЧ (DO = True), а также начинает плавное увеличение значения AO до уровня, заданного параметром Ref. Каждую секунду значение AO увеличивается не более, чем на Delta. В течение FbkDelTm секунд ожидается получение сигнала обратной связи Fbk. Если сигнал не получен, то внутренний объект аварии AL_Fbk фиксирует аварийное состояние ПЧ и дальнейший запуск ПЧ блокируется. Также после подачи сигнала на запуск и после достижения AO значения 30% начинается отсчет времени ожидания сработки датчика перепада давления. Если в течение PdsDelTm секунд не получен сигнал от датчика перепада внутренний объект AL_Pds фиксирует аварийное состояние и запуск двигателя прекращается. Для повторного пуска ПЧ требуется осуществить сброс аварийных блокировок.

Внутренние параметры и аварийные блокировки		
Внутренние параметры	DO	Выход управления ПЧ

Внутренние параметры и аварийные блокировки		
	AO	Выход управления производительностью ПЧ, %
	Sta	Общий статус ПЧ (0-остановлен, 1-в работе, 2-авария)
Аварийные блокировки	AL_Aut	Переключатель режимов ПЧ находится не в положении Авто.
	AL_HStart	ПЧ запущен переключателем (положение Ручн)
	AL_Fbk	Отсутствует обратная связь в течение времени, большего, чем FbkDelTm.
	AL_AlmSta	Получение аварийного сигнала от ПЧ.
	AL_Pds	Отсутствует сигнал наличия перепада давления через PdsDelTm секунд после достижения ПЧ производительности 30%.

Пример использования блока FCtrl:

```
function init_lua_main()
    require("SeHVAC")
    SuFa = FCtrl() -- инициализация приточного вентилятора
end

function poll_deal()
    SuFa:setParam(getIO("SuWrkDelTm"), getIO("SuFaPdsDelTm"), 2) -- задание времени
обратной связи
    SuFa:setAutH(getIO("SuFaAutSta"), getIO("SuFaHSta")) -- задание положения
переключателя
    SuFaCmd, SuFaSpdRef, SuFaSta = Pu:run(      -- работа вентилятора
                        2,      -- режим запуска по внешнему сигналу
                        getIO("Start"), -- статус команды запуска
                        getIO("SuFaSpdSpt") -- уставка скорости ПЧ
                        getIO("SuFaWrkSta"),-- статус работы ПЧ
                        getIO("SuFaAlmSta"), -- авария ПЧ
                        getIO("SuFaPdsSta"), -- статус перепада
                        getIO("StopBtn"), -- кнопка экстренного останова
                        getIO("AlmRst")   -- сигнал сброса аварий
    )
    setIO("FaSuCmd", SuFaCmd)   -- управление запуском ПЧ
    setIO("SuFaSpdRef", SuFaSpdRef) -- управление производительностью ПЧ
    setIO("SuFaSta", SuFaSta)   -- отображение текущего статуса
    setIO("AL_SuFaFbk", SuFa.AL_Fbk.AlmSta) -- индикация аварии обратной связи
    setIO("AL_SuFaAlmSta", SuFa.AL_AlmSta.AlmSta) -- индикация неисправности ПЧ
    setIO("AL_SuFaPds", SuFa.AL_Pds.AlmSta) -- авария отстуствия перпада давления
end
```

3.2.14. AHU_DmpDig

Объект управления воздушной заслонкой с двухпозиционным управлением – открыть/закрыть.

Объект обладает следующим функционалом:

- Управление воздушной заслонкой в автоматическом режиме по команде открытия.
- Возможность подачи сигнала на запуск вентилятора до полного открытия заслонки.

- Отслеживание статусов конечных выключателей
- Отслеживание времени открытия для определения заклинивания заслонки
- Обратный отсчет до окончания открытия заслонки
- Фиксация аварийных блокировок и возможностью ручного сброса

Инициализация экземпляра:

```
SuDmp = AHU_DmpDig()
```

Методы блока:

AHU_DmpDig:setParam(FbType, MovTm, FaCmdDelTm)		
Входные параметры	FbType	Значение числового типа. Задание типа обратной связи: 0 – концевые выключатели отсутствуют 1 – есть только статус ОТКРЫТО 2 – есть статусы ОТКРЫТО и ЗАКРЫТО
	MovTm	Значение числового типа. Задание времени полного хода воздушной заслонки. Необходимо для корректного отслеживания заклинивания заслонки
	FaCmdDelTm	Значение числового типа. Задержка в секундах на запуск вентилятора после подачи сигнала на открытие заслонки.
Описание Метод задает параметры работы заслонки.		

AHU_DmpDig:run(OpnSta, ClsSta, Ref, Rst)		
Входные параметры	OpnSta	Значение типа boolean или числового типа со значениями 0/1. Сигнал от концевого выключателя ОТКРЫТО. Отслеживается только при FbType > 0
	ClsSta	Значение типа boolean или числового типа со значениями 0/1. Сигнал от концевого выключателя ЗАКРЫТО. Отслеживается только при FbType = 2
	Ref	Значение типа boolean или числового типа со значениями 0/1. Сигнал команды открытия заслонки.
	Rst	Сброс аварийных блокировок
Выходные параметры	out (внутренний параметр)	Значение типа boolean. Команда открытия заслонки
	FaCmd (внутренний параметр)	Значение типа boolean. Команда на запуск вентилятора. Этот сигнал подается через FaCmdDelTm секунд после подачи команды на открытие заслонки.

AHU_DmpDig:run(OpnSta, ClsSta, Ref, Rst)

Описание

Основной метод блока. Обеспечивает непосредственную работу заслонки.

После подачи команды на открытие (Ref = 1 или Ref = True) блок осуществляет следующие действия:

- Выдача сигнала на открытие заслонки
- Начало обратного отсчета до получения статуса ОТКРЫТО
- Отсчет времени до подачи сигнала на запуск вентилятора
- Отслеживание времени открытия для определения заклинивания заслонки.

Если за MovTm секунд не был получен статус ОТКРЫТО, то фиксируется аварийное состояние и команда на открытие заслонки снимается.

После снятия команды на открытие снимается сигнал на открытие заслонки и начинается отсчет времени закрытия. Если за MovTm секунд не получен сигнал ЗАКРЫТО, то выдается аварийный статус.

При отсутствии какого-либо концевого выключателя соответствующие части контроля конечных положений не участвуют в алгоритме.

Для сброса аварийных блокировок необходимо задать параметру Rst значение True или 1.

Внутренние параметры и аварийные блокировки

Внутренние параметры	out	Сигнал типа boolean. Выход управления заслонкой
	FaCmd	Сигнал типа boolean. Выход управления вентилятором
	EstTmOpn	Целочисленное значение. Время, оставшееся до открытия заслонки
	EstTmCld	Целочисленное значение. Время, оставшееся до закрытия заслонки
	OpnTmCnt	Время, прошедшее с начала открытия заслонки
	ClsTmCnt	Время, прошедшее с начала закрытия заслонки
Аварийные блокировки	AL_Opn	Авария заслонки: не открылась
	AL_Cls	Авария заслонки: не закрылась

Пример использования блока AHU_DmpDig:

```
function init_lua_main()
    require("SeHVAC")
    SuDmp = AHU_DmpDig() -- инициализация заслонки притока
end

function poll_deal()
    SuDmp:setParam(2, 150, 20) -- задание параметров заслонки
    SuDmpRef, SuFaCmd = SuDmp:run(      -- работа заслонки
                        getIO("SuDmpOpnSta"), -- статус сигнала ОТКРЫТО
                        getIO("SuDmpClsSta") -- статус сигнала ЗАКРЫТО
                        getIO("Start"),-- команда на запуск
                        getIO("AlmRst")   -- сигнал сброса аварий
    )
    setIO("SuDmpRef", SuDmpRef) -- выдача сигнала на открытие
    setIO("SuFaCmd", SuFaCmd)   -- управление запуском вентилятора
    setIO("SuDmpEstTm", SuDmp.EstTmOpn) -- оставшееся время до открытия
    setIO("AL_SuDmpOpn", SuDmp.AL_Opn.AlmSta) -- индикация аварии открытия
    setIO("AL_SuDmpCls", SuDmp.AL_Cls.AlmSta) -- индикация аварии закрытия
end
```

3.2.15. AHU_FltDig

Объект воздушного фильтра с контролем загрязнения с помощью реле перепада давления.

Объект обладает следующим функционалом:

- Контроль сигнала от реле перепада давления
- Формирование аварийного статуса при наличии перепада.

Инициализация экземпляра:

```
SuFlt1 = AHU_FltDig()
```

Методы блока:

AHU_FltDig:run(Pds)		
Входные параметры	Pds	Значение типа boolean или числового типа со значениями 0/1. Сигнал от реле перепада давления на фильтре
Выходные параметры	AL_Pds.AlmSta (внутренний параметр)	Значение типа boolean. Аварийный статус загрязнения фильтра.
Описание Основной метод блока. Обеспечивает контроль загрязнения. При получении сигнала от реле перепада давления на фильтре формируется аварийный статус.		

Внутренние параметры и аварийные блокировки		
Аварийные блокировки	AL_Pds	Авария загрязнения фильтра.

Пример использования блока AHU_FltDig:

```
function init_lua_main()
    require("SeHVAC")
    SuFlt1 = AHU_FltDig() -- инициализация фильтра
end

function poll_deal()
    SuFlt1:run(getIO("SuFlt1PdsSta")) -- отслеживание датчика перпада
    setIO("AL_SuFlt1", SuFlt1.AL_Pds.AlmSta) -- индикация аварии загрязнения
end
```

3.2.16. AHU_Summer

Объект определения текущего сезонного режима Зима или Лето.

Объект обладает следующим функционалом:

- Автоматическое определение сезонного режима по температуре наружного воздуха.

- Принудительно задание сезонного режима оператором
- Задание параметров автоматического перехода между сезонами.

Инициализация экземпляра:

```
Summer = AHU_Summer()
```

Методы блока:

AHU_Summer:setParam(SwToSummerVal, SwToWinterVal)		
Входные параметры	SwToSummerVal	Значение числового типа. Уставка перехода на летний режим
	SwToWinterVal	Значение числового типа. Уставка перехода на зимний режим
Описание Метод задает параметры перехода между сезонными режимами в автоматическом режиме по температуре наружного воздуха.		

AHU_Summer:run(Md, Oat)		
Входные параметры	Md	Значение числового типа. 0 – принудительно задан режим Зима 1 – принудительно задан режим Лето 2 – сезонный режим определяется по температуре наружного воздуха
	Oat	Значение числового типа. Значение температуры наружного воздуха
Выходные параметры	out (внутренний параметр)	Значение типа boolean. Статус сезонного режима: False – зимний режим True – летний режим
Описание Основной метод блока. Обеспечивает задание сезонного режима. Режим работы блока определяется параметром Md. При значениях, отличных от 2, сезонный режим определяется параметром Md, при значении Md, равном 2, сезонный режим определяется по температуре наружного воздуха (параметр Oat) и по уставкам перехода между режимами SwToSummerVal и SwToWinterVal		

.

Пример использования блока AHU_Summer:

```
function init_lua_main()
    require("SeHVAC")
    Summer = AHU_Summer() -- инициализация блока
end

function poll_deal()
    Summer:setParam(
                getIO("SwToSummerVal"),
                getIO("SwToWinterVal")) -- задание параметров сезонного перехода
    Summer:run(getIO("SummerSwitch"), getIO("Oat")) -- работа блока
    setIO("Summer", Summer.out) -- индикация текущего сезона
end
```

3.2.17. AHU_Start

Объект осуществляет общий запуск установки либо в ручном режиме, либо по расписанию. При аварийной блокировке запуск установки прекращается.

Инициализация экземпляра:

```
Start = AHU_Start()
```

Методы блока:

AHU_Start:run(Md, Schedule, Block)		
Входные параметры	Md	Значение числового типа. 0 – принудительно задан режим Останов 1 – принудительно задан режим Старт 2 – режим запуска по расписанию
	Schedule	Сигнал запуска по расписанию
	Block	Сигнал блокировки запуска
Выходные параметры	prestart (внутренний параметр)	Значение типа boolean. Статус команды на запуск установки
Описание Основной метод блока. Обеспечивает запуск установки. Режим работы блока определяется параметром Md. При значениях, отличных от 2, режим определяется параметром Md, при значении Md, равном 2, запуск установки осуществляется сигналом от расписания. При Block = True запуск установки при любом значении Md отменяется.		

Пример использования блока AHU_Start:

```
function init_lua_main()
    require("SeHVAC")
    Start = AHU_Start() -- инициализация блока
end

function poll_deal()
    Schedule = getIO("Schedule") -- получение значения расписания
    Block = getIO("CmnAlm") -- получение статуса общей аварии
    StartSwitch = getIO("StartSwitch") -- получение режима работы
    Start:run(StartSwitch, Schedule, Block) -- работа блока
    setIO("Start", Start.prestart) -- индикация запуска установки
end
```

3.2.18. AHU_Preheat

Объект предназначен для осуществления предстартового прогрева калорифера вентиляционной установки в зимнем режиме работы.

Объект обладает следующим функционалом:

- Выдача сигнала на предстартовый прогрев при поступлении сигнала запуска установки в зимнем режиме
- Выдача разрешающего сигнала на запуск при удачном прогреве
- Фиксация аварийного состояния при недостижении параметров прогрева за заданное в настройках время.

Инициализация экземпляра:

```
Preheat = AHU_Preheat()
```

Методы блока:

AHU_Preheat:setParam(minPreheatTm, maxPreheatTm)		
Входные параметры	minPreheatTm	Значение числового типа. Минимальное время в секундах, которое должен продлиться предварительный прогрев. Как правило равен времени полного хода регулирующего клапана. Предназначен для предотвращения запуска вентилятора при закрытом клапане.
	maxPreheatTm	Значение числового типа. Максимальное время в секундах, которое может продлиться прогрев. При недостижении требуемых параметров теплоносителя за это время фиксируется аварийное состояние и блокируется запуск установки.
Описание Метод задает параметры времени прогрева.		

AHU_Preheat:run(prestart, RwtTmpSpt, RwtTmp, SummerSta, Rst)		
Входные параметры	prestart	Значение типа boolean. Статус команды на запуск установки. Чаще всего этот сигнал поступает от блока AHU_Start.
	Rwt	Значение числового типа. Значение температуры обратного теплоносителя.
	RwtSpt	Значение числового типа. Текущее значение уставки температуры обратного теплоносителя в режиме прогрева.
	SummerSta	Значение типа boolean. Статус сезонного режима: False – Зимний режим True – Летний режим
	Rst	Сброс аварийной блокировки
Выходные параметры	start (внутренний параметр)	Значение типа boolean. Сигнал разрешения на запуск
	prehSta (внутренний параметр)	Значение типа boolean. Статус режима прогрева.
	alarm.AlmSta	Значение типа boolean. Статус аварийного состояния

AHU_Preheat:run(prestart, RwtTmpSpt, RwtTmp, SummerSta, Rst)
Описание Основной метод блока. Обеспечивает предварительный прогрев калорифера. У блока есть два режима работы: зимний и летний режимы, определяемые по параметру SummerSta. В зимнем режиме работы при поступлении сигнала на запуск (prestart) происходит запуск алгоритма предпускового прогрева – параметр prehSta получает значение True. Далее, этот сигнал следует подать на блок жидкостного нагревателя для открытия клапана нагрева на 100% (). В режиме прогрева контролируется температура обратного теплоносителя и уставки минимального и максимального времени прогрева. Если температура обратного теплоносителя доходит до уставки RwtSpt до окончания времени прогрева, то он считается успешным, статус prehSta снимается и выдается разрешение на открытие воздушных заслонок и запуск вентилятора – параметр start. Даже, если температура теплоносителя при начале прогрева уже достаточна для запуска, разрешение на запуск дается через время minPreheatTm. Это сделано для того, чтобы предотвратить запуск вентилятора при закрытом клапане. В летнем режиме алгоритм прогрева неактивен и сигнал start отображает значение входного параметра prestart.

Пример использования блока AHU_Preheat:

```
function init_lua_main()
    require("SeHVAC")
    Preheat = AHU_Preheat() -- инициализация блока
end

function poll_deal()
    Rwt = getIO("Htg1Rwt") -- получение значения температуры обратной воды
    PrhtRwtSpt = getIO("PrhtRwtSpt") -- получение уставки обр. воды
    prestart = getIO("prestart") -- получение команды на запуск
    summer = getIO("Summer") -- статус сезонного режима
    AlmRst = getIO("AlmRst") -- сигнал сброса аварии
    Preheat:setParam(60, 500) -- настройка параметров прогрева
    Preheat:run(prestart, PrhtRwtSpt, Rwt, summer, AlmRst) -- работа блока
    setIO("Start", Preheat.start) -- индикация запуска установки
    setIO(AL_Prht, Preheat.alarm.AlmSta) -- индикация статуса аварии
end
```

3.2.19. AHU_WatHtg

Объект отвечает за управление калорифером вентиляционной установки с жидкостным теплоносителем.

Объект обладает следующим функционалом:

- Управление регулирующим клапаном нагрева, поддержание заданной температуры притока.
- Контроль температуры обратного теплоносителя, ограничение закрытия клапана при недостаточной температуре.
- Контроль состояния капиллярного термостата угрозы заморозки.
- Управление и контроль работы циркуляционного насоса.
- Открытие клапана при предварительном прогреве.
- Защитное открытие клапана при аварийных ситуациях.

- Защитное снижение производительности вентилятора при недостаточной температуре притока.

Инициализация экземпляра:

```
WatHtg1 = AHU_WatHtg()
```

Методы блока:

AHU_WatHtg:setPid(G, Ti, Td, Dz, Stroke, FaRedEna)		
Входные параметры	G	Значение числового типа. Коэффициент пропорциональности ПИД
	Ti	Значение числового типа. Время интегрирования ПИД
	Td	Значение числового типа. Время дифференцирования ПИД
	Dz	Значение числового типа. Мертвая зона регулирования ПИД
	Stroke	Значение числового типа. Время полного хода клапана в секундах
	FaRedEna	Значение типа boolean или числового типа со значениями 0/1. Разрешение защитного снижения производительности вентилятора при невыходе на уставку в режиме Зима.
Описание Метод задает параметры ПИД-регулятора контура нагрева.		

AHU_WatHtg:getIO(Dat, Rwt, TsSta, VlvPosSta, PuFbk, PuAlmSta, PuAutSta, PuHSta)		
Входные параметры	Dat	Значение числового типа. Температура притока
	Rwt	Значение числового типа. Температура обратного теплоносителя
	TsSta	Значение типа boolean или числового типа со значениями 0/1. Статус сработки капиллярного термостата угрозы заморозки
	VlvPosSta	Значение числового типа. Обратная связь регулирующего клапана. При отсутствии этого сигнала необходимо подать сюда значение выхода управления клапаном
	PuFbk	Значение типа boolean или числового типа со значениями 0/1. Статус работы циркуляционного насоса. В случае отсутствия сигнала необходимо подать True.
	PuAlmSta	Значение типа boolean или числового типа со значениями 0/1. Статус аварии циркуляционного насоса. В случае отсутствия сигнала необходимо подать False.
	PuAutSta	Значение типа boolean или числового типа со значениями 0/1. Статус положения Авто трехпозиционного переключателя режимов управления
	PuHSta	Значение типа boolean или числового типа со значениями 0/1. Статус положения Ручн трехпозиционного переключателя режимов управления
Описание Метод передает значения физических сигналов в объект.		

AHU_WatHtg:setPu(FbkPuDelTm)		
Входные параметры	FbkPuDelTm	Значение числового типа. Задержка на получение статуса работы циркуляционного насоса после его запуска.

Описание

Метод задает параметры работы модуля управления циркуляционным насосом. Кроме задания времени обратной связи этот метод осуществляет передачу параметров положения трехпозиционного переключателя.

AHU_WatHtg:setSafeSet(DatLoSpt, FaWrkSta, CurRwtSpt, PrhtTmpSpt, FrSta, PwrSta, SummerSta, Rst)		
Входные параметры	DatLoSpt	Значение числового типа. Нижняя аварийная уставка температуры притока
	FaWrkSta	Значение типа boolean или числового типа со значениями 0/1. Статус работы приточного вентилятора
	CurRwtSpt	Значение числового типа. Текущая уставка температуры обратного теплоносителя
	PrhtTmpSpt	Значение числового типа. Текущая уставка обратного теплоносителя при предварительном прогреве
	FrSta	Значение типа boolean или числового типа со значениями 0/1. Статус пожарной тревоги
	PwrSta	Значение типа boolean или числового типа со значениями 0/1. Статус наличия питания шкафа управления
	SummerSta	Значение типа boolean или числового типа со значениями 0/1. Статус сезонного режима работы
	Rst	Значение типа boolean или числового типа со значениями 0/1. Сигнал сброса аварий

Описание

Метод осуществляет проверку аварийных ситуаций и обеспечивает защиту калорифера.

Метод реализует следующие защиты:

- Контроль состояния капиллярного термостата – при сработке блокируется работа вентилятора, регулирующий клапан открывается на 100%
- Контроль температуры притока. При снижении температуры притока до значения DatLoSpt происходит блокировка работы установки и открывается регулирующий клапан на 100%
- Контроль температуры обратного теплоносителя. При снижении температуры обратной воды ниже значения CurRwtSpt более, чем на 4°C фиксируется аварийная ситуация и регулирующий клапан открывается на 100%.
- Контроль наличия питания шкафа автоматики. При отсутствии питания регулирующий клапан открывается на 100%.
- Контроль пожарной тревоги. При сработке пожарной тревоги работа установки блокируется и регулирующий клапан открывается на 100%.

При открытии клапана при сработке защиты начинает отслеживаться температура обратного теплоносителя и при достижении уставки температуры прогрева (PrhtTmpSpt), при условии отсутствия пожарной тревоги и наличии питания шкафа автоматики, регулирующий клапан переводится в режим поддержания температуры обратной воды в целях предотвращения завышения температуры обратного теплоносителя в контуре вентиляции в ИТП. При этом блокировка запуска установки сохраняется и снимается сигналом сброса аварии.

AHU_WatHtg:runPu(MdPu, Rst)		
Входные параметры	Md	Значение числового типа. Режим работы насоса, определяемый программно. 0 – принудительный останов 1 – принудительный запуск при отсутствии аварий 2 – автоматический запуск алгоритмом.
	Rst	Сброс аварийной блокировки
Описание Метод работы с циркуляционным насосом контура нагрева. В зимнем режиме работы контура происходит запуск насоса и контроль обратной связи (статус работы или статус аварии насоса). Работа метода влияет на значения внутренних объектов, определяющих текущее состояние насоса.		

AHU_WatHtg:runVlv(Start, prehSta, SuTmpSpt, HMd, HVal)		
Входные параметры	Start	Статус запуска установки
	prehSta	Статус состояния предварительного прогрева калорифера
	SuTmpSpt	Уставка температуры притока
	HMd	Статус ручного режима управления клапаном
	HVal	Управление клапаном в ручном режиме
Описание Метод управления клапаном регулирования. В дежурном режиме (сигнал Start неактивен) регулирование осуществляется только для поддержания требуемой температуры обратной воды. При запуске установки к контуру поддержания обратной воды добавляется контур поддержания температуры притока – на клапан передается большее из двух воздействий. При активном сигнале статуса предварительного прогрева (prehSta) клапан остается в полностью открытом положении. После успешного окончания прогрева регулирование разрешается либо после получения сигнала работы вентилятора, либо при отключении установки. Для ручного управления клапаном необходимо задать параметру HMd значение 1 (True). После этого на клапан будет передаваться значение параметра HVal.		

Внутренние параметры и аварийные блокировки		
Внутренние параметры	PuCmd	Сигнал типа boolean. Выход управления насосом
	PuSta	Целочисленное значение. Общий статус насоса (0-остановлен, 1-в работе, 2-авария)
	ValveRef	Числовое значение. Выход управления клапаном регулирования (0-100%)
Аварийные блокировки	AL_Ts	Сработка термостата
	AL_DatLo	Низкая температура приточного воздуха
	AL_RwtLo	Низкая температура обратной воды
	AL_PuFault	Получен сигнал неисправности насоса
	AL_PuNoFbk	Отсутствует сигнал обратной связи насоса после подачи команды на запуск

Пример использования блока AHU_WatHtg:

```
function init_lua_main()
    require("SeHVAC")
    Htg1 = AHU_WatHtg() -- инициализация блока
    G = 2           -- параметры ПИД
    Ti = 120        -- параметры ПИД
```

```
    Td = 0          -- параметры ПИД
    Dz = 0.05       -- параметры ПИД
    Stroke = 120    -- параметры ПИД
    SptLowAirTmp = 7-- параметры защиты
    MdPu = 2        -- режим насоса (автоматический)
    FbPuDelTm = 10  -- задержка обратной связи насоса
    FaRedEna = 1  -- разрешение на защитное снижение скорости вентилятора
    Htg1:setPid(G, Ti, Td, Dz, Stroke, FaRedEna) -- задание параметров регулирования
end

function poll_deal()
    Dat = getIO("Dat")              -- температура притока
    Rwt = getIO("Rwt")              -- температура обратной воды
    TsSta = getIO("TsSta")          -- статус термостата
    PuWrkSta = getIO("PuWrkSta")    -- статус насоса
    VlvPos = getIO("VlvPos")        -- положение клапана
    FaWrkSta = getIO("FaWrkSta")    -- статус вентилятора
    FrSta = getIO("FrSta")          -- пожарная тревога
    PwrSta = getIO("PwrSta")        -- наличие питания
    SpTmpSpt = getIO("SuTmpSpt")    -- уставка притока
    CurRwtSpt = getIO("CurRwtSpt")  -- уставка обратной воды
    SptRwtPrht = getIO("SptRwtPrht") -- уставка прогрева
    Start = getIO("Start")          -- получение команды пуска
    Summer = getIO("Summer")        -- статус режима Лето
    Rst = getIO("Rst")              -- сброс аварий
    prehSta = getIO("prehSta")      -- статус прогрева
    HMd = getIO("VlvHMd")           -- ручной режим клапана
    HVal = getIO("VlvHVal")         -- ручное управление клапаном

    -- получение физический сигналов
    Htg1:getIO(Dat, Rwt, TsSta, VlvPos, PuWrkSta, false, true, false)
    -- задание параметров насоса
    Htg1:setPu(FbPuDelTm)
    -- задание параметров защит
    Htg1:setSafeSet(SptLowAirTmp, FaWrkSta, CurRwtSpt, SptRwtPrht, FrSta, PwrSta,
Summer, Rst)
    -- управление насосом
    Htg1:runPu(MdPu, Rst)
    -- управление клапаном
    Htg1:runVlv(Start, prehSta, SpTmpSpt, HMd, HVal)

    setIO("PuCmd", Htg1.PuCmd)
    setIO("VlvRef", Htg1.ValveRef)
end
```

Systême soft

Узнать обо всех продуктах
Systeme Soft

https://systemesoft.ru/